Approches de résolution multiobjective séquentielle et parallèle pour les réseaux de transports multimodaux

Hedi Ayed

UNIVERSITÉ PAUL VERLAINE- METZ
UFR MIM

École doctorale IAEM Lorraine

Approches de résolution multiobjective séquentielle et parallèle pour les réseaux de transports multimodaux

THÈSE

présentée et soutenue publiquement le 10 Novembre 2011

pour l'obtention du

Doctorat de l'université-Paul-Verlaine, Metz

(spécialité informatique)

par

Hedi Ayed

Composition du jury

Rapporteurs :	Pr. Pierre Borne	Ecole centrale de Lille
	Pr. Juan Carlos Burguillo-Rial	Université de Vigo, Espagne
Examinateurs :	Pr. Marie-Claude Portmann	LORIA-Nancy
	Pr. Jacques Carlier	Université de Compiègne
	Pr. Frédéric Guinand	LITIS Le Havre
	Pr. Imed Kacem	LITA-UPV Metz
Encadrants :	Pr. Zineb Habbas	LITA-Université Paul-Verlaine-Metz
	Dr. Djamel Khadraoui	Chercheur au CRPHT, Luxembourg

Laboratoire d'Informatique Théorique et Appliquée — EA 3097

Remerciements

Je tiens á saluer ici les personnes qui, de près ou de loin, ont contribué á la concrétisation de ce travail de thèse de doctorat. Ces remerciements sont rédigés dans un moment de doux relâchement intellectuel, sans véritable rigueur ni souci taxinomique. Dans un autre état d'esprit, ces remerciements auraient certainement été tout autres, et j'aurais peut-être oublier un des noms qui suivent. Mais j'ai choisi ce moment précis pour les écrire.

Tout d'abord, mes remerciements s'adressent à Zineb Habbas, Professeur à L'université de Metz, ma Directrice de thèse et à Monsieur Djamel Khadraoui, Docteur-Chercheur au Centre de Recherche Henri Tudor mon Co-Directeur de thèse, pour m'avoir proposé le sujet de thèse et qui m'ont encadré tout au long de ces années d'étude. Au travers de nos discussions, ils m'ont apporté une compréhension plus approfondie des divers aspects du sujet. Je salue aussi la souplesse et l'ouverture d'esprit de mes encadrants de thèse qui ont su me laisser une large marge de liberté pour mener á bien ce travail de recherche.

Je suis très reconnaissant au professeur **Pierre Borne** et au professeur **Juan Carlos Burguillo-Rial** d'avoir accepté le rôle de rapporteurs. Les commentaires et les questions de ces personnalités scientifiques, tant sur la forme du mémoire que sur son fond, ont contribué á améliorer de manière significative le document.

Je tiens à remercier aussi l'ensemble des examinateurs : le professeur émerite **Marie Claude Portmann**, le professeur **Jacques Carlier**, le professeur **Fréderique Guinand**, et le professeur **Imed Kacem**, pour leur présence au jury ainsi que pour leurs nombreuses questions et remarques pertinentes.

Ma gratitude s'adresse aussi á mes collègues de bureau du CRP Henri Tudor, Carlos Galvez et Gerald Arnould. Tout d'abord pour leur accueil, mais surtout pour les nombreuses discussions que nous avons eu dans les domaines de l'optimisation, programmation et développement, pour leurs conseils, et pour leur aide inestimable en fin de thèse. De même, je remercie mes collègues, Wided Guedria, Brahim Batouche, Mehdi Mouad et Moussa Ouedraogo, qui m'ont aidé á répondre á un certain nombre de questions pratiques, techniques, ou scientifiques, que je me suis posées.

Je remercie tous ceux qui ont contribué á faire du CRP un endroit sympathique où je me suis rendu avec plaisir pendant un peu moins de quatre ans, ainsi que le FNR (Fonds National de la Recherche Luxembourg) pour avoir financer ce travail.

Enfin, je tiens á remercier ma famille et mes amis, qui m'ont accompagné et soutenu au cours de ces années.

A ma très chère mère Mongia,
Tu es l'exemple de dévouement qui n'a pas cessé de m'encourager et de prier pour moi. Puisse Dieu, le tout puissant, te préserver et t'accorder santé, longue vie et bonheur.

A mon père Ayoub,
Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être. Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation.

A mon onéreuse épouse Wissal,
Ce travail te doit beaucoup... Qu'il soit pour toi le témoignage de mon infinie reconnaissance pour ces années de compréhension, de privations et d'efforts communs.

A Aziz,
notre petit, tout étonné que son papa ait enfin terminé "sa thèse".

A mes frères Nour et Amine, a ma soeur Zeineb,
affectueuses reconnaissances.

A ma belle-famille, mére, pére, fréres et soeures,
Vous qui m'admirez tant, soyez sûrs que ce travail est le résultat de votre confiance en moi. Soyez-en rémerciés.

Table des matières

Introduction générale

Introduction

1 Contexte de cette thèse

L'émergence des technologies de l'information dans les transports a conduit, au cours de la dernière décennie, à un nombre important d'idées novatrices et à des solutions qui ont contribué à la viabilité économique de nombreuses villes dans le monde. Les technologies de l'information jouent un rôle clé dans l'optimisation de l'expérience utilisateur de bout en bout, qu'elles soient utilisées dans le cadre du processus décisionnel, dans le choix d'un mode de déplacement à utiliser, ou pour vérifier l'état du service de transport. Leur influence s'est d'autant plus accrue que l'utilisateur est prêt à payer pour ces nouveaux services, via des applications mobiles, des messages texte ou même des abonnements personnalisés.

Cette évolution est fondamentale par rapport aux fiches horaires imprimées, à la radio et la télévision qui constituaient la source principale d'informations de voyage il n'y a de cela que quelques années. Dans les transports publics, les paiements s'effectuaient la plupart du temps «en liquide», et les smartphones qui inondent le marché de nos jours n'étaient alors qu'à l'état d'idées – ou au mieux de prototype – dans les cartons des industriels de la téléphonie mobile. Mais aujourd'hui la donne n'est plus la même : les utilisateurs de smartphones se sentent perdus dans les villes sans l'accès à l'information que cette technologie a permis. Les projections montrent même que d'ici 2020, la plupart des trajets réalisés et utilisant plusieurs modes de transport impliqueront forcément l'utilisation de smartphones afin de fournir des informations et des mises à jour en temps réel sur les horaires, les correspondances et les conditions de circulation, ainsi que pour le paiement des voyages.

Cette innovation des technologies a des impacts sur les modes de transport :

- **Le recours au transport multimodal** : Alors que la voiture reste le mode dominant pour se déplacer dans de nombreuses villes, cette tendance a commencé à baisser dans les zones où le transport public est fréquent et fiable, qui fournit une bonne qualité de l'information en temps réel, et disposant de systèmes de billetterie intégrés facile d'utilisation par les passagers. Cela conduira inévitablement à l'émergence de plus en plus de villes, voire même des pays, à choisir des systèmes de transport multimodaux entièrement intégrés. La voiture continuera effectivement à jouer un rôle important dans le choix modal pour des voyages au sein des villes jusqu'en 2020. Cependant, avec l'augmentation des coûts automobile (notamment le carburant), l'innovation et la technologie facilitera les changements qui ont déjà commencé à prendre place dans certaines villes. L'émergence des solutions d'auto-partage, de covoiturage et des véhicules électriques, avec des applications mises au point pour soutenir ces tendances, sont autant d'exemples qui illustrent ces changements de comportements.

- **La nécessité de traiter des réseaux dépendant du temps** : Des initiatives sont en cours dans le cadre de développements de solutions, qui permettront d'optimiser l'information voyageurs en temps réel pour les bus, trains et pour les autres modes de transport partagés. Les utilisateurs auront la route optimale pour leur voyage, en fonction de leur emplacement actuel ainsi que de la disponibilité en temps réel des places sur les trains de métro les bus, et sur le réseau partagé de transport de véhicules privés dotés de GPS et de téléphones mobiles.
- **Le Problème d'accessibibilité aux données** : Les défis principaux pour réussir des solutions multimodales en temps réel et intégrées résident dans la cohérence des données et des systèmes de transport. Dans certains cas les opérateurs de transport doivent entreprendre une analyse profonde afin d'harmoniser l'ensemble de données. Il est nécessaire dans ce contexte de décider de qui est propriétaire des données, et qui les gère. Les systèmes de transport entièrement intégrés que les villes commencent à adopter peuvent s'appuyer sur un certain nombre d'acteurs fournissant des ensembles de données ou d'accéder, chacun avec différents intérêts et besoins.

2 Problématique

Cette importance accrue des systèmes de transport a attiré la communauté scientifique à s'intéresser à la problématique de transport et à celle du transport multimodal plus récemment. En effet, le transport multimodal a fait l'objet de nombreux travaux dans la littérature cette dernière décennie.

Pour s'attaquer à ce problème complexe, plusieurs modèles sont envisageables pour représenter les réseaux de transport multimodaux et plusieurs méthodes de résolution sont possibles. C'est une problématique vaste et complexe. Plusieurs hypothèses sont plausibles et chacune d'elle peut changer radicalement la nature du problème à traiter. Même dans le cas où on ne s'intéresse qu'à la recherche d'itinéraires usagers dans un réseau multimodal, le problème reste encore vaste et complexe.

Dans l'état actuel des choses, il n'existe pas de solution satisfaisante en terme de transport multimodal et temps pour des raisons métiers. Par nature, le métier ne se prête pas à cette situation à cause des problèmes de modélisation des réseaux de transport et des problèmes d'accessibilité aux données totalement distribuées. Sur le plan scientifique aucune des solutions proposées dans la litérature n'est satisfaisante pour les réseaux temps réel.

2.1 Objectif de cette thèse

Deux récentes thèses dont la thématique est très connexe à la notre ont été présentées dans [46] et [18]. T. Grabener a principalement adapté deux algorithmes existant pour intégrer la notion Time-dependent ou de dépendance au temps. Il s'agit de l'algorithme multi-objectif dû à Martins et al. [68] et d'un deuxième algorithme (contraction hierarchie).

La comparaison de ces différentes approches a montré que l'algorithme de Martins présente de meilleures performances ([46]). Les travaux de la thèse de Bousquet([18]) se sont concentrés sur le développement d'un algorithme adapté à l'optimisation d'itinéraires multimodaux au sein d'un système d'information destiné aux usagers des réseaux de transport. Aucune de ces thèses, n'a proposé d'approches parallèles pour un tel problème. Ce qui est regrettable, car les réseaux de transport deviennent de plus en plus importants en taille.

Dans le cadre de cette thèse, notre objectif est le calcul d'itinéraires multimodaux dépendant du temps dans le contexte de réseaux de transport de grande taille et tenant compte de la nature inhéremment distribuée des réseaux de transport. Nous recherchons des solutions algorithmiques génériques qui a priori, ne limitent ni le nombre de modes, ni le nombre d'objectifs. Nous envisageons aussi bien des solutions séquentielles que des solutions basées sur des méta-heuristiques , voire même des solutions hybridant les deux paradigmes. Nous proposerons aussi des algorithmes parallèles pour traiter des réseaux de transport de grandeur réelle.

2.2 Nos hypothèses

- Nos solutions doivent respecter les contraintes métier à savoir, l'aspect distribué des réseaux de transport et la non-centralisation des données. En effet, plusieurs fournisseurs de services de transport existent, et chacun utilise sa propre base de données ainsi que sa propre structure des données. Cette distinction doit être prise en compte lors de la modélisation.

- Nos solutions ne se limiteront pas à un ensemble de modes précis, mais plutôt à tous les types de modes : privés et publics.

- Les algorithmes proposés doivent résoudre des requêtes de type source, destination et temps de départ. Le ou les chemins optimaux trouvés relient la source à la destination partant à un temps supérieur ou égale au temps du départ donné. Les algorithmes doivent supporter d'autres types de préférences, par exemple l'interdiction d'un mode, le besoin d'un parking, une distance maximale de marche à pieds ... etc.

- Les réseaux utilisés seront dépendants du temps, c'est à dire que tous les coûts (prix, impact environnemental, ...) dépendent du temps ou de l'instant de leur utilisation. Un exemple classique est illustré par le fait que le prix d'un voyage en train n'est pas le même tous les jours.

- Les coûts pris en compte dans cette thèse seront aussi génériques avec quelques contraintes du style :

  1. Les valeurs d'un même coût doivent être strictement additives. Par exemple le prix, doit être exprimé dans la même unité.

  2. Nous recherchons un itinéraire dont le temps d'arrivée est le plus tôt et non pas le chemin le plus court en temps. Par exemple si l'utilisateur souhaite partir après 8h, une solution avec un chemin qui débute à 9h et se terminant à 11h sera meilleur qu'un chemin débutant à 11h et se terminant à 12h.

  3. Les coûts doivent être strictement positifs et croissants.

3 Principales contributions

Pour répondre à toutes les contraintes citées ci-dessus, nous avons proposé un nouveau modèle pour représenter les réseaux multimodaux, intitulé "graphe de transfert". Ce modèle préserve les

propriétés du modèle classique de graphe pour le calcul du plus court chemin tout en offrant une représentation plus naturelle des réseaux dans la réalité. En effet, dans ce modèle les différents réseaux mono-modaux sont maintenus séparés et leurs accès aussi. Ce dernier point fait de lui un modèle qui respecte la forte contrainte métier. Nous avons généraliser ce modèle à celui d'hypergraphe de transfert pour représenter des réseaux de transport de grande taille. Dans l'hypergrahe de transfert la composante de base n'est plus un réseau unimodal mais un graphe de transfert représentant une ville, une région, voire même un pays.

Nous nous sommes ensuite appuyés sur ce modèle pour proposer différentes solutions au calcul d'itinéraires, en procédant de la manière suivante :

- Dans un premier temps nous nous sommes intéressés à la recherche du chemin multimodal mono-objectif dépendant du temps. La procédure est décrite dans ce que suit.

  1. Nous avons proposé un premier algorithme simple générique basé sur une étape de pré-calcul consistant à stocker dans une base de données les calculs intermédiaires indépendants de la requête, dans le but d'optimiser le temps de calcul. Nous avons implanté cette base de données de deux manières. La première basée sur l'algorithme de Dijkstra présente de bonnes performances en temps mais souffre d'un problème d'explosion mémoire. La deuxième basée sur la métaheuristique ACO se comporte de façon totalement symétrique en offrant de bonnes performances en terme d'espace mémoire mais qui souffre par contre d'un temps de calcul conséquent.

  2. Pour remédier aux inconvénients de l'une et l'autre de ces deux méthodes, nous avons proposé un algorithme hybride. Ce dernier, contourne la construction de la base de données en passant par la création d'une structure intermédiaire de graphe appelée graphe abstrait. Cet algorithme améliore le précédent en terme de temps et espace mémoire, mais s'avère toutefois limité pour les réseaux de grande taille.

  3. Dans le but de traiter des réseaux de très grandes tailles, à l'échelle d'une région voire même d'une ville ou d'un pays, nous avons proposé un algorithme parallèle en mémoire partagée basé sur le modèle d'hypergraphe de transfert.

- Après avoir résolu le problème dans un contexte mono-objectif, nous nous sommes intéressés au cas multi-objectif. Notons d'abord que la première approche basée sur la construction d'une base de données n'est pas généralisable au cas multi-objectif. Nous avons résolu ce dernier problème par une méthode exacte, puis à l'aide de l'algorithme génétique NSGA-II et enfin par un nouvel algorithme hybride qui hérite des avantages des deux précédents.

4 Organisation du document

Le reste de cette thèse est organisée de deux parties structurées comme suit :

- **Partie 1 : Étude de l'existant**.
  Les éléments clés de cette thèse sont l'optimisation mono-objectif et multi-objectif, le transport multimodal dépendant du temps et le problème du plus court chemin. Pour des raisons de clarté, nous avons structuré l'étude de l'existant en trois chapitres, plutôt qu'un seul gros chapitre regroupant des notions de nature diversifiée.

Dans le chapitre 1, nous introduisons d'une façon générale le domaine de transport et plus particulièrement le transport multimodal, en insistant sur l'importance de la mobilité dans notre vie pratique ainsi que sur la multimodalité et les solutions qu'elle apporte pour améliorer la mobilité des citoyens. Nous détaillons aussi dans ce chapitre les différentes façons de modéliser un réseau du transport multimodal.

Le chapitre 2 sera dédié à une étude sur la théorie de la complexité des algorithmes. Nous proposons ensuite différentes solutions algorithmiques pour résoudre des problèmes d'optimisation, aussi bien des solutions exactes que des métaheuristiques.

Enfin, le chapitre 3 est une synthèse des solutions trouvées dans la littérature sur la résolution du problème du plus court chemin, bien connu dans la théorie des graphes.

– **Partie 2 : Contributions**.
La deuxième partie de cette thèse concerne nos principales contributions. Elle se compose aussi de trois chapitres :

Dans le chapitre 4 nous proposerons deux modèles d'abstraction des réseaux de transport multimodaux et dépendants du temps, à savoir le graphe de transfert et l'hyper-graphe de transfert. Nous terminerons ce chapitre par les solutions que nous apportons pour acquérir les données aussi bien académiques que réelles.

Le chapitre 5 présente nos principaux résultats en matière de calcul d'itinéraires mono-objectif et dépendant du temps. Il est composé de deux grandes parties correspondant aux solutions séquentielles et parallèle. Dans ce contexte précis mono-objectif, nous nous intéressons au plus court chemin en terme de temps.

Le Chapitre 6 concerne nos résultats en matière de recherche d'itinéraire multi-objectif. L'organisation de ce chapitre sera similaire à celle du chapitre 5. Pour l'approche séquentielle, nous commencerons par une solution classique due à Martins, modulo une modification mineure. Ensuite, nous proposerons une solution basée sur NSGA-II. Dans la troisième partie, nous présenterons de façon détaillée une approche plutôt originale, dite hybride. Cette dernière sera plus performante que les deux précédentes. Nous terminerons ce chapitre par une petite reflexion sur la paralllélisation du plus court chemin multi-objectif.

Nous terminerons ce document par le chapitre 7 consacré à une synthèse de nos résultats et une présentation de quelques unes de nos perspectives.

Table des figures

Liste des Algorithmes

Liste des tableaux

Partie 1 : Etat de l'art

Chapitre 1

Transport Multimodal : concepts fondamentaux

Le transport multimodal est l'un des éléments clés de cette thèse. C'est pourquoi nous consacrons ce premier chapitre à une analyse approfondie du concept de transport en général et plus particulièrement du transport multimodal qui nous concerne dans cette thèse. Dans la section 1.1, nous nous intéresserons au transport multimodal d'un point de vue socio-économique pour en dégager une définition formelle. Il s'agit d'une étape nécessaire à la meilleure appréhension de l'ensemble des paramètres à considérer lors de la réalisation de cette thèse. Dans la section 1.2, nous introduirons la notion de réseau de transport multimodal ainsi qu'une architecture matérialisant ce type de réseau. Dans la section 1.3 nous passerons rapidement en revue différentes techniques de modélisation des réseaux de transport multimodaux citées dans la littératue. Enfin, la section 1.4 sera consacrée à la présentation détaillée de la modélisation des réseaux de transport à l'aide de graphe.

1.1 Introduction au transport multimodal

1.1.1 Transport multimodal du point de vue socio-économique

1.1.1.1 Modèle en couches

Pour comprendre le concept de transport multimodal d'un point de vue socio-économique, nous nous appuyons sur sa représentation par un modèle en couches. Le modèle de base défini dans [94] et présenté sur la figure 1.1 est un modèle en couches dont les principaux éléments sont les suivants :

- Couche 3 (la plus élevée) : *activités*
- Couche 2 : *services de transport*
- Couche 1 : *services de gestion de trafic routier*

Ces couches sont reliées entre elles par des flux d'échange de services – ou marchés – qui sont de deux types principaux :

1. *Le marché des transports* reliant les couches 2 et 3 ;
2. *Le marché des services de gestion de trafic* qui relie les couches 1 et 2.

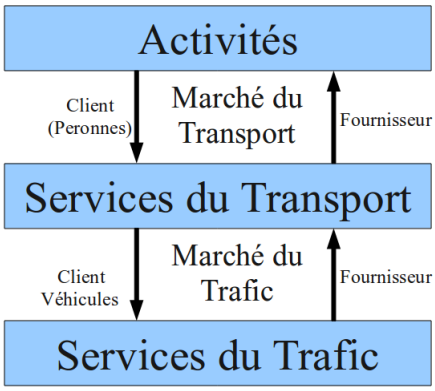

FIGURE 1.1 – Modèle de couches du système de transport

1.1.1.2 Services de transport

Le transport multimodal est associé à la deuxième couche du modèle de la figure 1.1. Il fait partie des services de transport en général, et en tant que tel partage leurs caractéristiques principales. Les services de transport couvrent aussi bien les transports publics que privés et intègrent l'ensemble des étapes composant un voyage du point de départ jusqu'à la destination, autrement dit de porte à porte. La qualité du service de transport dépend d'un grand nombre de paramètres, parmi lesquels le type de véhicule emprunté, les caractéristiques du réseau de transport ainsi que tous les attributs du service lui-même, qui incluent par exemple la fiabilité de l'organisme assurant le service de transport.

Les paramètres d'évaluation de la qualité de service de transport sont différenst selon que l'on considère le cas du transport public ou celui du transport privé :

- Dans le cas du transport public, le type de véhicule mis en œuvre et l'ensemble de ses caractéristiques, le type et la fréquence des liaisons, la pertinence et la disponibilité des informations à destination des utilisateurs, le coût du service de transport, etc. ne dépendent que du fournisseur du service de transport.

- Dans le cas du transport privé (véhicule personnel), le concept de «service de transport» est moins clairement défini, car le chauffeur du véhicule assure le transport pour lui-même : les rôles de prestataire de service et de consommateur sont joués par une seule et même personne physique. Cependant, tout comme dans le cas d'une société de transport public, les caractéristiques de la voiture et de son pilote déterminent la qualité du service de transport, tandis que la qualité du réseau utilisé ne dépend que de l'organisme qui en est gestionnaire.

1.1.1.3 Transport multimodal

Le transport multimodal implique l'utilisation successive de plusieurs services de transport durant un même trajet. Ces services peuvent être privés, publics ou une combinaison des deux. La figure 1.2 propose une vue détaillée de ce concept, représenté au sein de la catégorie «services de transport» du modèle en couches de la figure 1.1.

Un service de transport multimodal nécessite deux acteurs principaux pour fonctionner :

- **L'intégrateur de services de transport** : décide quels seront les services de transport

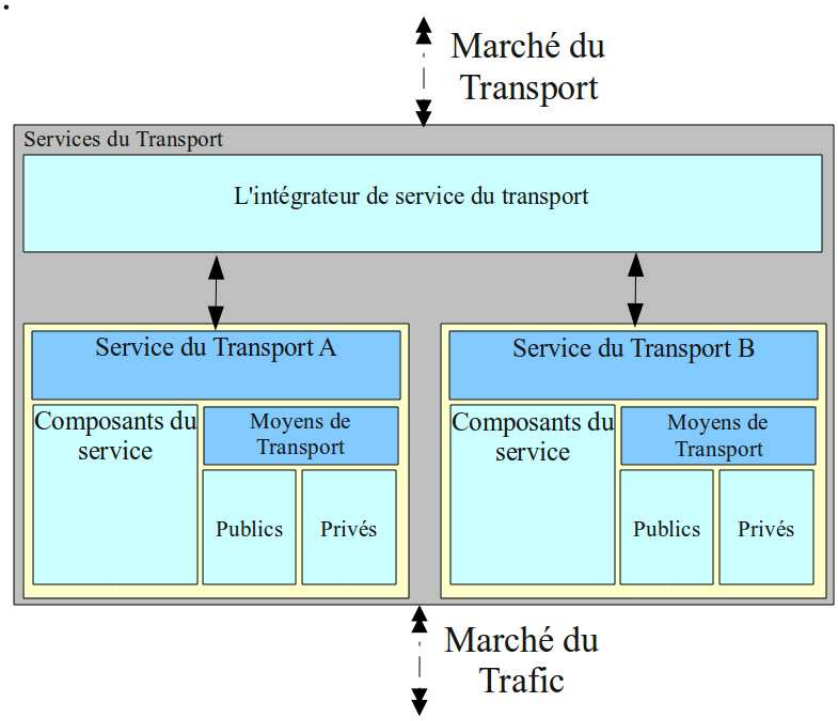

FIGURE 1.2 – Le transport multimodal et le modèle en couches

utilisés pour un trajet donné. Le voyage en question pourra être unimodal ou multimodal. C'est en général le voyageur lui-même qui joue le rôle d'intégrateur de services de transport. Dans certains cas, un organisme tiers – une agence de voyage par exemple – peut se charger de cette tâche.

– **Le fournisseur de service de transport** : tous les organismes de transports monomodaux de type urbains, régionaux ou nationaux, qu'ils soient publics ou privés font partie de cette catégorie. Les caractéristiques associées à chaque mode de transport déterminent le temps et le coût d'un voyage. En plus des véhicules eux-mêmes, chaque service de transport comporte plusieurs composants additionnels qui contribuent indirectement à la fourniture du service : les guichets et billetteries, les terminaux d'information, etc. sont autant d'exemples de tels composants.

Cette description détaillée des services de transport met en évidence de façon claire, trois caractéristiques du transport multimodal. En premier lieu, le transport multimodal requiert un ou plusieurs transferts entre différents services de transport privés ou publics. Ceci peut se traduire par une baisse de la satisfaction de l'utilisateur qui se voit confronté à un nouveau mode de transport dont le confort n'est pas forcément au même niveau que celui mode précédemment emprunté. Cela est typiquement le cas d'un voyageur passant d'un train confortable en première classe à un bus surchargé en période de pointe... En second lieu, il est indispensable d'assurer une bonne coordination et une transition aisée entre les différents modes de transport mis en jeu.

1.1.2 Définition formelle du transport multimodal

Le transport multimodal est défini dans cette thèse comme une combinaison de plusieurs modes de transport différents utilisés pour un seul et même voyage, pour lesquels le voyageur doit effectuer un transfert (voir figure 1.3).

Un mode peut être défini par le type de véhicule ou par la fonction de transport associée. La partie du voyage où un seul mode est utilisé est appelé une étape.

Un voyage dans lequel un vélo est utilisé pour accéder à une gare ferroviaire suivi d'un trajet en train, un voyage où un bus est utilisé pour aller de la gare à la destination finale sont deux exemples typiques de transport multimodal. Lors d'un voyage unimodal, seul un mode de transport, qu'il soit public ou privé est utilisé.

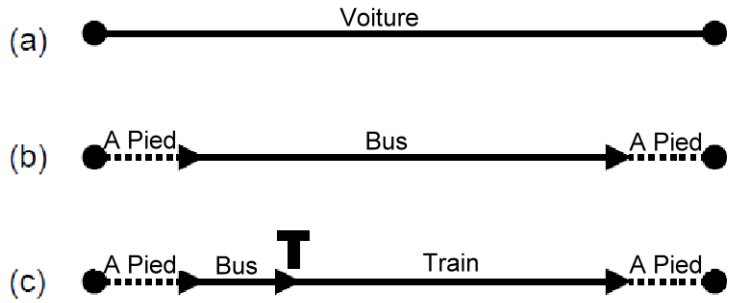

FIGURE 1.3 – Exemples de voyage unimodal (a,b) et multimodal (c)

Cette définition peut être précisée en examinant plus en détail le rôle des transferts entre différents modes, les types de modes de transport qu'il est possible d'utiliser et les services correspondants, les circuits prédéfinis ou encore la place du trajet pédestre dans un trajet multimodal.

1.1.2.1 Transfert entre modes

La notion de transfert joue un rôleessentiel dans un voyage multimodal. Lorsqu'un trajet comporte plusieurs modes, les voyageurs doivent changer de mode. Les nœuds de changement de mode sont appelés des nœuds de transfert. Mais le transfert n'est pas propre au seul transport multimodal. Cette notion existe aussi dans les réseaux de transports publics unimodaux. C'est pourquoi la définition d'un nøud de transfert doit être précisée.

Dans cette thèse, le terme transfert est utilisé pour les transferts de type intermodal, correspondant au cas où les voyageurs changent de service ou de mode de transport. L'inclusion des services de transport est essentielle car elle autorise le transfert d'un réseau de services de transport à un autre réseau de services de transport ayant d'autres caractéristiques. Dans ce dernier cas, on parlera de transfert intermodal par opposition au transfert intramodal qui correspond à un changement de bus par exemple.

Un transfert implique un temps de déplacement supplémentaire et/ou des frais de déplacement supplémentaire. Pour que le transport multimodal reste attractif, les coûts engendrés par le transfert entre mode et illustrés par la figure 1.4 doivent être compensés dans une certaine mesure, sans quoi l'utilisateur aura toujours tendance à préférer le transport monomodal.

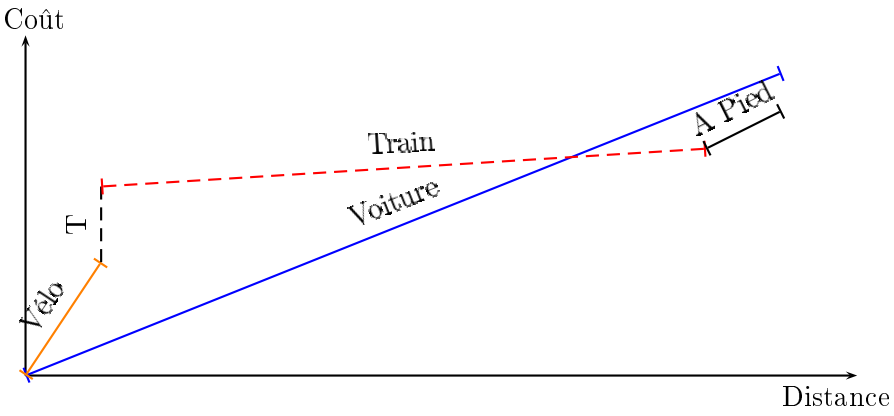

FIGURE 1.4 – Schéma du coûts/distance d'un voyage unimodal et multimodal

1.1.2.2 Modes et services de transport

Ces notions sont étroitement liées mais peuvent paradoxalement avoir des significations différentes. Le cas d'utilisation le plus courant du terme mode est celui qui le fait correspondre au type de véhicule emprunté par le voyageur, par exemple le vélo, la voiture ou les transports publics. Mais dans le cas des transports en commun, le terme mode est lié aux caractéristiques du service et non pas spécifiquement aux types de véhicules. Ainsi, une distinction peut être faite entre les modes de service, les modes privés (véhicules privés, voiture ou vélo) et les modes publics (bus, train, tram...). Comme le transport multimodal est fortement lié aux services de transport, le terme mode lui même est généralement lié aux modes de service. Les types de véhicules ont donc une importance secondaire. Par ailleurs, il convient de noter que dans le cas des services de transports publics, les différents types de services de transport peuvent avoir des caractéristiques différentes. Ces différences concernent plus particulièrement l'accessibilité, la rapidité, la fréquence, les tarifs, et les véhicules utilisés. Elles sont souvent fortement liées à différents niveaux fonctionnels du réseau : niveau urbain, régional, national. Le transport multimodal concerne donc les transferts entre modes de transport privé ou bien entre différents types de services fonctionnels de transports publics. Cette distinction dans les types de mode est illustré par la figure 1.5.

1.1.2.3 Voyages multimodaux et tournées multimodales

La définition du transport multimodal est basée sur celles des tournées (circuits) – comme l'illustre la figure 1.6 – et non sur celles des voyages. Les tournées multimodales et les voyages multimodaux ont chacun leurs propres caractéristiques. Une tournée composée d'un trajet en bus à l'aller et d'un trajet en véhicule privé au retour se compose de deux voyages unimodaux. Bien qu'une telle tournée pourrait être appelée une tournée multimodale (voir figure 1.7), il ressort clairement de cet exemple que la tournée multimodale est différente d'un voyage multimodal où deux ou plusieurs modes sont utilisés pour effectuer un seul et même voyage. Dans le cadre de cette thèse nous avons choisi de travailler sur le voyage multimodal.

Cet exemple montre que le concept de tournée est pertinent pour un voyage multimodal. Il est peu probable qu'un utilisateur utilisant un bus pour le trajet aller ait la possibilité d'utiliser sa voiture personnelle pour le retour : certains modes ne sont en général disponibles qu'au départ de certains lieu précis comme le domicile du voyageur. Par ailleurs, il doit être pris en compte

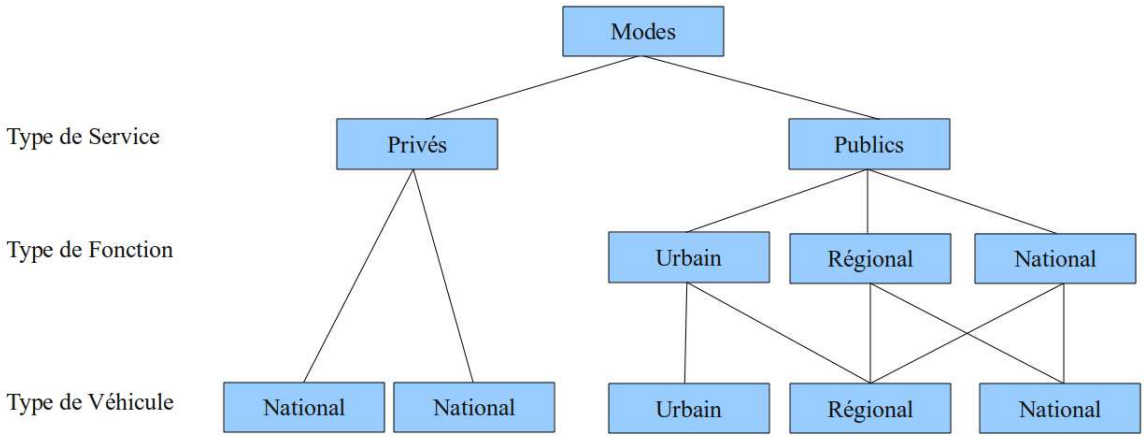

FIGURE 1.5 – Différents types de modes de transport

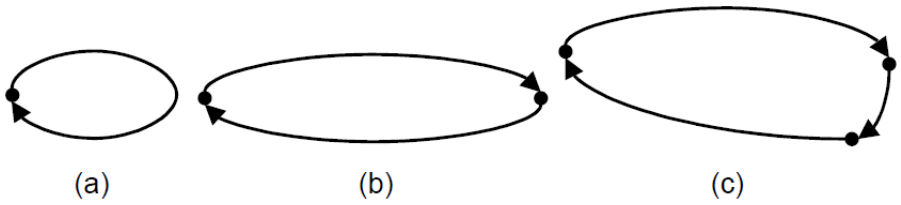

FIGURE 1.6 – Exemples de tournées comprenant 1 (a), 2 (b) et 3 voyages (c)

que si un mode privé est utilisé pour le premier voyage de la tournée, il doit être utilisé pour le retour. C'est le cas d'un voyageur ayant utilisé une voiture comme mode d'accès à la gare et qui doit retourner à la fin de sa tournée à la gare où sa voiture se trouve pour rentrer chez lui. Pour des raisons de clarté, seul le concept de voyage multimodal est utilisé dans le reste de la thèse.

1.1.2.4 Trajet piéton

La marche à pied fait presque toujours partie d'un voyage. Cela est évident dans le cas où les voyageurs marchent vers et depuis l'arrêt du service de transport public. Mais ça reste vrai pour le voyage en voiture qui nécessite d'aller et venir à la place de stationnement, bien que les distances considérées soient très courtes. La marche peut donc être considérée comme une composante incontournable au début et à la fin de tout voyage mais aussi pour les changements de modes. Le trajet piéton n'est donc pas considéré comme un mode séparé dans la définition d'un voyage multimodal.

1.1.3 Mobilité multimodale dans la pratique

Nous discutons ici les facteurs qui mettent en évidence l'importance de la mobilité multimodale de nos jours.

1.1.3.1 Répartition modale

Le tableau 1.1 montre la répartition modale pour tous les voyages avec une distinction entre les voyages unimodaux et multimodaux. Nous observons clairement que le pourcentage de dé-

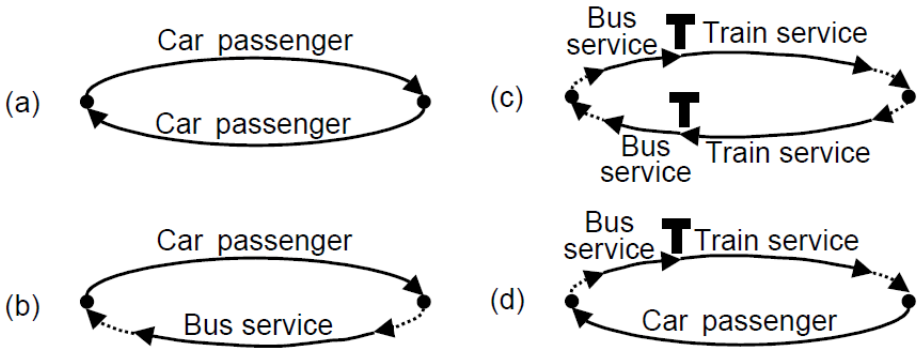

FIGURE 1.7 – Exemples de tournées multimodales et unimodales : (a) tournée unimodale avec 2 voyages unimodaux (b) tournée multimodale avec 2 voyages unimodaux (c) tournée multimodale avec 2 voyages multimodaux (d) tournée multimodale avec un voyage unimodal et un voyage multimodal

placements multimodaux est faible et ne représente que $2,9\%$ de tous les déplacements. Mais le taux de voyages multimodaux a augmenté de 25% cette dernière décennie et présente donc une croissance soutenue.

Mode Principale	Tournées %	Unimodale %	Multimodal %	% multimodal
Automobiliste	36.2	36.0	0.2	0.5
Voitures particulières	13.1	12.9	0.2	1.6
Train	2.1	0.4	1.7	80.5
Tram/Metro	0.9	0.7	0.2	20.4
Bus	2.0	1.6	0.4	21.2
Velo	27.6	27.5	0.0	0.1
Marche	16.0	15.9	0.1	0.7
Autres	2.1	2.1	0.0	1.7
Tous les Modes	100.0	97.1	2.9	2.9

TABLE 1.1 – Répartition entre voyages unimodaux et multimodaux

Les voyages multimodaux sont significatifs dans le cas d'utilisation du train : dans 80% des cas le train est associé à un autre mode de voyage. Le taux de déplacements multimodaux n'est que de 20% environ lorsqu'un bus ou un tram/métro est l'un des modes emprunté durant un voyage.

Trois facteurs dominent dans la distinction entre les voyages unimodaux et multimodaux classés comme suit, par ordre d'importance :

- **La distance parcourue :** Les déplacements longue distance font plus recours à des voyages multimodaux.

- **Le type de destination :** Les déplacements multimodaux sont orientés vers les principales villes et en particulier les centres-villes.

- **Le motif du voyage :** Les déplacements multimodaux sont principalement utilisés pour

se rendre au travail et à l'école.

1.1.3.2 Distance parcourue lors d'un voyage

L'importance de la longueur d'un voyage peut être clairement observée sur les figures 1.8 et 1.9. Elles représentent la distribution de la longueur d'un voyage et les variations de celle-ci par rapport au nombre d'étapes par voyage.

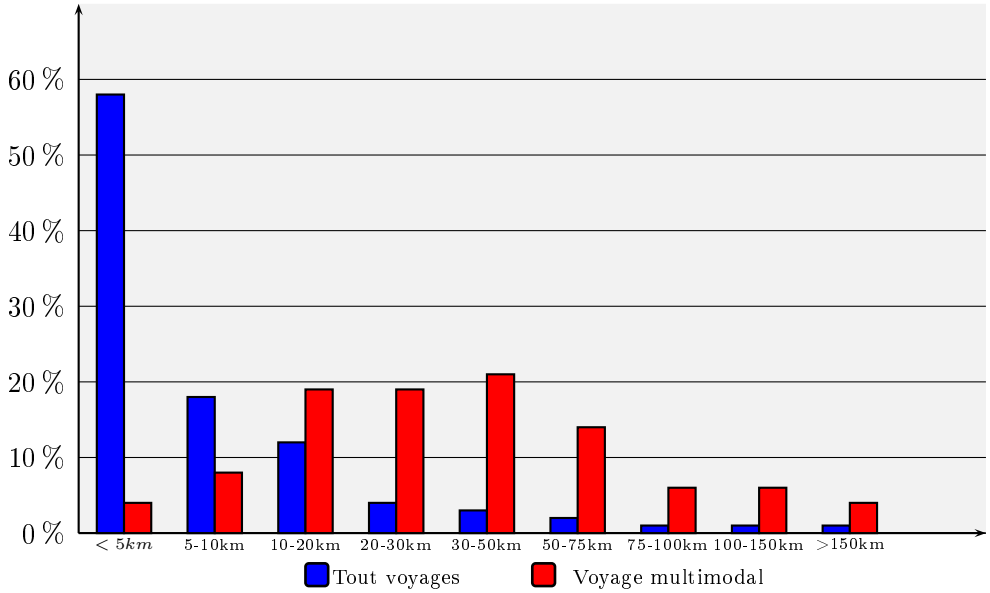

FIGURE 1.8 – Distribution des longueurs de voyage pour tous les voyages et pour les voyages multimodaux

La distance moyenne parcourue lors d'un déplacement multimodal est de 45 kilomètres, ce qui représente plus de 4,5 fois la distance moyenne des trajets unimodaux. Le transport multimodal représente ainsi plus de 12% de toute la distance parcourue. Le transport multimodal semble être une solution viable pour les voyages de plus de 10 kilomètres et devient une alternative intéressante pour les longs trajets de 30 kilomètres, avec une part modale de 15% environ.

1.1.3.3 Type de destination

Le second facteur discriminant dans les trajets multimodaux est le type de zone de destination. La figure 1.10 montre la répartition par rapport au type de zone considérée pour le départ du trajet. Pour les deux extrémités du trajet, les déplacements multimodaux sont moins importants dans les zones rurales. Ils sont par contre plus importants dans les villes importantes et les centre-villes. Ces deux figures montrent aussi que les parts respectives du transport multimodal et unimodal varient fortement en fonction de la zone de destination.

1.1.3.4 Motif du voyage

Le troisième facteur discriminant pour les voyages multimodaux est le motif du voyage. Comme on peut le voir dans le tableau 1.2, le transport multimodal joue un rôle important pour les trajets permettant de se rendre à l'école ou au travail. Le transport multimodal semble moins

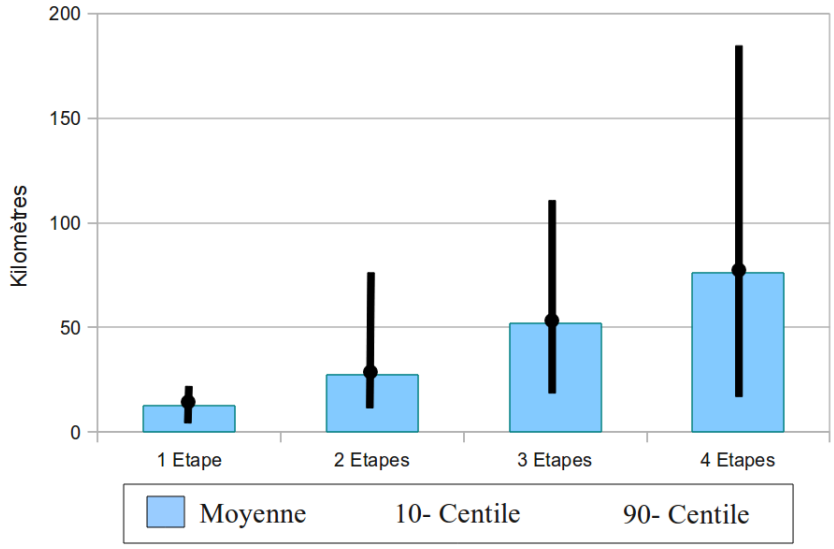

FIGURE 1.9 – Longueur des voyages par rapport au nombre d'étapes par voyage (NTS 1995-1997)

intéressant pour des voyages à caractère professionnel, ainsi que pour les tournées de ramassage de passagers.

	Tournées [%]	Multimodal [%]	Pourcentage multimodal
Travail	17.7	31.4	5.3
Sociale	15.6	14.8	2.8
Education	4.6	21.4	14.0
Shopping	24.5	9.4	1.1
Affaires, personels	2.2	1.0	1.3
Affaires, professtionnels	3.1	2.4	2.3
Loisirs	12.4	11.2	2.7
Tourisme	4.3	1.4	10
Soins personnels	3.1	1.3	1.2
Ramassage/Dépot	6.9	0.9	0.4
Autre	5.6	4.8	2.6
Totale	100.0	100.0	3.0

TABLE 1.2 – Motifs des voyages et distinction entre voyages unimodaux et multimodaux (NTS 1995-1997)

L'importance du multimodal pour les déplacements domicile-travail est facile à expliquer : la plupart des emplois et des entreprises sont concentrées dans les centre-villes qui disposent de la plus forte densité de réseaux de transport. De même, les trajets de courtes distances sont souvent liés au motif «shopping» qui se fait le plus souvent en véhicule personnel sauf au départ et à destination des grands centre-villes. Il n'y a cependant pas de tendance claire se dégageant des voyages de loisirs, par exemple.

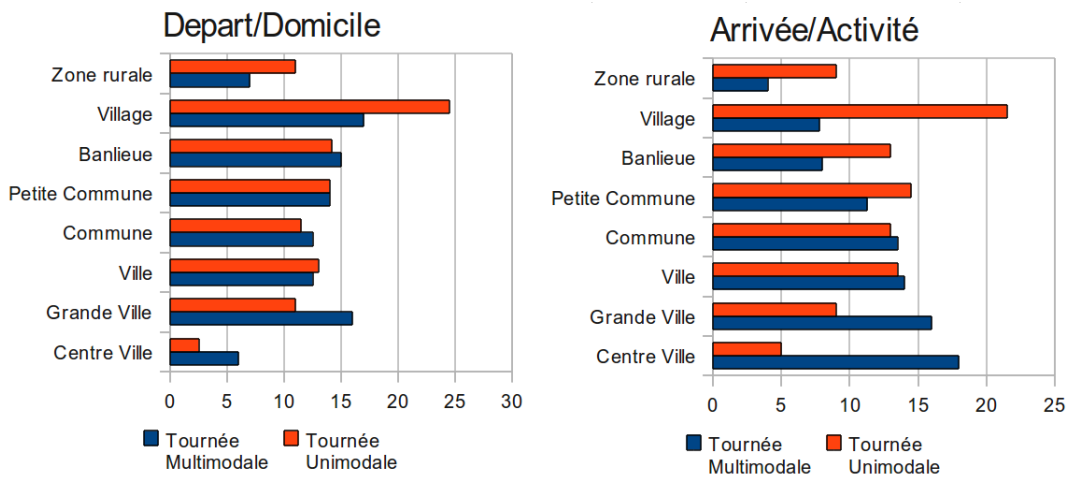

FIGURE 1.10 – Répartition des voyages unimodaux et multimodaux par rapport au type de zone pour le départ (à domicile) et l'arrivée (activité) (NTS 1995-1997)

1.2 Réseaux de transport multimodal

Il y a deux catégories de réseaux de transport qui devraient être considérées :

- Les réseaux de services de transport : il s'agit des réseaux de service de type bus ou train.

- Les réseaux de services de trafic ou réseaux physiques : il s'agit là des réseaux routiers ou des réseaux ferroviaires.

Un réseau de service de transport est toujours lié à un réseau physique. Ainsi un réseau de service de bus est basé sur un réseau routier et un réseau de service train s'appuie sur un réseau ferroviaire. Du point de vue du voyageur, l'important est de savoir si un service de train peut être utilisé pour son voyage. Un chemin de fer, sans services de trains associé est inutile pour le voyageur. Toutes les contraintes et les possibilités découlant du réseau ferroviaire sont déjà intégrées dans les services de trains disponibles. Pour les modes privés, le réseau physique détermine souvent si l'utilisation d'un mode privé est envisageable pour un voyage donné.

Les caractéristiques d'un réseau peuvent être vues sous deux angles : celui de l'utilisateur du réseau (les voyageurs) et celui de l'opérateur du réseau. Les principales caractéristiques de tout réseau de transport du point de vue du voyageur sont le coût du trajet qui dépend principalement de :

- L'accessibilité de l'espace : le nombre et la répartition des points d'accès où le voyageur peut entrer et sortir du réseau. Des exemples typiques sont les arrêts de bus, les parkings, et les aéroports ;
- L'accessibilité en temps : la répartition des possibilités par unité de temps pour le voyageur qui utilise le réseau. Cette caractéristique est très commune pour les services de transport publics ou d'avion et peut être décrite par des horaires ou des fréquences de liaison. Pour le transport privé, l'accessibilité en temps est généralement illimitée.
- La vitesse du réseau : la vitesse moyenne qui dépend de la structure du réseau et de ses caractéristiques physiques.

La qualité d'un voyage est généralement mesurée par le temps de déplacement, négligeant ainsi d'autres caractéristiques telles que son coût ou le confort. Cela est dû au fait que les réseaux

de transport déterminent surtout les durées de déplacement, tandis que les autres caractéristiques n'ont qu'une relation indirecte avec le réseau de transport lui-même.

Il y a deux approches principales pour décrire un réseau de transport. La plus commune considère un réseau comme un ensemble de nœuds reliés par des arêtes. Certains des nœuds sont des nœuds d'entrée et de sortie ou des nœuds d'accès, tandis que les autres nœuds intermédiaires représentent les passages où aucun accès ou sortie n'est possible (voir figure 1.11). Pour les réseaux de transports publics, la représentation comprend les lignes de transport public, qui sont un ensemble d'arêtes, et les horaires associés. Ce type de description est particulièrement adapté pour la modélisation des réseaux de transport.

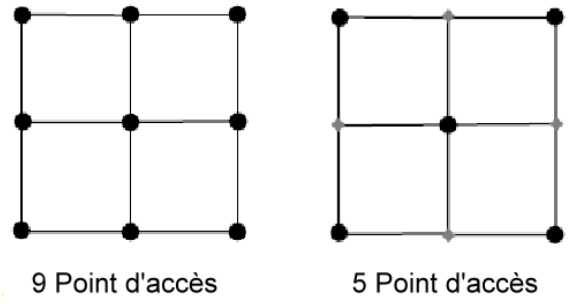

FIGURE 1.11 – Exemple d'un réseau avec de différents nombre de point d'accès

La seconde approche porte sur des réseaux génériques en utilisant des formes spécifiques comme les réseaux en grille ou radiaux, et des paramètres telles que la largeur de la route, le nombre de voies, l'espacement des arrêts, et la fréquence des liaisons. Ces variables globales déterminent les principales caractéristiques du réseau, à la fois du point de vue voyageur que investisseur. Le degré de précision de cette seconde approche est nettement moins bon que dans la première approche, mais il permet une analyse des caractéristiques fondamentales des réseaux. Un exemple typique d'une telle analyse approfondie peut être trouvée dans [103].

La définition de ces variables globales est fortement liée aux différentes structures de réseau dont certains exemples sont présentés sur la figure 1.12. La largeur des routes, par exemple, est une variable adaptée à la densité du réseau dans le cas d'un réseau en grille ou d'un réseau triangulaire, mais pas pour un réseau radial. Dans ce cas, le nombre de radiales serait plus approprié pour représenter ce paramètre.

1.2.1 Modes de transport

Il existe principalement 4 modes de transport :

1. Le transport routier est une activité de transports terrestres, qui s'exerce sur la route. Elle englobe à la fois le transport routier de personnes et le transport routier de marchandises. Le transport routier présente comme principaux avantages un coût relativement réduit et un service dit en «porte à porte».
2. Le transport maritime : il a pour avantage de recouvrir les zones de livraison les plus étendues du globe, il permet donc de desservir le monde entier.
3. Le transport aérien : tout comme pour le transport maritime, le transport aérien permet de desservir beaucoup de destinations à travers le monde. Il est approprié pour les transports nécessitant des délais courts tout en assurant une sécurité maximale.

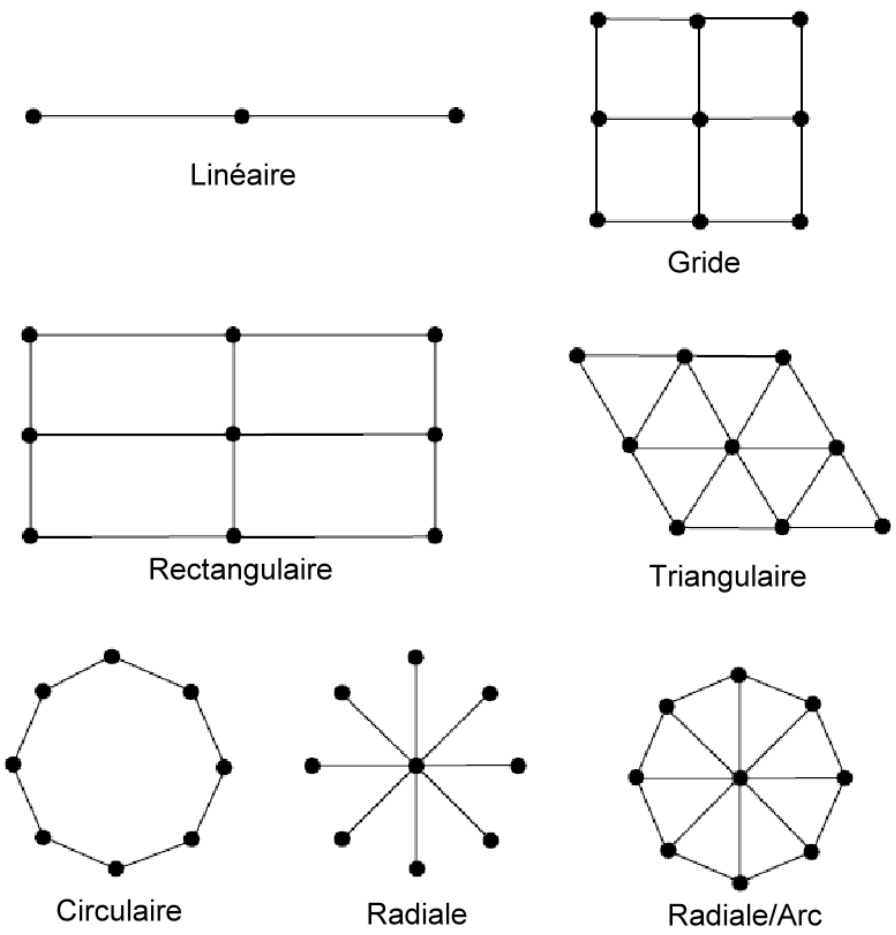

FIGURE 1.12 – Exemples de structure des réseaux

4. Le transport ferroviaire : est un système de transport guidé servant au transport de personnes et de marchandises. Il se compose d'une infrastructure spécialisée, de matériel roulant et de procédures d'exploitation faisant le plus souvent intervenir l'humain, même si dans le cas des métros automatiques cette intervention se limite en temps normal à de la surveillance.

Mode	Type	Catégorie
Train	Public	Ferroviaire
Tram/metro	Public	Ferroviaire
Bus	Public	Routier
Velo	Privé	Routier
Voiture	Privé	Routier
Avion	Public	Aérien
Bateau	Public	maritime

TABLE 1.3 – Différents modes du transport multimodal

Le tableau 1.3 représente une liste non exhaustive de différents modes de transport multimodaux. D'autres modes et services peuvent exister, comme les vélos en libre service, les taxis, le co-voiturage ou le transport à la demande. Chacun de ces modes possède ses propres caractéristiques et propriétés, comme la capacité d'une voiture de co-voiturage, les contraintes spécifiques au fournisseur du co-voiturage, ou même les places disponibles des bornes de vélos public.

1.2.1.1 La voiture et le vélo

Ce sont deux modes de réseaux routiers, qui sont caractérisés par une grande liberté de mouvement puisque le déplacement entre deux points quelconques du réseaux s'effectue de façon autonome et sans contrainte du temps. Évidemment, une voiture ne peut être laissée que dans un parking. Cette contrainte forte de ce type de service doit être prise en compte lors de la modélisation du réseau. En effet, il faut s'assurer qu'il n'est pas possible de laisser une voiture dans un autre endroit que les parkings.

1.2.1.2 Transports en commun

Le transport en commun, ou transport collectif, consiste à transporter plusieurs personnes ensemble sur un même trajet souvent dans un même véhicule. La notion de transport public est différente puisqu'ils regroupent tous les transports qui sont organisés pour le compte d'un tiers [1]. Par exemple les taxis traditionnels sont un transport public mais pas un transport en commun. La nuance reste cependant ténue puisqu'un taxi pourra par exemple être affrété dans le cadre d'un transport à la demande et être alors considéré comme un service de transport en commun.

Les transports en commun se caractérisent par un nombre restreint de possibilités et surtout par des horaires de fonctionnement très précis. Nous considérons que les transports en commun sont représentés par une succession d'arrêts qui forment une ligne. Chaque ligne peut être parcourue régulièrement par des bus, métros, trains, etc. qui respectent scrupuleusement la même séquence d'arrêts. Cependant, rien n'empêche que le temps de parcours entre deux arrêts soit différent selon l'heure de la journée. Par exemple un bus pris dans la circulation mettra plus de temps en heure de pointe qu'en heure creuse [46].

1.2.1.3 Vélos en libre service et Co-voiturage

Ces modes ont la particularité d'utiliser le réseau routier traditionnel mais ne sont accessibles que sous un ensemble de contraintes prédéfini par le fournisseur de service. Ils ne sont d'abord accessibles qu'à des endroits particuliers du réseau (borne de vélo, parkings, point de ramassage). Le vélo en libre service est caractérisé aussi par une disponibilité : un voyageur ne peut utiliser ce service, que si un vélo est présent dans la borne libre-service et ne peut laisser un vélo que si une place est disponible. Une réservation au préalable n'est souvent pas possible avec ce mode de transport.

La particularité du co-voiturage est que le voyageur doit être, tout comme le vélo en libre service d'ailleurs, abonné chez le fournisseur de service. Cet abonnement, crée chez le fournisseur de service un système de profil, un ensemble de contraintes et d'exigences fixées par le demandeur et par le fournisseur de service, ce qui rend la tâche plus complexe, puisque la disponibilité d'un tel service dépendra du passager lui même.

1. $http://www.bourgogne.developpement-durable.gouv.fr/article.php3?id_article = 126$

1.3 Modélisation d'un réseau du transport multimodal

Après avoir présenté les caractéristiques du transport multimodal dans la section précédente, on s'intéresse dans cette section à sa modélisation. Le choix du modèle a une influence directe sur les algorithmes à développer. Il est donc important d'étudier les différentes méthodes présentées dans la littérature. Nous présentons dans cette section les différents modèles utilisés pour le transport multimodal. Comme nous l'avons déjà mentionné dans la section 1.2, la manière la plus classique et la plus simple de modéliser un réseau du transport est la représentation sous forme de graphes. Mais il existe toutefois d'autres méthodes, certes moins répandues, pour modéliser les réseaux du transport.

1.3.1 Modélisation par les graphes

La modélisation des réseaux de transport multimodal par les graphes est de loin la méthode la plus utilisée dans la littérature par la communauté scientifique. Il existe cependant plusieurs façons d'utiliser les graphes selon le type de réseaux de transport à modéliser. Dans ce travail, nous ne nous intéressons au graphe que pour la modélisation des réseaux de transport multimodaux dépendant du temps et de façon plus spécifique, pour la résolution du problème du plus court chemin entre une source unique et une destination unique à partir d'un temps du départ fixe. Cette hypothèse, nous permet de cerner l'état de l'art sur le sujet qui risque d'être beaucoup trop vaste dans un cadre plus général.

La modélisation seule, d'un réseau routier ou d'un réseau de transport en commun est quelque chose de très naturelle. Cependant, la modélisation de la notion de réseaux dépendant du temps la rend beaucoup plus complexe. La multimodalité suscite aussi une autre question : comment modéliser le fait qu'un même tronçon puisse être emprunté par un bus, une voiture, un vélo ou encore un piéton [46] ?

Deux principales approches ont été proposées pour la modélisation de la notion de dépendance au temps pour le problème de plus court chemin : l'approche Time-expanded [70, 72, 73, 84, 87, 95], et l'approche Time-dependent [21, 26, 64, 74, 79]. La caractéristique commune de ces deux approches est que la résolution de la requête correspond à une extension du problème classique du plus court chemin.

1.3.1.1 Approche Time-expanded

Un graphe «time-expanded» ou graphe Espace-temps est un simple graphe dont les nœuds correspondent à des événements de temps spécifique (départ ou arrivée à une station) et dont les arêtes sont des connexions élémentaires entre les deux événements. les graphes Time-expanded sont de graphes statiques, de grande taille mais généralement sparses.

1.3.1.2 Approche Time-dependent

Pour éviter qu'un nœuds puisse changer en fonction d'un événement dépendant du temps, dans le graphe «Time-dependent», chaque nœud représente une gare (station, localité, parking, etc.), et deux nœuds sont reliés par une arête si leurs stations correspondantes sont reliées par une connexion élémentaire. Le coût d'un arc dépend du son temps d'accès. Les coûts des arêtes sont affectés «à la volée».

1.3.2 Modélisation par automates cellulaires

Étudiés en mathématiques et en informatique théorique, les automates cellulaires sont à la fois des modèles de système dynamiques discrets et des modèles de calcul. Le modèle des automates cellulaires se distinque par la grande simplicité de sa définition et la complexité pouvant être engendrée par certains comportements macroscopiques : l'évolution dans le temps de l'ensemble des cellules ne se réduit pas simplement à la règle locale qui définit le système. A ce titre il constitue un des modèles standards dans l'étude des systèmes complexes. Les automates cellulaires appliqués au trafic routier se sont beaucoup développés au cours des 15 dernières années, et permettent aujourd'hui des comparaisons directes avec des mesures empiriques [5].

1.3.3 Modélisation basée sur les activités

Ce modèle a été initié depuis les années 1970 avec comme objectif d'étudier les interactions entre un système de transport et l'occupation du sol. Bien que de nombreuses études aient été réalisées, la modélisation dynamique des systèmes de transport basée sur l'enchaînement des activités est récente [90, 99].

1.3.4 Réseaux de Petri et algèbre des dioïdes

Plusieurs systèmes complexes, dont les systèmes de transport public, peuvent être définis comme des systèmes dont la dynamique est exclusivement gouvernée par un modèle d'équations mathématiques dans une structure algébrique. Les auteurs de ce travail [75], présentent un modèle formalisé graphiquement par les réseaux de Petri (RdP) [30] et mathématiquement par l'algèbre des dioïdes [9].

1.4 Modèle de graphe : une étude détaillée

Nous revenons ici au modèle de graphe pour une présentation plus pointue, car c'est le modèle le plus répandu et sur lequel nous nous appuierons dans ce travail. Avant de détailler la modélisation des réseaux multimodaux dépendant du temps, nous devons d'abord distinguer les différents types de problèmes que nous souhaitons résoudre avec ces réseaux. En effet, il existe généralement deux types de problèmes de plus court chemins dans un réseau de transport multimodal :

- *Le problème de chemin le plus rapide (earliest arrival problem)* : dans ce type de problème, le but sera de répondre à une requête de type (A, B, t_0) constituée par une lieu de départ A, un lieu d'arrivé B et un temps du départ t_0. Un chemin est valide – c'est à dire est une solution réalisable – s'il existe un départ après t_0, dans ce cas le chemin optimal est celui qui permet d'arriver le plus tôt.
- *Cas réel* : d'autres critères d'optimisation importants concernent le nombre de transferts et le prix d'un chemin. Dans le problème de minimisation du nombre de transfert (*minimum number of transfers*), la requête à résoudre est ainsi : étant donné deux stations A et B, quel est le chemin qui minimise le nombre de transferts entre les modes ? Cette requête ne fait intervenir ni le temps de départ ni le temps d'arrivée. De même, on peut être amené à rechercher le chemin dont le coût est le plus bas.
- Une requête peut également contenir une séquence de stations intermédiaires ainsi que le temps d'arrêt souhaité au niveau de ces stations. En outre, certains modes peuvent être

exclus de l'ensemble des modes, par exemple, si l'on envisage d'utiliser un billet qui n'est valable que pour les trains locaux, les trains interurbains devraient être exclus. Il est possible d'envisager dans une requête l'heure d'arrivée et non l'heure de départ. Alternativement, les spécifications de temps peut être données par des intervalles plutôt que par une valeur précise.

Dans la modélisation des réseaux de transport multimodal par les graphes, deux aspects sont à prendre en compte. L'aspect statique des réseaux (stations, parkings, routes, bornes pour vélos, etc.) et l'aspect dynamique concernant le service de transport lui même (lignes, tables du temps, transferts, coûts, etc.). Nous commençons par définir l'aspect statique du modèle puisqu'il est unique quelque soit le type du problème à résoudre, nous aborderons ensuite les modèles dynamiques pour chaque type de problème envisagé.

Il existe plusieurs façons d'utiliser les graphes pour modéliser les réseaux de transport, selon les algorithmes qu'on souhaite appliquer, mais le modèle le plus utilisé est celui des multi-graphes valués. Chaque lieu – ou localité – (station, arrêt, parking, etc.) est défini par un nœud du graphe et deux nœuds sont reliés par une arête s'il existe une possibilité d'aller du premier nœud vers le deuxième. On associe à chaque arête un mode (bus, train, piéton...). Il est évidemment possible d'avoir deux ou plusieurs arêtes de différents modes entre la même paire de nœuds.

Définition 1.4.1

Formellement, un réseau multimodal est modélisé par un graphe $G = (V, E, M)$ où :
- *$V = \{v_1, ..., v_j\}$ est un ensemble de nœuds.*
- *$M = \{m_1, .., m_k\}$ est un ensemble de modes de transport.*
- *$E = \{e_1, ..., e_l\}$ est un ensemble d'arêtes. Une arête $e_x \in E$ peut être identifiée par $(v_p, v_q)_{m_r}$, où $v_p, v_q \in V$ et $m_r \in M$. L'arête e_x exprime qu'il est possible de passer d'une localité (nœud) v_p à v_q en utilisant le mode de transport m_r.*

La figure 1.13 présente un exemple d'un réseau multimodal modélisé par un multi-graphe.

Après avoir modélisé l'aspect statique des réseaux du transport, qui est considéré comme l'infrastructure commune quelque soit le type du problème à résoudre, nous nous intéressons maintenant à l'aspect dynamique du réseau, qui sera traité selon le type du problème. Pour cela nous avons besoin de définir concept très important pour la suite de cette thèse, à savoir « la connexion élémentaire» :

Définition 1.4.2

***Connexion Elémentaire** : Soit S un ensemble de services de transport multimodal (Transport en commun, Taxi, Co-voiturage, vélo, etc.), V un ensemble de lieux, une connexion élémentaire c est définie par un 5-tuples (s, v_1, v_2, t_d, t_a) où :*
- *$s \in S$ est service donné.*
- *$\{v_1, v_2\} \in V$ sont deux nœuds distincts de V*
- *t_d est le temps de départ de v_1*
- *t_a est le temps d'arrivée sur v_2, tel que $t_a \geq t_d$.*

Une connexion élémentaire est interprétée ainsi : Le service s (i.e. Bus 125), assure une connexion entre v_1 et v_2 partant à l'instant t_d et arrivant à t_a.

1.4.1 Problème du chemin le plus rapide

Dans cette section, nous examinons les modèles de résolution du problème le plus fondamental, à savoir trouver le chemin permettant d'arriver le plus tôt possible, en utilisant les approches

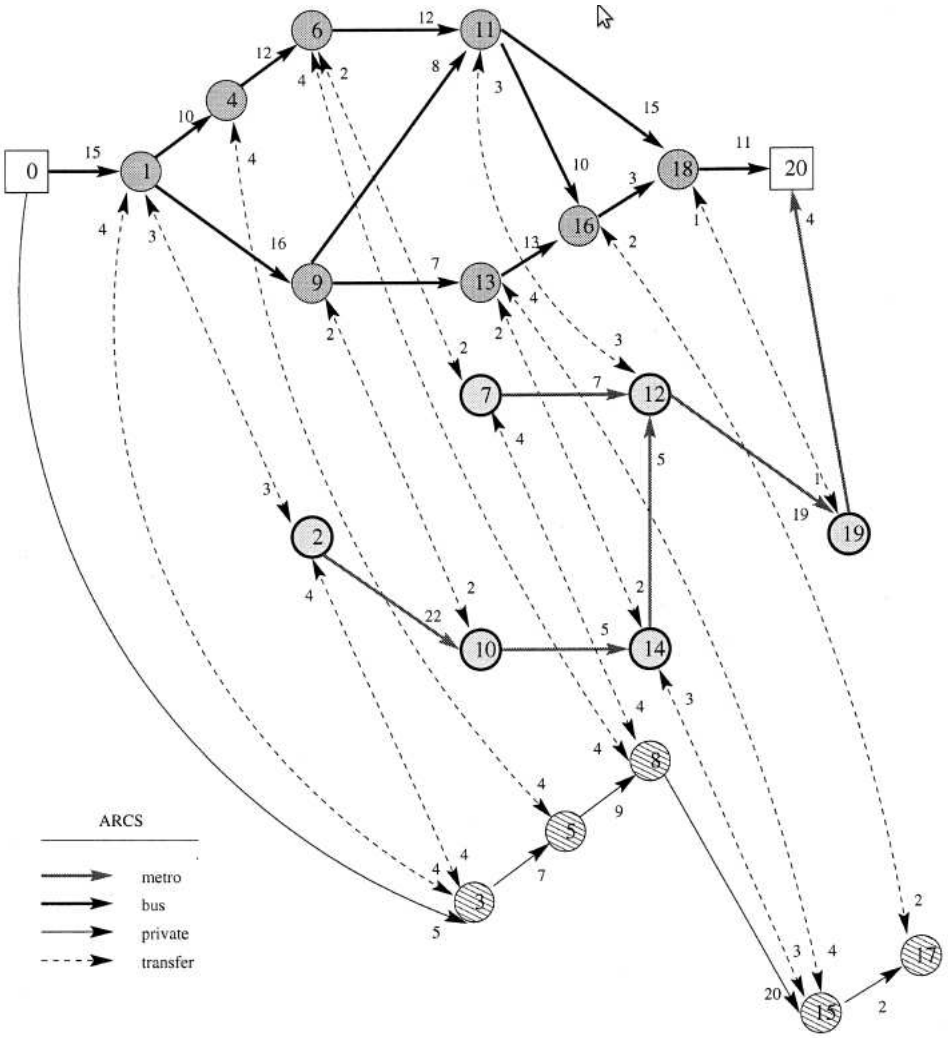

FIGURE 1.13 – Réseau multimodal modélisé par un multi-graphe [66]

Time-expandend et Time-dependent. Nous considérons dans cette section la spécification simplifiée du réseau suivante :

1. Le temps de transfert entre les modes à une station est négligeable
2. Chaque ligne de transport public est exploitée quotidiennement, c'est à dire l'intervalle de temps considéré s'étale sur une journée de 24 heures
3. Le mode marche à pied n'est pas utilisée dans le modèle et ne sera donc pas pris en compte dans le calcul.

1.4.1.1 Modèle Time-Expanded

Le modèle Time-expanded [96] est basé sur un graphe time-expanded orienté défini comme suit. Les noeuds du graphe correspondent à des paires localité et événement de temps (départ ou arrivée). Deux types d'arêtes existent.

– Arête de service : Pour chaque connexion élémentaire $c = (s, v_1, v_2, t_d, t_a)$ dans la grille des temps, il y a une *arête-service* dans le graphe, connectant le nœud de départ v_1 à l'instant

t_d et le nœud d'arrivée v_2 à l'instant t_a. v_1 et v_2 sont des localités.

– Arête pause : Pour chaque localité v, tous les nœuds sont ordonnés selon leur temps. Si $n_1, n_2, ...n_k$ est l'ensemble ordonné des nœuds de la localité v. L'ensemble des *arête-pause* représentant le temps d'attente à la localité v, est composé de l'ensemble des paires (n_i, n_{i+1}), $\forall i \in [1, k-1]$ et l'arête (v_k, v_1) .

(Voir [49] pour plus de détails). La figure 1.14 illustre un exemple d'un graphe time-expanded.

1.4.1.2 Modèle Time-Dependent

Le modèle dépendant du temps (Time-dependent) [21] est également basé sur un graphe orienté, appelé graphe dépendant du temps (ou graphe time-dependent). Dans ce graphe il y a un seul nœud par station (représentation classique), et il y a une arête e d'une localité A vers une localité B si il y a une connexion élémentaire de A à B. L'ensemble des connexions élémentaires de A à B est noté $C(e)$. Cette définition est illustrée dans la figure 1.14. Le coût d'une arête $e = (v, w)$ dépend de l'heure à laquelle cette arête sera utilisée lors de l'application de l'algorithme. En d'autres termes, si T est l'ensemble dénotant le temps, alors le coût (temps du parcours) d'une arête (v, w) est donné par la fonction $f_{(v,w)}(t) - t$, où t est le temps du départ de v, $f_{(v,w)} : T \to T$ est une fonction telle que $f_{(v,w)}(t) = t'$ et $t' \geq t$ est le temps d'arrivée le plus tôt possible à w.

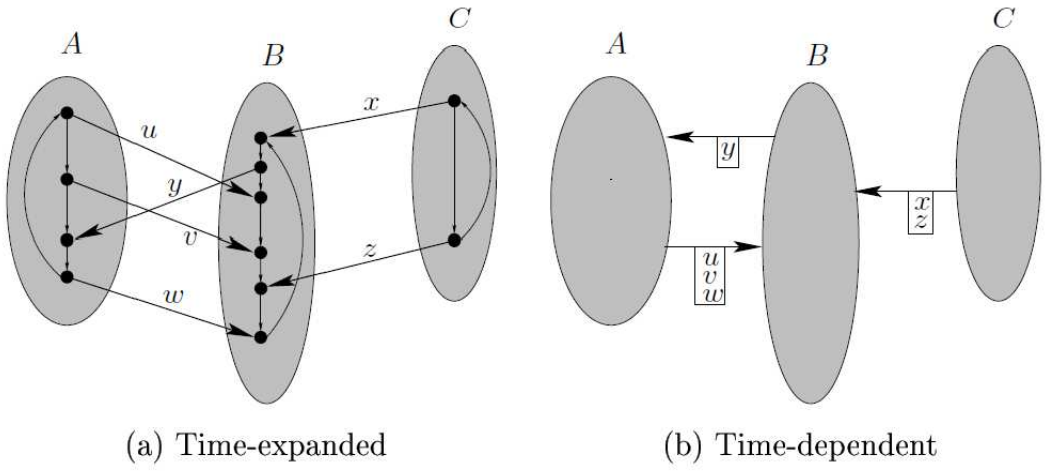

FIGURE 1.14 – Le graphe Time-expanded (a) et le graphe Time-dependent (b) avec trois stations A,B,C. Il y a 3 trains qui connectent A à B (connexions élémentaires u,v et w), un train de C à A passant par B (x, y) et un train de C à B (z). Exemple de [49]

1.4.1.3 Comparaison

Dans le scénario simplifié que nous étudions dans cette section, les graphes qui sont utilisés dans les deux approches sont fortement liés, connectant tous les nœuds qui appartiennent à la même station dans le graphe Time-expanded et la suppression des arêtes parallèles conduit au graphe Time-dependent. Cependant, le coût d'une arête doit être calculé dans l'algorithme avec un graphe Time-dependent, ce qui est coûteux en terme de temps de calcul.

1.4.2 Le modèle réel

Dans cette section, nous expliquons comment les approches introduites jusqu'à présent pour le problème simplifié du chemin le plus rapide, peuvent être étendues à des spécifications réalistes avec d'autres critères d'optimisation.

1.4.2.1 Le transfert

Pour intégrer les temps de transfert dans le modèle Time-expanded, le graphe *time-expanded réaliste* est construit comme suit [73,87,89]. À partir du graphe Time-expanded et pour chaque station, une copie de tous les nœuds de départ et d'arrivée de chaque localité est créée. Cette copie correspondra à un *nœud de transfert* (voir la figure 1.15. Les *arêtes-pause* relient alors les *nœuds de transfert*. Pour chaque nœud d'arrivée il y a deux arêtes sortantes : une arête au départ du même service, et une seconde arête au nœud de transfert avec une valeur de temps supérieur ou égal à la somme des temps des nœuds d'arrivée et le minimum de temps nécessaire pour changer de train à la localité en question.

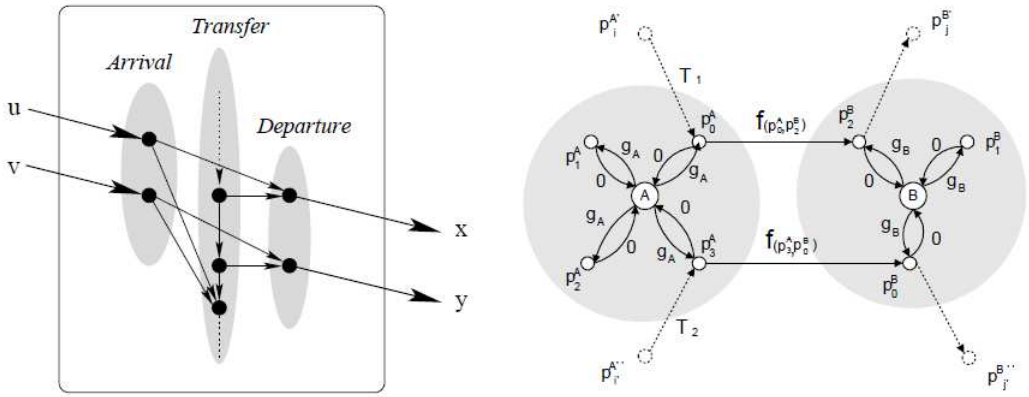

FIGURE 1.15 – Modélisation du transfert avec le modèle Time-expanded (gauche) et le modèle Time-dependent (droite)

Le modèle original «Time-dependent» est étendu dans [88], en utilisant des informations sur les itinéraires proposés par les services. Par conséquent, on suppose qu'on a un ensemble de lignes d'un service donné, par exemple le train, et leurs horaires respectifs. Dans la suite de ce travail, nous décrivons la construction du graphe «Time-dependent réaliste». On dit que les stations $S_0, S_1, ..., S_{k-1}, k > 0$ forment un *itinéraire du service* s'il y a un service qui démarre de S_0 et visite consécutivement S_1, S_2, ..., S_{k-1} dans l'ordre. S'il y a plus qu'une ligne de service suivant le même itinéraire (à l'égard de l'ordre dans lequel ils visitent les nœuds ci-dessus), alors on dit qu'ils appartiennent tous au même itinéraire du service.

L'ensemble de nœuds du graphe Time-dependent réaliste se compose :

- D'un ensemble S de nœuds représentant les localités.
- Pour chaque localité $s \in S$, d'un *nœud itinéraire* pour chaque itinéraire de service qui passe par s, noté p_i^s, où i est un index.
- Par trois types d'arêtes :
  1. une arête de chaque localité $s \in S$ vers tous les *nœud itinéraire* appartenant à s ;
  2. une arête de chaque noeud itinéraire vers sa localité modélisant ainsi la descente d'une ligne de service ;

3. pour chaque itinéraire de service, des arêtes connectant les localités correspondantes, modélisant ainsi la ligne du service lui même.

1.4.2.2 Marche à pied

Dans l'approche Time-dependent une *arête-à-pied* d'une localité A vers une localité B peut être facilement modélisée comme une arête dans le graphe Time-dependent réaliste, entre les deux nœuds représentant les localités A et B. Pour le problème du chemin le plus rapide, le coût de cette arête est le temps de marche. La modélisation de l'arête-à-pied dans le cas Time-expanded est évidemment faite par l'expansion de temps. Pour chaque nœud de transfert A, une arête est ajoutée vers le premier nœud de transfert possible de B.

1.4.2.3 Problème du nombre minimum de transferts

Le graphe Time-expanded réaliste, ainsi que le graphe Time-dependent réaliste, peuvent être utilisés pour résoudre *le problème du nombre minimum de transferts* avec une méthode similaire [72,73,87,89] : les arêtes qui modélisent les transferts, sont affectées à un coût égal à 1, et toutes les autres arêtes à zéro. Dans le cas Time-expanded, toutes les arêtes entrantes des nœuds de transfert ont un coût égal à 1, alors que dans le cas de Time-dependent, les arêtes qui représentent la desserte d'un service, en dehors de celles appartenant à la localité de départ, sont affectées à un coût égal à 1, et toutes les autres arêtes à zéro.

Conclusion

Une grande partie de ce chapitre concerne la problématique du transport multimodal dans un contexte purement socio-économique. Cette présentation "non scientifique" du problème nous a semblé utile pour bien aborder sa modélisation et sa résolution en tenant compte de tous ses paramètres. En effet, nous avons pu mettre en évidence que la modélisation du problème reposait sur les caractéristiques des différents modes de transport utilisés qu'ils soient publics ou privés, et sur leur mode de gestion. Elle dépend aussi d'un grand nombre de paramètres (coût, temps, distance, ...) pouvant influer sur le type de transport (modal, multimodal) et sur la façon de représenter le réseau. Pour finir, concernant les réseaux de transports multimodaux dépendant du temps, faisant l'objet de cette thèse, nous avons présenté de façon synthétique les différents modèles connus dans la littérature.

Chapitre 2

État de l'art sur l'optimisation

2.1 Théorie de la complexité

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté des problèmes en informatique. Elle se distingue de la théorie de la calculabilité (la théorie de la calculabilité – appelée aussi parfois théorie de la récursion – est une branche de la logique mathématique et de l'informatique théorique) qui s'attache à savoir si un problème peut être résolu par un ordinateur. La théorie de la complexité se concentre donc sur les problèmes qui peuvent effectivement être résolus, la question étant de savoir s'ils peuvent être résolus efficacement ou pas en se basant sur une estimation (théorique) des temps de calcul et des besoins en mémoire informatique [2].

2.1.1 Complexité algorithmique

Un algorithme a besoin de deux ressources importantes pour résoudre un problème : le temps et l'espace. La complexité en temps d'un algorithme est le nombre d'étapes requises pour résoudre un problème de taille n. La complexité est généralement définie en terme de l'analyse du pire cas.

L'objectif de la détermination de la complexité du calcul d'un algorithme n'est pas d'obtenir un décompte précis, mais une borne asymptotique sur le nombre d'instructions réalisées. la notation Big-O (Grand O) fait usage de l'analyse asymptotique. C'est l'une des notations les plus populaires dans l'analyse des algorithmes.

Définition 2.1.1

Complexité Big-O (Grand O) : *Un algorithme a une complexité* $f(n) = O(g(n))$ *si il existe une constante positive* n_0 *et* c *telle que* $\forall n \geq n_0, f(n) \leq g(n)$.

Dans ce cas, la fonction $f(n)$ est majorée par la fonction $g(n)$. La notation Big-O peut être utilisée pour calculer le temps ou la complexité en espace d'un algorithme.

2. $http://www.techno-science.net/?onglet=glossaire\&definition=6231$

Définition 2.1.2

Algorithme polynomial : *Un algorithme est polynomial si sa complexité est $O(p(n))$, où $p(n)$ est une fonction polynomiale de n.*
Un fonction polynomiale de n de degré k, peut être définie comme suit :
$p(n) = a_k.n^k + a_{k-1}.n^{k-1} + ... + a_1.n^1 + a_0.n^0$ *où* $a_k > 0$
Un tel algorithme admet une complexité polynomiale de $O(n^k)$.

Définition 2.1.3

Algorithme exponentiel : *Un algorithme est exponentiel si sa complexité est de $O(c^n)$, où c est une constante strictement positive supérieure à 1.*

Le tableau 2.1 illustre la façon dont le temps de recherche d'un algorithme croît avec la taille du problème en utilisant des algorithmes avec différentes complexités. Le tableau montre clairement l'explosion combinatoire des complexités exponentielles par rapport aux complexités polynomiales. Attendre un résultat de calcul pendant un temps très long n'est pas réaliste et autant dire que le problème n'a pas de solution.

Complexité	Taille = 10	Taille = 20	Taille = 30	Taille = 40	Taille = 50
$O(x)$	$0.00001s$	$0.00002s$	$0.00003s$	$0.00004s$	$0.00005s$
$O(x^2)$	$0.0001s$	$0.0004s$	$0.0009s$	$0.0016s$	$0.0025s$
$O(x^5)$	$0.1s$	$0.32s$	$24.3s$	$1.7mn$	$5.2mn$
$O(2^x)$	$0.001s$	$1.0s$	$17.9mn$	12.7 jours	35.7 ans
$O(3^x)$	$0.059s$	$58.0mn$	6.5 ans	3855 siècles	2.10^8 siècles

TABLE 2.1 – Temps du calcul d'un algorithme en fonction de la taille du problème utilisant différentes complexités (de [41])

Deux autres notions sont utilisées pour analyser les algorithmes : la notation Big-Ω (Grand Ω) et la notation Big-Θ (Grand Θ).

Définition 2.1.4

Complexité Big-Ω : *Un algorithme a une complexité $f(n) = \Omega(g(n))$ s'il existe des constantes positives n_0 et c telles que $\forall n > n_0, f(n) \geq c.g(n)$. La complexité de l'algorithme de $f(n)$ est minorée par la fonction $g(n)$.*

Définition 2.1.5

Complexité Big-Θ : *Un algorithme a une complexité $f(n) = \Theta(g(n))$, s'il existe des constantes positives n_0, c_1 et c_2 telles que $\forall n > n_0, c_1.g(n) \leq f(n) \leq c_2.g(n)$ La complexité de l'algorithme $f(n)$ est strictement majorée par la fonction $g(n)$.*

Il est plus facile de trouver d'abord la complexité $Big - O$ d'un algorithme, puis de déduire successivement les complexités Big-Ω et Big-Θ. La notation Big-Θ, définit une borne exacte (inférieure et supérieure) sur la complexité en temps d'un algorithme.

L'analyse asymptotique des algorithmes caractérise le taux de croissance de leur complexité en temps, en fonction de la taille du problème (les problèmes d'évolution). Elle permet une comparaison théorique de différents algorithmes, en termes de complexité dans le pire cas. Elle ne précise pas le temps d'exécution pratique de l'algorithme pour une instance donnée du problème. En effet, le temps d'exécution d'un algorithme dépend des données d'entrée. Pour une analyse plus complète, on peut aussi étudier les complexités moyennes, même si ces dernières sont plus difficiles à obtenir.

2.1.2 Complexité des problèmes

La complexité d'un problème est équivalente à la complexité de son meilleur algorithme de résolution. Un problème est soluble (ou facile) s'il existe un algorithme polynomial pour le résoudre. Un problème est non soluble (ou difficile) si aucun algorithme polynomial n'existe pour résoudre le problème. La théorie de la complexité des problèmes traite *les problèmes de décision* qui ont toujours une réponse binaire (oui ou non).

Problème du nombre premier : L'un des problèmes de décision les plus populaires est celui du nombre premier, qui consiste à déterminer si un nombre donné est premier ou non. La réponse donc est «oui» si le nombre est premier, et «non» s'il n'est pas.

Un aspect important de la théorie de la calculabilité est de caractériser un problèmes en classes. Une classe de complexité représente l'ensemble de tous les problèmes pouvant être résolus moyennant une certaine capacité de calcul. Deux classes importantes de complexités se dégagent : les classes P et NP (voir la Figure 2.1).

Un problème p est NP-difficile (NP-Hard) si tout problème de la classe NP peut être efficacement réduit à p par une réduction polynomiale. En d'autres termes, un problème est NP-difficile si il y a un algorithme en temps polynomial qui transforme toute instance d'un problème NP en une instance du problème p, ce qui signifie que p est au moins aussi difficile que tous les autres problèmes de la classe NP.

Un problème est dit NP-complet si il s'agit d'un problème NP-difficile appartenant à la classe NP. Autrement dit, les problèmes NP-complets sont «les problèmes les plus difficiles de la class NP». La grande découverte du début des années 70 est que des milliers de problèmes de recherche de la vie courante sont NP-complets. Nous pouvons citer sans pour autant être exhaustifs le problème de coloration de graphe, le problème TSP (Traveling Salesman Problem), le problème de satisfaisabilité... Du point de vue théorique, ces problèmes sont équivalents et correspondent tous à des codages différents d'un seul et unique problème.

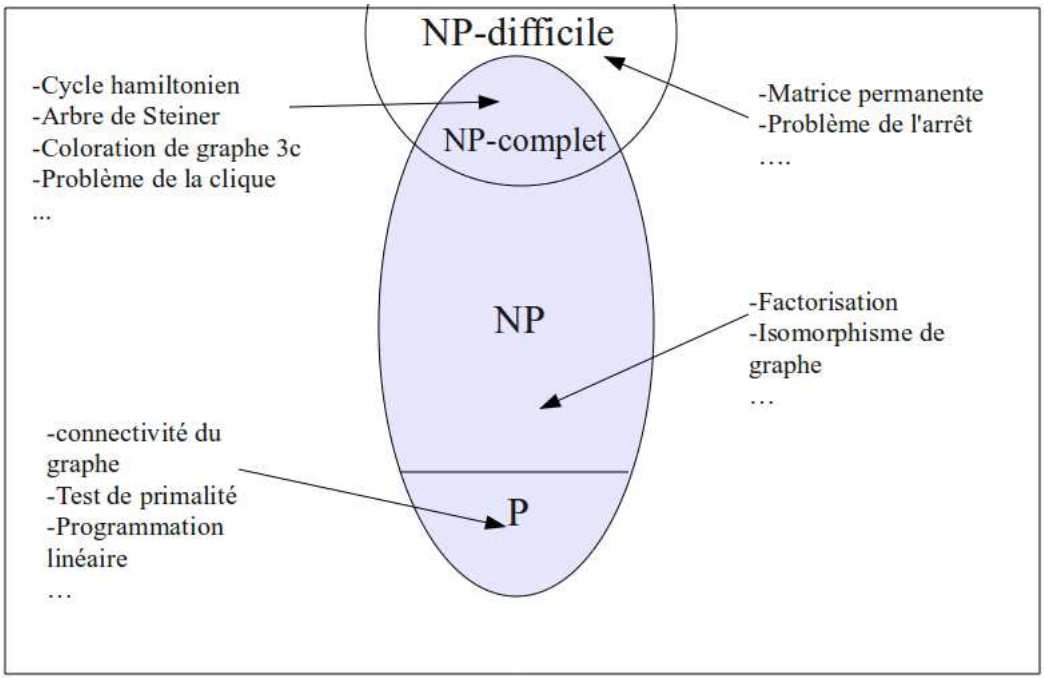

FIGURE 2.1 – Classes des problèmes

2.2 Modèles d'optimisation

En tant que scientifiques, ingénieurs et managers, nous avons toujours à prendre des décisions. La prise de décision est partout. Comme le monde devient de plus en plus complexe et concurrentiel, la prise de décision doit être abordée d'une façon rationnelle et optimale. La prise de décision peut se décomposer en plusieurs étapes successives comme présenté sur la figure 2.2 extraite de [100] :

- **Formulation du problème :** Dans cette première étape, un problème de décision est identifié. Puis, une déclaration initiale du problème est faite. Cette formulation peut être imprécise. Les facteurs internes et externes de l'objectif (s) du problème sont décrites. De nombreux décideurs peuvent être impliqués dans la formulation du problème.

- **Modélisation du problème :** Dans cette étape importante, un modèle mathématique abstrait est créé pour représenter le problème. Le modélisateur peut s'inspirer des modèles qui existent dans la littérature. Cela permettra de réduire le problème à des modèles d'optimisation bien étudiés. Habituellement, les modèles que nous résoudrons sont des simplifications de la réalité. Ils impliquent des approximations et parfois évitent les processus qui sont complexes à représenter dans un modèle mathématique. Une question intéressante qui peut se poser est la suivante : pourquoi résoudre de manière exacte les problèmes d'optimisation de la vie réelle qui sont flous par nature ?

- **Optimisation du problème :** Une fois que le problème est modélisé, la procédure de résolution génère une «bonne» solution pour ce dernier. La solution peut être optimale ou suboptimale. Notons que nous cherchons une solution pour un modèle abstrait du problème et non pas pour le problème original réaliste. Par conséquent, les performances de la solution obtenue, sont indicatives. Pour résoudre un problème, il est possible de réutiliser des algorithmes génériques déjà connus dans littérature ou de développer des algorithmes ad-hoc tenant compte des spécificités du problème à résoudre.

- **Implémentation de la solution :** La solution obtenue est testée en pratique par le décideur et est mise en œuvre si elle est «acceptable». Certaines connaissances pratiques peuvent être introduites dans la solution à mettre en œuvre. Si la solution est inacceptable, le modèle et/ou l'algorithme d'optimisation doit être amélioré et le processus décisionnel est réitéré.

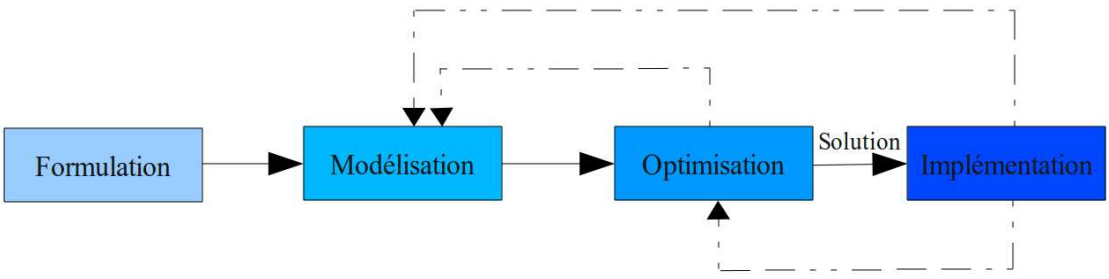

FIGURE 2.2 – Processus classique dans la prise de décision

2.2.1 Modèle d'optimisation classique

Comme mentionné précédemment, les problèmes d'optimisation se retrouvent dans de nombreux domaines : sciences, ingénierie, gestion et affaires.

Un problème d'optimisation peut être défini par le couple (S, f), où S représente l'ensemble des solutions réalisables, et f la fonction objectif à optimiser. La fonction objectif assigne à chaque solution $s \in S$ de l'espace de recherche un nombre réel indiquant sa valeur. La fonction objectif f permet de définir une relation d'ordre total entre toute paire de solutions dans l'espace de recherche.

Définition 2.2.1

Optimum Global : *Une solution $s^* \in S$ est un optimum global si elle admet une meilleure fonction objectif dans l'espace de recherche S, d'où $\forall s \in S, f(s^*) \leq f(s)$* [3].

Ainsi, le but principal dans la résolution d'un problème d'optimisation est de trouver une solution globale optimale s^*. Beaucoup de solutions globales optimales peuvent exister pour un problème donné. Ainsi, le problème peut également être défini comme la recherche de l'ensemble des solutions globales optimales.

Différentes familles de modèles d'optimisation sont utilisés dans la pratique pour formuler et résoudre les problèmes décisionnels (figure 2.3). Les modèles qui ont le plus de succès sont basés sur la programmation mathématique et la programmation par contraintes.

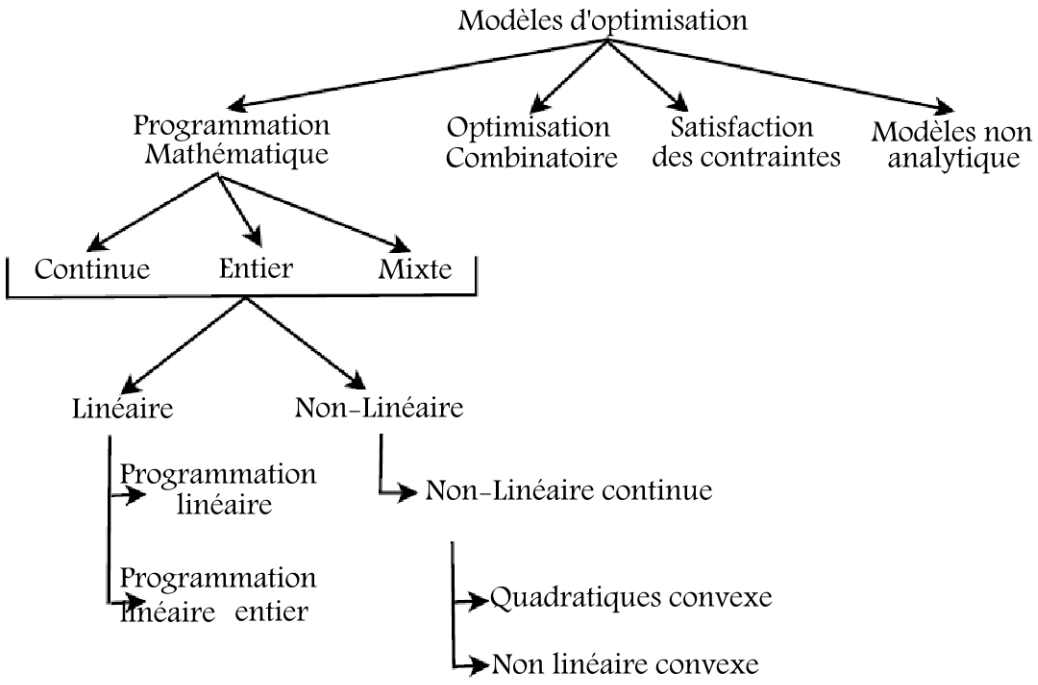

FIGURE 2.3 – Les modèles d'optimisation classiques. Les différentes classes se chevauchent parfois [100].

Un modèle couramment utilisé dans la programmation mathématique est le problème de programmation linéaire LP (Linear Programming), formulé comme suit :

Min $c.x$

sous contraintes : $\begin{cases} A.x \geq b \\ x \geq 0 \end{cases}$ où x est un vecteur de variables de décision continues, et c

et b (resp. A) sont des vecteurs constants (resp. matrice) de coefficients.

Dans un problème d'optimisation de programmation linéaire, la fonction objectif $c.x$ à optimiser et les contraintes $A.x \leq b$ sont toutes des fonctions linéaires. La programmation linéaire est l'un des modèles les plus satisfaisants pour la résolution des problèmes d'optimisation. En effet, pour ces problèmes d'optimisation linéaire continue, il existe des algorithmes efficaces exacts, pour ne citer que la méthode du simplex [29] ou les méthodes de points intérieurs [61]. L'efficacité de ces algorithmes est due au fait que l'espace réalisable du problème est un ensemble convexe et la fonction objectif est une fonction convexe. Ensuite, la solution globale optimale est nécessairement un nœud du polytope représentant l'espace réalisable (voir figure 2.4). En outre, toute solution optimale locale est un optimum global.

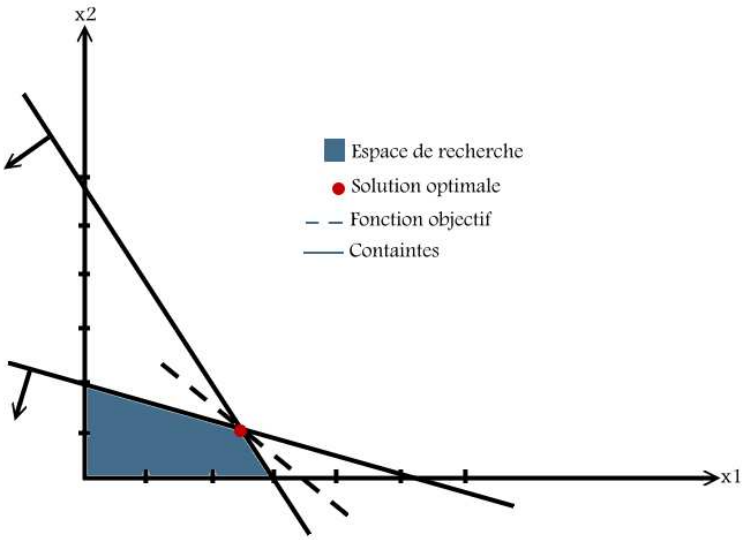

FIGURE 2.4 – Illustration graphique du modèle LP et de sa résolution

Les modèles de programmation non linéaire (NLP) font face à des problèmes de programmation mathématique où la fonction objectif et/ou les contraintes sont non linéaires [15]. Un problème d'optimisation non linéaire continu consiste à minimiser une fonction $f : S \subset R^n \rightarrow R$ dans un domaine continu. Les modèles non-linéaires continus sont cependant beaucoup plus difficile à résoudre, même s'il y a de nombreuses possibilités de modélisation qui peuvent être utilisés pour linéariser un modèle [4, 44]. Les techniques de linéarisation introduisent généralement des variables et des contraintes supplémentaires dans le modèle et dans certains cas un certain degré d'approximation [43].

La théorie de l'optimisation continue en termes d'algorithmes d'optimisation est plus développée que l'optimisation discrète. Cependant, il y a beaucoup d'applications réelles qui doivent être modélisées par des variables discrètes. Les modèles continus sont inappropriés pour ces problèmes. En effet, dans de nombreux problèmes d'optimisation pratiques, les ressources sont indivisibles (machines, personnes, etc.) Dans un modèle de programmation en nombres entiers, les variables de décision sont discrètes [77].

Lorsque les variables de décision sont à la fois discrètes et continues, nous avons affaire à des

problèmes de programmation en nombres entiers mixtes (MIP). Ainsi, les modèles MIP généralisent les modèles LP et les modèles IP. Résoudre des problèmes MIP s'est considérablement amélioré ces dernières années avec l'utilisation de techniques d'optimisation avancées telles que la relaxation et les approches de décomposition.Pour les modèles IP et le MIP, les algorithmes énumératifs tels que celui basée sur la séparation peut être utilisé pour de petites instances. La taille n'est pas le seul indicateur de la complexité du problème, mais aussi sa structure. Les algorithmes basés sur des métaheuristiques, contribuent de façon compétitive à la résolution des problèmes de cette classe, et ont pour but d'obtenir des solutions aux problèmes réputés être trop complexes pour être résolus par des méthodes exactes. Les métaheuristiques peuvent être aussi être utilisées pour générer de bonnes bornes inférieures ou supérieures aux algorithmes exacts et à améliorer leur efficacité. Notons qu'il y a quelques problèmes faciles, comme les problèmes de «Network flow», où la programmation linéaire génère automatiquement des valeurs entières. Les deux approches, programmation en nombres entiers et métaheuristiques, ne sont donc pas utiles pour résoudre ces classes de problèmes.
Une classe plus générale des problèmes IP sont les problèmes d'optimisation combinatoire. Cette classe de problèmes est caractérisée par des variables de décision discrètes et un espace de recherche fini. Cependant, la fonction objectif et les contraintes peuvent prendre n'importe quelle forme [86].
La popularité de problèmes d'optimisation combinatoire provient du fait que dans de nombreux problèmes du monde réel, la fonction objectif et les contraintes sont de nature différente (non linéaire, non analytiques, la boîte noire, etc), alors que l'espace de recherche est fini.
Une autre approche populaire pour modéliser les problèmes de décision est la programmation par contraintes (CP), un paradigme de programmation qui intègre des outils de modélisation plus riches que les expressions linéaires des modèles MIP. Un modèle est composé d'un ensemble de variables. Chaque variable a un domaine fini de valeurs. Dans le modèle, les contraintes symboliques et mathématiques liées à des variables peuvent être exprimées. Les modèles déclaratifs de CP sont flexibles et sont en général plus compacts que dans les modèles MIP.

2.2.2 Autres modèles d'optimisation

On remarque une utilisation croissante des problèmes d'optimisation dans le monde réel où les données sont bruitées et la fonction objectif change de façon dynamiquement. Trouver des solutions robustes pour certains problèmes de conception est un autre défi important dans l'optimisation. Une transformation vers des problèmes déterministes et statiques est souvent proposée pour résoudre ces problèmes. En outre, certaines adaptations peuvent être proposées pour les métaheuristiques en termes d'intensification et de diversification de la recherche pour s'attaquer à cette classe de problèmes [58].

2.2.2.1 Optimisation sous incertitude

Dans de nombreux problèmes d'optimisation concrets, les données d'entrée sont soumis aux parasites. Il existe différentes sources de parasites. Par exemple, l'utilisation d'un simulateur stochastique ou un dispositif de mesure intrinsèquement bruyants tels que les capteurs, introduit un parasite additif dans la fonction objectif. Pour une solution x donnée dans l'espace de recherche, une fonction objectif bruyante peut être défini mathématiquement comme suit :

$$f_{noisy}(x) = \int_{-\infty}^{+\infty}[f(x) + z]p(z)dz$$

2.3 Techniques de résolution des problèmes d'optimisation

Selon sa complexité, un problème d'optimisation peut être résolu par une méthode exacte ou une méthode approximative (figure 2.5 [100]). Les méthodes exactes [4] l'objectif est d'obtenir des solutions optimales et garantir leur optimalité. Pour les problèmes NP-complets, les algorithmes sont non polynomiaux (à moins que $P = NP$). Les méthodes approximatives (ou heuristiques) génèrent des solutions de haute qualité en un temps raisonnable pour une utilisation pratique, mais sans garantie de trouver une solution optimale globale.

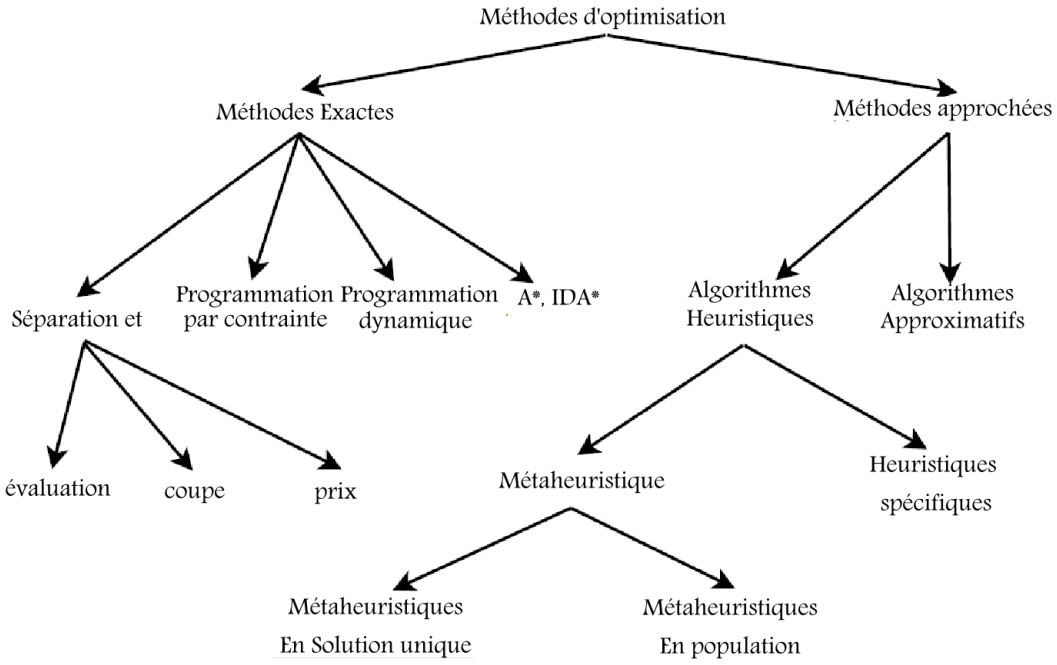

FIGURE 2.5 – Méthodes d'optimisations ([100])

2.3.1 Méthodes exactes

Dans cette classe on trouve des algorithmes classiques : programmation dynamique (dynamic programming), méthodes de séparation [5] (branch and bound, branch and cut, et branch and price) développées dans la communauté de la recherche opérationnelle, programmation par contraintes. On trouve aussi des algorithmes de recherche A^* (A^*, IDA^*) [63] développés dans la communauté de l'intelligence artificielle [91]. Ces méthodes énumératives peuvent être considérées comme des algorithmes de recherche arborescente. La recherche est effectuée par exploration de l'espace de recherche et par décomposition d'un problème en sous problèmes.

2.3.1.1 Programmation dynamique

La programmation dynamique est basée sur la division récursive d'un problème en sous problèmes plus simples. Cette procédure est fondée sur le principe de Bellman stipulant que «La

4. Dans la communauté de l'intelligence artificielle, ces algorithmes sont également nommés algorithmes complets.

5. Connus aussi par la méthode diviser et régner

sous-politique d'une politique optimale est elle même optimale» [13]. Cette méthode d'optimisation échelonnée est le résultat d'une séquence de décisions partielles. La procédure permet d'éviter une énumération totale de l'espace de recherche par des élagages des séquences de décision partielle qui ne peuvent pas conduire à la solution optimale.

La programmation dynamique est similaire à la méthode «diviser et régner» en ce sens que, une solution d'un problème dépend des solutions de ses sous-problèmes. La différence significative entre ces deux méthodes est que la programmation dynamique permet aux sous-problèmes de se superposer. Autrement dit, un sous-problème peut être utilisé dans la solution de deux sous-problèmes différents. Tandis que l'approche «diviser et régner» crée des sous-problèmes qui sont complètement séparés et peuvent être résolus indépendamment les uns des autres. Une illustration de cette différence est montrée par la figure 2.6. Dans cette figure, le problème à résoudre est à la racine, et les descendants sont les sous-problèmes, plus faciles à résoudre. Les feuilles de ce graphe constituent des sous-problèmes dont la résolution est triviale. Dans la programmation dynamique, ces feuilles constituent souvent les données de l'algorithme. La différence fondamentale entre ces deux méthodes devient alors claire : les sous-problèmes dans la programmation dynamique peuvent être en interaction, alors que dans la méthode «diviser et régner», ils ne le sont pas.

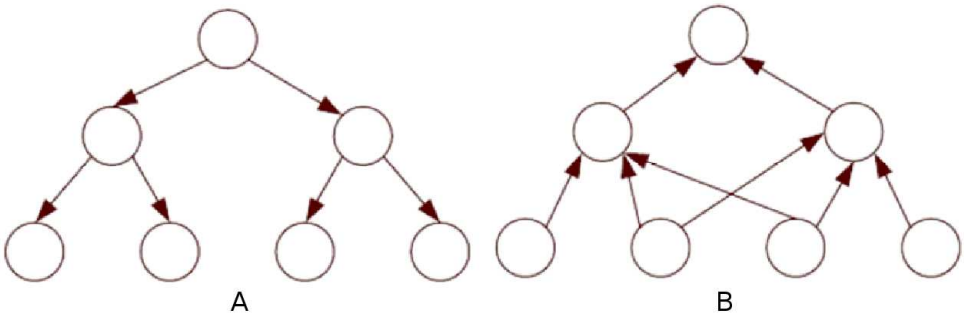

FIGURE 2.6 – Différence entre la programmation dynamique (B) et la méthode diviser et régner (A)

Une seconde différence entre ces deux méthodes est, comme l'illustre la figure 2.6, que la méthode «diviser et régner» est récursive : les calculs se font de haut en bas. Tandis que la programmation dynamique est une méthode dont les calculs se font de bas en haut : on commence par résoudre les plus petits sous-problèmes. En combinant leur solution, on obtient les solutions des sous-problèmes de plus en plus grands.

2.3.1.2 La méthode diviser et régner et la méthode A^*

L'algorithme de séparation et A^* sont basés sur une énumération implicite de toutes les solutions du problème d'optimisation considéré. L'espace de recherche est exploré par une construction dynamique d'un arbre dont la racine représente le problème à résoudre et son espace entier de recherche associés. Les feuilles sont les solutions possibles et les nœuds internes sont des sous-problèmes de l'espace totale. L'élagage de l'arbre de recherche est basé sur une fonction englobante qui élague une arborescence qui ne peut contenir la solution optimale.
La construction d'un tel arbre et son exploration sont effectuées en utilisant deux opérateurs principaux : séparation (branching) et l'élagage (pruning) (figure 2.7). L'algorithme se déroule en plusieurs itérations au cours desquelles la meilleure solution trouvée est progressivement améliorée. Les nœuds générés et non encore traités sont maintenus dans une liste dont le contenu initial est limité uniquement au nœud racine. Les deux opérateurs interviennent à chaque itéra-

tion de l'algorithme. La stratégie de séparation détermine l'ordre dans lequel les branches sont explorées. Beaucoup de stratégies de séparation peuvent être appliquées telles que la profondeur d'abord, la largeur d'abord, et la stratégie de meilleures d'abord. La stratégie d'élagage élimine les solutions partielles qui ne mènent pas à des solutions optimales. Cela se fait par le calcul de la borne inférieure associée à une solution partielle. Si la borne inférieure d'un nœud (solution partielle) est plus grande que la meilleure solution trouvée à ce jour ou une borne supérieure connue du problème, l'exploration du nœud n'est pas nécessaire. L'algorithme se termine si il n'y a pas plus de nœuds à explorer ou tous les nœuds sont éliminés. Ainsi, les concepts les plus importants dans la conception d'une branche et dans l'efficacité de l'algorithme sont liés à la qualité des bornes et à la stratégie de séparation.

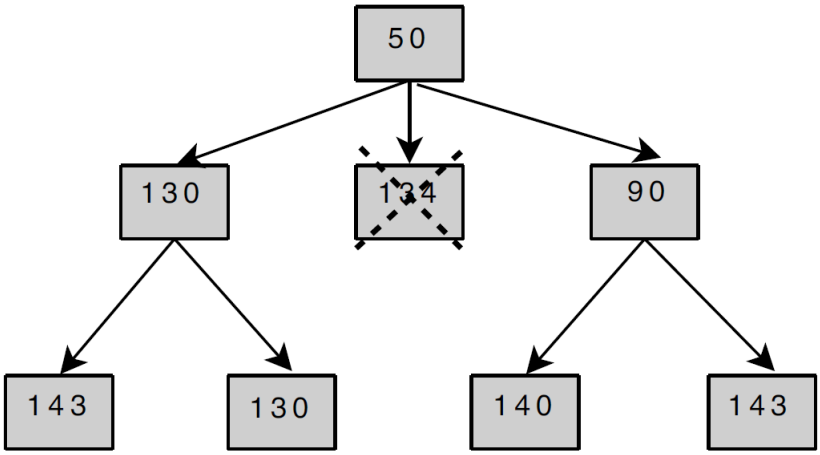

FIGURE 2.7 – Algorithme de séparation. Cette figure montre les nœuds réellement explorés dans le problème, par exemple, en supposant une stratégie de recherche en profondeur d'abord et de gauche à droite. Le sous-arbre enraciné au deuxième nœud sur le niveau 2 est élagué, car le coût de ce nœud (134) est supérieure à celle de la solution la moins chère déjà trouvé (130)

2.3.1.3 Programmation par contraintes

La programmation par contraintes est un langage construit autour des concepts de l'arbre de recherche et les implications logiques. Les problèmes d'optimisation en programmation par contraintes sont modélisés au moyen d'un ensemble de variables liées par un ensemble de contraintes. Les variables prennent leurs valeurs sur un domaine fini d'entiers. Les contraintes peuvent avoir des formes mathématiques ou symboliques. Les contraintes globales se réfèrent à un ensemble de variables du problème. Un exemple de telles contraintes globales est *all-different*$(x_1, x_2, ..., x_n)$, qui précise que toutes les variables $x_1, x_2, ..., x_n$ doivent être différentes.
Actuellement, nous voyons deux branches de la programmation par contraintes, la satisfaction de contraintes (constraint satisfaction) et la résolution de contraintes (constraint solving). Les deux partagent la même terminologie, mais les origines et les technologies de résolution sont différentes. Pour plus de détails sur la programmation par contraintes voir [11].

Satisfaction de contraintes Les problèmes de satisfaction de contraintes [102] ont été un sujet de recherche en Intelligence Artificielle durant de nombreuses années. Un problème de sa-

tisfaction de contraintes (CSP) est défini comme suit :

- Un ensemble de variables $X = x_1, x_2, ..., x_n$
- Pour chaque variable x_i, un ensemble fini de valeurs possibles D_i (son domaine),
- Un ensemble de contraintes restreignant les valeurs que les variables peuvent prendre simultanément.

Une solution d'un CSP est une affectation d'une valeur de son domaine à chaque variable, de telle façon que toutes les contraintes soient satisfaites à la fois. Il existe plusieurs types de problèmes CSP :

- Recherche d'une seule solution, avec aucune préférence.
- Recherche de toutes les solutions.
- Recherche de la solution optimale, ou du moins une bonne solution, étant donné une certaine fonction objectif définie en termes de certaines variables ou de toutes les variables.

Un CSP peut être résolu par des algorithmes énumératifs ou algorithmes de recherche systématique à travers les affectations possibles de valeurs aux variables. Les méthodes de recherche se divisent en deux grandes catégories, celles qui explorent l'espace des solutions partielles (ou les affectations de valeur partielle), et celles qui explorent l'espace d'assignations de valeurs complètes (pour toutes les variables) stochastiquement [102].

Optimisation de contraintes Dans de nombreuses applications réelles, on ne cherche pas à trouver une solution, mais une bonne solution. La qualité de la solution est généralement mesurée par une fonction qui dépend de l'application appelée fonction objectif. Le but est de trouver une telle solution qui satisfait toutes les contraintes tout en minimisant ou maximisant la fonction objectif, respectivement. Ces problèmes sont considérés comme des problèmes de satisfaction d'optimisation des contraintes (CSOP).

2.3.2 Algorithmes approximatifs

Dans la classe des méthodes approximatives, deux sous-classes d'algorithmes peuvent être distinguées : les algorithmes d'approximation et algorithmes heuristiques. Contrairement aux méthodes heuristiques, qui trouvent habituellement d'assez «bonnes» solutions dans un délai raisonnable, les algorithmes d'approximation fournissent une qualité de solution probable et un temps de fonctionnement raisonnable.

Les méthodes heuristiques trouvent des «bonnes» solutions au problème des instances de grande taille. Elles permettent d'obtenir des performances acceptables à des coûts acceptables pour un large éventail de problèmes. En général, les heuristiques n'ont aucune garantie de convergence. Elles peuvent être classées en deux familles : les heuristiques spécifiques et les métaheuristiques.

- Les heuristiques spécifiques sont adaptées et conçues pour résoudre un problème spécifique et/ou d'une instance donnée.

- Les métaheuristiques sont des algorithmes génériques qui peuvent être appliquées pour résoudre presque n'importe quel problème d'optimisation. Elles peuvent être considérées comme des méthodes de niveau supérieur pouvant être utilisées comme une stratégie conduisant à la conception d'heuristiques sous-jacente pour résoudre des problèmes d'optimisation spécifiques.

2.3.2.1 Algorithmes d'approximation

Dans les algorithmes d'approximation, il y a une garantie sur les bornes de la solution obtenue à partir de la solution optimale globale [54]. Un algorithme ϵ-approximation génère une solution rapprochée à un facteur ϵ de la solution optimale globale [104].

Définition 2.3.1

ϵ-approximation algorithm *Un algorithme a un facteur d'approximation ϵ si sa complexité en temps polynomial et pour n'importe quelle instance d'entrée, il produit une solution*[6] *telle que :*

$$a \leq \epsilon.s \quad if \epsilon > 1$$
$$\epsilon.s \leq a \quad if \epsilon < 1$$

où s est la solution optimale globale, et le facteur ϵ définit la garantie de performance relative. Le facteur ϵ peut être une constante ou une fonction de la taille de l'instance du problème traité.

Un algorithme dont le facteur d'approximation est ϵ produira une solution avec des performances absolues garanties, si la propriété suivante est vérifiée :

$$(s - \epsilon) \leq a \leq (s + \epsilon)$$

L'objectif d'un algorithme d'approximation pour un problème est de trouver les **limites restreintes** dans le pire des cas. L'étude des algorithmes d'approximation informe plus sur la difficulté du problème et peut aider à concevoir des heuristiques efficaces. Cependant, les algorithmes d'approximation sont spécifiques au problème d'optimisation cible. Cette caractéristique limite leur applicabilité. Par ailleurs, dans la pratique, des approximations réalistes sont trop loin de la solution optimale globale, faisant que ces algorithmes ne soient pas pas très utiles pour de nombreuses applications réelles.

2.3.2.2 Les métaheuristiques

Contrairement aux méthodes exactes, les métaheuristiques permettent d'aborder des instances des problèmes de grande taille en offrant des solutions satisfaisantes dans un délai raisonnable. Il n'y a aucune garantie de trouver des solutions globales optimales ou des solutions encore bornées. Les métaheuristiques sont de plus en plus populaires ces 20 dernières années. Leur utilisation dans de nombreuses applications montre leur efficacité pour résoudre les problèmes importants et complexes.

Dans la conception d'une métaheuristique, deux critères contradictoires doivent être pris en compte : l'exploration de l'espace de recherche (diversification) et l'exploitation des meilleures solutions trouvées (intensification) (figure 2.8 de [100]). Les régions prometteuses sont déterminées par l'obtention des «bonnes» solutions. Dans l'intensification, les régions prometteuses sont explorées plus en détail dans l'espoir de trouver de meilleures solutions. Dans la diversification, les régions non explorées doivent être visitées pour être sûr que toutes les régions de l'espace de recherche sont régulièrement explorées et que la recherche ne se limite pas uniquement à un nombre réduit de régions. Dans cet espace de conception, les algorithmes de recherche extrêmes en termes de l'exploration (respectivement l'exploitation) sont basés sur la recherche aléatoire (respectivement recherche itérative d'amélioration locale). Dans la recherche aléatoire, à chaque

itération une solution aléatoire est générée dans l'espace de recherche. Dans l'algorithme de recherche locale de base, à chaque itération on sélectionne la meilleure solution voisine qui améliore la solution actuelle.

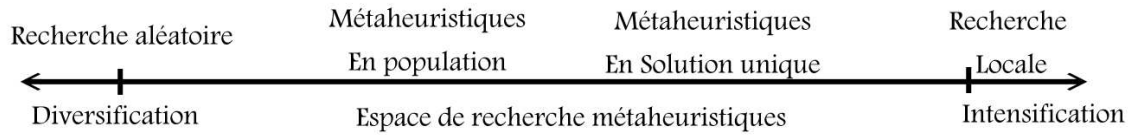

FIGURE 2.8 – Deux critères contradictoires dans la conception d'une métaheuristique : exploration (diversification) par rapport à l'exploitation (intensification)

Beaucoup de critères peuvent être utilisés pour classifier les métaheuristiques :

- **L'inspiration naturelle** . Plusieurs métaheuristiques sont inspirés des processus naturels : les algorithmes évolutionnaires et les systèmes artificiels provenant de la biologie ; les colonies de fourmis, colonies abeilles, l'optimisation en essaim de particules et le recuit simulé issu de de la physique.

- **L'usage de la mémoire** : Certains algorithmes métaheuristiques sont sans mémoire, aucune information extraite dynamiquement n'est utilisée durant la recherche. Certains représentants de cette classe sont la recherche locale, le GRASP (Greedy Randomized Adaptive Search Procedure), et le recuit simulé. Alors que d'autres métaheuristiques utilisent une mémoire qui contient des informations extraites dynamiquement pendant la recherche. Par exemple, les mémoires à court terme et à long terme dans la recherche tabou.

- **déterministe ou stochastique**. Une métaheuristique déterministe résout un problème d'optimisation en prenant des décisions déterministes (par exemple, la recherche locale, recherche tabou). Dans les métaheuristiques stochastiques, certaines règles sont appliquées au hasard lors de la recherche (par exemple, recuit simulé, algorithmes évolutionnaires). Dans les algorithmes déterministes, utiliser la même solution initiale conduira à la même solution finale, alors que dans les métaheuristiques stochastiques, différentes solutions finales peuvent être obtenues à partir de la même solution initiale. Cette caractéristique doit être prise en compte dans l'évaluation de la performance des algorithmes métaheuristiques.

- **La recherche avec population ou solution unique**. Les algorithmes basés sur la solution unique (par exemple, la recherche locale, recuit simulé) manipulent et transforment une solution unique lors de la recherche tandis que dans algorithmes à base de population (par exemple : essaim de particules, les algorithmes évolutionnaires) font évoluer toute une population de solutions. Ces deux familles ont des caractéristiques complémentaires : les métaheuristiques basées sur une seule solution sont orientées exploitation et ont le pouvoir d'intensifier la recherche dans les régions locales. Les métaheuristiques basées sur une population sont orientées exploration et permettent une meilleure diversification dans l'espace de recherche tout entier.

- **Recherche itérative ou gloutonne («greedy»)**. Dans les algorithmes itératifs, nous commençons par une solution complète (ou par une population de solutions) qui sera transformée à chaque itération en utilisant des opérateurs de recherche. Les algorithmes gloutons considèrent au départ une solution vide, et à chaque étape une variable de décision

du problème est assignée jusqu'à ce qu'une solution complète soit obtenue. La plupart des métaheuristiques sont des algorithmes itératifs.

2.3.2.3 Algorithmes de type Greedy (glouton)

Pour un problème d'optimisation, un algorithme glouton est un algorithme qui cherche à construire une solution optimale pas à pas, sans jamais revenir sur ses décisions, en retenant à chaque étape la solution qui semble la meilleure localement. Tous les problèmes n'admettent pas une solution gloutonne.

Lorsqu'un problème peut se résoudre à l'aide d'un algorithme glouton, la solution produite par l'algorithme n'est pas forcément optimale. Pour assurer que l'algorithme glouton donne une solution optimale pour un problème, il faut montrer que ce problème satisfait les propriétés suivantes :

1. Propriété du choix glouton : il existe toujours une solution optimale commençant par un choix glouton, c'est-à-dire qu'un choix optimal local peut mener à une solution optimale globale.

2. Propriété de sous-structure optimale : trouver une solution optimale contenant le premier choix glouton se réduit à trouver une solution optimale pour un sous-problème de même nature.

Pour montrer que l'algorithme ne trouve pas toujours une solution optimale, il suffit de trouver un contre-exemple.

Les algorithmes gloutons sont des techniques basiques très répandues car ils sont simples à concevoir. Par ailleurs, ils ont en général une complexité réduite par rapport à des algorithmes itératifs. Cependant, dans la plupart des problèmes d'optimisation, la vue locale des algorithmes gloutons, diminue leurs performances par rapport aux algorithmes itératifs. Les questions de conception principale d'une méthode gloutonne sont les suivantes :

- **Définition de l'ensemble des éléments**. Pour un problème donné, on doit trouver une solution comme un ensemble d'éléments. Ainsi, les solutions partielles manipulées peuvent être considérées comme des sous-ensembles d'éléments.

- **Heuristique de la sélection d'éléments**. À chaque étape, une heuristique est utilisée pour sélectionner l'élément suivant faisant partie de la solution. En général, cette heuristique choisit le meilleur élément de la liste actuelle en termes de sa contribution en réduisant localement la fonction objectif. Ainsi, l'heuristique calcule pour chaque élément la valeur de la fonction objectif. L'optimalité locale ne garantit pas une optimalité globale. L'heuristique peut être statique ou dynamique. Dans le cas des heuristiques statiques, la fonction objectif ne varie pas d'un élément de la liste à l'autre, alors que dans le cas des heuristiques dynamiques, les bénéfices sont mis à jour à chaque étape.

2.4 Optimisation par métaheuristique

Dans cette section nous allons particulièrement détailler les méthodes d'optimisation métaheuristique et plus précisément l'algorithme génétique et les colonies de fourmis car elles constitueront les briques de bases des algorithmes que nous proposerons plus loin dans cette thèse. Nous commençons par présenter les principaux concepts communs aux méta-heuristiques ensuite nous présentons brièvement les méthodes basées sur une solution unique avant de détailler celles qui sont basées sur une population.

2.4.1 Optimisation par colonie de fourmis (ACO)

ACO pour Ant Colony Optimization est l'une des métaheuristiques inspirées de la nature ayant eu le plus de succès. Les fourmis artificielles s'inspirent des fourmis biologiques en se basant sur l'observation de leur comportement alimentaire. Pour aller du nid à une source de nourriture, les fourmis semblent trouver non seulement un chemin aléatoire, mais surtout un très «bon» chemin, en termes de distance, ou de façon équivalente, en termes de temps de déplacement. La résolution du problème est totalement inspirée de leur comportement naturel.

Ce succès des fourmis biologiques est entièrement expliqué par leur type de communication et par leur façon de décider où aller : tout en se déplaçant, les fourmis déposent une substance chimique appelée phéromone sur le terrain, et elles ont tendance à choisir des itinéraires marqués par des concentrations de phéromone forte.

Soit deux voies d'abord inexplorées séparant la fourmilière d'une source de nourriture, l'une des voies étant plus longues que l'autre. Au début de l'exploration les fourmis choisissent d'abord au hasard. Mais les fourmis qui ont choisi, la route la plus courte sont les premières à atteindre la nourriture et retrouver leur nid. Ainsi, la phéromone aurait tendance à s'accumuler plus rapidement sur la route la plus courte. De ce fait, les fourmis seraient tentées de suivre la route la plus courte car la plus chargée de phéromone.

Les problèmes combinatoires résolus par la métaheuristique ACO sont généralement codés par la construction d'un graphe $G = (V, A)$, un graphe complet dont les nœuds V sont **des composants de solutions,** et les arcs A sont les connexions entre les composants. Trouver une solution revient à construire un chemin réalisable dans G. L'algorithme ACO est essentiellement l'interaction de trois procédures [35] : "ConstructAntsSolutions", "UpdatePheromones" et "DeamonActions", représentée par l'algorithme 1.

Algorithme 1 : Ant Colony Optimization (ACO)

```
1 répéter
2     ConstructAntsSolutions;
3     UpdatePheromones;
4     DeamonActions;
5 jusqu'à condition d'arrêt non atteinte ;
```

ConstructAntsSolutions est le processus par lequel les fourmis artificielles construisent des chemins du graphe d'une façon incrémentielle et stochastique. Pour une fourmi donnée, la probabilité de p_{kl} pour aller d'un nœud k à un nœud successeur possible l est une fonction croissante de τ_{kl} et $\eta_{kl}(u)$, où τ_{kl} est la quantité de phéromone sur l'arc (k, l), et $\eta_{kl}(u)$ est la valeur heuristique de l'arc (k, l), qui devrait être une estimation raisonnable de la qualité de l'arc (k, l). La valeur heuristique peut dépendre de la marche partielle u.

EvaporatePheromone est le processus par lequel la phéromone diminue. La phéromone diminue pour chaque fourmi et chaque arc par une opération de mise à jour locale. Diminuer la phéromone sur les arcs sélectionnés est important, afin d'éviter une convergence trop rapide de l'algorithme pour des solutions sous-optimales, et de favoriser l'exploration de nouveaux domaines de l'espace de recherche. Diminuer la phéromone permet de simuler le phénomène naturel d'évaporation biologique.

DeamonActions sont des opérations centralisées qui utilisent des connaissances globales sur le processus de recherche de l'algorithme. Ici, le terme «centralisée» est utilisé par opposition au terme «localisée», et représente les opérations qui ne peuvent être effectuées par les fourmis uniques (car ils fonctionnent localement). Les DaemonActions peuvent inclure : l'évaluation et la comparaison de la valeur de la fonction objectif des solutions différentes, produites par les fourmis ; l'application d'une heuristique de recherche locale pour des solutions produites par les fourmis, afin d'intensifier la recherche à proximité de certaines solutions choisies ; le recueil de l'information globale qui peut être utilisée afin d'augmenter la concentration de phéromone sur certaines solutions ou sur certains composants de la solution, afin de biaiser la recherche et de l'intensifier autour des solutions avec les caractéristiques indiquées (cette mise à jour centralisée de la phéromone est aussi appelée mise à jour globale).

2.4.2 L'algorithme génétique

L'AG (Algorithme Génétique) est une méthode de recherche stochastique globale qui imite la métaphore de l'évolution biologique naturelle. Les AG fonctionnent sur une population de solutions potentielles appliquant le principe de la survie des plus aptes à produire des approximations de plus en plus proches d'une solution. À chaque génération, une nouvelle série d'approximations est créée par le processus de sélection des individus selon leur qualité dans le domaine du problème et leur capacité à se reproduire ensemble à l'aide des opérateurs empruntés au domaine de la génétique naturelle. Ce processus conduit à l'évolution des populations vers une sélection d'individus qui sont mieux adaptés à leur environnement que les individus dont ils sont issus, tout comme dans l'adaptation naturelle.
Les individus, ou les approximations actuelles, sont codés comme des chaînes : *les chromosomes*, composés d'un ensemble d'alphabet, de sorte que les génotypes (les valeurs des chromosomes) sont particulièrement bien mis en correspondance avec la variable de décision (phénotypique) de domaine. La représentation la plus couramment utilisée dans les GAs est l'alphabet binaire 0, 1, bien que d'autres représentations peuvent être utilisés, par exemple ternaire, entier, réels évalués, etc. Pour le cas d'un problème à deux variables, x_1 et x_2 peuvent être mappées sur la structure des chromosomes selon le schéma de la figure 2.9.

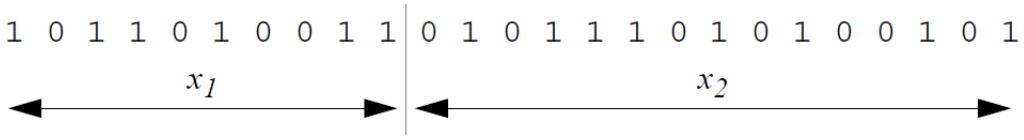

FIGURE 2.9 – Codage de deux variables par des chromosomes

Sur cette figure, x_1 est codé avec 10 bits et x_2 avec 15 bits, éventuellement reflétant le niveau de précision ou une gamme de variables de décision individuelle. Examiner la chaîne

de chromosomes seule ne fournit aucune information au sujet du problème que nous tentons de résoudre. C'est seulement avec le décodage du chromosome en ses valeurs phénotypiques que tout sens peut être appliqué à la représentation. Cependant, comme décrit ci-dessous, le processus de recherche fonctionne sur ce type de codage des variables de décision, plutôt que les variables de décision elles-mêmes, excepté pour le cas où les valeurs réelles sont utilisées.
Ayant décodé le chromosome en sa représentation dans le domaine des variables de décision, il est possible d'évaluer la performance (ou fitness) des membres individuels d'une population. Ceci est fait par une fonction objectif qui caractérise la performance d'un individu dans le domaine du problème. Dans le monde naturel, ce serait la capacité d'un individu à survivre dans son environnement actuel. Ainsi, la fonction objectif établit la base pour la sélection des paires d'individus qui seront accouplés pendant la reproduction.

Pendant la phase de reproduction, chaque individu est affecté d'une valeur de fitness tirée de sa mesure de performance brute donnée par la fonction objectif. Cette valeur est utilisée dans la sélection de partialité envers plusieurs individus. Les individus de bonne qualité (meilleures valeurs fitness par rapport à l'ensemble de la population), ont une forte probabilité d'être sélectionnés pour l'accouplement tandis que les individus moins aptes ont une probabilité proportionnellement plus faible d'être sélectionnés.

Une fois que les individus ont été affectés d'une valeur fitness, ils peuvent être choisis comme faisant partie de la population, avec une probabilité en fonction de leur qualité relative, et pourront être ensuite recombinés pour produire la prochaine génération. Les opérateurs génétiques manipulent les individus (les gènes) des chromosomes directement, en utilisant l'hypothèse que les codes de certains gènes individuels, en moyenne, produisent des meilleurs individus. L'opérateur de recombinaison est utilisé pour échanger des informations génétiques entre les paires, ou les groupes, des individus.

Considérons les deux parents codés en binaires suivant :

$$\begin{array}{lcccccccc} P_1 & = & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0, \\ P_2 & = & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0. \end{array}$$

Si une position i est sélectionnée de façon aléatoire, avec une probabilité uniforme dans l'intervalle $[1, l-1]$, alors deux nouvelles chaînes descendantes sont produites. La progéniture deux ci-dessous sont produites lorsque le point de croisement $i = 5$ est sélectionné :

$$\begin{array}{lcccccccc} O_1 & = & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0, \\ O_2 & = & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0. \end{array}$$

Cette opération de croisement n'est pas nécessairement effectuée sur toutes les chaînes dans la population. En effet, le croisement est appliqué selon une probabilité Px lorsque les paires sont choisies pour la reproduction. Un opérateur supplémentaire génétique, appelé mutation, est ensuite appliqué à des chromosomes avec de nouveau une probabilité fixé, Pm. La mutation provoque un changement dans la représentation génétique individuel en fonction de quelques règles probabilistes. Dans un codage binaire, la mutation entraîne la modification d'un seul bit, $0 \Rightarrow 1$ ou $1 \Rightarrow 0$. Ainsi, par exemple, la mutation du quatrième bit de O_1 conduit à la nouvelle chaîne :

$$\begin{array}{lcccccccc} O_{1m} & = & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

La mutation est généralement considérée comme un opérateur de fond qui assure que la probabilité de chercher un sous-espace particulier de l'espace de problème n'est jamais nul. Cela a pour effet d'avoir tendance à empêcher la possibilité d'une convergence vers un optimum local, plutôt que l'optimum global.
Après le croisement et la mutation, les chaînes individuelles sont ensuite, si nécessaire, décodées, la fonction objectif évaluée, une valeur de fitness attribuée à chaque individu et les individus sélectionnés pour l'accouplement en fonction de leur qualité. Le processus se poursuit à travers les générations subséquentes. De cette façon, la qualité moyenne des individus d'une population devrait augmenter. Les individus de bonne qualité sont conservés et les individus les moins bons meurent. L'AG se termine lorsque certains critères sont satisfaits, par exemple un certain nombre de générations, un écart à la moyenne dans la population, ou lorsque un point particulier de l'espace de recherche est rencontré.

Conclusion

La thématique principale de cette thèse est l'optimisation. Nous avons donc consacré un chapitre à cette thématique, pour bien introduire la théorie de la complexité. Nous avons ensuite fait un état de l'art des modèles les plus connus pour l'optimisation en détaillant plus particulièrement le modèle classique qui sera adopté dans cette thèse. Ce modèle classique définit un problème à résoudre comme étant une fonction à optimiser en respectant un ensemble de contraintes statiques. La solution à ce problème est une affectation de l'ensemble des variables, qui minimise la fonction objectif – dans le cas mono-objectif – ou un ensemble d'affectations non-dominées dans le cas multi-objectif.

Après avoir présenté la méthode classique utilisée pour modéliser un problème d'optimisation, nous avons retracé les méthodes utilisées pour résoudre ce type de problème. Nous avons aussi mis l'accent sur deux méthodes métaheuristiques : l'algorithme génétique et l'optimisation par colonies de fourmis (ACO).

Après cette introduction générale de notre thématique de la thèse, dans le chapitre suivant nous examinerons de façon plus spécifique le problème du plus court chemin dans un contexte théorique d'abord, et dans le contexte du transport multimodal ensuite.

Chapitre 3

Problème du plus court chemin en théorie des graphes

Ce chapitre présente un état de l'art sur le problème du plus court chemin, articulé autour des concepts classiques de complexité et d'optimisation multi-critéres, ainsi que des notions plus spécifiques à notre problème à savoir les concepts de modalité et de réseaux dépendants du temps.

3.1 Plus court chemin statique

Soit $G = (N, A)$ un graphe simple orienté où N est un ensemble de nœuds de cardinalité n et A un ensemble d'arêtes de cardinalité m. Soit $c : A \to R$ une fonction d'étiquetage des coûts associant un coût c_{ij} à chaque arête $(i, j) \in A$. Soit un nœud $i \in N$, notons $FS(i)$, la fonction de sortie du nœud i, correspondant à l'ensemble des arêtes sortantes, exprimée formellement $FS(i) = \{(i, j) \in A\}$, et soit $Adj[i]$ l'ensemble des sommets adjacents à i. De même pour un nœud $i \in N$, notons $FE(i)$, la fonction entrante du nœud i définie par $FE(i) = \{(j, i) \in A\}$.

Étant donné un nœud racine(source) $s \in N$, le problème de l'arbre des Chemins les plus court (Shortest Path Tree SPT) consiste à trouver un arbre T orienté tel que pour chaque nœud $i \in N$ connecté à s, le seul chemin de s à i dans T est un des plus courts chemins de s à i dans G. Les nœuds i et j sont connectés si et seulement il existe un chemin de i à j dans G. Si chaque nœud $i \in N$ est connecté à s alors T est un arbre couvrant notée T^*.

Une solution existe pour le problème SPT si et seulement si il n'y a pas de cycle orienté de coût négatif dans G [13].

3.1.1 Algorithme du plus court chemin de Bellman-Ford [13]

La plupart des algorithmes de résolution du problème SPT sont relativement basiques. L'idée est de construire un arbre couvrant T de racine s en explorant tous les chemins de l'arbre partant du nœud s. Plus concrètement, l'arbre T initial est de coût minimal «fictif». À chaque itération, l'algorithme met à jour l'arbre jusqu'à obtention d'un arbre T^* de coût minimum.

Soit $C = \{C_1, C_2, ..., C_n\}$ un ensemble d'étiquettes de coûts C_i représentant chacune le coût du chemin enraciné en s dans l'arbre T et ayant comme destination un nœud i. Le coût C_i correspond à chaque étape de l'algorithme au meilleur coût du chemin reliant les nœuds s et i.

Notons par $\pi[i] : N \to N$ la fonction prédécesseur, où $\pi[i] = \{j|(i, j) \in T\}$. Cette fonction est une description implicite de l'arbre T permettant de décrire de façon aisée le chemin de s à

i. c_{ij} est le coût de l'arête (i, j)

À chaque itération, via un processus connu sous le nom de numérisation (scanning), pour un nœud i sélectionné, la condition de Bellman 3.1.1 est vérifiée pour toutes les arêtes sortantes $FS(i)$ de i.

Définition 3.1.1

Condition de Bellman $C_j \leq C_i + c_{ij}, \forall (i, j) \in A$.

Le coût réduit d'un arc C est une fonction $\bar{c} : A \to R$ définie par :

Définition 3.1.2

Coût réduit $\bar{c}(i, j) = c_{ij} + C_i - C_j$

L'idée de Belleman est décrite par l'algorithme 2 ci-dessous.

Algorithme 2 : L'algorithme générique SPT

```
1  InitializesSPT(Q);
2  tant que Q non vide faire
3      i ← Q.Select();
4      pour chaque j ∈ Adj[i] faire
5          si C_i + c_ij < C_j alors
6              Update(i, j);
7              Q.Enqueue(j);
8          fin
9      fin
10 fin
```

La procédure de mise à jour de T la plus rudimentaire est décrite par l'algorithme 3 La fonction *Update* peut mettre à jour les étiquettes des nœuds , autres que le nœud courant.

Des approches plus efficaces ne propagent pas les paramètres des étiquettes sur les sous-arbres, mais maintiennent un registre Q d'étiquettes mises à jour, sur lequel les conditions de Bellman sont vérifiées.

Algorithme 3 : Fonction naïve de $Update(i, j)$

```
1  C_j ← C_i + c_ij;
2  π[j] ← i;
```

De nombreuses variantes de cet algorithme basique diffèrent par le traitement de Q et en particulier par la façon d'appliquer une règle de sélection des nœuds candidats dans Q et par la manière de mettre à jour l'arbre des plus courts chemins.

Le pseudo-code complet de l'algorithme de BF (Bellman-Ford) est présenté dans l'algorithme 4. La complexité dans le pire des cas de cet algorithme est en $O(mn)$.

3.1.2 L'algorithme de Dijkstra

L'un des algorithmes des plus courts chemins fondamentaux est sans doute dû à Dijkstra en 1959 [34]. Il est basé sur le même principe que l'algorithme de Bellman dont le but est de décrire tous les chemins les plus courts de s vers tous les nœuds du graphe en construisant un arbre de recouvrement SPT.

Algorithme 4 : L'algorithme de Bellmann-Ford

```
1  pour chaque i ∈ N faire
2      C_i = ∞;
3      π[i] = nill;
4  fin
5  C_s = 0;
6  pour k = 1n − 1 faire
7      tmp_C_j = C_j ∀j ∈ N;
8      change = faux;
9      pour chaque (i, j) ∈ A faire
10         d_ij = tmp_C_i + c_ij;
11         si d_ij < tmp_C_j alors
12             C_j = d_ij;
13             π[j] = i;
14             change = vrai;
15         fin
16     fin
17     si change = faux alors
18         Stop : C est optimale;
19     fin
20 fin
21 Stop : détection cycle négatif;
```

Dans la suite du manuscrit nous notons l'algorithme de Dijkstra DSP (Dijkstra's Shortest Path). DSP est une technique de fixation d'étiquettes séparant les nœuds en deux ensembles : un ensemble A, contenant tous les nœuds pour lesquels un chemin de longueur minimale de s à d a été trouvé. Ces nœuds sont considérés permanents. Un deuxième ensemble B correspond aux nœuds non permanents ou temporaires. Le pseudo-code de DSP est présenté par l'algorithme 5.

La complexité de l'algorithme DSP dépend de son implémentation. Les implémentations les plus simples mémorisent les sommets dans une liste et la complexité dans ce cas est en $O(n^2 + m) \approx O(n^2)$.

L'algorithme de Dijkstra peut être implémenté plus efficacement en stockant le graphe sous forme de listes d'adjacence et en utilisant une pile binaire ou pile de Fibonacci comme une file d'attente prioritaire pour extraire le nœud de coût minimal. Avec une pile binaire, l'algorithme est en $O((m + n)logn)$ et avec une pile de Fibonacci, il est en $O(m + nlogn) \approx O(nlogn)$.

3.2 Plus court chemin dynamique

La grande majorité de la littérature sur le problème du plus court chemin a porté sur les réseaux à topologie et à coûts fixes. Depuis quelques temps, en raison de l'intérêt porté aux réseaux de transport et aux réseaux de données, il y a eu un regain d'intérêt pour une une classe de problèmes de plus courts chemins dynamiques.

Algorithme 5 : L'algorithme de Dijkstra

```
1  pour chaque i ∈ N faire
2      C_i = ∞;
3      π[i] = nill;
4  fin
5  C_s = 0;
6  B = N;
7  tant que B ≠ ∅ faire
8      i = nœud de B avec minimum C_i;
9      B = B \ {i};
10     A = A ⋃ {i};
11     pour chaque j ∈ Adj[i] & j ∈ B faire
12         tmp = C_i + c_ij;
13         si tmp < C_j alors
14             C_j = tmp;
15             π[j] = i;
16         fin
17     fin
18 fin
```

3.2.1 Que signifie dynamique ?

Le terme «dynamique» a deux significations. La première, ne faisant pas l'objet de cette thèse, concerne le calcul du plus court chemin dans un graphe dont la topologie peut changer. L'objectif des travaux de recherche pour ce cas précis est de prendre en compte les changements de topologie progressivement tout en visant l'optimalité des algorithmes. Dans le cas où le poids d'une arête change pendant le calcul du plus court chemin, il est souhaitable qu'en moyenne le recalcul soit moins coûteux que le calcul du plus court chemin. Il faut donc des stratégies pour éviter de tout recalculer.

Dans cette thèse, nous nous intéressons à la seconde signification du terme «dynamique» dans la littérature, liée au temps. Le but est de modéliser la dépendance temporelle des arêtes du graphe et en particulier, d'étudier l'impact de cette dépendance sur la recherche de chemins optimaux. Les arêtes d'un graphe «dynamique» sont étiquetées par des propriétés qui déterminent «quand» et pendant «combien» de temps une arête peut être parcourue.

Notons que dans cette deuxième catégorie, le temps peut être modélisé comme une partie d'un ensemble de valeurs continues [80] ou de façon discrète [20, 23, 84]. La discussion ici ne considère que le modèle discret.

Dans le cas où la dépendance du temps est modélisée par discrétisation, les algorithmes qui en découlent et l'analyse de leurs performances se basent sur une version élargie des réseaux statiques dans lesquels, les nœuds sont reliés par des arêtes temporelles. Ces réseaux dits «espace-temps» ne correspondent pas à la meilleure approche à utiliser en tant que telle mais dégagent toutefois de nombreuses propriétés sur les graphes dynamiques qui permettront la conception de meilleurs algorithmes.

Les problèmes dits TDSP (Time Dependent Shortest Paths) ont de nombreuses applications [79, 81], y compris dans les domaines du transport et de la logistique [10]. Les tâches d'optimisation associées et leur conceptualisation remontent à 1966, quand Cooke & Halsey ont

proposé une façon de discrétiser le temps [31]. Dans ce problème, le temps de traversée d'une arête du réseau est supposé être prévisible ou déjà connu en fonction de l'heure de départ [31], avec plus ou moins de certitude [59]. Le problème TDSP est au moins NP-dur [6, 19, 31]. Mais cette notion de classification sera toujours liée à des caractéristiques du réseau et à la taille du problème à résoudre, comme le stipule bien Impagliazzo dans son article [57] en faisant remarquer que «un problème pas facile n'est pas forcement difficile». Par ailleurs, selon la façon dont le problème est défini, il peut même ne pas être dans la classe NP. C'est le cas de TDSP en considérant l'hypothèse FIFO, appelée aussi une propriété de non-dépassement [79] et les propriétés qui en découlent. Sous cette hypothèse, il existe des solutions efficaces en temps polynomial [2,19,31,78,81]. Néanmoins, les performances seront également influencées par la complexité des fonctions des dépendance du temps du réseau, à savoir la taille des grilles horaires.

Le problème TDSP peut être ramené à deux variantes fondamentales : l'arrivée au plus tôt (*Earliest Arrival : EA)* ou le départ le plus tard (*Latest Departure : LD*. Dans [31], les auteurs présentent différentes versions de ces deux variantes.

Soit $G = (N, A)$ un graphe orienté et pondéré. G est dit graphe dynamique si le poids d'un arc est une valeur qui dépend du temps. Par conséquent, le poids d'un arc $(i, j) \in A$ est décrit par la fonction $c_{ij}(t)$ (ou $w_{ij}(t)$) avec t la date de départ du nœud i. Pour ce type de graphe, le temps peut être considéré comme continu ou discret. Cependant, pour simplifier le problème, le temps est généralement supposé discret ou continu avec c une fonction constante par intervalles par rapport au temps. Dans ce dernier cas, le temps est divisé en périodes égales de longueur T. Durant chaque période T, la fonction de poids est supposée constante. Il faut noter que, contrairement aux réseaux de Pétri, le temps n'est pas réinitialisé au niveau de chaque nœud. La figure 3.1 présente un exemple d'un tel cas. Dans cet exemple, le poids de chaque arc est défini pour trois intervalles de temps. Supposons que $T = 5$ unités de temps. Le poids de l'arc (1,2) est égal à 2 pour $t \in [0; 5[$, il est égal à 5 pour $t \in [5; 10[$ et il est égal à 4 pour $t \in [10; 15[$.

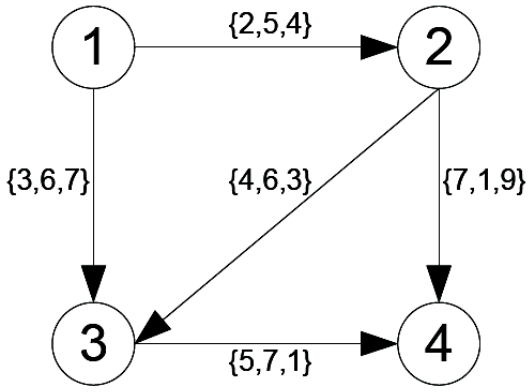

FIGURE 3.1 – Exemple d'un graphe dynamique (source [53])

La contrainte FIFO (*First In, First Out*) est particulièrement importante. Cette contrainte stipule qu'emprunter un arc (i, j) garantit le fait d'arriver en j au plus tôt [47].

Les graphes FIFO sont une sous-classe des graphes dynamiques déterministes. Un graphe dynamique déterministe est dit graphe FIFO si tous ses arcs sont des arcs FIFO. Un arc $(i, j) \in A$ est appelé arc FIFO si sa fonction de poids est définie de telle manière qu'à partir du nœud i pour le nœud j, $t' \geq t'$ implique nécessairement d'arriver à j plus tard que $t + c_{ij}(t)$.

Définition 3.2.1

Arc FIFO *: Soit $G = (N, A)$ un graphe déterministe. Un arc $(i, j) \in A$ est dit arc FIFO s'il*

vérifie la propriété suivante :
$\forall t, t^{'} \geq 0, t \leq t^{'} \Rightarrow t + c_{ij}(t) \leq t^{'} + c_{ij}(t^{'})$

3.2.2 Problème EA avec intervalle de temps

Ce problème, connu dans la littérature sous le nom "allFP", correspond au problème de tous les plus courts chemins avec intervalle de temps. Étant donné un intervalle I de temps avec des temps de départ et arrivée définis par l'utilisateur, un nœud source s et un nœud destination d, «allFP» cherche un ensemble de tous les chemins les plus courts de s à d : un pour chaque sous-intervalle de I.

3.2.2.1 Algorithme à temps discret

Un algorithme efficace à temps-discret pour les réseaux dynamiques a été présenté par Chabini [23]. Il a proposé un algorithme efficace avec discrétisation du temps dans un réseau dynamique. Chabini définit les problèmes de plus court chemin comme dynamiques ou dépendants du temps (interchangeable), et distingue différentes version du problème du plus court chemin dynamique en fonction de plusieurs caractéristiques, dont celle du réseau FIFO qui permet un temps de calcul polynomial. Beaucoup de réseaux de transport présentent un comportement FIFO, les réseaux de transport terrestres peuvent être généralement considérés comme étant de cette classe.

Limité à une représentation en temps discret, l'algorithme de Chabini [23] trouve, approximativement, un voyage avec un plus petit temps total de trajet (*least total travel time* (LTT)) en faisant exploser uniformément le graphe en k nœuds temporels, permettant au problème TDSP d'être résolu comme étant un problème de plus court chemin à source statique dans ce dernier. Cette approche présente deux inconvénients :

- La méthode LTT approchée diffère de la méthode LTT optimale, appelée LTT erreur (manière cumulative sur les sentiers), par sa sensibilité au paramètre k avec lequel la dimension du temps est discrétisée.
- L'augmentation de la valeur de k peut conduire à un résultat plus proche de l'optimal, le graphe explosé est k fois plus grand que l'original.

3.2.2.2 Algorithme basé sur Bellman-Ford

Orda & Rom (1990) [79] traitent le problème du plus court chemin dépendant du temps sans la restriction d'une représentation en temps discret, ni à celle des domaines de nombres entiers positifs. Leur algorithme (6) est une généralisation de l'algorithme de Bellman-Ford [13] à un graphe dynamique $G(N, A)$.

Dans cet algorithme, $g_i(t)$ est la fonction qui renvoie le temps d'arrivée le plus tôt au nœud i depuis le nœud source s, tandis que $h_{i,j}(t)$ renvoie le temps d'arrivée le plus tôt à v_j à partir de v_s et en prenant l'arc (i, j), pour $t \in T$. Ces deux fonctions sont mises à jour jusqu'à convergence vers des valeurs correctes. Partant du calcul du meilleur temps de départ t^*, le chemin p^* optimal peut être construit sur la base des fonctions $g_i(t)$ et $h_{i,j}(t)$.

L'algorithme détermine les chemins tout en affinant les fonctions $g_i(t)$ du temps d'arrivée pour l'intervalle T. Bien que cet algorithme modélise le temps par une fonction continue et que les coûts sont des entiers positifs, la complexité temporelle élevée en $(O(nm\alpha(T)))$, où $\alpha(t)$ est le temps nécessaire à une opération de fonction dans l'intervalle T, $n = |N|$ et $m = |A|$, le rend inapplicable aux graphes dynamiques de grande taille.

Algorithme 6 : Algorithme de Orda & Rom [79]

pour tous les $\underline{i \in N}$ **faire** $g_i(t) \leftarrow \infty \forall t \in T$;
pour tous les $\underline{(i,j) \in A}$ **faire** $h_{ij}(t) \leftarrow \infty \forall t \in T$;
$g_s(t) \leftarrow t \forall t \in T$;
répéter
    pour tous les $\underline{(i,j) \in A}$ **faire** $h_{ij}(t) \leftarrow g_k(t) + c_{ij}(g_{ij}(t))$;
    pour tous les $\underline{i \in N}$ **faire** $g_i(t) \leftarrow min_{j \in Adj[i]}\{h_{ij}(t)\}$;
jusqu'à $\underline{\text{fonctions } g \text{ restent inchangées}}$;
retour $(t^* \leftarrow argmin_{t \in T}\{g_d(t) - t\}, p^*)$

3.2.2.3 Algorithme A^*

Kanoulas et al. [38] ont proposé une amélioration de l'algorithme A^* pour traiter les problèmes TDSP. Mais encore une fois la sélection du chemin et celle du temps sont couplées, même de façon plus étroite que dans l'algorithme d'Orda & Rom (6). Cela conduit dans le pire des cas où tous les chemins sont énumérés, à une complexité temps/espace exponentielle par rapport à la taille du graphe. Du point de vue pratique, les performances de l'approche de Kanoulas et al. [98] se résument comme suit :

- Cet algorithme n'est efficace que dans le cas où l'estimation permet une réduction de l'espace de recherche et que s et d sont proches dans le graphe ;
- L'estimation pour réduire l'espace de recherche étant difficile dans les graphes généraux, cet algorithme ne permet par conséquent pas le traitement de graphes dynamiques de grandes taille où le nœud s a toutes les chances d'être éloigné du nœud d.

Compte tenu du fait que les heuristiques A^* fournissent généralement de meilleurs résultats dans le cas du plus court chemin statique, beaucoup de tentatives pour adapter cet algorithme aux réseaux dynamique dépendants du temps ont été effectuées (Huan et al. [56], Delling & Nannicini [76], Zhao & Ohshima [108]).

3.2.2.4 Algorithme de plus court chemin à deux-étapes

Dans Ding et al [28], les auteurs ont également utilisé les notions de raffinement du temps et de sélection du chemin pour résoudre le problème du plus court chemin dépendant du temps, en particulier lorsqu'un intervalle de temps de départ est donné. Il s'ensuit que la principale caractéristique de leur algorithme est le découplage du raffinement du temps et de la sélection du chemin, donnant lieu à un algorithme en deux-étapes (nommé two-step least travel-time, par les auteurs). Le pseudo-code de leur approche est décrit par l'algorithme 7 qui traite généralement le problème TDSP sous l'hypothèse FIFO. Les auteurs montrent également qu'il est possible de l'appliquer pour les réseaux non-FIFO. Cet algorithme prend en entrée un graphe en fonction du temps G et une requête (s, d, T), un nœud de départ s, un nœud destination d, et un intervalle $T = [t_s, t_d]$ de temps de départ en entrée. La sortie de l'algorithme est un chemin optimal $p^* = v_s - v_d$ correspondant au temps de départ optimal t^*. Cet algorithme fait appel à des fonctions «délai-arête» (edge-delay) linéaires et continues par morceaux. Les problèmes de discontinuités dans les fonctions font que l'algorithme de Dijkstra ne s'applique pas dans ce cas. Pour traiter les fonctions présentant des discontinuités, nous nous référons aux travaux de Dell'Amico & Pretolani [31].

La première étape de l'algorithme de Ding et al. [28] détermine les fonctions de temps d'arrivée minimal $g_i(t)$ pour chaque nœud i du graphe dans l'intervalle du temps T donné (algorithme 9).

Algorithme 7 : Algorithme à deux étapes

$g_i(t) \leftarrow timeRefinement(G, s, d, T)$;
si $\neg g_d(t) = \infty$ pour l'ensemble de $[t_s, t_d]$ **alors**
    $t^* \leftarrow argmin_{t \in T}\{g_s(t) - t\}$;
    $p^* \leftarrow pathSelection(G, g_i(t), s, d, t^*)$;
sinon
    retour $\emptyset$;
fin

Le principal avantage de cet algorithme réside dans sa capacité à identifier un sous-intervalle pour lequel la fonction temps-arrivée est déjà bien définie et que, par conséquent, n'a pas besoin d'être calculée de nouveau.

Algorithme 8 : Fonction timeRefinement de Ding et al. (2008)

$g_s(t) \leftarrow t$ for $t \in T$;
pour chaque $i \neq s$ **faire**
    $g_i(t) \leftarrow \infty$ for $t \in T; \tau_i \leftarrow t_s$;
fin
Soit Q une file de priorité contenant initialement les paires $(\tau_i, g_i(t))$;
pour tous les nœuds $i, j \in N$, ordonnés par $g_i(\tau_i)$ dans l'ordre croissant;
tant que $|Q| \geq 2$ **faire**
    $(\tau_i, g_i(t)) \leftarrow dequeue(Q)$;
    $(\tau_k, g_k(t)) \leftarrow head(Q)$;
    $\Delta \leftarrow min\{w_{f,i}(g_k(\tau_k))|(f, i) \in A\}$;
    $\tau_i^{'} \leftarrow max\{t|g_i(t) \geq g_k(\tau_k) + \Delta\}$;
    pour chaque $(i, j) \in A$ **faire**
        $g_j^{'}(t) \leftarrow g_i(t) + w_{i,j}(g_i(\tau_i))$ for $t \in [\tau_i, \tau_i^{'}]$;
        $g_j(t) \leftarrow min\{g_j(t), g_j^{'}(t)\}$ for $t \in [\tau_i, \tau_i^{'}]$;
        $update(Q, (\tau_j, g_j(t)))$;
    fin
    $\tau_i \leftarrow \tau_i^{'}$;
    si $\tau_i \geq t_d$ **alors**
        return $\{g_i(t)|i \in N\}$;
    sinon
        $enqueue(Q, (\tau_i, g_i(t)))$;
    fin
fin
retour $\{g_i(t)|i \in\}$;

La seconde et dernière étape de l'algorithme consiste à effectuer une sélection du plus court chemin selon la procédure décrite par l'algorithme 9.

Algorithme 9 : Fonction pathSelection

```
1  j ← d;
2  p* ← ∅;
3  tant que j ≠ s faire
4      pour chaque (i, j) ∈ A faire
5          si (g_i(t*) + w_ij(g_j(t*)) = g_j(t*)) alors
6              j ← i;
7          fin
8          p* ← (i, j).p*;
9      fin
10 fin
11 retour p*;
```

3.2.3 Problème EA avec un temps du départ fixe

Lorsque l'objectif à minimiser est le temps et que le temps de départ est fixé par l'utilisateur, il est possible d'utiliser l'algorithme de Dijkstra avec une modification mineure (voir algorithme 10). Cette approche a été proposée initialement dans Dreyfus [36]. Les conditions exactes d'optimalité de cette approche ont été présentées dans [45].

Algorithme 10 : Algorithme de Dijkstra dépendent du temps

```
1  pour chaque i ∈ N faire
2      C_i = ∞;
3      π[i] = nill;
4  fin
5  C_s = t_0;
6  B = N;
7  tant que B ≠ ∅ faire
8      i = nœud de B avec minimum C_i;
9      B = B \ {i};
10     pour chaque j ∈ Adj[i]&j ∈ B faire
11         si C_j > F(i, j, C_i) alors
12             C_j = F(i, j, C_i);
13             π[j] = i;
14         fin
15     fin
16 fin
```

Des améliorations de cet algorithme existent, à savoir l'utilisation de bornes inférieures ou la recherche bidirectionnelle. Mais l'utilisation de ces méthodes usuelles dans le cas de graphes dynamiques n'est pas évidente. Par exemple, la recherche bidirectionnelle nécessiterait de connaître à l'avance l'instant d'arrivée à la destination. Cependant, d'importants efforts ont été faits dans ce domaine et les techniques haute performance présentées dans le cadre de coûts statiques ont été adaptées avec succès [12, 32, 76].

3.2.4 Chemin de moindre coût

Lorsque la fonction de coût à optimiser n'est pas le temps mais qu'elle dépend elle même du temps, il n'est plus possible d'utiliser les méthodes classiques telles que Dijkstra ou Bellman-Ford. Cela est dû au fait que choisir une arête minimisant le coût à un certain temps risque de mener à une situation de cul de sac où il n'est plus possible d'avancer dans le graphe. Ce problème ne peut être résolu, sans avoir enregistré au préalable tous les choix effectués a chaque étape, permettant ainsi un retour arrière. Cette technique qui risque de conduire au parcours de l'arbre complet de recherche est inacceptable.
Nous présentons dans cette section les rares travaux de la littérature qui traitent le problème du plus court chemin à moindre coût dans un graphe dynamique dépendent du temps. Cette étude reprend une synthèse déjà effectuée dans la thèse récente de T. Gräbener [46].

Il est peu probable qu'il existe un algorithme en temps polynomial. Ce problème est en effet NP-difficile puisqu'il permet de résoudre le problème du plus court chemin avec des contraintes de ressources. Le premier article qui traite de ce problème est dû à Orda & Rom [79], où les auteurs montrent que le chemin de moindre coût peut être de longueur infinie. Un algorithme est proposé pour résoudre ce problème avec des conditions particulières sur les fonctions de coût pour que le chemin optimal soit de longueur finie. L'exécution de l'algorithme s'arrêtera , mais rien ne garantit qu'il se termine en temps polynomial. La différence entre les deux types d'objectifs est développée dans Ahuja [2] où les auteurs étudient le chemin le plus rapide et celui de moindre coût dans un réseau routier prenant en compte les feux en montrant la différence de complexité.

Dans certains cas, le problème de moindre coût a toutefois une complexité polynomiale. Il s'agit des graphes dits "cost-consistent". Cette contrainte supplémentaire stipule que quitter un nœud plus tôt ne coûte pas plus cher que le quitter plus tard. Cette notion est présentée dans [85]. Les auteurs présentent également l'adaptation de leur algorithme Chrono-SPT pour calculer le chemin de moindre coût sur de tels graphes.

3.3 Plus court chemin multi-critères

Le problème d'optimisation mono-objectif tout comme le problème de plus court chemin dépendant du temps vu jusque là, est souvent simple. Dans le problème du plus court chemin, un seul objectif correspond à la recherche du chemin le plus rapide, le plus court ou le moins cher. DAns le cas multi-objectifs, les arcs du graphe sont étiquetés par deux fonctions de pondération (ou plus), telles que trouver le chemin le moins cher et le plus rapide. Résoudre un problème d'optimisation multi-objectif implique souvent la recherche de multiples solutions *efficaces*, appelé l'ensemble optimal de Pareto, avec la propriété que l'un des critères sur lesquels une solution a été trouvée peut être amélioré, mais seulement au détriment des autres critères.

3.3.1 Pareto Optimalité

3.3.1.1 Définitions

Formellement, le problème d'optimisation multi-objectif est défini comme étant le problème de la recherche d'un vecteur de variables de décisions $\bar{x}^* = [x_1^*, x_2^*, ..., x_n^*]^T$ qui optimise la fonction vectorielle $F(\bar{x})$ dont les éléments sont les fonctions objectif.

$$F(\bar{x}) = \begin{cases} f_1(x) \\ f_2(x) \\ . \\ . \\ . \\ f_n(x) \end{cases}$$

où $n \geq 2$, de l'espace S qui représente une série de variables de décision. L'objectif est de trouver l'ensemble des solutions qui satisfont l'ensemble des l'ensemble des fonctions objectives. Pour que $\bar{x}^* \in S$ soit une solution *Pareto optimale*, deux conditions doivent être vérifiées :
$\forall \bar{x} \in S$:

$$\forall i \in [1..n], f_i(\bar{x}^*) \leq f_i(\bar{x}) \exists i \in [1..n], f_i(\bar{x}_i^*) < f_i(\bar{x}_i)$$

$\bar{x}^*$ est un Pareto optimum (ou solution non dominée) s'il n'existe aucun autre vecteur $\bar{x}$ qui réduirait un critère, sans augmenter simultanément la valeur d'un autre. L'ensemble des points non dominés dans l'espace S est connu comme l'ensemble Pareto optimal et il est rare que cet ensemble ne contienne qu'une seule solution.
Le problème d'optimisation multi-critères, est en général, NP-difficile. La taille de l'ensemble des solutions pareto-optimales varie exponentiellement avec la taille du problème, même quand le nombre de critères est fixé [52].

3.3.1.2 Indicateurs de qualité

En général, les outils d'indicateurs de qualité peuvent être utilisés de deux façons : soit comme des programmes indépendants, ou comme méthodes invoquées dans d'autres programmes. Nous décrivons ensuite comment calculer différents indicateurs.

Generational Distance Cet indicateur de qualité a été introduit par Van Veldhuizen et Lamont [105] pour mesurer la distance qui sépare les éléments de l'ensemble des vecteurs non-dominés trouvés, de ceux de l'ensemble pareto. Cet indicateur est défini comme suit :

Définition 3.3.1

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}$$

Où n est le nombre de solutions trouvées et d_i est la distance euclidienne (mesurée dans l'espace objectif) entre chacune de ces solutions et l'élément le plus proche de l'ensemble pareto optimal. Une valeur de $GD = 0$ indique que tous les éléments générés sont dans le front pareto. Afin d'obtenir des résultats fiables, les ensembles non dominés sont normalisés avant de calculer cette distance.

Spread L'indicateur de marge ou spread est un indicateur de diversité qui mesure le degré de propagation obtenu entre les solutions [60]. Cette métrique est définie comme suit :

Définition 3.3.2

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}}$$

où d_i est la distance euclidienne entre deux solutions successives, $\bar{d}$ est la moyenne de ces distances, et d_f et d_l sont les distances euclidiennes entre les solutions extrêmes du front pareto exacte de l'espace objectif. Cette métrique prend une valeur nulle pour une distribution idéale. Avant d'appliquer cette métrique, les valeurs de la fonction objectif sont normalisées.

Hypervolume Cet indicateur mesure la qualité d'une solution dans l'espace objectif, en mesurant la distance entre le front pareto ou l'ensemble des solutions non dominées Q (région fermée dans la ligne discontinue de la figure 3.2, $Q = \{A, B, C\}$) et un point de référence W de l'espace. Mathématiquement, pour chaque solution $i \in Q$, un hypercube v_i est construit avec les point de référence W et la solution i considérées comme les extrémités de la diagonale de l'hypercube. Le point de référence peut simplement être trouvé par la construction d'un vecteur des pires valeurs de la fonction objectif. Par la suite, une union de tous les hypercubes est calculée et leurs hypervolume (HV) est donc :

Définition 3.3.3

$$HV = volume\Big(\cup_{i=1}^{|Q|} v_i \Big)$$

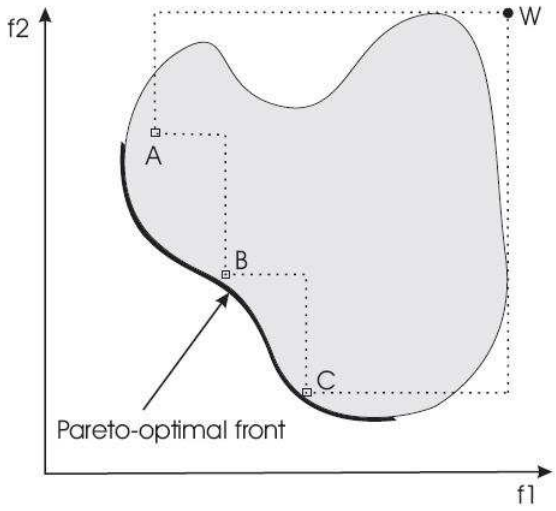

FIGURE 3.2 – Hypervolume

3.3.2 Approches exactes

Pour résoudre le problème de plus court chemin multi-critères, l'approche la plus commune est de généraliser les algorithmes mono-critère. Au lieu d'une étiquette scalaire, à chaque nœud i est associé un vecteur d'étiquettes de M dimensions correspondant aux coûts des chemins non-dominés, allant du nœud source s à tout nœud i. Cette méthode énumérative ou de génération peut être décomposée en deux classes principales : les algorithmes d'étiquetage(*labelling algrorithms* et les algorithmes de classement(*ranking algorithms RK*). La première classe peut être à son tour divisée en *algorithmes à fixation d'étiquettes LS*, dans lesquels une étiquette est fixée

en permanence à chaque itération, et *les algorithmes à correction d'étiquette LC*, dans lesquels toutes les étiquettes deviennent permanentes seulement à la dernière itération. Une étude théorique détaillée des propriétés des algorithmes à fixation d'étiquettes ainsi que des algorithmes à correction d'étiquettes peut être trouvée dans [69].
Comme pour les techniques de classement (RK), la première approche a été proposée par Clímaco et Martins [25] pour le problème du plus court chemin à double objectifs. Une séquence des k plus courts chemins relativement au premier objectif, jusqu'à ce que le chemin de valeur minimale , relativement au deuxième objectif soit obtenu. Une extension de cet algorithme au contexte multi-objectif a été proposée par Azevedo et Martins [8].
Il y a aussi une différence importante entre les algorithmes d'étiquetage mono-objectif et leur version multi-objectif. Lorsque le but est de trouver simplement le chemin optimal de s à t, un algorithme mono-objectif (par exemple Dijkstra) peut être arrêté dès que l'étiquette du nœud t est élue permanente. Il n'est pas possible de procéder ainsi dans le cas d'objectifs multiples. Il est possible que plusieurs chemins (dont le nombre est inconnu) $s-t$ efficaces existent. Par conséquent, l'algorithme doit être exécuté jusqu'à ce que tous les chemins efficaces soient trouvés au niveau de tous les nœuds.
D'autres approches intéressantes existent pour déterminer l'ensemble pareto, comme la méthode complète en deux phases (2P) [3], la méthode de partitionnement parallèle (PP) [65] et les divers paradigmes de programmation (par exemple, la programmation dynamique, la programmation en nombre entiers, la programmation linéaire, la programmation logique...).
La méthode en deux phases(2P) a montré de bonnes performances [3, 71, 107]. Comme son nom l'indique, cette méthode divise le problème en deux phases. Dans la première phase, elle calcule des solutions efficaces qui définissent les points extrêmes de l'ensemble des vecteurs objectifs réalisables. Dans la deuxième phase, une approche énumérative (par exemple un algorithme d'étiquetage ou de classement) est utilisée pour calculer les solutions restantes. Cet algorithme est efficace car la méthode énumérative est limitée à de petites zones de l'espace de recherche. Raith et Ehrgott [3] ont montré que dans les réseaux routiers réels la méthode en deux phases est plus performante que la technique LS ou LC. Par ailleurs, pour la deuxième phase, les approches d'étiquetage semblent être préférables à celles de classement, même si le résultat dépend toujours des instances à résoudre.

3.3.2.1 Algorithme à fixation d'étiquette : Martins

L'algorithme de Dijkstra est sans doute l'algorithme à fixation d'étiquette le plus célèbre. Toutefois, il ne traite que des problèmes mono-objectif avec contraintes additives (coût positif).

L'algorithme proposé par Hansen [50] est une première extension de la DSP pour deux coûts additifs. Ce travail a encore été généralisé par Martins [67] pour prendre compte un nombre quelconque d'objectifs. De même que PSD, l'algorithme de Martins vise à calculer l'ensemble des chemins efficaces d'un sommet à tous les autres sommets. Il utilise toujours deux ensembles différents d'étiquettes pour étiqueter les chemins : les étiquettes permanentes correspondent aux sous-chemins non-dominés, et les étiquettes temporaires sont associées à des chemins qui pourraient être dominés au cours des prochaines itérations. Notons que l'opérateur mono-objectif classique «min» est désormais remplacé par un test de dominance. À chaque itération, l'algorithme sélectionne l'étiquette la plus petite au sens de l'ordre lexicographique, parmi l'ensemble des étiquettes temporaires. Le nœud de plus petite étiquette devient permanent et l'information est propagée à tous les nœuds successeurs temporaires. Quand il n'y a plus d'étiquette temporaire, l'algorithme s'arrête. Chaque étiquette permanente correspond à un chemin unique optimal.
La preuve de validité de l'algorithme est donnée par Ehrgott et Gandibleux dans [37]. Les tra-

vaux de Martins ont récemment été étendus par Gandibleux et al. [40] pour traiter à la fois les métriques additives et concaves.
Une approche différente a été proposée par Martins et Santos [69], sous la forme d'un algorithme d'étiquetage multi-objectif général pour les graphes orientés, basé sur le principe d'optimalité. Cet algorithme général est le point de départ pour deux implémentations spécifiques, basées soit sur la fixation d'étiquette ou sur la correction d'étiquette. La correction d'étiquette est plus facile à mettre en œuvre, car elle ne nécessite pas de structures de données complexes pour l'ordre lexicographique (comme pour la version de fixation d'étiquette). Mais puisque les étiquettes sont continuellement mises à jour jusqu'à la dernière itération, il en résulte une plus grande complexité temporelle et spatiale [33].
L'approche de fixation d'étiquette est décrite par l'algorithme 11. Le principe de base est de parcourir un graphe pour construire un arbre de recherche ST. La fonction $h()$ crée une association entre un nœud dans le graphe et sa position dans ST, par exemple $h(1) = s$. La fonction $path(x)$ renvoie le chemin à partir du nœud s jusqu'au nœud $h(x)$, représentée par la branche de 1 à x dans l'arbre ST. X est l'ensemble des nœuds feuilles de ST qui n'ont pas encore été analysés. Z_v est l'ensemble des chemins non-dominés de s à un nœud générique v. À la fin de l'algorithme, chaque branche de l'arbre ST final définit un chemin non-dominé de s à t. L'opérateur $\oplus$ indique une concaténation des deux sous-chemins.

Algorithme 11 : Algorithme de Martins

1 $cnt = 1$;
2 $h(cnt) = s$;
3 $Z_s = path(1)$;
4 $X = \{cnt\}$;
5 **tant que** $X \neq \emptyset$ **faire**
6 x = plus petit élément lexicographiquement de X;
7 $X = X \setminus \{x\}$;
8 $i = h(x)$;
9 **pour chaque** $(i, j) \in A$ **faire**
10 $path(cnt) = path(x) \oplus (i, j)$;
11 **si** $path(cnt)$ n'est pas dominé dans Z_j **alors**
12 $cnt = cnt + 1$;
13 $add(x, cnt)$ dans l'arbre ST;
14 $h(cnt) = j$;
15 $X = X \bigcup \{cnt\}$ (lexicographiquement);
16 $Z_j = Z_j \bigcup \{path(cnt)\}$;
17 supprimer tous les chemins dominés de Z_j;
18 enlever les nœuds de l'arbre correspondant de X;
19 **fin**
20 **fin**
21 **fin**
22 Z_t détient l'ensemble des chemins non-dominés de s à t;

La complexité de l'algorithme de Martins peut être évaluée comme suit. La boucle «tant que» dépend de la taille de X, et plus précisément, de la valeur maximale atteinte par le compteur cnt (lignes 12 et 15). Nous pouvons observer que CNT est incrémenté à chaque fois qu'un chemin

non-dominé est ajouté. Ainsi, on peut supposer que sa valeur maximale est la taille de l'ensemble non-dominé complet. Soit ω indiquant ce nombre. L'extraction de x de X est immédiate si X est trié. La boucle «Pour» est proportionnelle au nombre d'arêtes sortantes d'un nœud. Soit D le degré du réseau moyen, D peut être considéré comme le nombre maximal d'arêtes sortantes d'un nœud. Le test de non-dominance à la ligne 11 nécessite M comparaisons, pour chaque chemin stocké dans Z_j, dont la taille maximale est, là encore, ω. L'insertion d'un nouvel élément dans X tout en gardant X lexicographiquement trié nécessite $Mlog_2(\omega)$ comparaisons. Enfin, les transferts (lignes 17 et 18) peuvent être négligés, car ils peuvent être efficacement réalisés en un seul passage tout en vérifiant la non-domination du $path(cnt)$.
En résumé, la complexité de l'algorithme 11 peut être écrite comme :

$$O(\omega.D.M\omega.Mlog_2(\omega)) = O\left(DM^2\omega^2log(w)\right)$$

3.3.2.2 Algorithmes à correction d'étiquette (LC)

De même que la fixation d'étiquette, l'approche à correction d'étiquette a été largement étudiée aussi. L'idée originale est due à Vincke [106], qui a conçu la première méthode LC pour obtenir toutes les solutions pareto optimales pour le problème du chemin bi-critère. Plus tard, l'algorithme proposé par Brumbaugh-Smith et Shier dans [22] a longtemps été le plus cité, et ce n'est que récemment qu'il a été amélioré par d'autres chercheurs (par exemple Skriver et Andersen [97] et Sastry [93]). Enfin, une étude intéressante sur l'impact de la sélection d'étiquettes par opposition à la sélection des nœuds avec les stratégies de LC est réalisée par Guerriero et Musmanno [48]. La conclusion, confirmée par d'autres ouvrages (voir [82]), est que la politique de sélection d'un nœud nécessite un plus grand effort de mise en œuvre et est en général moins efficace que la sélection d'étiquettes.
Actuellement, l'algorithme de référence LC est due à Skriver et Andersen, qui a conçu un PCA (Path Computation Algorithm) pour résoudre le problème de plus court chemin bi-critère (M = 2) dans un graphe orienté [97]. L'idée de base émane de l'algorithme de Brumbaugh-Smith et Shier, qui a été amélioré en éliminant les arêtes dominées le plus tôt possible, afin que la procédure de recherche subséquente soit simplifiée.

3.3.2.3 Algorithme de classement (RK)

Les méthodes de classement, à l'instar d'autres approches, ont d'abord été proposé pour des problèmes de bi-critères, pour lesquels ils représentent toujours une approche très efficace. Dans l'algorithme conçu par Clímaco et Martins [23], les chemins de s à t sont énumérés en basant sur la première fonction objectif jusqu'à ce que le chemin le plus court dans le deuxième critère est obtenu.
L'avantage de cette procédure est qu'elle n'a pas à calculer les plus courts chemins de s à tous les autres sommets dans le réseau. Mais d'autre part, il s'est avéré ne pas être compétitif, car le critère de la dominance est vérifié uniquement au niveau du nœud destination t. Bien que les mêmes auteurs aient proposé une extension de plus de deux critères, Ehrgott et Gandibleux prétendent que la généralisation de cette approche à plus de trois objectifs n'est pas possible sans la connaissance du point nadir qui est difficile à obtenir quand $M > 2$ [37]. Une extension de l'algorithme de Clímaco et Martins au contexte multi-objectifs a été plus tard conçu sur des bases différentes par Azevedo et al. [8]. En outre, ils ont aussi amélioré son efficacité par le biais du principe d'optimalité. Dans [24], Clímaco et al. applique un algorithme de classement à un problème spécifique, c'est à dire le calcul des chemins non-dominés pour les flux vidéo dans un

réseau ATM. L'aspect notable est que, au contraire de ses ancêtres, il représente également des contraintes pratiques telles que les retards, la bande passante, etc... Le résultat de certains tests de calcul sur les réseaux aléatoires avec au maximum quelques milliers de nœuds ont montré que lorsque les contraintes sont strictes, l'algorithme atteint rapidement son objectif, à l'inverse, lorsque les contraintes sont lâches, il pourrait même ne pas terminer en raison de la dimension de l'espace de recherche.

Récemment, une technique de marquage afin d'obtenir les chemins non-dominés classés selon une relation d'ordre total sur R^M a été proposée par Martins et al. [68] et peu après améliorée par Paixao et Santos [83]. En pratique, l'algorithme peut être utilisé pour produire seulement les k meilleurs «chemins», avec k variant de 1 au nombre total de chemins efficace.

3.3.2.4 Méthode deux phases 2P

La méthode de deux phases divise le calcul de l'ensemble pareto en deux étapes distinctes. Dans la première phase, les solutions efficaces *extrêmes* sont identifiées. Ces solutions peuvent être facilement calculées. Dans la deuxième phase, le reste des solutions non-dominées sont calculées avec une approche énumérative, et les informations obtenues dans la première phase sont utilisées pour restreindre l'espace de recherche pour l'algorithme énumératif, de sorte que son exécution devienne plus efficace. Parfois, une phase d'initialisation est également nécessaire pour trouver une ou deux solutions initiales permettant de lancer la première phase.
Historiquement, la première méthode 2P pour le problème de plus court chemin bi-objectif a été conçue par Mote et al. [71] qui ont utilisé un algorithme de type simplex pour trouver un sous-ensemble Pareto-optimal, et une méthode de correction des étiquettes pour trouver tous les chemins optimaux restants. Une formulation de la classe plus générale des problèmes d'optimisation combinatoire bi-objectifs peut être trouvée dans l'ouvrage de Ulungu et Teghem [107], où la deuxième phase est réalisée à l'aide de l' algorithme de séparation (Branch and bound).
Dans un article très récent, Raith Ehrgott analysent les différentes méthodes pour les deux phases ainsi que pour l'initialisation [3]. Suite à des expériences menées sur des réseaux générés de façon aléatoire, les réseaux de type grille et les réseaux routiers, ils concluent que l'algorithme de Dijkstra est la meilleure approche pour l'initialisation, «La fixation d'étiquette dichotomie» est la meilleure pour la première phase (avec quelques inconvénients pour les réseaux de petite grille), et les algorithmes LC et LS sont tous deux plus adaptés pour la deuxième phase.
L'approche dichotomique, choisie pour la première phase, est une méthode qui génère par itération les poids et résout le problème de la somme pondérée mono-objectif jusqu'à ce que toutes les solutions extrêmes soient trouvées. À cet effet, tout algorithme peut être utilisé. Dans le document en question, les auteurs ont choisi l'algorithme de Dijkstra, comme suggéré par Raith et Ehrgott. Deux solutions initiales (notées p_1 et p_2) sont choisis en fonction de l'ordre lexicographique, chaque solution étant la meilleure en fonction d'un Γ. Les poids de $F(p)$ sont déterminés par ces équations :

$$\begin{cases} \alpha_1 = f_2(p_1) - f_2(p_2) \\ \alpha_2 = f_1(p_2) - f_1(p_1) \end{cases} \tag{3.1}$$

Un tel choix permet d'obtenir une solution efficace ($p3$) dont l'image dans l'espace objectif est la distance maximale de la ligne droite reliant les points image des deux premières solutions (voir figure 3.3). Si cette solution est distincte des deux solutions qui la définissent, deux nouveaux sous-problèmes peuvent être formulés, l'un entre p1 et p3 et l'autre entre

p3 et p2. Sinon, le processus est terminé. Lorsque tous les problèmes et la somme pondérée des sous-problèmes ont été résolus, un ensemble complet de solutions extrêmes est obtenu.

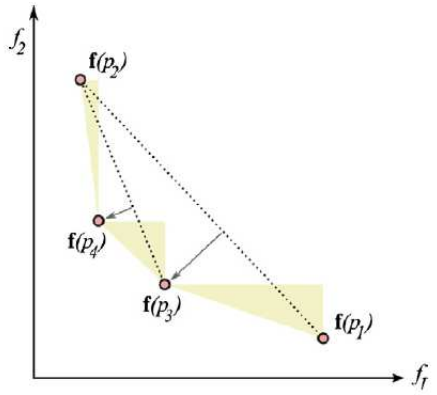

FIGURE 3.3 – Exemple de l'approche dichotomie pour l'espace bi-objectif. Quatre solutions extrêmes sont trouvées, permettant d'identifier trois régions triangulaires où appliquer la phase 2 du processus [42].

3.3.3 Approches incomplètes

Compte tenu de la grande complexité du problème consistant à trouver l'ensemble des solutions efficaces, une approche plus rapide peut être obtenue par négociation de la qualité par rapport à la vitesse. Cela conduit à concevoir des algorithmes qui donnent une représentation grossière de l'ensemble pareto, mais qui fonctionne en temps polynomial. Soit ε une constante représentant une précision souhaitée. La propriété de domination peut être relâchée et on peut établir qu'un chemin p $\varepsilon-$domine q si :

$$f(p) \neq f(q) \wedge f_i(p) \leq (1+\varepsilon) f_j(q), \forall j \in [1, M] \tag{3.2}$$

Suite à la définition de Sahni [52], nous pouvons affirmer qu'un algorithme est ε-approximation si et seulement s'il génère une solution $\bar{p}$ telle que :

$$\frac{|f(\bar{p}) - f(p^*)|}{|f(p^*|} \leq \varepsilon \tag{3.3}$$

où p^* est la solution optimale et f est la fonction objectif.

Plusieurs méthodes ont été présentées dans la littérature pour construire le front pareto ε-approché. La technique la plus répandue est en effet la discrétisation des coûts des chemins afin de réduire suffisamment la taille du problème pour le rendre soluble en temps polynomial. Dans [92], l'auteur a présenté trois techniques connues sur le nom «mise à l'échelle» et «arrondissement» (*scaling and rounding*), le partitionnement d'intervalle (*interval partitioning*), et la séparation (*separation*).
La méthode de Scaling and rounding (SR) est la plus utilisée dans la construction d'algorithmes approchée pour la QoS de routage. Elle consiste à élargir chaque coût d'arête et

l'arrondir à une valeur entière, puis à utiliser un algorithme exact pour résoudre le problème avec mise à l'échelle (entier) des coûts, qui n'est pas NP-complet. Sahni prouve que si le facteur d'échelle est choisi de manière appropriée, un algorithme d'approximation en temps polynomial est obtenu. Beaucoup d'échelles valides et de règles d'arrondissement ont été proposées.
Le Partitionnement par Intervalles (IP) consiste à diviser l'espace objectif en une série d'intervalles et à garder au plus une étiquette à chaque nœud pour chaque intervalle. Cela réduit le nombre d'étiquettes à conserver à chaque nœud, , minimisant ainsi le temps d'exécution. Des exemples de partitionnement d'intervalle sont proposés dans [51].
Une approche similaire est également à la base de l'algorithme de séparation. Dans ce cas, deux étiquettes ne sont conservées que si la distance d'au moins une composante des vecteurs objectif est supérieure à un seuil prédéfini. En pratique, si deux chemins menant à un nœud examiné ont des vecteurs d'objectif très proches, un seul des deux est conservé.

Conclusion

Après cette introduction générale de la thématique de la thèse, dans le chapitre suivant nous traiterons plus en détail le problème de plus court chemin dans un contexte théorique d'abord, et dans le contexte du transport multimodal ensuite grâce à un état de l'art plus spécifique.

Plusieurs méthodes algorithmiques peuvent exister pour résoudre le problème du plus court chemin dépendant du temps, toutes dépendent des contraintes et hypothèses à prendre en compte. Nous avons essayé d'évoquer dans ce chapitre ces différents modèles ainsi que les solutions existantes dans la littérature.

Partie 2 : Contributions

Chapitre 4

Graphe et hypergraphe de transfert

4.1 Motivation

Dans cette thèse, notre objectif est le calcul d'itinéraires multimodaux dépendant du temps dans le contexte de réseaux de transport grande distance et tenant compte de la nature inhéremment distribuée des réseaux de transport. Pour celà nous proposons dans ce qui suit deux modèles pour représenter les réseaux de transport multimodaux et dépendant du temps.

- Dans la section 4.3, nous proposons le graphe de transfert, un modèle pour représenter les réseaux de transport multimodaux.
- Dans la section 4.5, nous proposons l'hypergraphe de transfert, une généralisation du graphe de transfert, pour la modélisation des réseaux de grande taille géographiquement distribué.
- dans la section 4.6 nous présentons trois générateurs différents indispensables pour le déploiement de nos solutions.

Avant d'introduire les deux modèles, nous présentons dans la section 4.2 quelques définitions utiles.

4.2 Définitions préliminaires

Nous désignons par $G = (V, E, M)$ un graphe orienté multimodal, où

- $V = \{v_1, ..., v_j\}$ est un ensemble de noeuds,
- $M = \{m_1, .., m_k\}$ est un ensemble de modes de transport (par exemple, train, bus, voiture ... etc.),
- $E = \{e_1, ..., e_l\}$ est un ensemble d'arêtes. Une arête $e_x \in E$ peut être identifiée par un couple $(v_p, v_q)_{m_r}$, où $v_p, v_q \in V$ et $m_r \in M$. L'arête e_x exprime qu'il est possible de passer d'une localité (noeud) v_p à v_q en utilisant le mode de transport m_r.

Définition 4.2.1

Travel : *Soit $e = (v_i, v_j)_{m_k}$ une arête, un travel t est une paire $t = (t_{v_i}, t_{v_j})$ où t_{v_i} dénote le temps du départ du noeud v_i et t_{v_j} le temps d'arrivée au noeud v_j, tels que $t_{v_i} \leq t_{v_j}$. Un vecteur c_t est associé à chaque travel, indiquant le coût (éventuellement nul) induit par la transition de v_i à v_j à l'instant t_{v_i} (par exemple, la distance, le prix, le confort... etc.).*

Définition 4.2.2

Graphe multimodal dépendant du temps (TDMG) : *Un graphe multimodal dépendant du temps est un 4-uplet $G = (V, E, M, T)$ où*
- *V est un ensemble de noeuds,*
- *E un ensemble d'arêtes,*
- *M un ensemble de modes.*
- *$T = \bigcup_{e_i \in E} \tau_{e_i}$ sachant que à chaque arête $e_i \in E$ est associée un ensemble de travels $\tau_{e_i} = \{(t_{s_1}, t_{a_1}), ..., (t_{s_k}, t_{a_k})\}$, tels que $\mid \tau_{e_i} \mid = k \geq 1$ et $t_{s_j} \leq t_{a_j} \forall j \in \{1..k\}$.*

Définition 4.2.3

Chemin multimodal : *Un chemin multimodal $p_{v_1.v_k} = (v_1 \rightarrow v_k)$ est une séquence d'arêtes entre les noeuds v_1 et v_k, $((v_1, v_2)_{m_1}, ..., (v_{k-1}, v_k)_{m_{k-1}})$, où $\forall i, j \in \{1..k\}, v_i, v_j \in V, (v_i, v_{i+1})_{m_i} \in E, m_i \in M$, et $i \neq j \Leftrightarrow v_j \neq v_i$.*

Définition 4.2.4

Coût dépendant du temps : *Si P dénote l'ensemble de tous les chemins dans un graphe multimodal dépendant du temps G, la fonction $f(p, t_0), f : P \times T \rightarrow R^c$ représente le vecteur de coûts du chemin p avec un temps de départ t_0 où $c \geq 1$ reprsente le nombre de coûts*

Définition 4.2.5

Plus court chemin dans TDMG : *Etant donnés un graphe multimodal dépendant du temps $G = (V, E, M, T)$, deux noeuds $s, d \in V$ et un temps du départ $t_0 \in \{t_1, ..., t_l\}$, le problème de plus court chemin dans un graphe multimodal dépendant du temps consiste à calculer un chemin p de s vers d avec un temps de départ $t_s \geq t_0$ tel que $f(p, t_s)$ est minimal.*

4.3 Le graphe de transfert

Définition 4.3.1

Noeud de transfert : *Soient $G_i = (V_i, E_i, M_i, T_i)$ et $G_j = (V_j, E_j, M_j, T_j)$ deux réseaux de transport correspondant à deux modes de de transport M_i et M_j distincts, un noeud $v_i \in V_i$ est un noeud de transfert si $v_i \in V_j$ aussi.*

Définition 4.3.2

Graphe de transfert : *Soit $G = (V, E, M, T)$ un graphe multimodal dépendant du temps, un graphe de transfert (Transfer_Graph) associé à G est une paire $G_t = (C, T_r)$ où :*
- *$C = \{C_1, ..., C_k\}$ est un ensemble de k composants.*
- *T_r est un ensemble d'arêtes virtuelles.*
- *Chaque composant $C_i = (V_i, E_i, M_i, T_i, TV_i)$ correspond à une représentation unique d'un réseau de transport unimodal où $TV_i \subset V_i$ est l'ensemble des noeuds de transferts :*

- $E\ \forall i \in \{1..k\}, \bigcup V_i = V, \bigcup E_i = E, \bigcup T_i = T$ *and* $\bigcup M_i = M$.
- $T_r = \bigcup_{i \in \{1..l\}} t_{r_i}$ *où* $t_{r_i} = (v_x, v_y)$ *représente un transfert du mode* m_x *vers le mode* m_y *dans le noeud* v_x *(ou* v_y *les deux noeuds représentent la même localité physique).*

La figure 4.1 et le tableau 4.1 illustrent un exemple d'un graphe de transfert composé de deux modes : Mode1 et Mode2, reliés par trois noeuds de transfert : a, c, et e . Chaque composant contient les arêtes appartenant à seulement un mode de transport. Dans cet exemple, deux heures de départ sont possibles par arête. Le passage du noeud a à b ne peut s'effectuer que dans le Mode 1 avec deux options possibles : au départ du noeud a à 1 et arrivant à b à 3 ou au départ à 3 et arrivant à 4 (voir tableau 4.1).

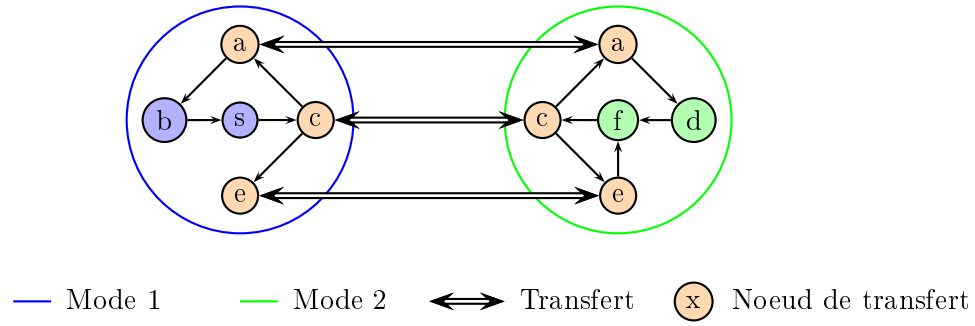

FIGURE 4.1 – Exemple d'un graphe de transfert

Mode 1			Mode 2		
Arêtes	Temps	Coûts	Arêtes	Temps	Coûts
$a \to b$	$1 \to 3$	1	$a \to d$	$1 \to 4$	1
	$3 \to 4$	2		$12 \to 15$	2
$b \to s$	$3 \to 5$	1	$d \to f$	$4 \to 6$	1
	$4 \to 8$	2		$7 \to 9$	2
$s \to c$	$6 \to 8$	1	$f \to c$	$6 \to 7$	1
	$7 \to 10$	2		$8 \to 9$	3
$c \to a$	$2 \to 3$	1	$c \to a$	$3 \to 9$	7
	$9 \to 11$	1		$8 \to 11$	9
$c \to e$	$9 \to 10$	3	$c \to e$	$10 \to 11$	3
	$10 \to 11$	1		$13 \to 15$	1
			$e \to f$	$3 \to 6$	1
				$7 \to 9$	2

TABLE 4.1 – Un exemple de dépendance du temps du graphe de la figure 4.1

4.4 Le "Relevant Graph"

Les chemins dans un graphe de transfert G_t peuvent êtres divisés en deux groupes : les chemins intra-composants et les chemins inter-composants. Un chemin inter-composant est un chemin qui utilise au moins deux composants distincts. Un chemin intra-composant est un chemin qui relie deux noeuds $s, d \in C_i$ passant uniquement par des noeuds du composant C_i. En particulier, les chemins intra-composants peuvent être répartis de la manière suivante :

- $P^{*k}_{s.d}$: Ensemble de chemins reliant un noeud source s (spécifié par la requête) et un noeud destination d dans un même composant C_k.
- $P^{*k}_{s.-}$: Ensemble des chemins reliant un noeud source s d'un composant à n'importe quel noeud de transfert de ce même composant C_k.
- $P^{*k}_{+,d}$: Ensemble de chemins reliant tout noeud de transfert à un un noeud destination d dans le même composant C_k.
- $P^{*k}_{+,-}$: Ensemble de chemins reliant toutes les paires de noeuds transfert du même composant C_k.

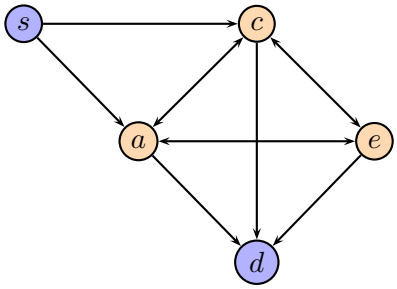

FIGURE 4.2 – Le "Relevant Graph" calculé dépuis le graphe de transfert de la figure 4.1

Chemins Relevants	Arêtes	Chemins	Temps		Coût
Chemins Relevants "têtes"	$(s \to a)$	$(s \to c \to a)_1$	6	$\to$11	2
$(P^{*1}_{s.-} \bigcup P^{*2}_{s.-})$	$(s \to c)$	$(s \to c)_1$	6	$\to$ 8	1
			7	$\to$10	2
Chemins Relevants "intermédiaire"	$(a \to c)$	$(a \to d \to f \to c)_2$	1	$\to$ 7	3
$(P^{*1}_{+.-} \bigcup P^{*2}_{+.-})$	$(a \to e)$	$(a \to d \to f \to c \to e)_2$	1	$\to$11	6
			1	$\to$15	4
	$(c \to a)$	$(c \to a)_1$	2	$\to$ 3	1
			9	$\to$11	1
		$(c \to a)_2$	3	$\to$ 9	7
			8	$\to$11	9
	$(c \to e)$	$(c \to e)_1$	9	$\to$10	3
			10	$\to$11	1
		$(c \to e)_2$	10	$\to$11	3
			13	$\to$15	1
	$(e \to a)$	$(e \to f \to c \to a)_2$	3	$\to$11	11
Chemins Relevants "queue"	$(e \to c)$	$(e \to f \to c)_2$	3	$\to$ 7	2
$(P^{*1}_{+.d} \bigcup P^{*2}_{+.d})$	$(a \to d)$	$(a \to d)_2$	1	$\to$ 4	1
			12	$\to$15	2
	$(c \to d)$	$(c \to a \to d)_2$	3	$\to$15	9
			8	$\to$15	11
	$(e \to d)$	$(e \to f \to c \to a \to d)_2$	3	$\to$15	13

TABLE 4.2 – Chemins Relevants calculés depuis le graphe de transfert. Figure 4.1

Etant donné un graphe de transfert $G_t = (C, T_r)$, pour toute paire origine-destination $s, d \in V$, les ensembles de chemins $P^{*k}_{s.d}$, $P^{*k}_{s.-}$, $P^{*k}_{+,d}$ et $P^{*k}_{+,-}$ peuvent être calculés pour tout composant $C_k \in C$. Sur base de ces chemins et de la requête utilisteur à un instant

t_0, il est possible de construire un graphe réduit à partir duquel tous les chemins inter-composants peuvent êtres calculés. Nous appelons ce graphe, **le "Relevant Graph"** noté G_r. Les noeuds de ce graphe réduit sont les noeuds source et destination s, d, et tous les noeuds de transfert de chaque composant du C. Les arêtes du graphe G_r sont les chemins calculés au auparavant réduits à des simples arcs. Un chemin est représenté uniquement par son noeud de départ et son neoud d'arrivée. Le "Relevant Graph" est formellement défini comme suit :

Définition 4.4.1

Graphe Relevant *Etant donnés $G_t = (C, T_r)$ un graphe de transfert et une requête utilisateur de recherche d'itiniéraire d'un noeud de départ s à un noeud d'arrivée d au temps de départ t_0. Si $\forall C_k = (V_k, E_k, M_k, T_k, TV_k) \in C$, $P_k = P^{*k}_{s.d} \bigcup P^{*k}_{s.-} \bigcup P^{*k}_{+,d} \bigcup P^{*k}_{+,-}$ est l'ensemble des chemins Relevant de chaque composant C_k, le "Relevant Graph" correspondant est donc $G_r = (V_r, E_r)$ où :*

- $V_r = (\bigcup_{\forall k} TV_k) \bigcup \{s, d\}, \forall C_k \in C$.
- $E_r = \bigcup_{\forall k} P_k, \forall C_k \in C$

Dans le tableau 4.2 et la figure 4.2 un exemple d'un "Relevant Graph" (G_r) est présenté. G_r est un multigraphe monomodal simplifié dépendant du temps dont la taille est beaucoup plus réduite que celle du transfert graphe associé comme le montrent les deux figures 4.1 et 4.2. La résolution de la requête peut être faite en appliquant un algorithme de plus court chemin classique, intégrant la notion de dépendance du temps sur le "Relevant Graph".

4.5 L'hypergraphe de transfert

Dans cette section, nous présentons un modèle d'abstraction des réseaux de transport distribués, multimodaux et dépendant du temps. Ce modèle appelé hypergraphe [7] compose de deux niveaux d'abstraction : le niveau région et le niveau modalité. Au niveau supérieur, le graphe est divisé en régions en tenant comptes de leur localisation géographique. Chaque région est également divisée selon les modes de transport disponibles (voir Figure 4.3).

La figure 4.3 montre un exemple d'un hypergraphe composé de deux régions. Les deux régions sont représentées par des graphes de transfert (voir section 4.3), reliés par des arêtes virtuelles. La région 1 admet trois composantes interconnectées, les noeuds E, D, et H sont des noeuds de transfert. Dans cet exemple, deux heures de départ (travel) sont associées à chaque arête (voir tableau 4.3 un exemple de la grille du temps). D, G et K sont des noeuds "ponts" permetant de quitter la region 1 en destination vers la region 2.

Nous définissions formellement cette notion d'hypergraphe comme suit :

7. Notons que la notion d'hypergraphe utilisée dans cette thèse est différente de celle connue dans la théorie des graphes.

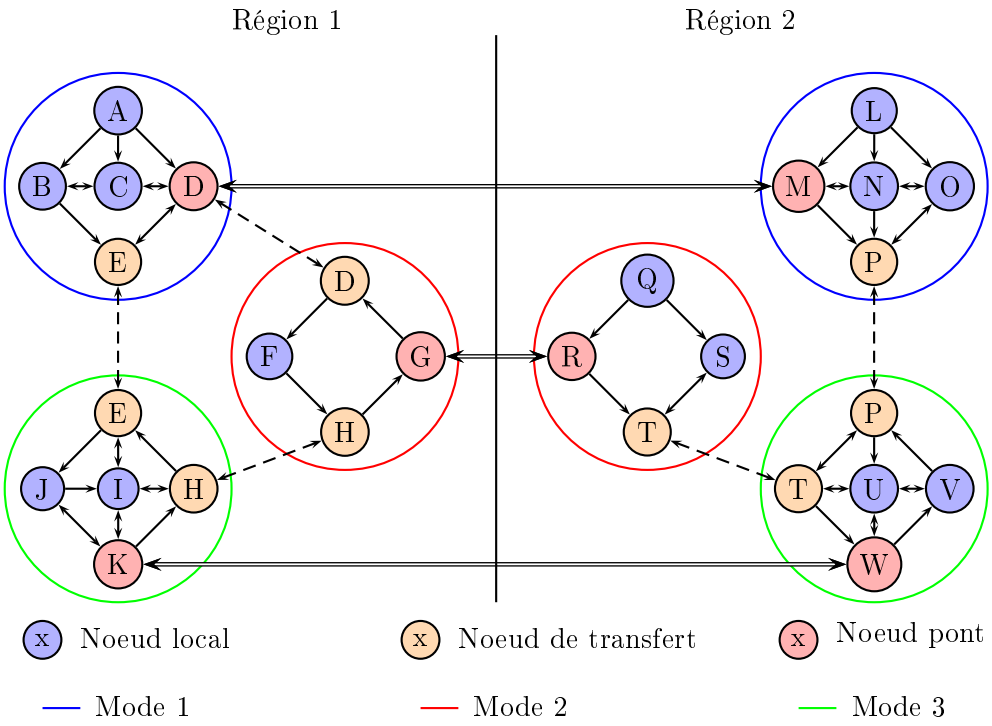

FIGURE 4.3 – Exemple d'un hypergraphe de transfert

Définition 4.5.1

Pont : *Etant donné deux graphes de transfert G_{t_i}, G_{t_j},* ***pont(ou Bridge)*** *$b = (v_p, v_q)m_r$ est une arête particulière qui relie deux régions(deux graphes de transfert), où $v_p \in G_{t_i}$, et $v_q \in G_{t_j}$. Le pont b signifie qu'il est possible de passer du noeud v_p de la région i au noeud v_q de la région j en utilisant le mode m_r.*

Définition 4.5.2

Hypergraphe de Transfert : *Un Hypergraphe de transfert est un graphe $G_h = (R, B)$ où $R = \{G_{t_1}, G_{t_2}, ..., G_{t_n}\}$ est un ensemble de n regions (transfert graphes) et $B = \{b_1, b_2, ..., b_m\}$ est l'ensemble des ponts connectant les regions.*

Region 1					
Mode 1			Mode 2		
Arêtes	Temps	Coûts	Arêtes	Temps	Coûts
$A \to C$	$2 \to 3$	8	$D \to F$	$2 \to 4$	4
	$4 \to 6$	3		$5 \to 9$	1
$E \to D$	$4 \to 7$	2	$F \to H$	$5 \to 8$	4
	$8 \to 9$	9		$8 \to 9$	9
$B \to E$	$8 \to 9$	6	$H \to G$	$9 \to 10$	8
	$12 \to 14$	1		$11 \to 16$	3

TABLE 4.3 – Un exemple de travels du graphe donnée par l'exemple de la figure 4.3

4.6 Génération des données

Nous évoquons ici une problématique particulièrement importante, surtout pour ce qui concerne le transport, à savoir la non disponibilité des données. A notre connaissance et sans doute à cause des contraintes métiers, il n'existe pas des "benchmarks" accessibles à la communauté de recherche publique, dans ce domaine particulier comme il en existe pour d'autres applications et d'autres problèmes d'optimisation.

Pour valider les solutions algorithmiques, nous proposons ici notre propre générateur de données. La création d'un générateur de données transport est une tâche particulièrement difficile, car il s'agit de produire des réseaux quasi réels en se basant sur le hasard. A cet effet, nous avons proposé trois générateurs différents : le premier, totalement stochastique, est utilisé surtout pour valider les méthodes d'un point de vue académique. Le deuxième générateur est plus réaliste et son but est de générer des réseaux séparés en plusieurs villes connectées entre elles. Le troisième générateur est beaucoup plus proche de la réalité car il est basé sur des informations collectées à partir de Open-Street-Map.

4.6.1 Générateur I : Stochastique

Le but de ce générateur est de créer un graphe multimodal dépendent du temps à l'aide d'un algorithme totalement stochastique. Les graphes générés seront surtout utilisés pour réaliser des tests académiques qui n'envisagent pas d'analyse sur la topologie du réseau. Les seuls paramètres pris en compte dans ce générateur concernent le nombre de nœuds, le nombre d'arêtes et la densité du réseau de transport .
Dans une première étape, ce générateur crée un ensemble de n nœuds, puis les connectent par un ensemble de $(n-1)$ arêtes créées de façon aléatoire pour assurer la connectivité du graphe. Ensuite $(m-n+1)$ arêtes additionnelles sont créées pour avoir au total m arêtes. Les arêtes créées sont unidirectionnelles, chacune est associée à un mode donné (le nombre de modes est fixé au préalable). La deuxième étape consiste à générer les grilles de temps ou ensemble de travels associés aux arêtes pour avoir un réseau dépendant du temps. La grille des temps est générée de façon aléatoire en créant pour chaque travel (t_1, t_2) un temps de départ t_1 et un temps d'arrivée t_2 tel que $t_2 \geq t_1$. Les deux valeurs temporaires sont choisies aléatoirement sans aucune loi de distribution. Une fois qu'un travel est créé, son vecteur de coût est ensuite généré. La même procédure est utilisée pour créer les noeuds de transfert entre les modes.

Ce générateur n'étant réalisé qu'à des fins purement académiques, il ne présente aucune intelligence additionnelle pour maintenir l'aspect stochastique. Les avantages de ce générateur sont :

- Les graphes sont générés entièrement au hasard.
- Le temps de génération est négligeable.
- Sa complexité mémoire est faible et par conséquent des graphes de très grande taille peuvent être traités.

Caractéristiques	**Solutions**
Les réseaux de transport sont plus présents et plus denses dans les centres villes et sont de plus en plus éparse en s'éloignant du centre.	Le graphe généré représente virtuellement une ville composée de plusieurs zones. Une zone centrale, représentant le centre ville, et le reste des zones représentent la périphérie. Les noeuds de graphe sont crées dans les zones avec une distribution plus dense dans le centre ville, et de plus en plus éparse dans le reste des zones.
Les réseaux publics sont constitués de lignes de services, ainsi les localités (gares, stations) sont liées ente elles via des lignes et non pas par une simple connexion comme dans le cas d'un réseau privé.	La même propriété est préservée, c'est à dire que cette fois les nœuds sont créés ensuite, plutôt que de créer des arêtes directement, des lignes virtuelles sont d'abord créées. Chaque ligne se compose d'un ensemble de nœuds et est associée à un mode unique. La création de la ligne est entièrement aléatoire. Une fois qu'une ligne est créée, une grille d'horaire est générée pour garantir la continuité dans le temps d'une ligne. Des vecteurs de coûts sont aussi associés à chaque ligne dans la grille d'horaire. Finalement, les lignes sont transformés en arêtes et les arêtes en grille des temps.
Les coûts ainsi que le temps de parcours d'une ligne sont sensibles au type du service fourni. Par exemple, le prix d'un métro qui ne quitte pas le centre ville est moins cher qu'un métro de banlieue.	Plusieurs types de lignes sont créées.Les oûts cost et les grilles de temps sont générées en fonction des types de lignes.

TABLE 4.4 – Caractéristiques des réseaux du transport réels et les solutions apportées par le générateur II

4.6.2 Générateur II : Un générateur réaliste

Le premier générateur ne peut malheureusement être suffisant car les graphes générés sont trop loin des caractéristiques des réseaux réels. En effet les réseaux de transport ont des propriétés spécifiques. Ce deuxième générateur est proposé pour créer des graphes en tenant compte des caractéristiques de réseaux de transport dans le monde réel. Les propriétés essentielles de ces réseaux et les solutions apportées sont présentées dans le tableau 4.4.

Ce générateur commence par créer un ensemble de zones. Le nombre de zones créés, est généré de façon aléatoire mais dépend de la taille du graphe souhaité. Ensuite un ensemble de n nœuds sont créés. Chaque nœud appartient à une et une seule zone, selon la règle suivante : plus une zone est proche du centre plus la probabilité de contenir un nœud est grande (voir figure 4.4). Ensuite un ensemble de lignes spécifiques sont créées

pour assurer la connectivité du réseau, puis le reste des lignes sont créées pour atteindre le nombre d'arêtes souhaité (voir figure 4.4). Pour chaque ligne générée, une grille des temps est créée au hasard en respectant le type de la ligne et une distribution correcte espace-temps. C'est à dire que le temps moyen entre deux nœuds successifs dépend de leur emplacement géographique, un système de coordonnées est mis en place pour calculer la distance euclidienne entre deux points. Ensuite les lignes sont transformées en des arêtes ainsi que les grilles en des "travels". La dernière étape consiste à créer les noeuds transferts et leurs coûts

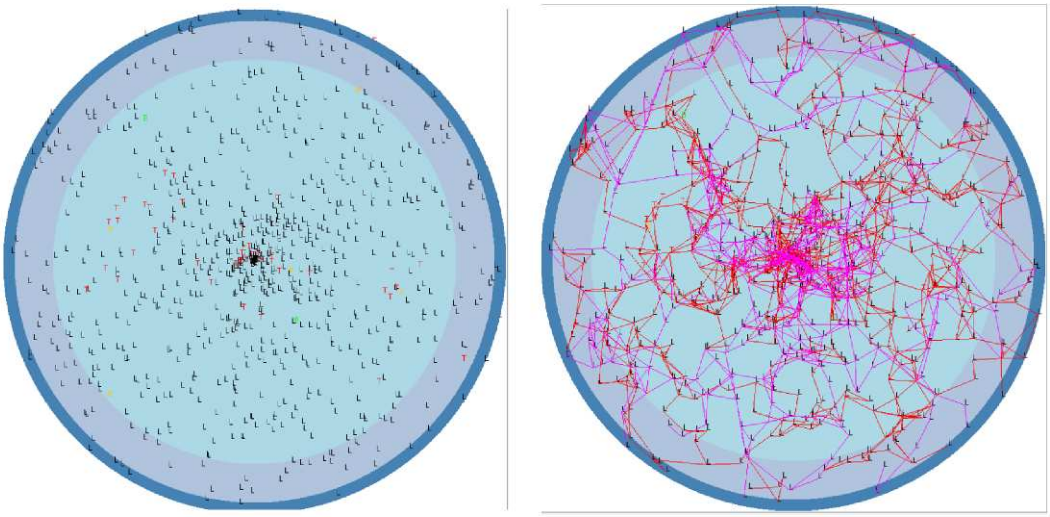

FIGURE 4.4 – Exemple du fonctionnement du générateur II : Un exemple d'une ville avec 3 zones, (a) présente la distribution dans l'espace des nœuds crées, (b) les différentes lignes avec 2 modes.

4.6.3 Générateur III : OSM

L'inconvénient majeur du deuxième générateur réside au niveau de la création des lignes, étape cruciale de ce dernier. En effet, le choix des nœuds d'une ligne est crucial, puisque le hasard peut créer des lignes avec une très mauvaise distribution spatiale. Donc l'objectif de ce dernier générateur est de créer des lignes plus réalistes. Pour y parvenir, nous avons utilisé des cartes géographiques réelles issues de OpenStreetMap[8].
Ce générateur fonctionne de la manière suivante : un fichier OSM de la ville (région) ciblée est donné en entrée, l'algorithme scanne ce fichier pour extraire toutes les informations statiques, les nœuds, les stations, les parkings, les lignes de service (bus, train subway..) .... Ensuite un graphe est créé exactement de la même manière qu'avec le générateur II sans avoir à générer les lignes. L'inconvénient majeur de ce générateur est qu'il est plus lent et surtout nécessite plus de mémoire pour scanner le ficher en entrée. Il nous a permi toutefois de déployer nos résultats sur quelques réseaux réels.

8. fr.wikipedia.org :OpenStreetMap est un projet qui a pour but de créer des cartes libres du monde sous licence libre, en utilisant le système GPS et d'autres données libres.

Conclusion

Bien que court, ce chapitre est le résultat d'un travail de réflexion et d'analyse important durant la première année de cette thèse. En effet, la définition d'un modèle de représentation des réseaux de transport, répondant aux contraintes et hypothèses fixées dans l'introduction de ce document, constitue un pré-requis indispensable pour les travaux qui ont suivi. Nous avons proposé le modèle appelé graphe de transfert qui permet la représentation des réseaux du transport multimodaux, en différents composants monomodaux indépendants pour des raisons de performance, de flexibilité et d'extensibilité. Nous avons ensuite étendu ce modèle à celui d'hypergraphe de transfert, qui, tout en préservant les propriétés du modèle précédent, permet en plus de modéliser les réseaux de transport de très grandes tailles (des réseaux allant jusqu'à 400.000 noeuds).

Pour finir, nous avons abordé dans ce chapitre, une problématique assez connue dans le domaine de l'optimisation à savoir l'acquisition des pour la phase de tests et de validation des algorithmes. En effet, dans le domaine de transport, il est difficile de garantir une centralisation des données car pour des raisons métier, les opérateurs de transport gardent leur données souvent confidentielles. Pour valider nos différentes approches nous avons proposé trois générateurs pour la réalisation des tests académiques et réels.

Chapitre 5

Calcul d'itinéraire mono-objectif : Approches Séquentielle et Parallèle

5.1 Introduction

Dans le chapitre 4, nous avons proposé le graphe de transfert comme modèle d'abstraction des réseaux de transport multimodaux. Nous avons aussi introduit la notion de "Relevant Graph", une version simplifiée du graphe de transfert permettant d'optimiser les algorithmes de calcul de plus court chemin que nous développons dans ce chapitre. Plus précisément, nous développons ici trois algorithmes pour le calcul du plus court chemin dans les réseaux de transport Multimodaux, dépendant du temps et mono-objectifs.

Les deux algorihmes séquentiels sont basés sur le graphe de transfert et "Relevant Graph" qui en découle. L'algorithme parallèle est basé sur l'hypergraphe de transfert, une généralisation des graphes de transfert pour modéliser des réseaux de transports de très grande taille. Dans la suite de ce chapitre, nous présenterons ces différents algorithmes, leurs propriétés théoriques ainsi que les résultats d'implémentation, aussi bien sur des cas académiques que sur des cas réels.

5.2 Approche Séquentielle

5.2.1 Notations

Dans la suite de ce chapitre, nous supposons que $G = (V, E, M)$ désigne un graphe multimodal orienté, où $V = \{v_1, ..., v_j\}$ est un ensemble de noeuds, $M = \{m_1, .., m_k\}$ est un ensemble de modes, et $E = \{e_1, ..., e_l\}$ est un ensemble d'arêtes. Une arête $e_x \in E$ est un couple de la forme $(v_p, v_q)_{m_r}$, où $v_p, v_q \in V$ et $m_r \in M$. Nous supposons aussi qu'une requête utilisateur doit spécifier le noeud de départ s, le noeud d'arrivée d ainsi que le temps de départ de l'utilisateur t_0 de l'utilisateur. Comme nous considérons le problème du chemin le plus rapide dans un réseau dépendant du temps et mono-objectif, nous supposons donc que le seul coût à optimiser est le temps d'arrivée de l'utilisateur ou le temps de trajet de ce dernier. Cette hypothèse n'est nécessaire que dans la cas de l'algorithme exact de Dijkstra intégrant la notion de temps. Les algorithmes basés sur des métaheuristiques peuvent optimiser un autre critère que le temps.

5.2.2 Algorithmes basés sur le "Relevant Graph"

Pour satisfaire une requête utilisateur recherchant un itinéraire optimal entre les noeuds s et d à l'instant t_0, il est nécessaire de calculer les différents ensembles de chemins $P^{*k}_{+,-}(t)\forall t \in \{t_1, ..., t_l\}$, $P^{*k}_{s.d}(t_0)$, $P^{*k}_{s,-}(t_0)$ et $P^{*k}_{+,d}(t)\forall t \in \{t_1, .., t_l\}$, $\forall C_k \in C$. Nous appelons ces ensemble de chemins, pour y faire référence, des chemins Relevants.

Certains de ces ensembles sont dépendants de la requête utilisateur et d'autres pas. En effet, l'ensemble $P^{*k}_{+,-}(t)\forall t \in \{t_1, ..., t_l\}$ est indépendant des noeuds s et d. Cet ensemble peut donc être calculé en phase de prétraitement de tout algorithme. Par conséquent, la résolution du plus court chemin basée sur le graphe de transfert peut se décomposer de la façon suivante :

1. *Phase de Pré-calcul* : Calcul et sauvegarde des ensembles $P^{*k}_{+,-}(t)\forall C_k \in C, \forall t \in \{t_1, ..., t_l\}$.

2. *pour une requête donnée* : Calcul des ensembles $P^{*k}_{s.d}(t_0)$, $P^{*k}_{s,-}(t_0)$ et $P^{*k}_{+,d}(t)\forall t \in \{t_1, .., t_l\}$, $\forall C_k \in C$.

3. *Construction du "Relevant Graph" G-r* : Utiliser le pré-calcul ainsi que les ensembles de l'étape 2 pour constuire le "Relevant Graph" G_r.

4. *Calcul de l'itinéraire* : Trouver le plus court chemin sur le graphe G_r par un algorithme exact ou non.

Cette décomposition permet de réduire considérablement la complexité en temps du problème. En effet, le pré-calcul concerne le calcul de l'ensemble des chemins les plus difficiles en termes de temps de calcul. De plus, ce pré-calcul fournit un avantage important dans les systèmes réels devant résoudre des milliers de requêtes en des temps courts. C'est le cas des Web service tels que mappy ou google par exemple. Bien entendu ce pré-calcul suppose que le réseau est robuste.

5.2.3 Algorithme MTDSPP1 : Calcul du plus court chemin basé sur le graphe de transfert

L'algorithme MTDSPP1 (Multimodal Time Dependent Shortest Path Problem version 1) est décrit schématiquement par la figure 5.1.

L'algorithme 12 est l'algorithme générique de calcul du plus court chemin basé sur le graphe de transfert.

L'étape de pré-calcul étant la plus coûteuse nous proposons ici différentes faons de la calculer.

5.2.3.1 Algorithme de Dijkstra pour le pré-calcul

L'algorithme de Dijkstra est l'un des algorithmes de calcul de plus court chemin les plus performants dans le contexte de graphes valués avec des coûts non-négatifs. Dans

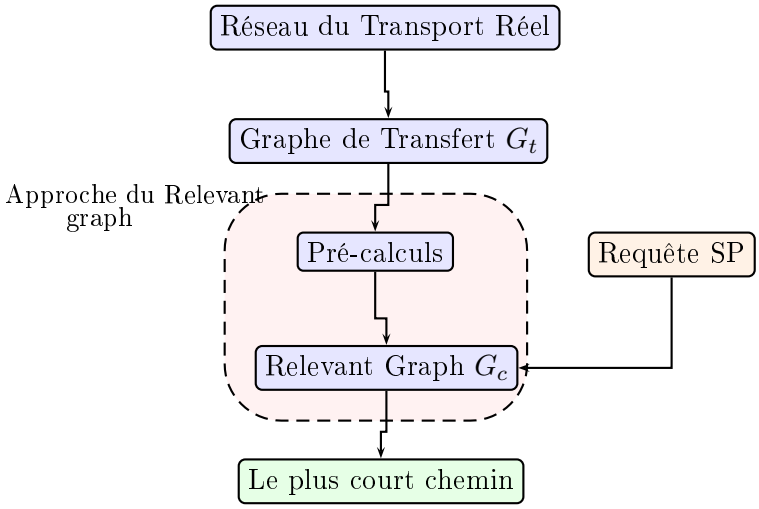

FIGURE 5.1 – Approche du "Relevant Graph"

Algorithme 12 : Calcul du plus court chemin basé sur le graphe de transfert G_T

```
1 DB ← Precalculation(G_t);
2 tant que Request(s, d, t_0) faire
3     R_g ← Build(DB, s, d, t_0);
4     SPP(G_r);
5 fin
```

cette section, nous présentons une variante de cet algorithme pour réaliser la phase de pré-calcul. Afin de décrire formellement et simplement cette approche, nous avons besoin d'introduire les trois définitions suivantes.

Définitions utiles :

Définition 5.2.1

(Arêtes Entrantes et Arêtes Sortantes) *: Soit $G = (V, E)$ un graphe et $v_i \in V$ un noeud de G*

- *$E^-(v_i) = \{(v_j, v_i), \forall (v_j, v_i) \in E\}$ est l'ensemble des arêtes entrantes de v_i.*
- *$E^+(v_i) = \{(v_i, v_j), \forall (v_i, v_j) \in E\}$ est l'ensemble des arêtes sortantes de v_i*

Définition 5.2.2

(Arête accessible) *: Soit un graphe multimodal dépendant du temps $G = (V, E, M, T)$, $\tau_{e_i} = \{(t_s, t_a)_1, ..., (t_s, t_a)_k\}$ est l'ensemble de k travels valides à t de e_i. Un travel (t_s, t_a)*

Algorithme 13 : Précalcul avec Dijkstra

```
1 pour chaque C_k ∈ G_t faire
2     pour chaque v ∈ TransferVertex(C_k) faire
3         pour chaque t ∈ DepartTime(v) faire
4             DijkstraOneToMany(C_k, v, t);
5         fin
6     fin
7 fin
```

est dit valide à un temps t si $t_s > t$. L'arête e_i est accessible à un temps t si $|\tau_{e_i}| > 0$.

Définition 5.2.3

***(Travel le plus rapide)** : Soit un graphe multimodal dépendant du temps $G = (V, E, M, T)$, $\tau_{e_i} = \{(t_s, t_a)_1, ..., (t_s, t_a)_k\}$ est l'ensemble de k travels valides à t de e_i. Un travel $(t_s, t_a) \in \tau_{e_i}$ est dit la plus rapide si $\forall (t_s^{'}, t_a^{'}) \in \tau_{e_i}, t_a \leq t_a^{'}$.*

Dijkstra dépendant du temps : Le pseudo-code de l'approche de Dijkstra est présenté dans l'algorithme 14. Comme on peut le voir, la fonction principale est $DijkstraOneToMany(Ck, v, t)$. Cette fonction est appelée pour chaque composant monomodal C_k, chaque noeud de transfert v, et pour chaque temps du départ t possible de v. Par ailleurs, cette fonction calcule les plus courts chemins entre le noeud de transfert v et tous les noeuds de transfert du composant C_k pour un temps de départ t.

Cette fonction est une variante de l'algorithme classique de Dijkstra et son pseudo-code est présenté dans l'algorithme 14.

La partie principale de cet algorithme consiste en une boucle (lignes 7 à 23) qui s'arrête quand tout le graphe est exploré ou tous les sommets de transfert sont marqués (ligne 23). Dans la première étape de cette boucle, toutes les arêtes sortant du noeud courant sont analysées (lignes 8 à 19). Si une arête est accessible au temps courant t, et si son noeud de destination n'est pas marqué alors l'algorithme met à jour le coût de ce dernier si le nouveau coût est meilleur (ligne 13-14). Cette arête est donc choisie comme candidate pour une prochaine itération.
Après avoir traité toutes les arêtes sortantes (boucle principale), l'arête dont le noeud destination possède le coût minimal dans la liste des candidats, est sélectionnée pour la boucle suivante et le temps courant ainsi que le noeud courant sont mis à jour (lignes 21 et 22). Enfin, le nœud courant est marqué (ligne 23).

5.2.3.2 Utilisation de ACO pour le pré-calcul :

Nous utilisons la métaheuristique ACO pour explorer le graphe en vue de trouver les chemins plus courts entre les noeuds de transfert au niveau de chaque composante du graphe. Une colonie de fourmis est appliquée à chaque composante du graphe. Le pseudo-code de cette approche est représenté par l'algorithme 15

Algorithme 14 : DijkstraOneToMany(C_k, v, t)

```
1  currentTime ← t;
2  currentNode ← s;
3  Mark(s);
4  pour chaque v ∈ V faire
5      cost[v] ← ∞;
6  fin
7  répéter
8      pour chaque edge ∈ E+(currentVertex) faire
9          si accessibleEdge(edge, currentTime) alors
10             vertex ← destinationVertex(edge);
11             si notMarked(vertex) alors
12                 currentCost = bestArrivalTime(edge,currentTime);
13                 si (currentCost < cost[vertex]) alors
14                     cost[vertex] ← currentCost;
15                     candidateEdges ← candidatesEdge ∪ {edge};
16                 fin
17             fin
18         fin
19     fin
20     nextEdge ← selectBestEdge(candidateEdges) ;
21     currentVertex ← destinationVertex(nextEdge);
22     currentTime ← cost[currentVertex];
23     Mark(currentVertex);
24 jusqu'à CandidateEdges = {} OR StopCondition() ;
```

La fonction $FindBestPaths(v, Ck, DB)$ calcule les plus courts chemins du noeud source v vers tous les noeuds de transfert dans la composante C_k. Le pseudo-code de cette fonction est présenté par l'algorithme 16.

Au début, les fourmis trouvent les chemins menant de v à tous les noeuds de transfert de la composante C (ligne 3). Les chemins trouvés sont ensuite classés, décomposés et sauvegardés (lignes 4-5). Finalement, la table de phéromone est mise à jour selon les chemins trouvés (ligne 6). Ces étapes sont exécutées tant que le nombre de cycles spécifié n'est pas atteint (ligne 2). Dans les paragraphes suivants, nous décrivons en détail chacune de ces étapes.

- Dans la première étape de la fonction $FindBestPaths(v, Ck, DB)$, un ensemble de fourmis artificielles explorent la composante C_k du graphe à partir du noeud de transfert v. Les fourmis construisent des chemins sans boucles en ajoutant progressivement les arêtes jusqu'à ce qu'il n' y ait plus d'arêtes accessibles. Quand une nouvelle arête est ajoutée au chemin, le travel ayant le meilleur temps d'arrivée est choisi. La sélection du prochain noeud de l'ensemble des noeuds sortants est probabiliste et il est effectué selon la formule suivante :

Algorithme 15 : Précalcul avec ACO

```
1 pour chaque C_k ∈ G_t faire
2     pour chaque v ∈ TransferNodes(C_k) faire
3         FindBestPaths(v,C_k,DB);
4     fin
5 fin
```

Algorithme 16 : FindBestPath(vertex,Component,DB)

```
1 BD ← Initialization();
2 répéter
3     P ← ExploreGraph(v,C);
4     ClassifyPaths(P);
5     BD ← DecomposeAndStore(P);
6     UpdatePheromones(C,P);
7 jusqu'à nbCycle < MaxCycle ;
```

$$p((i,j)^t) = \frac{[\tau^t_{(i,j)}]^\alpha \cdot [\eta^t_{(i,j)}]^\beta}{\sum_{(i,l)\in E^+(i)}[\tau^t_{(i,l)}]^\alpha \cdot [\eta^t_{(i,l)}]^\beta} \forall (i,j) \in E^+(i) \tag{5.1}$$

Où :
- $p((i,j)^t)$: est la probabilité de choisir l'arête (i,j) à l'instant$t$
- $\tau^t_{(i,j)}$: la valeur du phéromone c_{ij} à l'instant t
- $\eta^t_{(i,j)} = \frac{1}{cost(c_{ij})}$
- $cost(i,j)$ est le coût du chemin courant si on lui ajoute l'arête (i,j)

- Dans la deuxième étape de la fonction (la ligne 4 de l'algorithme 15), les chemins déjà trouvés sont classés. Les chemins qui contiennent deux ou plusieurs sommets de transfert sont classés comme des chemins complets sinon ils sont considérés incomplets. Cette information est utilisée ultérieurement lors de la mise à jour de la phéromone.
- Dans la troisième étape (ligne 5), les chemins complets sont traités pour extraire tous les sous-chemins possibles entre les sommets de transfert. Ce processus découle du fait que tout sous-chemin d'un plus court chemin est un plus court chemin aussi. Le tableau 5.1 présente un exemple de la décomposition d'un chemin.

Dans la dernière étape de la fonction (la ligne 6 de l'algorithme 15), la table de phéromone de la composante C_k est mise à jour. Puisque nous considérons un problème dépendant du temps, la phéromone est associée aux travels et non pas aux arêtes. Les quantités de phéromone des chemins décomposés sont mises à jour en tenant compte de leur classification précédente. Nous augmentons la quantité de phéromone pour tout travel (t_{v_i}, t_{v_j}) appartenant à un chemin complet en utilisant l'équation 5.2, et nous la diminuons pour ceux qui appartiennent à des chemins incomplets, en utilisant l'équation 5.3.

Chemins	Travel
Chemin d'origine	
$p_{1.5} = \{(\mathbf{1}, 2), (2, \mathbf{3}), (\mathbf{3}, \mathbf{4}), (\mathbf{4}, 5)\}$	$\{(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_5)\}$
Chemin décomposé	
$p_{1.4} = \{(\mathbf{1}, 2), (2, \mathbf{3}), (\mathbf{3}, \mathbf{4})\}$	$\{(t_1, t_2), (t_2, t_3), (t_3, t_4)\}$
$p_{1.3} = \{(\mathbf{1}, 2), (2, \mathbf{3})\}$	$\{(t_1, t_2), (t_2, t_3)\}$
$p_{3.4} = \{(\mathbf{3}, \mathbf{4})\}$	$\{(t_3, t_4)\}$

TABLE 5.1 – Exemple de décomposition d'un chemin, les noeuds 1, 3 et 4 sont des noeuds de transfert

$$\tau_{ij}^{(t_{v_i}, t_{v_j})} = \tau_{ij}^{(t_{v_i}, t_{v_j})} + (\delta \frac{1}{cost(p)}) \tag{5.2}$$

$$\tau_{ij}^{(t_{v_i}, t_{v_j})} = \tau_{ij}^{(t_{v_i}, t_{v_j})} - (\gamma \frac{1}{cost(p)}) \tag{5.3}$$

Où γ et $\delta \in [0, 1]$ sont des paramètres à spécifier dépendament du problème à résoudre (taille, densité, ...) et $cost(p)$ représente le coût du chemin p, dans ce cas le temps d'arrivé du chemin.

En outre, les quantités de phéromone sont également modifiées afin de simuler le processus naturel d'évaporation de la phéromone. La modification est faite conformément à la formule classique :

$$\tau_{ij}^{(t_{v_i}, t_{v_j})} = (1 - \rho) \ \forall (t_{v_i}, t_{v_j}) \in E_k \tag{5.4}$$

Où $\rho \in [0, 1]$ est un paramètre d'évaporation.

5.2.3.3 Comparaison des deux approches de pré-calcul

Pour comparer ces deux approches d'un point de vue pratique, nous avons considéré cinq instances instances du problème du plus court chemin, dans un graphe dépendant du temps avec trois modes du transport. On note $|V|$ le nombre de noeuds du graphe, $|E|$ le nombre d'arêtes et $|M|$ celui de modes. Les instances traitées sont : $(100, 300, 3)$, $(500, 1000, 3)$, $(1000, 3000, 3)$, $(2100, 6000, 3)$ et $(4000, 15000, 3)$. Le nombre de travels par arête est 10.

Le tableau 5.2 montre les performances des deux approches pour effectuer l'étape de pré-calcul en comparant les temps CPU et l'espace mémoire nécessaire.

- En terme de temps CPU, l'algorithme de Dijkstra est légèrement meilleur que la métaheuristique pour les petites instances du problème. Cependant, les performances de Dijkstra sont nettement supérieures dans le cas des grandes instances.

– En ce qui concerne l'espace mémoire, l'approche ACO est plus performante que l'algorithme de Dijkstra pour des instances de grande taille. En fait, l'espace mémoire requis par la métaheuristique ne dépasse pas 104MB pour la plus grande instance, alors que l'algorithme de Dijkstra nécessite près de cinq fois cette valeur. L'instance (4000, 15000) n'a pas pu être traitée ni par l'algorithme de ijkstra, faute d'espace mémoire, ni par ACO, faute du temps CPU nécessaire.

instances			**temps CPU(s)**		**Mémoire (MB)**	
$\|V\|$	$\|E\|$	$\|M\|$	Dijkstra	ACO	Dijkstra	ACO
100	300	3	**1.2**	2.2	**20**	30
500	1000	3	**2.7**	6.7	**40**	46
1000	3000	3	**25.0**	80.3	170	**68**
2100	6000	3	**100.0**	1100.0	510	**104**
4000	15000	3	−−	−−	−−	−−

TABLE 5.2 – Comparaison Dijkstra-ACO

La figure 5.2 présente un exemple d'exécution de l'algorithme de Dijkstra et de l'algorithme ACO sur l'instance (1000,3000, 3). Nous constatons clairement que l'algorithme de Dijkstra nécessite un espace mémoire strictement croissant en fonction de temps. Par contre , la quantité mémoire exigée par l'algorithme ACO est inférieure à celle exigée par l'algorithme de Dijkstra.

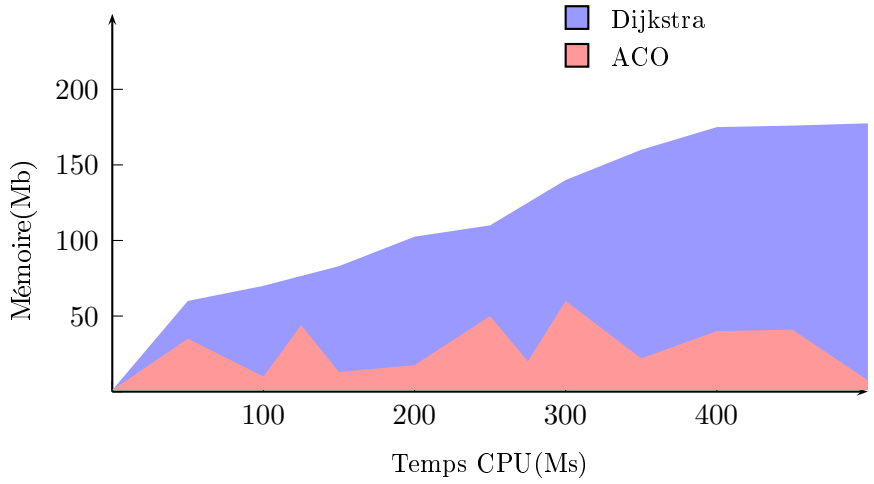

FIGURE 5.2 – Occupation mémoire : Dijkstra-ACO sur l'instance (1000,3000,3)

5.2.4 Algorithme MTDSPP2 : Approche hybride

Jusque là, nous avons présenté l'algorithme de résolution du problème du plus court chemin multimodal dépendant du temps et mono-objectif basé sur le graphe de transfert ainsi que deux implémentations de cette approche : une approche exacte et une approche orientée métaheuristique. Nous avons mis en évidence expérimentalement que ces deux approches présentent des limitations soit en terme de temps CPU, soit en terme d'espace mémoire occupé. Ces limitations sont dûes à la création de la base de donnée et à son stockage. Pour pallier à ces limites, nous proposons une solution alternative MTDSPP2

(Multimodal Time Dependent Shortest Path Problem version 2) qui fait face à ces deux problèmes de limitation de temps d'une part et d'espace mémoire d'autre part. Nous appelons cette méthode alternative méthode hybride.

La figure 5.3 permet de situer apriori cette nouvelle méthode hybride par rapport à la première méthode précédente. Cette figure permet de constater que la méthode hybride, de façon informelle, contourne la création de la base de donnée, en passant par une nouvelle structure intermédiaire dénommée graphe abstrait.

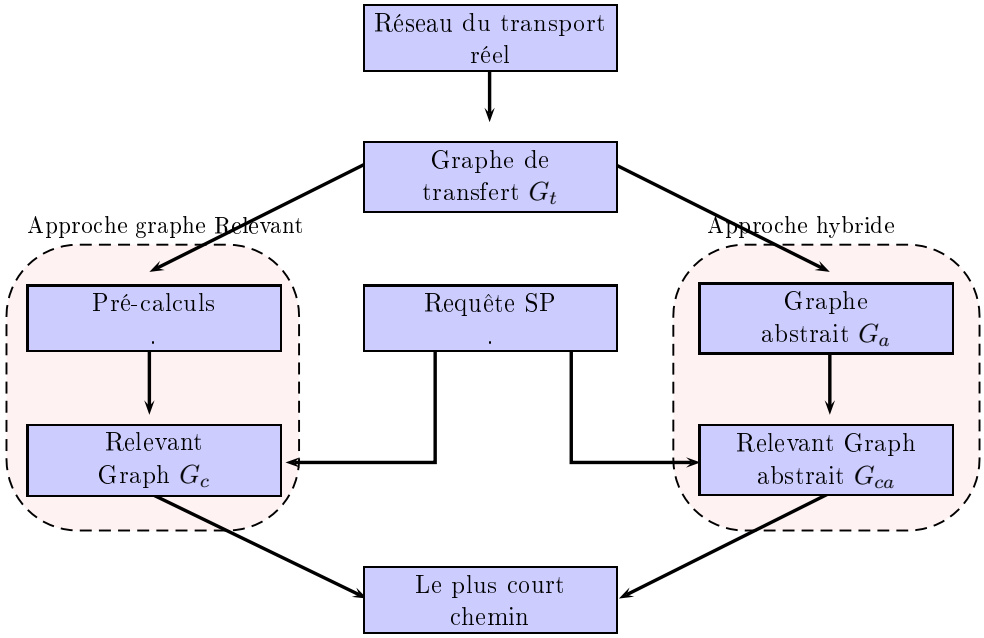

FIGURE 5.3 – Approche hybride pour le graphe de transfert

Dans ce qui suit, nous allons présenter formellement cette nouvelle structure et l'algorithme qui en découle. Pour ce faire, nous présentons au préalable certaines définitions utiles pour la formalisation.

5.2.4.1 Quelques définitions

Définition 5.2.4

(Chemin local) *: Soit un graphe de transfert $G_t = (C, T_r)$, $C_k = (V_k, E_k, M_k, T_k, TV_k) \in C$, un chemin $p_{v_1.v_j} = ((v_1, v_2), ..., (v_{j-1}, v_j))$ est dit local si $\forall i \in \{2, .., (j-1)\}, v_i \notin TV_k$. LP dénote l'ensemble de tous les chemins locaux de G_t.*

Ainsi, le graphe abstrait peut être défini comme suit :

Définition 5.2.5

Graphe abstrait : *Soit un graphe de transfert $G_t = (C, T_r)$, nous définissons un graphe abstrait comme un graphe $G_a = (V_a, E_a)$, où $V_a = \bigcup_{C_k \in C} TV_k$, et $E_a =$*

$(e_1, e_2, ..., e_l)$. Chaque arête $(v_i, v_j) \in E_a$ enveloppe un ensemble de chemins locaux physiques entre v_i et v_j.

La figure 5.4 et le tableau 5.3 présentent un exemple d'un graphe abstrait correspondant au graphe de transfert de la figure 4.1. Cette nouvelle stucture ne comporte que les noeuds de transfert du graphe de transfert initial. Les noeuds intermédiaires supprimés sont remplacés par des arêtes spécifiques étiquetées par des chemins locaux.

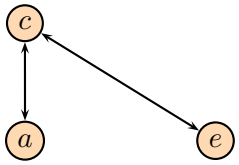

FIGURE 5.4 – Graphe abstrait du graphe de transfert de la figure 4.1

Arête	Chemin	Temps	Coût
$(a \to c)$	$(a \to b \to s \to c)_1$	$--$	$--$
	$(a \to d \to f \to c)_2$	$--$	$--$
$(c \to a)$	$(c \to a)_1$	$--$	$--$
	$(c \to a)_2$	$--$	$--$
$(c \to e)$	$(c \to e)_1$	$--$	$--$
	$(c \to e)_2$	$--$	$--$
$(e \to c)$	$(e \to f \to c)_2$	$--$	$--$

TABLE 5.3 – Chemins locaux du graphe abstrait de la figure 5.4

Définition 5.2.6

Travel dominant : *soit un TDMG $G = (V, E, M, T)$, et deux travels associées (t_{s_1}, t_{a_1}) et (t_{s_2}, t_{a_2}). On dit que (t_{s_1}, t_{a_1}) domine (dans un contexte mono-objectif dépendant du temps) (t_{s_2}, t_{a_2}) si $t_{s_1} \geq t_{s_2}$ et $t_{a_1} \leq t_{a_2}$.*

Définition 5.2.7

Chemin dominé : *soit $p_{v_i.v_j}$ et $p'_{v_i.v_j}$ deux chemins de v_i à v_j et $T_{p_{v_i.v_j}}$, $T_{p'_{v_i.v_j}}$ les ensembles de ces travels, respectivement. On dit que $p_{v_i.v_j}$ domine $p'_{v_i.v_j}$ si et seulemnt si $\forall tr' \in T_{p'_{v_i.v_j}}$, $\exists tr \in T_{p_{v_i.v_j}}$, tel que tr domine tr'.*

Pour une bonne perception de ces notions plutôt abstraites, nous donnons ici par un exemple la notion de chemin dominé. Soit $p_{s.d} = s \leftarrow a \leftarrow b \leftarrow d$ avec les deux travels $\{(2,5),(4,7)\}$ et $p'_{s.d} = s \leftarrow c \leftarrow e \leftarrow d$ avec les deux travels $\{(1,6),(3,8)\}$. p domine p' puisque la travel $(2,5)$ du chemin p domine le deux travels $\{(1,6),(3,8)\}$ de p'.

L'ensemble des chemins réels correspondant aux chemins physiques donnés dans le tableau 5.3 sont résumés dans le tableau 5.4.

Arête	Chemin	Temps	Coût
$(c \to a)$	$(c \to a)_1$	$2 \to 3$	1
		$9 \to 11$	1
	$(c \to a)_2$	$3 \to 9$	7
		$8 \to 11$	9
$(c \to e)$	$(c \to e)_1$	$9 \to 10$	3
		$10 \to 11$	1
	$(c \to e)_2$	$10 \to 11$	3
		$13 \to 15$	1
$(a \to c)$	$(a \to d \to f \to c)_2$	$1 \to 7$	3
$(e \to c)$	$(e \to f \to c)_2$	$3 \to 7$	2
$(s \to c)$	$(s \to c)_1$	$6 \to 8$	1
		$7 \to 10$	2
$(a \to d)$	$(a \to d)_2$	$1 \to 4$	1
		$12 \to 15$	2

TABLE 5.4 – Chemins réels correspondant aux chemins locaux du tableau 5.3

5.2.4.2 Graphe Relevant

A partir du graphe abstrait et d'une requête donnée, le "Relevant Graph" abstrait (the abstract relevant graph) est construit par transformation des chemins locaux physiques en des chemins réels dépendant du temps. Cette transformation, suivie d'une étape de filtrage pour ne conserver que les chemins "pertinents", permet de réduire le temps de calcul final. Le "Relevant Graph" abstrait est formellement défini comme suit :

Définition 5.2.8

(Graphe Relevant abstrait) *Soit un graphe abstrait $G_a = (V_a, E_a)$ associé au graphe de transfert $G_t = (C, T_r)$, $C_k = (V_k, E_k, M_k, T_k, TV_k)$, et étant une requête utilisateur avec un noeud source s, un noeud destination d et un temps du départ t_o, le "Relevant Graph" abstrait est le graphe $G_{ca} = (V_{ca}, E_{ca})$ où :*

$V_{ca} = V_a \bigcup \{s, d\}$

$E_{ca} = E_a(t) \bigcup P^{*g}_{s.d}(t_o) \bigcup P^{*g}_{s.-}(t_o) \bigcup P^{*g}_{+.t}(t)\ \forall t \in T_k, \forall C_k \in C$

Où $E_a(t)$ est l'ensemble des plus courts chemins réels obtenus à partir des chemins locaux physiques pour tout temps t du départ possible.

Le "Relevant Graph" abstrait du graphe abstrait de la figure 4.1 est décrit par la figure 5.5.

La transformation des chemins locaux physiques en chemins dépendant du temps consiste à instancier pour chaque arête d'un chemin physique un travel par un des travels de la liste des travels accessibles. Chaque chemin physique génère au plus un chemin dépendant du temps. Après avoir transformé tous les chemins physiques d'une arête, on procède au filtrage c'est à dire à supprimer les chemins qui ne peuvent dans aucun cas contribuer à une solution finale. Ce filtrage est basé sur la notion de dominance 5.2.2.

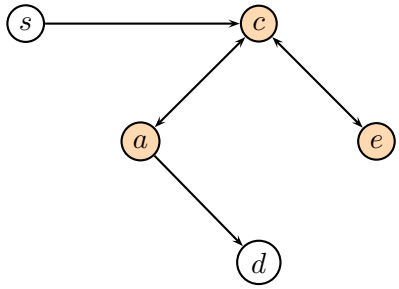

FIGURE 5.5 – Graphe Relevant abstrait de la figure 4.1

Algorithme 17 : Pseudo-code de l'approche Hybride

```
1 G_a ← CreateAbstractGraph(G_t);
2 répéter
3     G_ca ← GraphRelevantAbstrait(A_g,s,d,t);
4     Filtering(G_ca);
5     ShortestPath(G_ca,s,d,t);
6 jusqu'à Request(s,d,t) ;
```

5.2.4.3 Description de l'algorithme hybride

Le pseudo-code de l'approche hybride est décrit par l'algorithme 17. Il est quasi similaire à l'approche "Relevant Graph" (voir algorithme 12).

L'étape précalcul est remplacée ici par la construction du graphe abstrait (ligne 1). Ensuite, pour chaque nouvelle requête (ligne 2), le "Relevant Graph" abstrait est construit (ligne 3). Cette étape est suivie d'un filtrage qui permet d'illiminer les chemins dominés afin de simplifier le graphe, sans pour autant perdre des informations pertinentes (ligne 4). Enfin, le plus court chemin est calculé par la même fonction que dans l'approche "Relevant Graph" en utilisant la métaheuristique ACO ou l'algorithme de Dijkstra(ligne 5). La fonction $CreateAbstractGraph(G_t)$ crée le graphe abstrait qui est une sorte de base donnée des chemins locaux physiques. Nous avons implémenté cette fonction, grâce à une recherche tabou, qui depuis un noeud de transfert source parcourt le graphe, en évitant les chemins déjà considérés, tout en construisant des chemins locaux.

5.2.4.4 Comparaison théorique

Afin de valider l'approche hybride du point de vue théorique, nous devons montrer qu'elle est correcte par rapport à l'approche "Relevant Graph". En d'autres termes, nous devons monter que tout chemin obtenu par l'approche "Relevant Graph" (Dijkstra et ACO) peut être trouvé par l'approche hybride. Bien entendu, la complétude n'est possible que dans le cas où nous utilisons une méthode exacte pour ces deux approches. Pour valider donc la méthode théoriquement, nous supposons que nous appliquons un algorithme exact énumeratif pour la création du graphe abstrait et l'algorithme de Dijkstra pour le pré-calcul avec l'approche Relevant.

Avant d'établir la proposition principale, nous avons besoin d'introduire les lemmes

suivants :

Lemme 1

Si $p_{v_i.v_j}(t)$ est un plus court chemin local dans le "Relevant Graph" G_r au temps t, alors $p_{v_i.v_j}(t)$ est aussi un plus court chemin de G_{ca}.

Preuve 1

La preuve est triviale. C'est un simple raisonnement par l'absurde.

Lemme 2

Si $p_{v_1.v_k}$ est un plus court chemin non local dans le "Relevant Graph" G_r, alors le même chemin peut être trouvé dans le "Relevant Graph" abstrait G_{ca}.

Preuve 2

Un chemin non local possède au moins un noeud de transfert en dehors des noeuds source et destination. Par conséquent, si $p_{v_1.v_k}$ est un plus court chemin non local dans G_r, alors il peut être décomposé en une succession de sous chemins locaux comme suit : $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{(k-1)} \rightarrow v_k$ avec :

- *$v_j \in TV$ $1 \leq j \leq k$, et*
- *$v_j \rightarrow v_{j+1}$ est un chemin local $1 \leq j \leq (k-1)$*

D'après la propriété de Bellmann [27], nous pouvons affirmer que $v_j \rightarrow v_{j+1}$ $1 \leq j \leq (k-1)$ est un plus court chemin local. En plus, $v_j \rightarrow v_{j+1}$ est un plus court chemin local dans G_r, selon la lemme 1.

Finalement, $v_1 \rightarrow v_k$ peut être obtenu dans G_{ca}, puisqu'il est composé d'une succession de sous-chemins qui eux mêmes peuvent être obtenus dans G_{ca}

Proposition 5.2.1

Soit $G = (V, E, M, T)$ un graphe de transfert, G_r son "Relevant Graph" et G_{ca} son "Relevant Graph" abstrait. Si $p_{v_i.v_j}(t_o)$ est un plus court chemin de v_i à v_j avec un temps du départ t_o dans G_r, alors $p_{v_i.v_j}(t_o)$ est un plus court chemin dans G_{ca}.

Preuve 3

Afin de montrer que $p_{v_i.v_j}(t)$ est le plus court chemin dans G_r à un temps t alors il est le plus court chemin dans G_{ca} à l'instant t, nous devons distinguer deux cas :

Cas 1 *: Si $p_{v_i.v_j}(t_1)$ est un plus court chemin local dans G_r à t alors il y est aussi dans G_{ca} selon le Lemme 1.*

Cas 2 *: Si $p_{v_i.v_j}$ est un plus court chemin non local dans G_r à t alors il y est aussi dans G_{ca} selon le Lemme 2.*

5.2.4.5 Implémentation et analyse des résultats

Dans cette section, une implémentation de l'approche hybride est proposée. En outre, plusieurs tests permettant de sélectionner les paramètres sont analysés. Enfin, les tests de l'algorithme sur différentes instances du problème sont réalisés afin de comparer cette approche hybride avec les deux algorithmes présentés dans les sections précédentes.

Utilisation de ACO pour le graphe abstrait : La construction du graphe abstrait nécessite le calcul des chemins locaux entre les sommets de chaque composante. Une méthode exacte n'est pas adaptée dans ce cas car le nombre de noeuds de transfert dans un graphe est important et la génération de tous les chemins locaux nécessitera un temps de calcul important. Ce calcul peut être fait, même si l'incomplétude de telles solutions risque d'influencer la qualité du resultat final.

Nous proposons une implémentation de la méthode Hybride basée sur la métaheuristique ACO comme méthode de recherche Tabou pour la construction du graphe abstrait. La méthode ACO admet plusieurs paramètres : les valeurs initiales de phéromone, la fréquence de mise à jour ou le taux d'évaporation. Cependant, parmi tous ces paramètres, le nombre de fourmis et le nombre de cycles sont les plus critiques. Le tableau 5.5 montre les performances de l'algorithme ACO pour la construction des chemins locaux de l'instance (1000,3000) en faisant varier le nombre de fourmis et le nombre de cycles.

- Après analyse de ce tableau, nous concluons que dans un contexte séquentiel (pas de fourmis en parallèle) le nombre de chemins trouvés ainsi que le temps CPU augmentent de façon proportionnelle au produit du nombre de cycles et du nombre de fourmis.

- Le tableau 5.5 montre que la variation du nombre d'itérations allant de 100 (10*10) à 3200 (80*40), a fait augmenter considérablement le temps du calcul (de 13.4 à 107.9 sec) et que cette variation n'a pas beaucoup influencé le resultat de l'éxécution (une différence de 50 chemins uniquement). Ce qui nous a conduit dans le reste des tests, à fixer le nombre de cycle à 20 et le nombre des fourmis à 20 aussi.

Comparaison des différentes approches : Le tableau 5.6 présente d'un côté, les performances de précalculs avec les deux méthodes du "Relevant Graph", à savoir Dijkstra et ACO, et d'un autre côté les performances de la construction du graphe abstrait par l'approche hybride. Les résultats de ce tableau sont donnés en termes du temps de calcul et d'espace mémoire. Ces valeurs ont été calculées sur 100 requêtes indépendantes pour quinze instances du problème, en faisant varier chaque fois le nombre de noeuds, le nombre d'arêtes et le nombre de modes.

- Du point de vue temps CPU, l'approche "Relevant Graph" basée sur l'algorithme de Dijkstra donne les meilleurs résultats dans la quasi totalité des cas , bien que toutefois que les performances de l'approche hybride sont comparables.

Nb Cycle	**Nb Fourmis**	**Nb Chemins locaux**	**CPU Temps (s)**
10	10	$3.29 \cdot 10^4$	13.4
10	20	$3.32 \cdot 10^4$	18.2
10	30	$3.32 \cdot 10^4$	22.7
10	40	$3.33 \cdot 10^4$	26.0
20	10	$3.32 \cdot 10^4$	18.1
20	20	$3.33 \cdot 10^4$	26.0
20	30	$3.33 \cdot 10^4$	32.0
20	40	$3.33 \cdot 10^4$	38.2
40	10	$3.33 \cdot 10^4$	26.0
40	20	$3.33 \cdot 10^4$	38.3
40	30	$3.33 \cdot 10^4$	57.8
40	40	$3.33 \cdot 10^4$	61.2
80	10	$3.33 \cdot 10^4$	39.2
80	20	$3.33 \cdot 10^4$	60.7
80	30	$3.34 \cdot 10^4$	82.6
80	40	$3.34 \cdot 10^4$	107.9

TABLE 5.5 – Performance de ACO pour la recherche des chemins locaux : instance (1000,3000,3)

Instances			**Temps CPU(s)**			**Espace Mémoire(MB)**		
$\|V\|$	$\|E\|$	$\|M\|$	Relevant		Hybride	Relevant		Hybride
			Dijkstra	ACO		Dijkstra	ACO	
100	300	3	**1.2**	2.2	1.5	**20**	30	40
500	1000	3	**2.7**	6.7	**2.7**	**40**	46	56
1000	3000	3	**25.0**	80.3	26.0	170	68	**67**
2000	6000	3	**100.0**	1100.0	130.0	510	**104**	107
4000	15000	3	−−	−−	**642.0**	−−	−−	**170**
100	300	4	**1.1**	2.1	1.4	**20**	22	31
500	1000	4	**2.5**	5.3	2.6	**39**	**39**	47
1000	3000	4	**23.3**	77.2	25.4	165	60	**55**
2000	6000	4	**98.1**	1030.0	122.6	506	**96**	101
4000	15000	4	−−	−−	**640.2**	−−	−−	**166**
100	300	5	**0.9**	2.1	1.4	**18**	20	30
500	1000	5	2.3	4.9	**2.2**	38	**32**	45
1000	3000	5	**22.8**	76.1	24.6	162	55	**52**
2000	6000	5	**97.6**	1022.2	120.2	501	**89**	98
4000	15000	5	−−	−−	**636.2**	−−	−−	**160**

TABLE 5.6 – Comparaison entre Relevant Graph et approche hybride.

– Du point de vue espace mémoire, l'approche "Relevant Graph" basée sur ACO et l'approche hybride semblent fournir les meilleurs résultats pour les grandes instances.

– Nous pouvons conclure ici que l'approche hybride présente un bon compromis temps CPU et espace mémoire.

Aprés avoir montré les performances de la méthode pour la création de la base de donnée (Le pré-calcul pour la méthode Relevant et le graphe abstrait pour l'hybride) nous nous intéressons ici aux performances de la dernière étape de calcul à savoir l'application de Dijkstra sur la structure finale (Relevant ou Relevant abstrait).

Le tableau 5.7 présente les résultats de cette comparaison. Nous constatons clairement que la méthode Hybride est performante en terme temps CPU. En effet, elle est la plus rapide pour toutes les instances. De plus la méthode hybride permet de résoudre l'instance (4000, 15000, 3) qui n'a pas été résolue par les deux autres approches.

Instances			**Temps CPU(s)**		
			Relevant		**Hybride**
$\|V\|$	$\|E\|$	$\|M\|$	Pré-calcul Dijkstra	Pré-calcul ACO	Abstrait ACO
100	300	3	0.012	0.011	**0.009**
500	1000	3	0.054	0.056	**0.049**
1000	3000	3	0.248	0.239	**0.226**
2000	6000	3	1.053	1.054	**0.996**
4000	15000	3	−−	−−	**2.726**

TABLE 5.7 – Comparaison des deux approches

Analyse de la méthode hybride selon différents paramètres : La méthode Hybride est une méthode incomplète, puisque l'étape de la construction du "Relevant Graph" abstrait est réalisée par un algorithme stochastique. Nous voulons donc montrer l'influence de cette incomplétude sur la qualité de la solution finale. Nous analysons cet algorithme selon les paramètres suivants :

– le nombre de travels par arête ;
– la densité des instances traitées.

La figure 5.6 montre le comportement de l'algorithme Hybride sur 4 différentes instances du problème, en variant à chaque fois le nombre de travels par arête pour mesurer la qualité de la solution finale. Dans plus de 95% des cas, l'algorithme trouve la solution optimale et ce pour toutes les instances. On remarque aussi que l'augmentation du nombre de travels par arête n'a pas d'influence sur la qualité de la solution finale.

La figure 5.7 présente le comportement de l'algorithme par rapport à la densité des instances traitées. L'analyse est faite en comparant la qualité de la solution finale. Pour les 5 instances traitées, nous avons fixé le nombre de noeuds et augmenté le nombre d'arêtes ainsi que la densité du graphe. On remarque bien que pour toutes les instances traitées, cette augmentation dégrade la qualité de la solution finale. Ceci n'est pas surprenant car

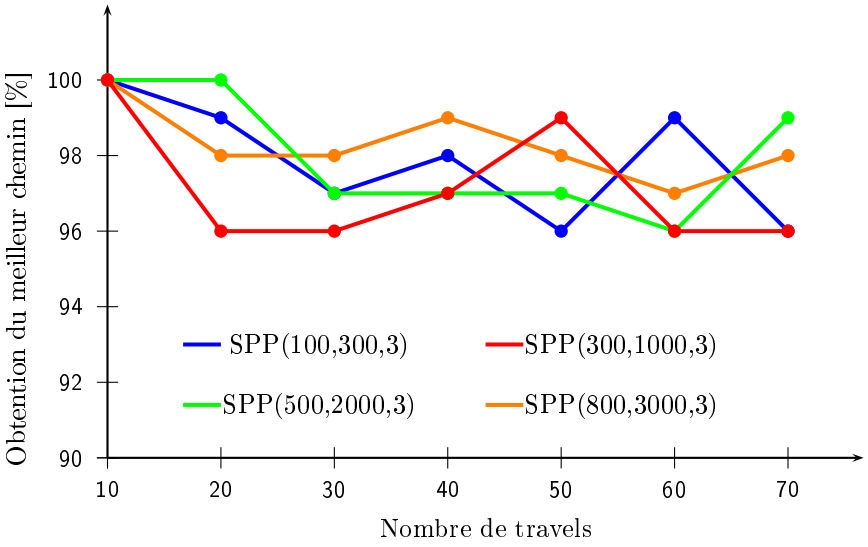

FIGURE 5.6 – Influence du nombre de travels

en augmentant la densité du graphe, on augmente de façon exponentielle le nombre de chemins locaux physiques possibles. En gardant le même nombre de cycles et le même nombre de fourmis, on diminue forcément le nombre de chemins trouvés et on détériore la qualité de la solution finale.

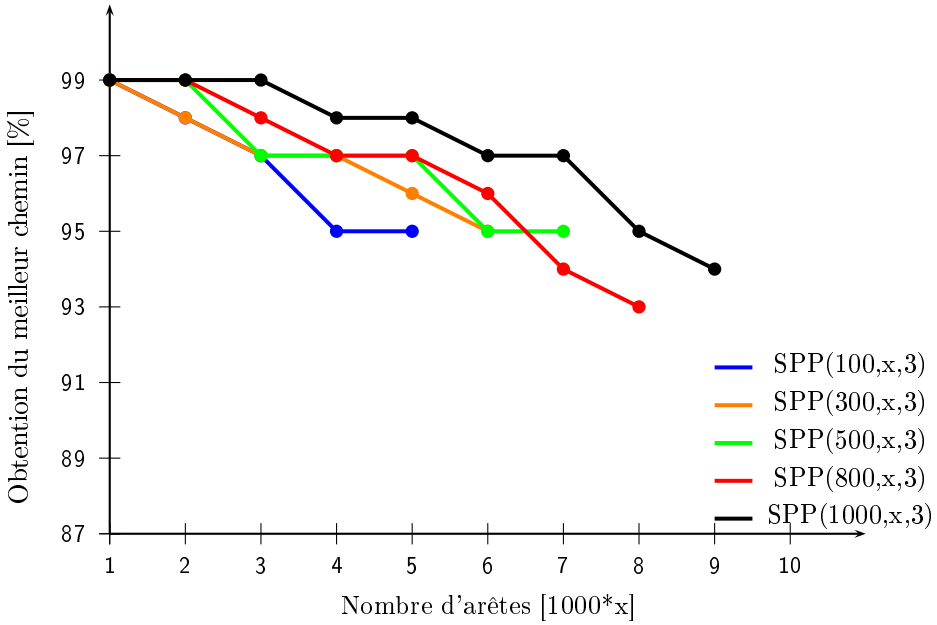

FIGURE 5.7 – Influence du nombre d'arêtes : le nombre de noeuds est fixe

Comparaison avec d'autres travaux : Le problème de transport multimodal dépendant du temps est peu étudié dans la littérature et l'absence de benchmarks ne facilite pas la comparaison de nos travaux à ceux déjà proposés. Pour autant qu'on sache, il y' a deux

Nb		Temps CPU(s)				Temps CPU normalisé (s)			
Noeuds	Arêtess	Ap.1 [17]	Ap.2 [110]	R.G [39]	Hybride	Ap.1 [17]	Ap.2 [110]	Relevant [39]	Hybride
50	200	0.33	0.54	0.001	0.001	0.33	0.71	**0.01**	**0.01**
100	300	0.53	0.81	0.012	0.009	0.53	1.07	0.16	**0.12**
200	600	1.27	−−	0.025	0.020	1.27	−−	0.33	**0.27**
500	1500	4.31	6.19	0.054	0.051	4.31	8.17	0.72	**0.68**
1000	3000	6.55	13.59	0.248	0.226	6.55	17.94	3.30	**3.01**
2000	6000	−−	−−	1.053	0.996	−−	−−	14.01	**13.25**
4000	15000	−−	−−	−−	2.726	−−	−−	−−	**36.26**

TABLE 5.8 – Comparaison avec les travaux existants

principaux travaux relatifs à notre problème [17, 109].
Dans [109] les auteurs modélisent le problème par des graphes multimodaux non-hiérarchiques et proposent plusieurs structures de données préalablement précalculées, afin d'optimiser le calcul. Bielli et al. dans [17] considèrent le problème de transport multimodal entre les villes. Le problème est géographiquement divisé et représenté de façon abstraite par un graphe hiérarchique où les villes sont représentées par des graphes, interconnectés entre eux par des liens directs. Les plus courts chemins à l'intérieur des sous-graphes sont préalablement précalculés, ce qui réduit l'espace de recherche.

Le tableau 5.8 présente une comparaison entre ces travaux, l'approche "Relevant Graph" et l'approche hybride sur différentes instances du problème. Les implémentations ont été effectuées sur trois machines différentes (Pentium II, Ultra-SPARC III, et T7300). Pour une comparaison réaliste, les temps de calcul ont été normalisés selon deux benchmarks [9] et [10]. Selon ces critères, les machines T7300 et ultra-SPARC III sont 13,3 et 1,32 fois plus rapides que le Pentium II, respectivement. Ces ratios sont appliqués aux temps de calcul pour obtenir le tableau 5.8.

Une simple lecture du tableau normalisé nous laisse observer que l'approche "Relevant Graph" et l'approche hybride sont toujours plus performantes que les deux autres méthodes citées précédemment. De façon plus globale, l'approche hybride présente de meilleures performances pour toutes les instances sélectionnées. En outre, contrairement aux trois autres travaux, l'approche hybride peut être appliquée à la plus grande des instances sélectionnées.

5.3 Approche Parallèle

5.3.1 Motivation

Pour pouvoir traiter des réseaux de très grande taille (100.000 Nœuds et plus) nous avons proposé une nouvelle approche originale appelée Hypergraphe de transfert. Ce modèle est une superposition des décompositions géographiques et des décompositions modales. Dans l'hypergraphe de transfert, nous distinguons deux niveaux d'abstraction. Le premier niveau d'abstraction correspond à la décomposition géographique où les nœuds du graphe représentent les régions (zone, ville, cartier, ...) et les arêtes représentent les connexions entre elles. Chaque région est représentée dans le deuxième niveau, par un

9. "Standard performance evaluation corporation (SPEC)" *http : //www.spec.org/*.
10. "Geekbench benchmark," *http : //www.primatelabs.ca/geekbench/*

graphe de transfert.
Pour résoudre le problème de plus court chemin dans l'hypergraphe de transfert, nous avons développé un algorithme basé sur le parcours parallèle de l'arbre de recherche. Cette nouvelle méthode nous a permis de résoudre des instances de grande taille . Nous avons pu en l'occurence résoudre un problème de 400.000 nœuds et 4.200.000 arêtes sur une machine parallèle alors qu'avec l'algorithme hybride la plus grande instance résolue était composée de 4000 noeuds.

5.3.2 Etat de l'art

Les algorithmes du plus court chemin constituent un élément clé dans les applications de transport. Le développement des systèmes de transport intelligents (Intelligent Transportation Systems ITS) et la nécessité qui en découle pour les gestion du trafic en temps réel et le trafic routier, nécessitent des algorithmes plus rapides intégrant la prise en charge de réseaux de transport grandeur réelle, s'alignant à l'échelle d'un pays ou d'un état.
Dans le but de prendre en compte cette notion de réseaux réels géographiquement distribués, le développement d'algorithmes de plus courts chemins parallèles a reçu un intérêt considérable de la communauté scientifique , cette dernière décennie. La plupart des travaux publiés dans ce domaine sont très souvent des résultats théoriques. Très peu d'implémentations ont été effectuées. De plus, la plupart des travaux ont porté sur la parallélisation d'algorithmes de plus courts chemin connus tels que Dijkstra ou Floyd [1].

- Dans [16], les auteurs ont parallélisé plusieurs algorithmes de type "label-correcting", sur une machine à mémoire partagée avec huit processeurs. Bien que certains de ces algorithmes présentent des accélérations superlinéaire accélérations pour des cas spécifiques, les propriétés de ces algorithmes et, leur scalabilité n'a pas été montrée. En outre, aucune expérience sur des des réseaux réels n'a été effectuée.

- Dans [55], Hribar et al. ont effectué une étude approfondie des algorithmes du plus court chemin parallèles, basée sur différents algorithmes de plus court chemin existants et différentes stratégies de décomposition. Les auteurs ont principalement discuté l'impact des différentes techniques de décomposition sur le coût des communications induites.

- Dans un article plus récent [101], Tang et al. ont proposé une parallélisation de l'algorithme de Dijkstra basée sur le partitionnement des graphes. Cet algorithme a été mis en oeuvre sur un cluster composé de quatre serveurs dual core interconnectés par un réseau Gigabit Ethernet. Les résultats obtenus montrent une bonne accélération sur trois réseaux de transports routiers. Cependant le plus grand réseau considéré est composé uniquement de 92792 sommets et de 26439 arêtes.

5.3.3 L'algorithme parallèle PMTDSPP

5.3.3.1 Présentation intuitive :

Nous proposons ici un nouvel algorithme PMTDSPP (Parallel MTDSPP) pour le problème du plus court chemin dans un réseau de transport multimodal et dépendant du temps. Cet algorithme est basé sur la décomposition géographique des réseaux de transport. Notre algorithme est très intuitif et part du principe simple qu'un voyageur qui veut aller d'un point source à une destination cherchera toujours à quitter la region où se trouve la source pour réjoindre la région où se trouve la destination. De même il cherchera à quitter le mode du réseau où se trouve la source pour rejoindre le mode du réseau où se trouve la destination dans le cas ou le deux noeuds ne font pas partie du même mode.

L'algorithme PMTDSPP reproduit ce comportement. En effet on part du noeud source pour réjoindre, la destination si elle existe dans la même région. On essaie de rejoindre les ponts pour changer de region. On rappele que dans notre cas on accepte le retour multiple sur une region.

De façon plus formelle le fonctionnement de cet algorithme revient à un parcours exhaustif d'un arbre de recherche. La figure 5.8 présente l'arbre de recherche exhaustif de l'hypergraphe présenté à la figure 4.3. La recherche commence au noeud source (A) et explore tous ses noeuds adjacents. Pour chaque noeud adjacent, un noeud fils est créé dans l'arbre. ce processus est réitéré pour tous les noeuds créés. La limite d'une branche de l'arbre est atteinte quand il n'y a plus de noeuds à visiter ou quand on arrive au noeud destination. Rappelons qu'on est dans un contexte dépendant du temps et que l'objectif est d'optimiser le temps d'arrivée, il faut donc à chaque étape du parcours choisir pour l'arête en cours, le travel qui optimise le temps d'arrivée. La recherche s'arrête quand plus aucune arête n'est accesible avec le temps courant.

Pour éviter l'exploration combinatoire de l'arbre complet de recherche, l'algorithme propose d'explorer l'arbre selon les principes suivants :

- **Concurrence** : Le parcours de l'arbre de recherche est associé à la construction dynamique de tâches devant être exécutées en parallèle par le noyau de l'algorithme. Chaque tâche exécutée génère elle même des tâches indépendantes et ce jusqu'à exploration de tout l'arbre. Les tâches créées sont exécutées indépendamment les unes des autres. Mais elles partagent toutefois l'unique objectif qui est celui de trouver le chemin le plus rapide vers une destination commune.

- **Compétitivité :** C'est l'aspect le plus important de notre algorithme. Le principe est le suivant : si un chemin partiel (la destination n'est pas encore atteinte) admet un coût supérieur au meilleur chemin complet connu, il doit être ignoré par l'algorithme puisqu'il ne contribuera en aucun cas à la solution optimale. Cela est vrai puisque le coût considéré (temps d'arrivée) est positivement additif. Cette propriété est garantie car les tâches collaborent via une mémoire partagée. En effet chaque fois qu'une tâche atteint la destination , elle partage cette information avec les autres tâches. De plus, chaque fois qu'une tâche doit créer de nouvelle tâches, elle vérifie d'abord le coût du meilleur chemin complet connu. La figure 5.9 montre un exemple de la simplification de l'arbre de recherche grâce à la concurrence et à la compétitivité.

5.3.3.2 Présentation formelle

Avant de présenter formellement l'algorithme PMTDSPP, nous introduisons d'abord formellement la notion de tâche intermodale. Une tache IMT (Inter Modal Task) consiste à résoudre le problème du plus court chemin entre une source s vers un ensemble de destinations $D = \{d_1, ...d_l\}$ au temps du départ t_0.

Définition 5.3.1

Tâche Inter Modale (IMT) *: Soit un Hypergraphe $G_h = (R, B)$, une region $r \in R$ et son transfert graphe $G_{t_r} = (C, T_r)$, un composant $C_i = (V_i, E_i, M_i, T_i, TV_i) \in C$. Si $s \in V_i$ est un noeud source, t_0 un temps du départ et $D = \{d_1, d_2, ..., d_l\} \subseteq V_i$ un ensemble de l noeuds destination, une Tache Inter Modale (IMT) est une activité de calcul de plus court chemin de s vers tout $d \in D$ à partir du temps t_0 dans le*

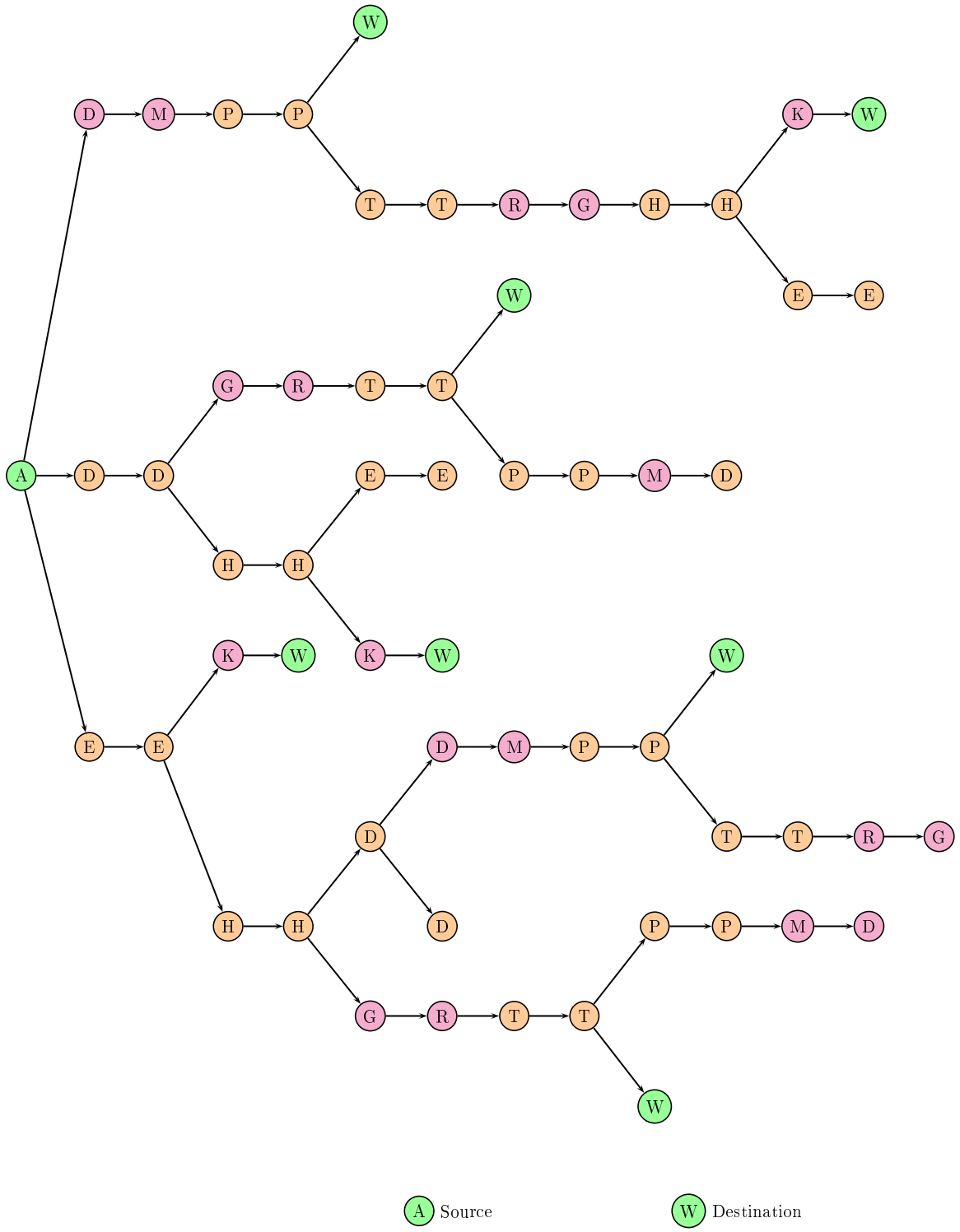

FIGURE 5.8 – L'arbre de recherche compet de l'exemple 4.3

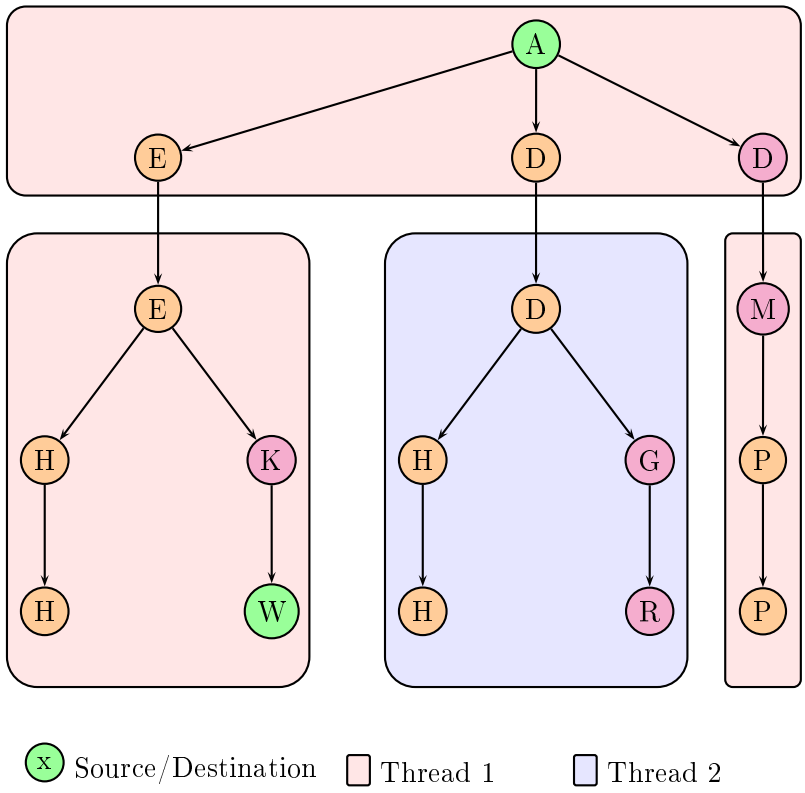

FIGURE 5.9 – L'arbre de recherche simplifié de l'exemple 4.3

composant C_i, notée IMT(R,s,t_0, D)

Pour calculer les IMT, nous avons considéré une version modifiée de l'algorithme one-to-many de Dijkstra. Une modification de ce dernier est nécessaire pour supporter la dépendance du temps. Cet algorithme a été introduit dans la section 5.2.3.1. Néanmoins, tout autre algorithme One-To-Many peut être également utilisé.

L'algorithme 18 décrit le pseudocode de PMTDSPP.
Cet algorithme est principalement divisé en trois étapes comme décrit ci-après :

1. Etape 1 : Il s'agit de l'étape d'initialisation dans laquelle l'algorithme identifie le composant où se trouve le noeud source ainsi qu'un ensemble de noeuds destination constitué par tous les noeuds de transfert ainsi que les ponts de ce composant et la destination globale (ligne 4). Enfin une première tâche (IMT) est créée dans le composant depuis la source vers l'ensemble des destinations créees au temps de départ t_0. Cette tâche est ajoutée à une pile partagée par les exécuteurs.

2. Etape 2 : Cette étape consiste à faire démarrer les éxecuteurs. Chaque exécuteur cherche dans la pile de façon itérative une tâche et ce tant que la pile n'est pas vide.

Algorithme 18 : Pseudo-code de l'algorithme PMTDSPP

```
1  Initialisation();
2  meilleurCoût ← ∞ ;
3  g ← composantDeNoeud(s);
4  destinations ← noeudsTransfert(g) ∪ noeudsPont(g) ∪ d;
5  tache ← IMT(g, s, destinations, t0);
6  pile ← pile ∪ tache;
7  pour chaque Executeur e faire
8      e.demarre(pile, d)
9  fin
10 tant que un exécuteur tourne faire
11     pause();
12 fin
```

3. Etape 3 : Cette étape consiste à tester périodiquement la condition d'arrêt, qui est dans notre cas la fin d'exécution de tous les exécuteurs.

La fonction qui exécute une tâche est présentée par l'algorithme 19. L'éxécuteur commence par dépiler une tâche (ligne 3). Il fait appel ensuite à une fonction de calcul du plus court chemin de type OneToMany. La fonction retourne donc un ensemble de chemins de la source (source de la tâche) vers l'ensemble des destinations. Chaque chemin p est analysé comme suit : si p atteint la destination globale (destination de la requête a résoudre) alors la procédure de mise à jour de la solution globale est appelée, "solutionGlobale(p, meilleurCout)" (ligne 9). Si la solution courante (le chemin p) admet un coût meilleur que la solution globale connue, alors cette dernière est modifiée. Dans le cas où le chemin p n'arrive pas à la destination globale, et donc forcément à un noeud de transfert ou un pont, la fonction "valide()" est appelée. Cette fonction teste si le coût du chemin p (le temps d'arrivée dans notre cas) n'est pas supérieur à "meilleurCout()", auquel cas la tâche est teminée. Cette fonction teste aussi si on est déjà passé par ce noeud avec un coût meilleur, dans ce cas la tâche est aussi terminée. Sinon, l'exécuteur procède à la création d'une nouvelle tâche (ligne 12-15).

La compétitivité de cet algorithme vient du fait que plus tôt des solutions complètes sont obtenues, et plus vite l'arbre de recherche est élagué. En effet, plus on optimise le meilleur coût global connu, moins des tâches sont exécutées, sans risque d'influencer les résultats. Donc,vu l'importance de l'ordre d'exécutions des tâches et pour plus d'otimisation nous avons créé trois types de piles dans lesquelles les tâches sont empilées de la manière suivante :

- Pile P_0 avec une priorité 0 (la plus élevée) : contient les tâches qui s'exécutent dans un composant qui contient la destination globale.

- Pile P_1 avec une priorité moins élevée que P_0 : contient les tâches interdites dans P_0 mais qui s'exécutent dans des regions contenant la destination globale.

- Pile P_2 : le reste des tâches ou les moins prioritaires y sont empilées
  .

Algorithme 19 : Pseudo-code de l'algorithme de l'exécuteur

```
1  tant que Vrai faire
2      si !pile.vide alors
3          t ← pile.dpiler();
4          chemins ← OneToMany(t.source, t.destinations, t.temps);
5          pour chaque p ∈ chemins faire
6              dernierNoeud ← p.DernierNoeud();
7              tempsArrive ← p.TempsArrive();
8              si dernierNoeud = d alors
9                  solutionGlobale(p, meilleurCout);
10             sinon
11                 si valide(tempsArrive, dernierNoeud) alors
12                     g ← composantDeNoeud(dernierNoeud);
13                     destinations ← noeudsTransfert(g) ∪ noeudsPont(g) ∪ d;
14                     tâche ← IMT(g, s, destinations, t0);
15                     pile ← pile ∪ tche;
16                 fin
17             fin
18         fin
19     fin
20 fin
```

Les exécuteurs, commencent par chercher une tâche dans la pile la plus prioritaire, à savoir la pile P_0, puisque les tâches de cette pile ont plus de chance de trouver une solution que les autres tâches. Si la pile P_0 est vide alors l'exécuteur examine la pile P_1, puisque les tâches de celle-ci sont susceptibles de créer des tâches de type P_0. Dans le cas négatif, l'exécuteur cherche alors dans la pile la moins prioritaire. L'ajout des nouvelles tâches respecte aussi la periorité des piles.

5.3.4 Propriétés théoriques de PMTDSPP

Proposition 5.3.1

L'algorithme PMTDSPP termine.

Preuve 4

Évident puisque dans le pire des cas, l'algorithme explore un arbre complet et fini de tâches, chacune correspondant à un calcul de plus court chemin de type One-To-Many.

Proposition 5.3.2

L'algorithme PMTDSPP est correct.

Preuve 5

La preuve est évidente si l'algorithme n'est pas optimisé, ce qui signifie que tout l'arbre est exploré. Mais dans notre cas, nous devons montrer que la réduction de certains

nœuds de l'arbre afin d'optimiser l'algorithme n'a pas d'influence sur sa correction.

Pour montrer cette proposition, nous devons différencier deux cas. En effet, l'algorithme met fin à une tâche (ne crée pas une successeur) dans deux cas possible : le coût de sous chemin courant dépasse le meilleur coût de la solution globale, ou le coût du noeud courant.

- ***cas 1 :*** *On suppose que l'algorithme a interrompu une tâche sur le noeud courant n d'un sous-chemin $p_{s.n}$, car son coût c est supérieur au meilleur coût connu $c^{'}$. On propose une démonstration, par contradiction. que la coupe effectuée n'influence pas le resultat final.*
  Supposons que si l'agorithme n'a pas réalisé de cut, le sous-chemin p aurait pu atteindre la destination finale d, et que ce chemin $p_{s.n.d}$ soit la solution optimale du problème. Le coût de $p_{s.n.d} = c + c^{''}$ avec $c^{''} > 0$ est le coût du chemin $p_{n.d}$ (le sous chemin que l'algorithme aura ajouté à p), comme $c > c^{'}$ alors $c + c^{''} > c^{'}$ et donc contradiction puis le coût du chemin $p_{s.n.d} = c + c^{''}$ est supposé être le coût optimale.

- ***cas 2 :*** *On suppose que l'algorithme a fait un cut d'une tâche sur le noeud courant n d'un sous-chemin $p_{s.n}$, car son coût c est supérieur au coût $c^{'}$ d'un autre chemin $p^{'}_{s.n}$ déjà exécuté sur n. Pour cela, on suppose que si l'agorithme n'a pas réalisé de cut, le sous-chemin p aurait pu atteindre la destination finale d, et que ce chemin $p_{s.n.d}$ soitla solution optimale du problème. Si on remplace le sous-chemin $p_{s.d}$, par $p^{'}_{s.d}$ dans $p_{s.n.d}$ on obtiendra un chemin encore plus court, car $c^{'} < c$ et donc contradiction puisque $p_{s.n.d}$ est supposé être le chemin le plus court global.*

5.3.5 Implémentaion et analyse des résultats

5.3.5.1 Environnement de programmation

Nous avons implémenté notre algorithme parallèle dans le langage de programmation Java utilisant l'API de threads pour l'exécution des tâches simultanées. Les tests ont été effectués sur le cluster Roméo II de l'Université de Reims, Champagne-Ardenne. Le cluster est composé de six nœuds de quatre processeurs dual-core de 16 Go de RAM, un noeud de huit processeurs dual-core de 128 Gb de RAM, et un noeud de 16 processeurs dual-core des 64 Go de RAM. Les nœuds sont interconnectés via le réseau d'interconnexion quadriques, permettant une communication bidirectionnelle haute performance avec plus de 900 Mbits/s entre les nœuds.

5.3.5.2 Apport de l'algorithme parallèle

Avant tout et afin d'évaluer les performances et les propriétés de notre algorithme parallèle, nous avons testé ses performances sur trois réseaux de transport dépendant du temps. Le tableau 5.9 montre une comparaison entre les performances de l'algorithme parallèle et les deux algorithmes précédents, *serial1* (l'approche séquentielle Relevant) et

serial2 (l'approche séquentielle hybride), sur les instances sélectionnées. Le symbole "-" signifie que l'instance n'a pas été résolue faute d'espace mémoire. Les résultats représentés dans le tableau 5.9 sont issus des expériences effectuées sur un PC Intel Dual -Core 2,4 Ghz avec 4 Go de RAM. Notons que l'exemple de réseaux de 4000 nœuds est le plus grand des réseaux résolus par nos algorithmes séquentiels précédents.

Comme nous pouvons le voir dans le tableau 5.9, l'algorithme parallèle ne présente aucun avantage sur les petites instances du réseau de transport. En fait, l'approche parallèle nécessite plus de temps CPU pour résoudre ce cas, que les autres algorithmes séquentiels. La raison est que pour un petit problème, le surcoût induit par l'étape de synchronisation ne peut pas être couvert par le court laps de temps de calcul. Néanmoins, une légère amélioration en termes de performance peut être observée pour des réseaux de transport moyen de 2000 noeuds et 6000 arêtes. Les deux dernières lignes du tableau montrent clairement l'avantage de notre nouvelle approche. Le réseau de 4000 noeuds et 15000 arêtes est résolu en 1.26 sec par notre approche parallèle alors que l'algorithme *serial2* prend le double de ce temps et cette instance ne peut pas été résolue en utilisant l'algorithme *serial1*. La plus grande instance sélectionnée dans le tableau 5.9 est résolue en 3.85 sec par l'algorithme parallèle et ne peut être résolue par les autres algorithmes séquentiels. Ce premier test montre clairement l'avantage de notre approche parallèle.

Instance		**Temps CPU(s)**		
$\|V\|$	$\|E\|$	Serial1 [39]	Serial2 [7]	Parallel
1000	3000	0.248	0.226	0.320
2000	6000	1.053	0.996	0.825
4000	15000	–	2,72	1.296
16000	167000	–	–	3.85

TABLE 5.9 – Comparaison Parallèle-séquentielle

5.3.5.3 Propriétés de l'algorithme parallèle

Dans ce paragraphe, nous analysons les performances de l'algorithme parallèle en termes de différents paramètres topologiques des réseaux de transport qui pourraient influer sur sa performance. Ces paramètres sont le nombre de noeuds n, le nombre de noeuds frontière (ponts et noeuds de transfert), le nombre de modes, de la taille de la grille des temps (nombre de travel), et la densité du réseau. Les instances sélectionnées sont composées de 4 modes et de 100 travels par arêtes.

Influence de la densité des graphes La figure 5.10 montre le comportement de l'algorithme PMTDSPP en terme de temps CPU, pour trois réseaux de transport avec 2000, 4000 et 6000 noeuds respectivement, en faisant varier leur densité. On augmente le nombre d'arêtes tout en gardant constant le nombre de noeuds.

Nous observons que le temps CPU n'est pas sensible à l'augmentation de la densité des graphes. Ce résultat n'est pas surprenant, car avec le plus grand nombre d'arêtes

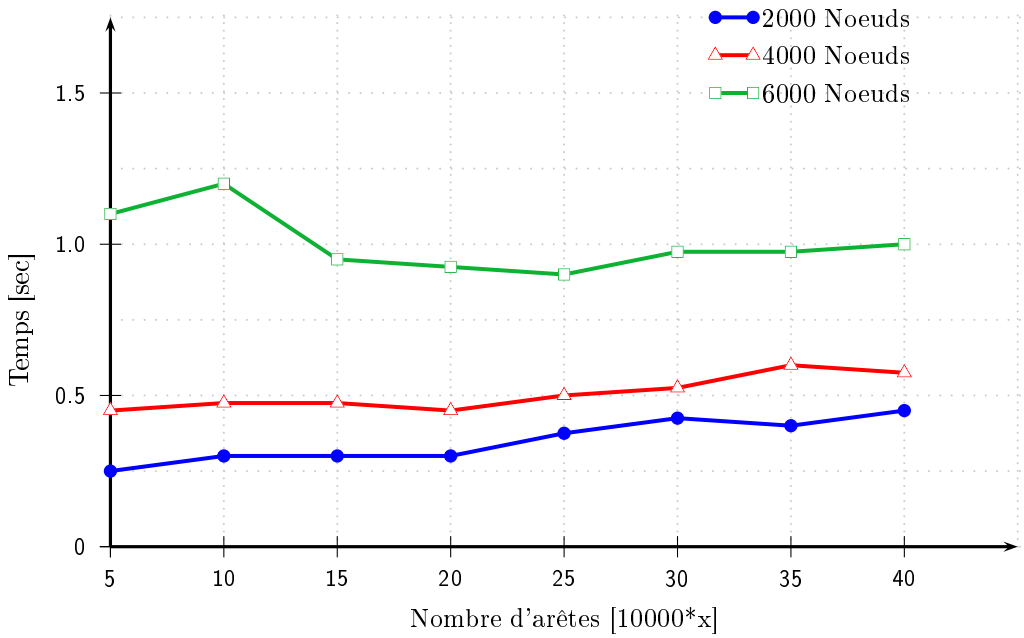

FIGURE 5.10 – Influence de la densité des graphes

considéré, le graphe reste sparse . Ces graphes sparses sont considérés toutefois comme très denses pour notre application.

Influence du nombre de noeuds de transfert La figure 5.11 montre les performances de l'algorithme de PMTDSPP pour trois réseaux en fixant le nombre de noeuds, la densité et en faisant varier le nombre de noeuds de transfert et de noeuds ponts.

De toute évidence, la complexité du problème augmente de façon linéaire par rapport à la proportion de noeuds de transfert. Cela est dû au fait que l'augmentation du nombre de noeuds frontière augmente le nombre de tâches générées, voir figure 5.12.

Influence du nombre de modes Le tableau 5.11 met en évidence une propriété intéressante de notre algorithme : la performance de l'algorithme s'améliore lorsque le nombre de modes augmente.

Nombre de Modes	**Nombre de processeurs**					
	1	2	4	8	16	32
2	56.13	28.55	16.58	15.31	15.30	15.30
4	18.38	8.59	4.80	4.72	4.71	4.70
8	12.51	5.66	2.88	1.67	1.63	1.63
10	10.47	4.69	2.35	1.10	0.97	0.96

TABLE 5.10 – Influence du nombre de modes

Ceci est éxpliqué par le fait qu'une tâche est exécutée dans les graphes mono-modaux. En augmentant le nombre de modes, et en gardant le même nombre de noeuds la taille des composants diminue et le calcul des tâches sera plus rapide. L'augmentation du nombre de modes permet un parallélisme à grain fin.

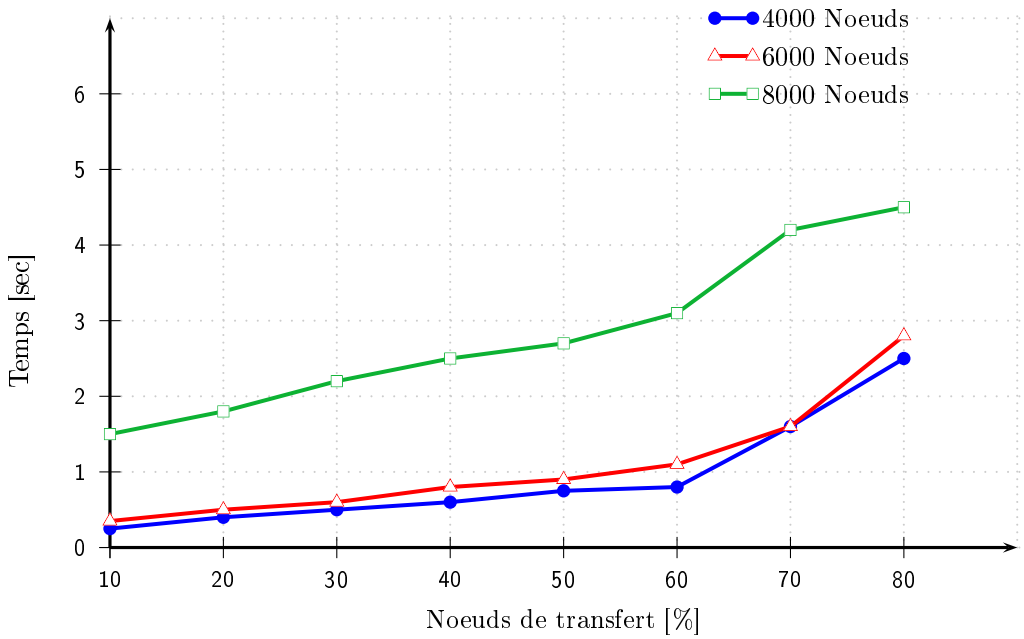

FIGURE 5.11 – Influence du nombre de noeuds de transfert

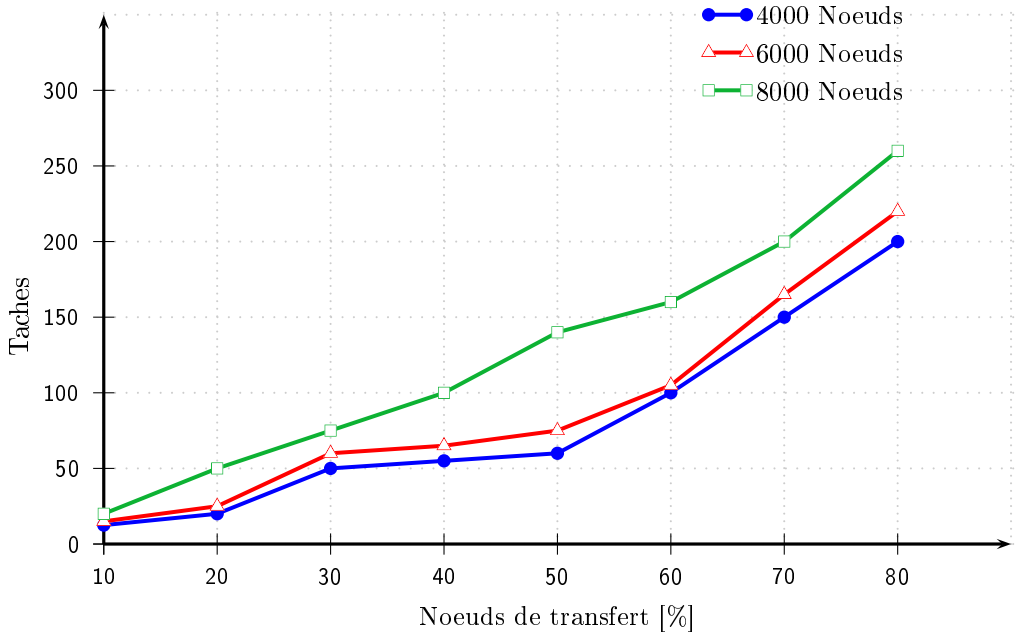

FIGURE 5.12 – Nombre de tâches créées par rapport au nombre de noeuds de transfert

Influence de la taille de la grille des temps Le tableau 5.11 résume les performances de notre algorithme en termes de taille des grilles de temps (nombre de travels). Nous avons constaté que notre algorithme est légèrement influencé par ce paramètre. C'est encourageant, car notre approche peut tenir compte de l'évolution des réseaux dans les grandes villes ou des grands pays.

Nombre de Travel	**Nombre de processeurs**					
	1	2	4	8	16	32
50	17.32	7.89	4.70	4.60	4.61	4.60
100	18.38	8.59	4.80	4.72	4.71	4.70
200	18.81	8.66	5.10	5.02	5.01	5.01
300	19.22	9.09	5.20	5.21	5.21	5.21

TABLE 5.11 – Influence de la taille de la grille des temps

Une étude sur la granularité La décomposition géographique des réseaux affecte le nombre de tâches générées. En effet, plus la granularité est élevée, plus le nombre de tâches générées est élevé et plus la complexité des tâches est réduite. Comme déjà décrit dans l'algorithme 19, une MIT est composée d'une étape de calcul et une étape de synchronisation. Bien sûr, l'augmentation du nombre de tâches implique que plusieurs étapes de calcul peuvent être exécutées simultanément et donc une augmentation de la synchronisation qui peut causer des problèms d'accès mémoire. Ce cas apparaît lorsque plusieurs exécuteurs veulent accéder à la mémoire simultanement, afin de mettre à jour le plus court chemin global ou le chemin local. La figure 5.13 montre les performances de l'algorithme vis-à-vis du grain de la décomposition. On considère quatre réseaux de transport avec un nombre de noeuds entre 2000 et 8000. En particulier, nous analysons le comportement de l'algorithme en termes de temps CPU en faisant varier le grain de décomposition de deux à seize régions. Au début, le temps CPU diminue considérablement. Ceci est justifié par le fait que, si nous aug mentons le grain nous augmentons aussi le nombre de tâches qui peuvent être exécutées simultanément. Mais augmenter le grain a une limite parceque les tâches générées doivent être synchronisées et le temps de synchronisation ne peut pas être couvert par le temps de calcul de taches de petite taille.

Efficacité, scalabilité et speedup Un algorithme parallèle doit être efficace et scalable. L'efficacité d'un algorithme parallèle est calculée de la manière suivante :

$$\frac{T_{seq}}{T_{par} \times N_p} \tag{5.5}$$

où T_{seq} est le temps du meilleur algorithme séquentiel.
et T_{par} est le temps mis par l'algorithme parallèle sur une machine cible à N_p.

Comme nous ne disposons pas de temps du meilleur algorithme séquentiel, nous considérons le temps mis par l'algorithme parallèle sur une machine mono-processeur.

– Un algorithme parallèle est efficace si son efficacité est égale à 1.

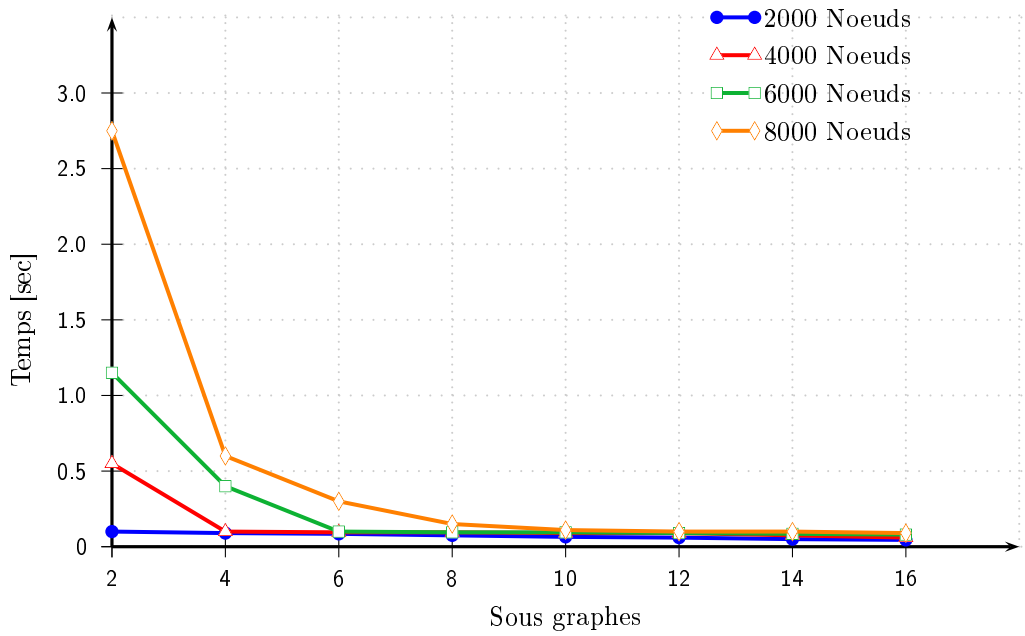

FIGURE 5.13 – Une étude sur la granularité

Instances		Nombre de processeurs					
Noeuds	Arêtes	1p	2p	4p	8p	16p	32p
16000	167000	3.85	2.05	1.20	1.05	1.05	1.05
63000	670000	18.38	8.59	4.80	4.72	4.71	4.70
144000	1500000	50.11	22.25	12.71	8.13	7.44	7.44
255000	2700000	107.65	51.66	26.75	15.16	14.81	14.80
400000	4200000	166.06	93.91	49.31	25.09	13.72	13.44

TABLE 5.12 – Performance de PMTDSPP en terme de temps CPU

- Un algorithme parallèle est scalable si le gain de parallélisme induit augmente de façon linéaire en fonction de la taille du problème.
- Le gain induit par le parallélisme peut aussi être calculé par l'accélération ou speedup qui se calcule comme le rapport entre le temps du meilleur algorithme séquentiel et le temps de l'algorithme parallèle.

La table 5.12 résume les temps d'exécution mis par l'algorithme PMTDSPP sur 5 instances différentes. Les tests sont tous effectués sur la machine Roméo. Nous avons utilisé au maximun 32 processeurs. Nous avons considéré 4 modes de transport et des grilles de temps de cardinalité 100. Les temps CPU diminuent avec l'augmentation du nombre de processeurs.

De plus la table 5.13 montre l'efficacité de PMTDSPP car nous observons que plus l'instance considérée est large et plus la scalabilité est importante.

La figure 5.14 montre l'accélération de PMTDSPP. Nous observons une accélération de 4 pour 4 processeurs et pour des plus grandes instances une accélération de 12 pour 16 processeurs.

Instances		Nombre de processeurs					
Noeuds	Arêtes	1p	2p	4p	8p	16p	32p
16000	167000	–	94%	80%	46%	23%	11%
63000	670000	–	107%	96%	49%	24%	12%
144000	1500000	–	113%	99%	77%	42%	21%
255000	2700000	–	104%	101%	89%	45%	23%
400000	4200000	–	88%	84%	83%	76%	39%

TABLE 5.13 – Efficacité de PMTDSPP

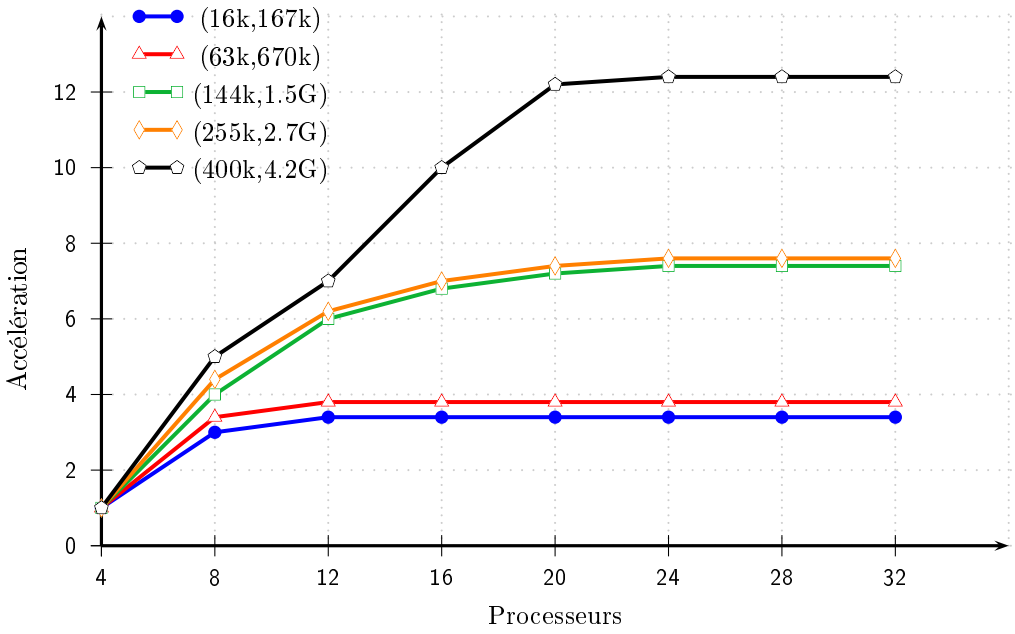

FIGURE 5.14 – Accélération de PMTDSPP

5.4 Cas réel : Le projet Carlink

Dans les sections précédentes nous avons testé nos approches sur des réseaux académiques. Dans cette section, nous allons montrer le déploiement de nos approches sur un réseau réel, une étude menée dans le cadre du projet Carlink [11].

Dans cette section, nous présentons d'abord la plate-forme Carlink ainsi que ses services et ses contraintes. Ensuite, nous montrons comment notre approche a été validée dans la plate-forme Carlink pour des scénarios réels.

5.4.1 Plateforme Carlink

Le projet *"Wireless Carlink Traffic Service Platform"* est conçu pour fournir une base pour un large éventail de problèmes comme le trafic commercial et gouvernemental, la sécurité et les services de transport (voir la figure 5.15). La plate-forme est une entité de réseau de communication ad-hoc sans fil avec une connectivité au réseau backbone via des stations de base. La plate-forme de communication elle-même est l'élément clé du projet

11. http ://carlink.lcc.uma.es/

Carlink. Mais les différents services créés au niveau de la plate-forme ont aussi un rôle important. Les trois services les plus importants sont : le trafic routier de services locaux météorologiques (RWS), le service de gestion du trafic (TMS) et le service de transport multimodal (MTS).

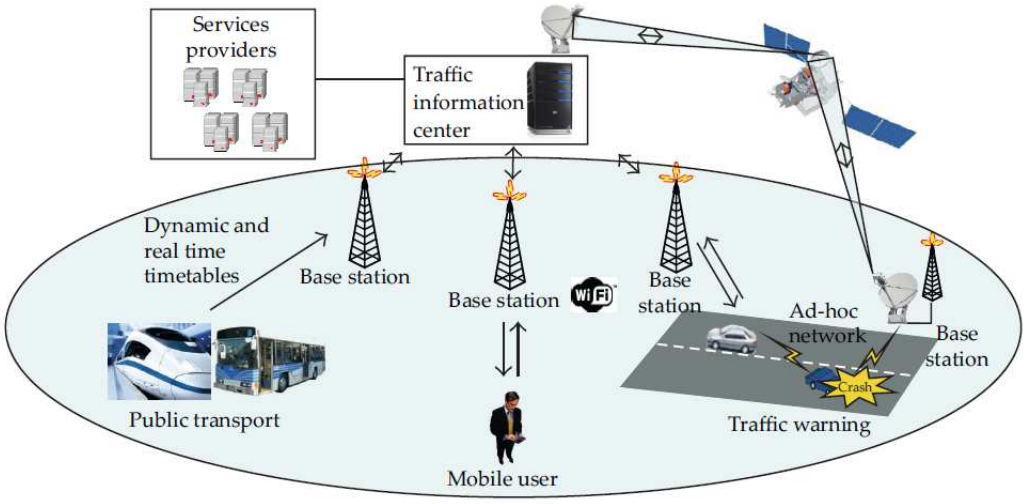

FIGURE 5.15 – Structure de la plate-forme Carlink

- Le RWS local est dérivé du FMI (Institut météorologique finlandais), un modèle de réseau météorologique routier [62] qui fournit automatiquement et sans intervention utilisateur, les opérations de sécurité routière mise en valeur par la collecte d'informations relatives à la sécurité des véhicules et la livraison des données consolidées.

- Le TMS fournit l'état du trafic en utilisant les informations de trafic statiques offerts par les autorités et les informations de trafic dynamique données par l'application de voiture flottante de Carlink. Cette application est composée de plusieurs voitures pour collecter des informations sur la densité de véhicules dans différents endroits du réseau routier et pour fournir des informations sur l'état du trafic.

- Enfin, le MTS calcule des itinéraires multimodaux en fonction d'une requête d'un voyage et un ensemble de contraintes spécifiées par l'utilisateur. Ce service est basé sur l'approche graphe de transfert.

La plateforme exploite les informations du RWS et TMS, ainsi que d'autres services, afin de générer des avertissements d'accidents ou d'incidents du trafic et des conditions météorologiques (voir la figure 5.15). Les MTS utilisent cette information pour calculer les meilleurs itinéraires en évitant les routes en mauvais état et les zones accidentelles qui risquent de réduire le temps de déplacement, ou encore recommander de changer le mode de transport si une route bloquée est détectée, par exemple, de garer la voiture près d'une gare et prendre le train à la place.

5.4.2 Environnement expérimental

La plate-forme sans fil du service trafic est divisée en trois parties différentes : Traffic Service Central Unit (TSCU), Traffic Service Base Station (TSBS), et Mobile End User (MEU). La plate-forme entière peut être vue comme une architecture d'échange de données entre la TSCU et la MEU, tandis que la TSB est vue comme des émetteurs-récepteurs de données bidirectionnels. Les entités MEUs communiquent entre elles de manière ad-hoc lorsqu'elles se croisent et peuvent également envoyer des données critiques directement à TSCU en utilisant une méthode GRPS à faible capacité de communication. Les noyaux de service sont des systèmes externes connectés à TSCU.

Le TSCU réalise les mises à jour des informations de trafic de la plate-forme depuis les noyaux de service. Le TSCU met à jour régulièrement les TSBs avec les informations du trafic les plus récentes liées à leur domaine, à travers le réseau fixe. Chaque TSBs diffuse l'information trafic de la plate-forme à chaque MEU qui passe près d'elle en utilisant une connexion sans fil ad-hoc. Enfin, les SMEs diffusent cette information lorsqu'ils se croisent (voir figure 5.16).

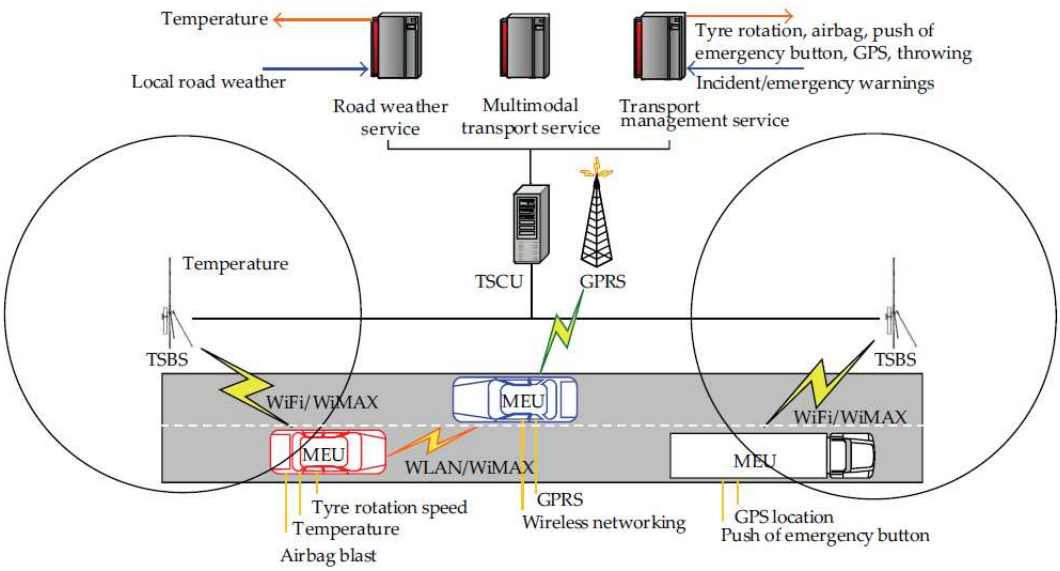

FIGURE 5.16 – Architecture de la plate-forme Carlink

Les MEU et TSBS sont équipés d'un système d'acquisition qui recueillent leurs propres informations (vitesse, données GPS, température de la route, etc) et communiquent régulièrement au TSCU. Ces données sont traitées par le TSCU, et si un événement critique est détecté dans une zone (par exemple, un accident, l'état des routes mauvaises), un message d'avertissement est livré en temps réel directement à chaque MEU connu qui peut être dans la zone d'avertissement en utilisant le GPRS ou une méthode de communication similaire.

L'avantage de cette architecture est que les utilisateurs MEU ainsi que les systèmes de diffusion TSBS, envoient régulièrement des informations sur l'état de la route aux TSCU. Ceci est particulièrement important lorsque un accident survient, parce que le TSCU est informé en temps réel et peut avertir les conducteurs dans les environs de la zone de

l'accident. En outre, puisque les voitures communiquent lorsqu'elles se croisent, le nombre de TSBS nécessaires dans les routes peuvent être considérablement réduits induisant ainsi une réduction du coût du système.

5.4.3 Résultats expérimentaux

La validation du système développé a été réalisée dans un scénario réel (voir figure 5.18). Nous avons considéré un scénario dont l'origine est la ville belge d'Arlon et la destination est la ville Luxembourg. Plusieurs milliers de navettes quotidiennes pratiquent ce voyage en voiture, train, bus, ou par une combinaison de ces modes de transport.

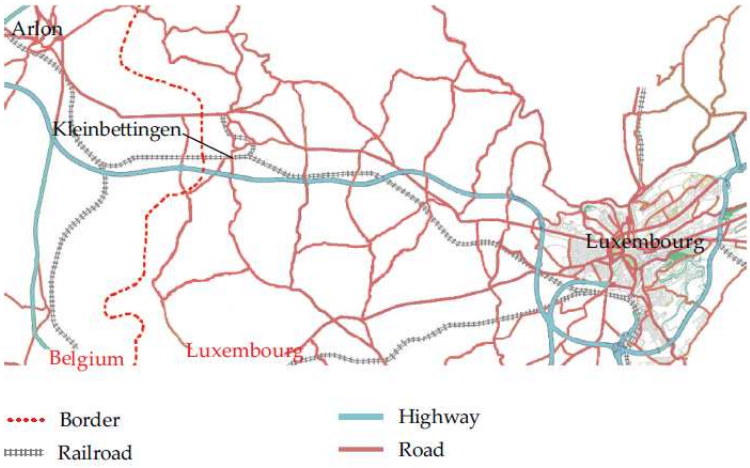

FIGURE 5.17 – Un scénario réel

FIGURE 5.18 – Captures d'écran de l'application pour les appareils mobiles. Sélection des préférences pour le voyage, Affichage de l'itinéraire.

5.4.4 Analyse des résultats

Un graphe de transfert pour représenter de façon abstraite ce scénario a été déployé dans la plate-forme Carlink et a été mis à disposition pour la MTS. Le graphe de transfert

est créé avec 73 nœuds, 49 points de transfert, et 226 arcs et contient trois composantes (train, voiture, bus). Nous considérons que la transition entre deux modes de transport se fait à pied et que son coût est représenté de façon implicite dans les coûts de transfert.

L'approche graphe de transfert est mise en œuvre dans le MTS de la plate-forme Carlink. Les requêtes des utilisateurs sont envoyées au MTS par l'intermédiaire d'une application cliente pour les appareils mobiles. Dans cette application, l'utilisateur peut sélectionner les préférences du voyageur (par exemple, les modes de transport, heure de départ, l'origine et la destination), envoyer la demande à la MTS, et obtenir le chemin optimal à partir de la plate-forme Carlink (figure 5.18).

Dans notre scénario, les préférences sont sélectionnées en optimisant le temps de déplacement pour aller d'Arlon à Luxembourg, en utilisant n'importe quel type de mode de transport et partant à 05h50. La première solution recommandée par le MTS consiste à aller directement en voiture via l'autoroute (tableau 5.14). Alors que nous nous déplaçons en voiture, le TMS a signalé à 06h00 une perturbation dans l'itinéraire actuel. Cette perturbation modifie les données de transport pour les transports routiers, les pré-calculs sont réeffectués pour les bus et les voitures, mais pas pour le train. Cette perturbation augmente le temps d'itinéraire pour la première solution, jusqu'à 80 minutes de retard. Automatiquement, le MTS recalcule un second itinéraire en tenant compte de cette perturbation et de la position GPS actuelle du voyageur. Cette information est renvoyée à l'utilisateur par l'application cliente. La deuxième solution devrait arriver 25 minutes avant la première et les différentes étapes de l'itinéraire sont les suivantes : parc à proximité de la gare la plus proche de Kleinbettingen, prendre le prochain train pour le Luxembourg, et enfin le bus jusqu'à la destination finale, comme indiqué dans le tableau 5.14.

Route	Mode de Transport	Temps	Etapes d'itinéraire (départ et arrivée) Arlon : Source	Highway E25/A4	Kleinbettingen parking	Kleinbettingen train station	Luxembourg Train station	Luxembourg bus station	Luxembourg 29 JFK
1	Voiture	40′	5 :50						6 :30
1 (Mis à jour)	Voiture	80′		6 :01					7 :21
2	voiture	9′		6 :01	6 :10				
	M. Pied	5′			6 :10	6 :15			
	Train	18′				6 :15	6 :33		
	M. Pied	5′					6 :33	6 :38	
	Bus	18′						6 :38	6 :56

TABLE 5.14 – Les itinéraires calculés pour le scénario de validation

5.4.5 Résultat de PMTDSPP sur le réseau carlink

Nous avons montré les performances de PMTDSPP sur des graphes académiques. Pour confirmer les bonnes propriétés de cet algorithme parallèle, nous montrons ici ses performances pour un cas réel. Nous avons considéré comme scénario le réseau de transport de la ville de Berlin, composé de 3122 noeuds, 4306 arêtes et 178 noeuds pont. Nous considérons pour ce réseau des décompositions en 2 et 4 régions 5.19.

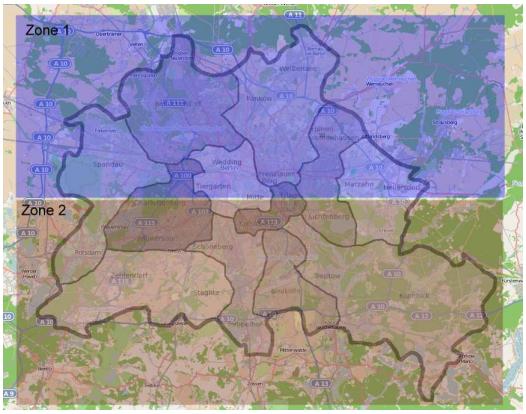

FIGURE 5.19 – Réseau de Berlin décomposé en 2 régions

La table 5.15 montre les performances de PMTDSPP en terme de temps CPU.

Instance			Nombre de processeurs				
Noeuds	Arêtes	Divisions	1p	2p	4p	8p	16p
3122	4206	2	2.37	1.19	0.59	0.42	0.57
3122	4206	4	1.69	0.87	0.52	0.42	0.49

TABLE 5.15 – Temps CPU(s) de PMTDSPP pour le réseau de Berlin

La table 5.16 montre l'efficacité de PMTDSPP pour le réseau de berlin.

Instance			Nombre de processeurs				
Noeuds	Arêtes	Divisions	1p	2p	4p	8p	16p
3122	4206	2	–	99%	100%	70%	25%
3122	4206	4	–	97%	81%	50%	21%

TABLE 5.16 – Efficacité de PMTDSPP pour le réseau de Berlin

Conclusion

Ce chapitre a résumé notre contribution dans une première partie de cette thèse dans laquelle nous avons simplifié quelque peu notre problème en ne considérant que l'aspect mono-objectif. Nous avons développé deux approches pour le résoudre. La première approche est séquentielle et elle est basée sur le modèle de graphe de transfert. La deuxième approche est parallèle et elle est basée sur l'hypergraphe de transfert. Nous avons validé ces approches sur des réseaux académiques et à l'aide de données réelles dans le cadre du projet (Carlink) du centre de recherche CRP Henri Tudor au Luxembourg.

Nous avons montré que ce problème est simple et peut être résolu de façon polynomiale par une simple version de l'algorithme de Dijkstra intégrant la notion de temps. Mais l'étape qui pose le plus de problèmes est l'étape de pré-traitement qui consiste à construire une base de donnée. Nous avons proposé d'une part deux versions différentes pour construire la base de données. La première utilise l'algortithme de Dijkstra très gourmant en espace mémoire et la deuxième fait appel à la matéheuristique ACO qui est plus gourmande en terme de temps CPU. D'autre part, nous avons proposé un algorithme hybride qui évite la construction de cette base de données qui n'est pas sans inconvénient, et qui présente un meilleur compromis temps-espace. Enfin, nous avons développé un algorithme parallèle ayant comme objectif la résolution de réseaux de grandes tailles.

Chapitre 6

Plus court chemin multi-objectif : Approches Séquentielle et Parallèle

6.1 Introduction

Dans le chapitre 5, nous avons plusieurs versions de l'algorithme MTDSPP pour résoudre le problème du plus court chemin multimodal dépendant du temps dans un contexte mono-objectif. L'unique coût à optimiser que nous avons considéré est le temps de parcours. Dans ce chapitre nous abordons le problème du plus court chemin multimodal dépendant du temps dans un contexte mono-objectif (MMTDSPP).
Nous avons déjà introduit dans le chapitre 6, différentes méthodes de résolution d'un problème d'otimisation multi-objectif de façon générale. Rappelons ici qu'il existe en fait trois grandes approches possibles :

- " Approches classiques ou apriori" : Ces approches consistent à trouver une fonction objectif qui résulterait de l'agrégation de tous les objectifs, agrégation ramenant le problème initial à un problème d'optimisation mono-objectif. Cette approche permet alors, dans le cas d'optimisation du plus court chemin d'utiliser les algorithmes de plus court chemins classiques bien connus.

- " Approches interactives " : Ces approches ont pour but de proposer des solutions intermédiaires à l'utilisateur qui, de façon interactive décide dans quelle direction il doit améliorer la solution pour s'approcher de la solution optimale. Ce processus est itérativement répété jusqu'à trouver une solution satisfaisante du point de vue de l'utilisateur ou tant que le nombre maximun de cycles autorisé n'est pas atteint.

- " Pareto optimalité ou approches apostériori" : Cette technique a pour but de calculer l'ensemble pareto le plus large possible. Elle n'exclue aucune solution satisfaisante du point de vue utilisateur au détriment d'un calcul combinatoire qui la rend non réalisable en pratique dans le cas de réseaux de grande taille.

Dans le cadre de cette thèse, nous développons une optimisation de l'ensemble pareto. Pour faire face au calcul combinatoire mentionné précédemment, nous distinguerons dans nos algorithmes les chemins physiques des chemins réels. Les chemins physiques ne tiennent pas compte des ensembles de travels ou des grilles de temps associés aux arêtes

du réseau. Les chemins réels correspondent à des instanciations particulières des chemins physiques en associant à chaque arête un travel figurant dans la grille des temps.

Cette transformation a déjà été abordée dans le chapitre 5 pour le problème mono-objectif. le problème est cependant beaucoup plus complexe ici. c'est pouquoi nous traitons cette transformation spécifique dans la section suivante.

6.2 Transformation d'un chemin physique

Cette fonction constitue la "brique de base" des algorithmes que nous développerons ultérieurement. Etant donné un chemin physique, cette fonction consiste à trouver des suites d'associations arc/travel parmi tous les "travels" possibles de chaque arc du chemin. Chaque suite constitue un chemin réel non dominé. La figure 6.1 représente un exemple d'un chemin physique composé de 3 arcs. Les chemins réels sont décrits par le tableau 6.1. Pour cet exemple, nous pouvons facilement constater que les deux chemins réels non dominés sont :

- $[\{(a \to b)(1 \to 2)\}, \{(b \to c)(2 \to 3)\}, \{(c \to d)(4 \to 5)\}] : t_1 = 5,\ c_1 = 8$
- $[\{(a \to b)(1 \to 3)\}, \{(b \to c)(3 \to 6)\}, \{(c \to d)(6 \to 9)\}] : t_2 = 9,\ c_2 = 7$

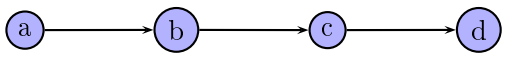

FIGURE 6.1 – Exemple d'un chemin chysique

Arc	Travels	Coûts
$a-b$	$(1 \to 2)$	$(2,4)$
	$(1 \to 3)$	$(3,2)$
$b-c$	$(2 \to 3)$	$(3,3)$
	$(3 \to 6)$	$(6,2)$
$c-d$	$(4 \to 5)$	$(5,1)$
	$(6 \to 9)$	$(9,3)$

TABLE 6.1 – Les travels du chemin physique donné par la figure 6.1

Pour calculer les chemins réels nous avons proposé deux approches :

- **L'approche exacte** : Cette approche s'appuie sur l'algorithme de Martins pour construire un graphe à partir des travels. Le principe est de relier deux travels appartenant à deux arcs successifs du chemin physique de telle façon que le temps de départ du deuxième travel soit supérieur au temps d'arrivée du premier travel. La construction de ce graphe ne prend en compte que les travels non dominés. Ce préfiltrage réduit l'espace du recherche et permet à un algorithme telque celui de

Martins, d'obtenir des résultats intéressants (voir l'exemple de la figure 6.2).

– **L'approche basée sur l'agrégation** : Il s'agit d'une méthode incomplète basée sur les fontions d'agrégation. Le principe est de générer au hasard des fonctions d'agrégation. Le nombre de fonctions à générer est un paramètre dépendant de la taille du problème. Chaque fonction produit un chemin optimal devant être comparé aux chemins déjà obtenus afin de ne mémoriser que les chemins non dominés.

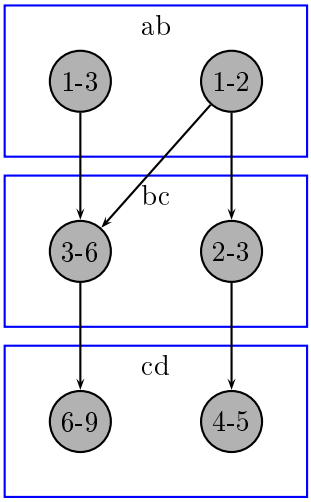

FIGURE 6.2 – Le Graphe des travels de l'exemple 6.1

Nous présentons dans la figure 6.3 le résultat d'exécution de ces deux algorithmes obtenus à partir d'un chemin physique de 10 arcs dans un contexte bi-objectif. Nous avons considéré le temps et la distance du chemin comme objectifs à optimiser. L'algorithme d'agrégation a été éxécuté avec 200 fonctions différentes générées aléatoirement. Bien évidemment, le front pareto obtenu par l'approche d'agrégation est incomplet. L'algorithme de Martins trouve bien toute les solutions, mais il est par contre moins rapide. Le tableau 6.2 montre bien que pour certains chemins physiques, l'algorithme de Martins peut s'avérer moins efficace.

Le tableau 6.2 met bien en évidence les limites de l'algorithme de Martins pour la transformation d'un chemin physique en chemin réel. D'une part, ce tableau compare l'algorithme de Martins et l'algorithme basé sur l'agrégation. D'autre part, il souligne le pourcentage de chemins trouvés par l'algorithme d'agrégation. Les résultats mentionnés en vert soulignent le fait que le temps d'éxécution est en-dessous d'une certaine limite (ici 0.1 seconde), au delà de laquelle, l'éxécution est considérée trop coûteuse et donc pas optimale (résultats mentionné en rouge). Sans perte de généralité, nous émettons l'hypothèse que le résultat est insuffisant si le nombre de chemins réels trouvés représente moins de 10% de l'ensemble de tous les chemins réels. Le symbole $\succ$ indique que le résultat ne peut être obtenu par l'algorithme de Martins au bout de 60 secondes.

Ce tableau montre aussi que l'algorithme d'agrégation est extrêmement rapide par raport à l'algorithme de Martins. Par contre, pour les problèmes à 3 et 4 objectifs, la qualité des solutions trouvées est insuffisante. Nous validons de façon expérimentale de cette fa-

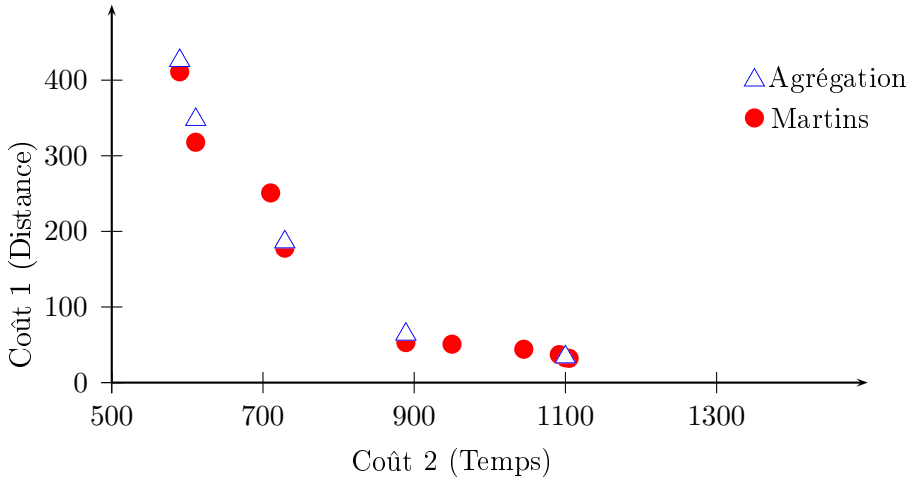

FIGURE 6.3 – Comparaison de deux fronts

Nb Coûts	Taille Chemin	Martins		Agrégation	
		Taille Front	CPU(s)	Taille Front	CPU(s)
2	5	11	0.002	4 - 36%	0.004
	10	10	0.002	6 - 60%	0.007
	20	4	0.003	2 - 50%	0.012
3	5	158	0.009	25 - 15%	0.005
	10	445	0.090	15 - 03%	0.009
	20	967	1.069	19 - 02%	0.016
4	5	1630	0.745	26 - 01%	0.004
	10	≻	≻	≻	≻
	20	≻	≻	≻	≻

TABLE 6.2 – Comparaison entre l'algorithme de Martins et l'algorithme d'agrégation pour le problème de transformation d'un chemin physique

çon, les performances de l'algorithme de Martins. Au delà de deux objectifs, l'algorithme d'agrégation semble s'imposer. Dans le but d'affiner les résultats obtenus par l'algorithme d'agrégation, il faut augmenter le nombre de fonctions générées tout en observant le temps d'exécution. L'idéal est de trouver un compromis entre le nombre de chemins trouvés et le temps d'exécution. Le tableau 6.3 présente le résultat de ce test en faisant varier le nombre de fonctions de 1000 à 5000.

6.3 L'algorithme MMTDSPP1 basé sur L'algorithme de Martins

6.3.1 Principe de l'algorithme de Martins dépendant du temps

L'algorithme de Martins dependant du temps est décrit par l'algorithme 20. Comme les arcs dépendent du temps, l'ajout d'un arc à un chemin n'est possible que s'il est associé à un travel. La recherche du plus court chemin porte donc sur les couples formés d' arcs ainsi que des travels valides. Ce qui se traduit dans cet algorithme tout simplement par l'ajout d'une boucle supplémentaire (ligne 12) qui parcourt tous les travels non dominés à partir du temps d'arrivée au noeud x. Ce temps est enregistré au préalable par la fonction

Nb Coûts	Taille Chemin	Nb Fonctions	CPU(s)	% Chemins
3	20	1000	61	04
		2000	123	05
		3000	185	06
		4000	246	06
		5000	308	07
4	5	1000	19	03
		2000	39	05
		3000	59	05
		4000	79	06
		5000	99	07
	10	1000	37	01
		2000	75	01
		3000	114	02
		4000	151	02
		5000	192	02
	20	1000	71	$\succ$
		2000	140	$\succ$
		3000	211	$\succ$
		4000	281	$\succ$
		5000	351	$\succ$

TABLE 6.3 – L'algorithme d'agrégation avec différents nombres de fonctions

Time() (ligne 18). Notons aussi que l'ajout d'un arc à un chemin est réalisé cette fois par la fontion de concaténation qui prend en paramètre l'arc à ajouter ainsi que le travel associé.

Algorithme 20 : L'algorithme Martins and Santos dépendant du temps

```
1  cnt ← 1;
2  h(cnt) ← s;
3  Z_s ← path(1);
4  X ← {cnt};
5  Time(cnt) ← t_0;
6  tant que X ≠ ∅ faire
7      x ← lexico(X);
8      X ← X\x;
9      i ← h(x);
10     pour tous les (i, j) ∈ E faire
11         time ← Time(x);
12         pour tous les tr ∈ NoDominatedTravel((i, j), time) faire
13             path(cnt) ← path(cnt) ⊕(i, j, tr);
14             si path(cnt) n'est pas dominé in Z_j alors
15                 cnt ← cnt + 1;
16                 ajouter (x, cnt) à ST ;
17                 h(cnt) ← j;
18                 Time(cnt) ← tr.arrival;
19                 X ← X ∪ cnt ;
20                 Z_j ← Z_j ∪ path(cnt);
21                 Supprimer tous les chemins dominés de Z_j;
22                 Supprimer les sous-arbres correspondants dans ST;
23             fin
24         fin
25     fin
26 fin
   Sorties : Z_t
```

6.3.2 Implémentation et analyse des résultats

Nous montrons ici les résultats d'implémentation de l'algorithme de Martins (Algorithme 20). Chaque instance du problème est testée sur un échantillon de 100 requêtes générées au hasard. Les graphes sont aussi générés au hasard. Nous avons analysé cet algorithme selon les trois paramètres suivants :

- le nombre de travels par arc ;
- le nombre d'objectifs ;
- la taille des instances.

Le tableau 6.4 présente les résultats obtenus pour 2, 3 et 4 objectifs et pour 6 instances de problèmes de 1000 à 32.000 noeuds. Pour chaque instance et pour un nombre fixe d'objectifs, nous avons fait varier le nombre de travels de 10 à 300. Le signe $\succ$ indique que l'exécution dépasse largement 60 secondes par requête, ce qui est dans notre cas inacceptable. La couleur verte dans le tableau indique un temps optimal et réaliste. La

couleur rouge souligne un résultat non réaliste.

Travels	(1k,2k)	(2k,4k)	(4k,8k)	(9k,16k)	(16k,33k)	(32k,66k)
Nombre de Coûts = 2						
10	0.05	0.11	0.3	0.7	1.86	4.42
90	0.47	1.13	2.68	7.02	18.36	64.25
210	1.19	2.72	6.74	9.43	33.65	≻
300	1.45	3.92	12.08	24.59	≻	≻
Nombre de Coûts = 3						
10	0.9	2.65	7.06	26.23	≻	≻
90	44.32	≻	≻	≻	≻	≻
210	≻	≻	≻	≻	≻	≻
300	≻	≻	≻	≻	≻	≻
Nombre de Coûts = 4						
10	10.27	36.53	≻	≻	≻	≻
90	≻	≻	≻	≻	≻	≻
210	≻	≻	≻	≻	≻	≻
300	≻	≻	≻	≻	≻	≻

TABLE 6.4 – Martins dépendant du temps

6.4 Algorithme génétique

Pour pallier aux limites des algorithmes exacts, nous nous sommes orientés vers les métaheuristiques. Nous avons principalement développé un algorithme génétique basé sur NSGA-II connu comme l'algorithme multi-objectif le plus performant de la famille d'algorithmes évolutionnaires. Le principe de cet algorithme est le suivant :

6.4.1 principe de l'algorithme NSGA-II

L'algorithme NSGA-II commence par créer une population initiale ou un ensemble de solutions réalisables de taille N. A chaque itération de l'algorithme, une population Q_t est créée à partir de la population parente P_t aussi de taille N. Les deux populations sont alors combinées pour former une population R_t , de taille 2N . Un classement en couches de l'ensemble pareto (*non dominated sorting*) est ensuite calculé entre les membres de R_t , à partir duquel on construit la nouvelle population P_{t+1}. Les fronts pareto sont successivement cumulés à P_{t+1} en commençant par le premier front pareto, puis le deuxième front et ainsi de suite. Comme la population R_t est beaucoup plus grande que la population P_{t+1} , tous les fronts ne pourront pas être cumulés. Si un front ne peut pas être ajouté en entier, une stratégie de niche, appelée peuplement (*crowding*), est employée pour sélectionner les individus à ajouter. Enfin, un tournoi prenant en compte le peuplement (*crowded tournament*) est utilisé pour générer Q_{t+1} à partir de P_{t+1}. Pour une description plus détaillée de l'algorithme NSGA-II, nous renvoyons le lecteur à l'article [60].

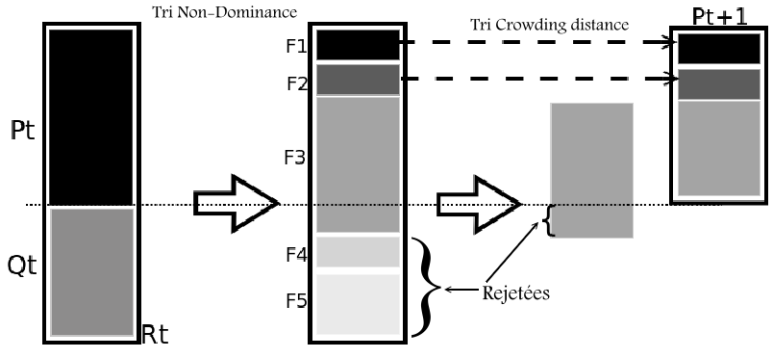

FIGURE 6.4 – Procédure principale de NSGA-II

6.4.2 L'algorithme MMTDSPP2 : Basé sur NSGA-II

Etant donné un graphe dépendant du temps $G(V, E, T)$, l'objectif de l'algorithme est de calculer un ensemble de chemins non dominés, depuis un noeud source $s \in V$ vers un noeud destination $d \in V$ à partir d'un temps $t \in T$.

6.4.2.1 Génération de la population initiale

En raison de la grande diversité des populations initiales, nous pouvons distinguer d'une part les P-métaheuristique qui sont naturellement des algorithmes de recherche orientées exploration, des S-métaheuristiques qui sont plutôt orientées exploitation. La détermination de la population initiale est souvent négligée dans la conception des métaheuristiques. Pourtant, cette étape joue un rôle crucial dans l'efficacité de l'algorithme et une attention particulière devrait lui être accordée.

Le critère principal à prendre en compte dans cette étape de génération de population initiale, est celui de la diversification. Si la population initiale n'est pas bien diversifiée, une convergence prématurée peut se produire pour toute P-métaheuristique. Par exemple, cela peut se produire si la population initiale est générée en utilisant une heuristique gloutonne ou une S-métaheuristique (recherche locale, recherche tabou) pour chaque solution de la population, voir [100] pour plus de détails.

Compte tenu de l'importance de cette étape de génération de population initiale, nous avons élaboré deux solutions pour la générer afin d'étudier son influence sur la qualité des solutions obtenues. La première méthode est basée sur un choix glouton alors que la deuxième est basée sur le principe de la métaheuristique ACO (Ant Colony Optimization). Nous montrerons, par la suite que la deuxième approche est meilleure du point de vue qualité de la population générée. Ceci contribue à l'obtention du meilleur front. L'objectif des deux approches est de construire N chemins réalisables respectant les contraintes citées précédemment citées.

Approche Gloutonne : Cette solution consiste à parcourir le graphe de départ de façon aléatoire. L'approche gloutonne est présentée par l'algorithme 21. La boucle principale de la fonction commence à la ligne 2, et s'arrête quand le nombre maximal de cycles est

atteint, ou quand la taille de l'ensemble des chemins trouvés est égale à n (taille de la population). Après l'étape d'initialisation (lignes 3-7), l'itération principale est décrite par la boucle "*tantque*", ligne 8-22. En commençant par le noeud source (ligne 4 dans l'étape d'initialisation), chaque étape de la boule consiste à choisir un noeud suivant au hasard parmi les noeuds candidats. La fonction $Succ[n, t, p]$ retourne l'ensemble des noeuds candidats en fonction du noeud courant n, du temps t ainsi que du chemin en cours p. Un noeud $n^{'}$ est considéré comme candidat, s'il est successeur de n et qu'il existe un arc $c(n, n^{'}, t^{'})$, tel que $t^{'} \geq t$ ($t^{'}$ est le temps du départ de l'arc), et que l'ajout de l'arc c au chemin p, ne viole aucune contrainte. Si le noeud D existe dans la liste de noeuds candidats (lignes 9-13) alors une copie du chemin courant est créée, le noeud D est ajouté avec la fonction $Ajout[copie, D]$, ensuite la fonction $Ajout(Result, copie)$ s'occupe de l'ajout du chemin complet *copie* à l'ensemble *Result*. Après avoir, éventuellement ajouté un chemin à l'ensemble *Result*, la fonction teste (ligne 13) si la taille de l'ensemble pareto a atteint la taille de la population, auquel cas, la fonction est interrompue. Dans un deuxième temps (lignes 15-21), la fonction sélectionne, parmi la liste des candidats, le noeud suivant. Cette sélection est basée sur un choix glouton. Si la liste de candidats est vide, b est alors mis à *vrai* indiquant la fin de la recherche pour le cycle en cours.

Algorithme 21 : Glouton-G.P.I(Graphe G, Requete R(S,D,t_0)

Sorties : Un ensemble de chemins réalisables(Result[])

```
1  i ← maxCycle ;
2  tant que i > 0 et taille[Result] < N faire
3      i ← i-1;
4      n ← S;
5      t ← t0;
6      b ← faux;
7      p ← CheminVide() ;
8      tant que b = faux faire
9          si D ∈ Succ[n, t, p] alors
10             copie ← Copie[p];
11             Ajout[copie, D];
12             Ajout[Result, copie];
13             si taille[Result] = N alors Interruption;
14         fin
15         si Succ[n, t, p] ≠ nil alors
16             n ← Glouton[Succ[n,t]];
17             t ← Glouton[n, t];
18             Ajout[p, n];
19         sinon
20             b ← vrai;
21         fin
22     fin
23 fin
```

Approche ACO : Dans le but de diversifier davantage la population initiale, nous utilisons la mataheuristique ACO [35]. Dans le cadre de notre application, il existe deux types de diversification à mettre en oeuvre pour garantir une bonne qualité de recherche :

- Diversification spatiale : Il s'agit des chemins physiques obtenus dans l'espace du graphe.

- Diversification temporelle : Il s'agit des chemins réels tenant compte de la dimension réelle du temps.

L'algorithme est constitué de deux fonctions principales. La première consiste à créer des chemins physiques en utilisant l'algorithme AS (Ant System), pour garantir la diversification spatiale. La deuxième fonction est une heuristique gloutonne qui garantit la diversification temporelle 6.2. L'algorithme considère le graphe associé au réseau, comme comme l'espace de travail des fourmis. A chaque itération, chaque fourmi (ou éventuellement plusieurs fourmis en parallèle) recherche un chemin physique reliant le noeud source s au noeud destinataire d, selon une loi de probabilité tenant compte de la quantité de phéromone. Cette fonction est décrite par l'algorithme 22.

Algorithme 22 : AntSystem-G.P.I(Graphe G, Noeud S,Noeud D)

```
   Sorties : Un ensemble de chemins réalisables(Result[])
 1 Initialisation() ;
 2 i ← maxCycle ;
 3 tant que i > 0 faire
 4     i ← i-1;
 5     b ← faux;
 6     p ← CheminPhysique();
 7     n ← S;
 8     tant que b=faux faire
 9         Candidats[] ← ArcSortant(n, p);
10         si Candidats = nil ou D ∈ p alors
11             b ← vrai;
12         sinon
13             pour chaque arc ∈ Candidats faire
14                 CalculProba(arc);
15             fin
16             suivant ← SelectArc(Candidats);
17             Ajout[p, suivant];
18         fin
19     fin
20     MiseAjourPheromone(p);
21     si D ∈ p alors
22         Ajout[Physique,p];
23     fin
24 fin
25 Result ← Transform(Physique, Requete R(S,D,t0));
```

La première étape consiste à initialiser une table de phéromone (Ligne 1). L'initialisation de la phéromone concerne les arcs du graphe et elle dépend de la taille et de la densité de ce graphe. Après cette étape d'initialisation, la boucle principale (Lignes 3-24) qui s'éxecute $maxCycle$ fois, où $maxCycle$ est un paramètre fixé au départ, consiste à créer un chemin physique à chaque itération. Ce chemin physique représente une possibilité de relier la source et la destination. Si à la fin de l'itération, la fourmi a trouvé un chemin exact et complet alors ce dernier est stocké une fois dans un ensemble . La boucle commence par sélectionner l'ensemble des arcs valides sortant du noeud courant, c'est à dire les arcs qui peuvent être ajoutés au chemin phyique en cours de construction. Ceci est réalisé par la fonction $ArcSortant(n, p)$ (Ligne 9). Si cet ensemble est vide ou si le noeud destinataire est atteint, la recherche est suspendue (Ligne 10). Dans les autres cas, la fonction $CalculProba(arc)$ calcule la probabilité associée à chaque arc, élement de l'ensemble candidat. La probabilité associée à chaque arc est inversement proportionnelle à la quantité de phéromone. Intuitivement, celà signifie que plus un arc a été traversé et moins il est intéressant de le traverser de nouveau. Sur base de cette probalité, le prochain arc du chemin est choisi de façon aléatoire (Ligne 16). Après la construction d'un chemin physique, que celui-ci soit complet ou pas, une mise à jour de la phéromone est effectuée (Ligne 20). Dans le cas où le chemin physique obtenu est complet, il est rajouté à l'ensemble des chemins obtenus. Lorque l'ensemble des chemins physiques est totalement construit, la deuxième étape de l'algorithme fait appel à la fonction $Transform()$ pour transformer les chemins physiques en chemins réels en tenant compte des notions de temps et de coûts, voir section 6.2.

6.4.2.2 Comparaison des deux approches

La figure 6.5 illustre l'exécution des deux algorithmes sur un graphe de 75 noeuds et 150 arcs. Pour chacune des deux heuristiques le nombre de cycles est fixé à 20. Les éléments (noeud ou arc) découverts uniquement par la fonction 22 sont identifiés par la couleur verte. la couleur rouge identifie les éléments découverts uniquement par l'algorithme 21 et la couleur bleu permet de représenter les éléments découverts par les deux algorithmes. La source et la destination sont représentés par la couleur magenta. Nous remarquons clairement que la recherche est mieux répartie sur l'ensemble du graphe avec l'algorithme 22 qu'avec l'algorithme 21.

Afin de tester l'efficacité de l'approche ACO, nous avons mis en oeuvre un algorithme exact, de parcours exhaustif de l'arbre de recherche pour calculer tous les chemins physiques entre une source et une destination données. Evidemment, l'objectif de cette comparaison est de mesurer la qualité des solutions obtenues et non les temps d'éxecution. L'objectif de la métaheuristique ACO est de répartir la recherche sur tout le graphe, pour garantir une diversification de la population initilale. Par conséquent, notre but n'est pas de trouver tous les chemins physiques, mais plutôt des chemins physiques uniformément répartis sur l'ensemble de l'espace de recherche. Le tableau 6.5 représente un exemple d'éxécution de l'algorithme ACO sur un réseau de 40 noeuds et 80 arcs. L'instance du problème a été exécutée par l'algorithme exact pour déterminer le nombre de chemins physiques existant entre la source et la destination ainsi que le nombre d'arcs couverts par chacun de ces chemins. Pour cette instance, l'algorithme exact détermine 166 chemins physiques recouvrant 47 arcs. Le tableau 6.5 présente pour différentes éxécutions faisant varier le nombre de cycle des 100 cycles à 800 cycles , le nombre de chemins physiques

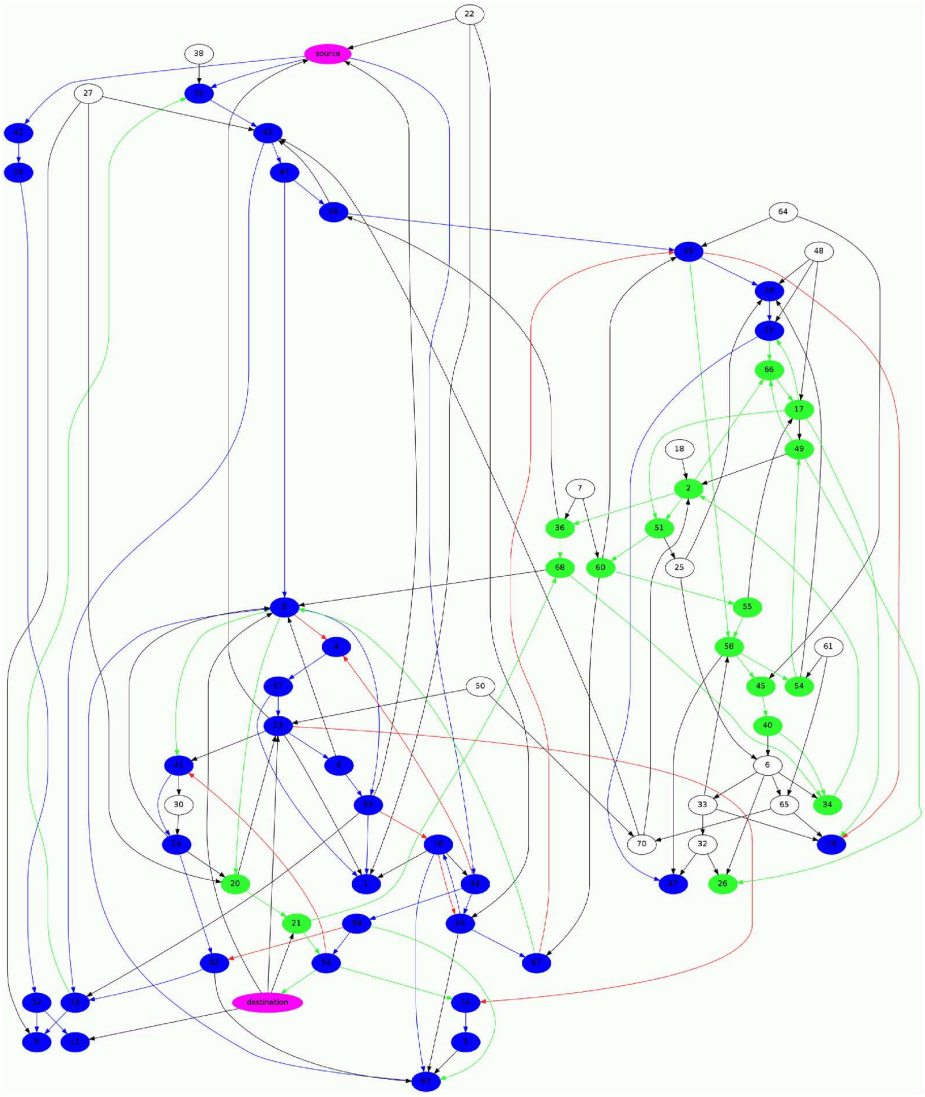

FIGURE 6.5 – Génération de la population initiale (75, 150), 20 cycles

trouvés ainsi que le nombre d'arcs. Nous constatons que, même lorsque le nombre de chemins physiques trouvés par la métaheuristique est peu élevé ($< 5\%$), la diversification est garantie car 72% des arcs sont couverts par ces chemins.

6.4.3 Déroulement de l'algorithme

6.4.3.1 Génération d'une nouvelle population

L'opérateur de sélection que nous avons mis on oeuvre est celui de l'algorithme NSGA-II. Les individus sont classés selon deux critères. Tout d'abord, les solutions sont classées en utilisant le concept de non-dominance. Toutes les solutions non dominées de la population sont affectées au rang 1 et sont ensuite retirées de celle-ci. Itérativement, les solutions non dominées sont déterminées et sont affectées au rang 2. Ce processus est ainsi réitéré jusqu'à épuisement de la population. Suite à cette étape, au niveau de chaque rang donné, les solutions sont classées de nouveau selon la distance de remplissage ("crowding distan-

Nombre de cycles	Chemins	Précision	Arêtes	Précision
100	8	%5	34	72%
200	13	8%	39	84%
300	14	9%	40	86%
400	15	9%	40	87%
500	20	12%	42	91%
600	23	14%	43	93%
700	27	16%	44	94%
800	30	18%	45	95%

TABLE 6.5 – Fiabilité de l'algorithme AS sur un réseau de 40 noeuds et 80 arcs

ce"). Pour plus de détails voir [60]. A chaque itération de l'algorithme, p descendants sont générés, où p est la taille de la population. Pour former une nouvelle poulation, la population âgée et sa descendance sont regroupées et classées selon les deux critères présentés ci-dessus. La meilleure partie de l'union des deux populations constitue la nouvelle population.

6.4.3.2 Mutation

Quelque soit la méthode utilisée pour la mutation, le changement stochastique d'un géne d'un individu peut naturellement engendrer une solution non réalisable. Celà est tout à fait normal puisque il y a beaucoup de contraintes à respecter, et la moindre modification au hasard du chemin peut violer une ou plusieurs contraintes. Pour éviter ceci, nous avons développé une méthode de mutation ah-doc, où l'opération du mutation prend en compte les contraintes du problème imposées par le modèle et par l'utilisateur.

Nous avons proposé et développé deux méthodes de mutation : une mutation temporelle et une mutation physique du chemin. La sélection d'une méthode ou d'une autre est basé sur un jeu de paramètres prédéfini qui dépend de la taille et du type du réseau d'entrée. Les deux méthodes de mutation sont définies comme suit :

- **Mutation temporelle** : Avec cette méthode on tente de minimiser un coût choisi au hasard. Pour celà, on choisit le coût à minimiser puis on parcourt les arcs du chemin initial afin de sélectionner à chaque fois le travel valide qui minimise le coût en question. Si aucun travel n'est valide, relativement au choix des travels précédents, la recherche s'arrête sans trouver de mutation possible.

- **Mutation physique** : Le principe de cette méthode découle de la nature du problème à traiter. Une propriété fondamentale du problème du plus court chemin dans un réseau du transport est que plus le chemin est court plus il est probable que sont coût soit optimal. Celà n'est pas toujours vrai, mais bien probable, notamment pour certains certains coûts, comme la distance ou le temps total. Cette méthode vérifie si le noeud de destination est successeur d'un noeud du chemin original, auquel cas un nouveau chemin est créé en concaténant la première partie du chemin original et le nouvel arc obtenu. La figure 6.6 illustre par un exemple cette mutation.

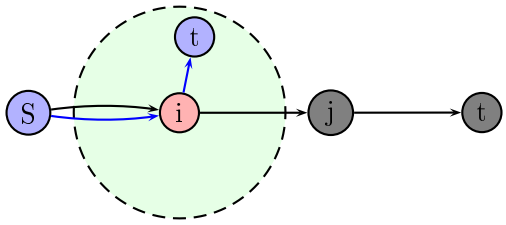

FIGURE 6.6 – Mutation physique d'un chemin

6.4.3.3 Croisement

Le croisement est la fonction principale de l'algorithme génétique. Le croisement (ou reproduction ou crossover) consiste à créer un individu à partir de deux individus parents. Il s'agit de recopier dans l'individu fils une partie du parent 1, jusqu'à atteindre une "cassure", puis ensuite la partie correspondante du parent 2. La difficulté réside dans le fait que les noeuds ne doivent pas être utilisés plusieurs fois dans un chemin. Il s'agit de la contrainte la plus forte. Pour éviter cette situation, un premier parent est sélectionné au hasard de la population courante puis on parcourt le reste de la population pour touver un deuxième parent qui partage au moins un noeud (qui jouera le rôle point de "cassure") avec le premier parent. Si aucun parent ne peut jouer le rôle de parent 2, alors un autre premier parent doit être déterminé. Ce processus se repète jusqu'à avoir une nouvelle population de même taille que la population courante, ou jusqu'à épuisement du nombre maximum d'essais autorisés. Dès que les deux parents sont sélectionnés, on cherche le premier point de cassure (qui doit forcément exister), en partant du noeud source d'un parent et du noeud destination de deuxième parent. Grâce à cette technique on évite la répétition des noeuds dans le chemin obtenu. Evidemment, la deuxième partie du fils doit être valide (il doit y avoir des travels valides par rapport au temps d'arrivée sur le noeud cassure). La figure 6.7 présente un exemple de croisement de deux parents. Il est aussi possible de permuter le rôle de deux parents pour trouver éventuellement un deuxième fils.

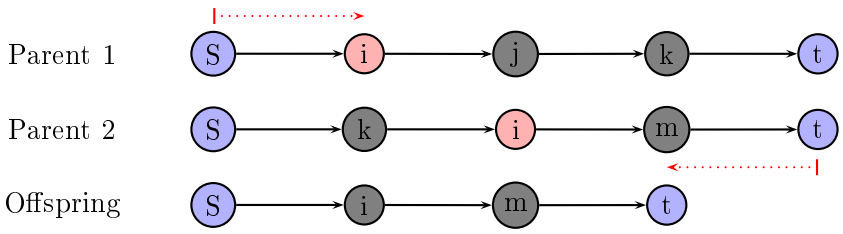

FIGURE 6.7 – Création d'un enfant suite à un croisement de deux parents

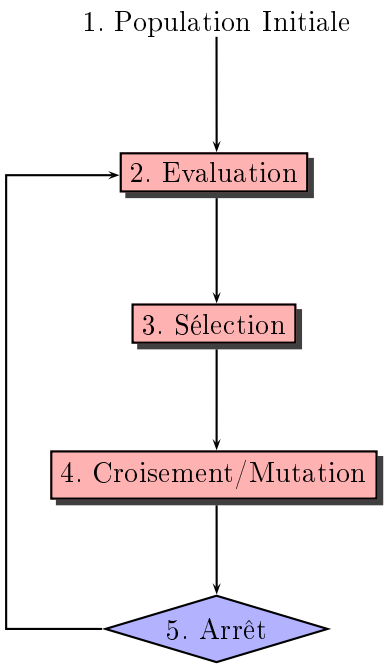

FIGURE 6.8 – Organigramme du fonctionnement de l'algorithme génétique

6.4.4 Implémentation et analyse des résultats

6.4.4.1 Génération de la population initiale

La qualité de la population initiale affecte les performances de l'algorithme génétique. En effet, plus la population initiale est diversifiée et plus l'algorithme génétique échappera aux optimums locaux. Pour garantir cette diversification nous avons donc utilisé l'algorithme ACO (voir algorithme 22). Nous discutons dans ce paragraphe les différents paramètres qui peuvent influencer la génération de la population initiale.

La figure 6.9 présente un exemple d'éxécution de l'algorithme ACO pour la génération de la population initiale pour 3 instances du problème. Il s'agit de montrer le comportement de l'algorithme (en nombre de chemins physiques trouvés) en augmentant le nombre de cycles. Il est clair que le nombre de chemins trouvés croît de façon exponentielle par rapport à l'augmentation du nombre de cycles éxécutés. Bien évidemment le temps d'éxécution de l'algorithme est proportionnel à l'augmentation du nombre de cycles. Un objectif serait donc de trouver le bon nombre de cycles à exécuter pour pouvoir générer un nombre suffisant de chemins physiques , servant à la création de la population initiale. Le principe de l'algorithme ACO pour la génération des chemins physiques est basé sur l'éxploration du graphe. Donc plus le graphe est dense, et plus il existe de possibilités de relier la source à la destination. Cette propriété est illustrée par la figure 6.10, où trois familles d'instances sont testées (1000, 2000 et 4000 noeuds) en augmentant à chaque fois la densité du graphe. Le nombre de cycles est fixé à 400 dans ce cas de figure.

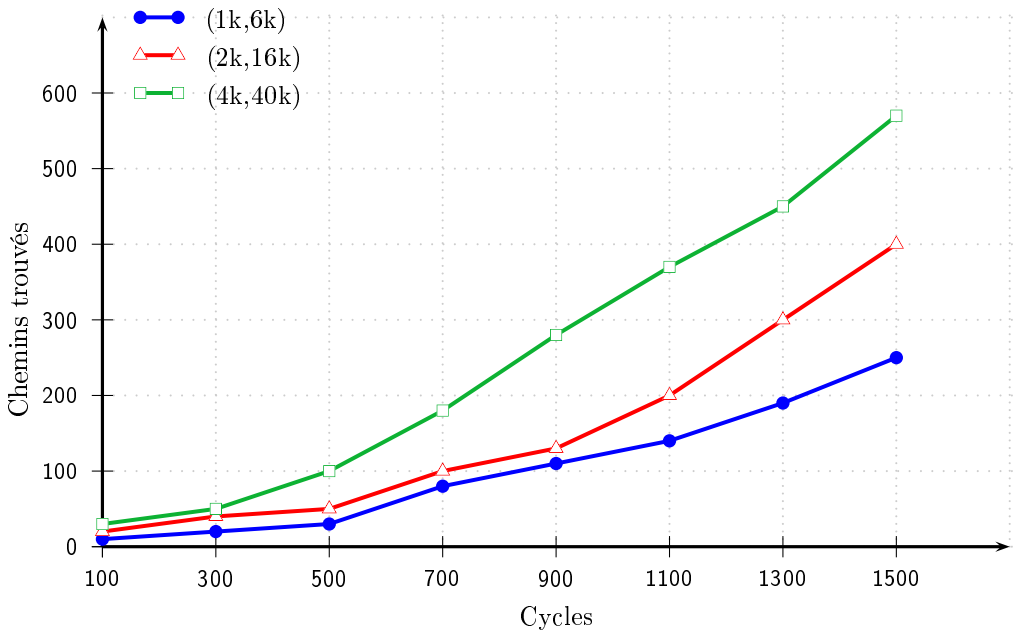

FIGURE 6.9 – Comportement de l'algorithme ACO avec l'augmentation du nombre de cycles

Nb Cycles	Temps[sec]	Chemins	Distance du point Nadir
Coûts = 2			
200	3,8	6	268,95
400	2,8	7	179,09
800	2,4	7	122,53
Coûts = 3			
200	3,3	40	298,74
400	5,4	43	244,51
800	4,6	35	128,73

TABLE 6.6 – Performance de l'algorithme NSGA-II pour l'instance (1000,3000)

Pour montrer l'influence de la qualité de la population initiale sur le résultat final, nous avons exécuter l'algorithme génétique avec deux différentes populations, une première générée avec 100 cycles d'ACO et une deuxième avec 400 cycles. La comparaison est réalisée par rapport au front complet. La figure 6.11 présente les résultats de cette comparaison. On remarque bien que le front obtenu avec 400 cycles (couleur jaune) est meilleur en qualité que celui obtenu avec 100 cycles (couleur rouge). Le tableau 6.6 montre le comportement de l'algorithme NSGA-II pour l'instance du problème (1000, 3000) avec 3 populations initiales générées avec 200, 400 et 800 cycles ACO. Le tableau 6.6 compare le temps de d'exécution, le nombre de chemins trouvés ainsi que la distance euclidienne par raport au front complet. On remarque bien que le nombre de chemins trouvés ainsi que le temps d'éxécution sont indépendants de la qualité de la population initiale. Par contre la qualité du front obtenu (distance du point Nadir) est fortement dépendante de la qualité de la population initiale. Cela est vrai quelque soit le nombre de coûts.

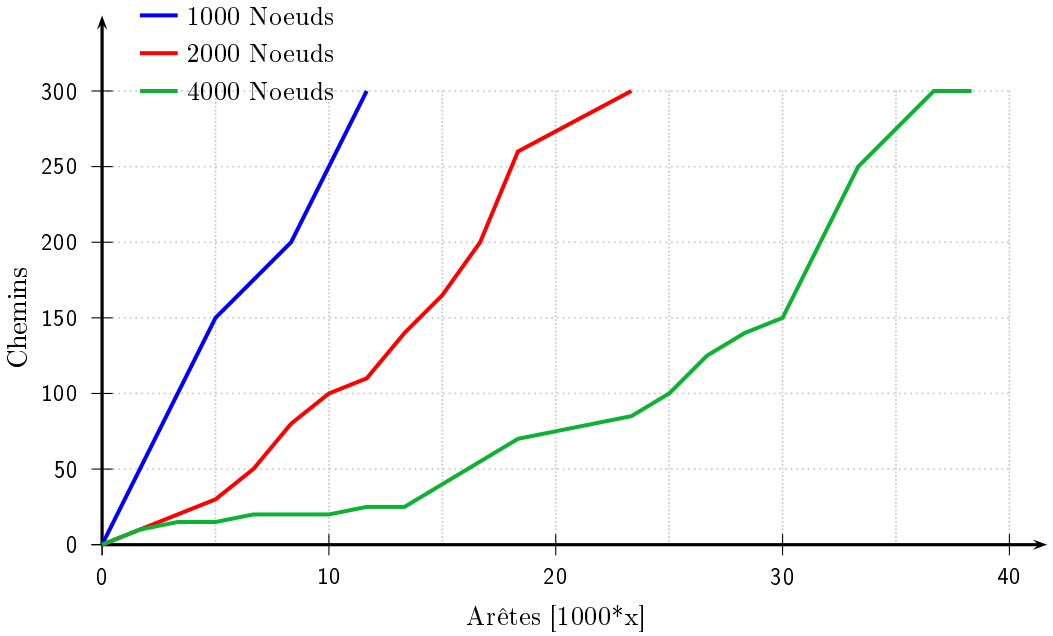

FIGURE 6.10 – Densité du graphe et nombre de chemins

6.4.4.2 Paramétrage de l'algorithme

Après avoir mis en évidence l'influence de la population initiale sur le comportement de l'algorithme, nous discutons ici l'enjeu d'un certain nombre de paramètres sur les performances de l'algorithme. Les paramètres en question sont le nombre de cycles, la taille de la population initiale ainsi que le probabilité de la mutation.

Influence du nombre de cycles : Le tableau 6.7 montre le comportement du l'algorithme NSGA-II sur l'instance (1000, 3000) lorque nous varions le nombre de cycles tout en maintenant fixée, la population initiale. Le tableau 6.7 montre bien que pour cette instance et pour une population initale donnée, l'augmentation du nombre de cycles n'améliore pas forcément la qualité de la solution obtenue. Cela est dû à la qualité de la population initiale. La couleur verte montre la limite pour laquelle l'augmentation du nombre de cycles ne fait qu'augmenter le temps de calcul.

Influence de la taille de la population : Pour connaître le comportement de l'algorithme vis-à-vis de la taille de la population, nous avons éxécuté l'algorithme sur l'instance (1000, 3000) pour 2 et 3 coûts, en variant la taille de la population de 10 à 300. Le résultat de ce test est illustré par la figure 6.12. On remarque bien que la qualité de la solution obtenue croît en augmentant la taille de la population et ce jusqu'à obtention d'une limite connue comme le point point idéal, au delà de laquelle l'augmentation de la taille de la population n'influence plus la qualité de la solution obtenue. Dans l'exemple, le point idéal est de 50 pour le problème bi-objectif et de 100 pour le problème tri-objectif.

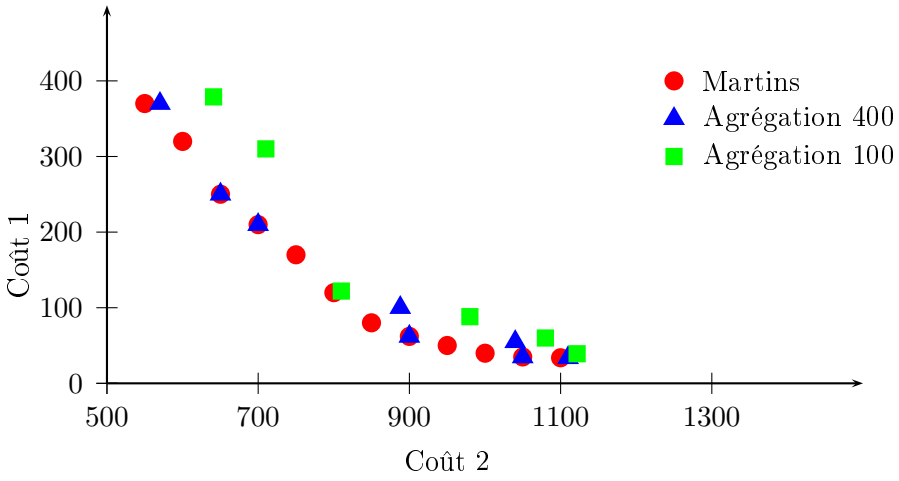

FIGURE 6.11 – Qualité du front et population initiale

Nombre de cycles	Temps [sec]	Chemins	Distance Nadir
Cost = 2			
10	0,3	7	218,95
20	0,6	7	192,7
40	1,6	7	184,3
80	2,5	7	216,6
160	4,8	7	196,9
320	6,3	7	208,8
Cost = 3			
10	0,7	40	313,95
20	1,1	37	315,04
40	1,7	38	308,8
80	3,9	35	307,9
160	6,5	41	252,24
320	12,2	41	268,34

TABLE 6.7 – Comportemant de l'algorithme G.A avec l'instance (1000,3000) pour différent nombres de cycles

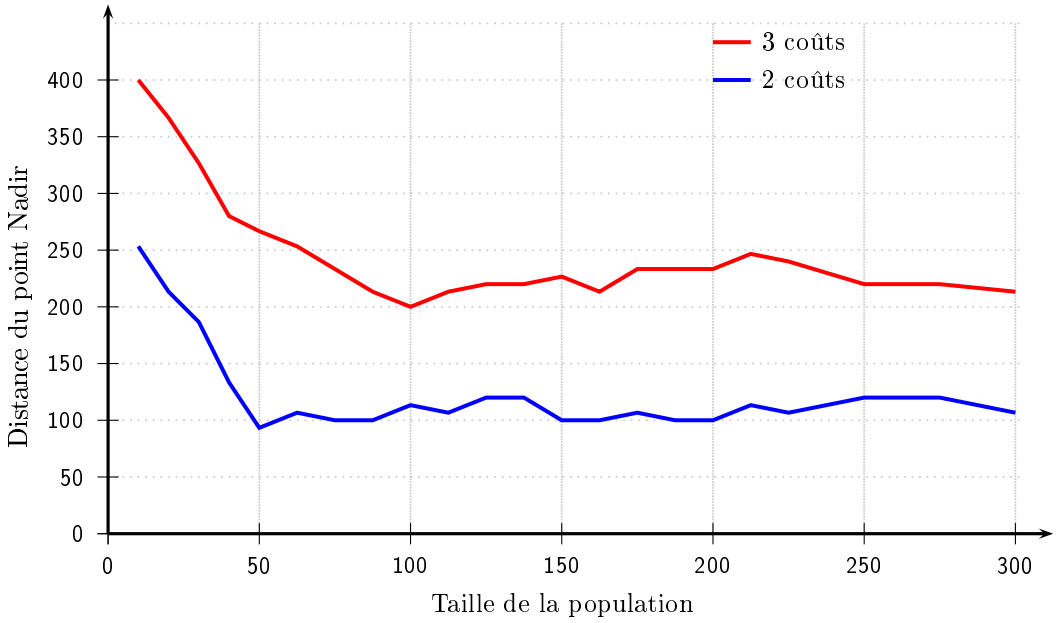

FIGURE 6.12 – Taille de la population et qualité de la solution

6.5 Algorithme MMTDSPP3 : Approche hybride

6.5.1 Motivation

Dans les sections précédentes nous avons mis en évidence les limites de l'algorithme de Martins dépendant du temps (MMTDSPP1) ainsi que celle de l'algorithme NSGA-II (MMTDSPP2), pour la résolution du problème du plus cout Multimodal multi-objectif et dépendant du temps. En effet l'algorithme NSGA-II retourne des résultats médiocres en terme de qualité du front pareto, pendant que l'algorithmer NSGA-II souffre d'un temps d'exécution élevé, notamment dans le cas où le nombre de d'objectifs est supérieur à 2.
Dans cette section, nous présentons un algorithme hybride combinant la rapidité des algorithmes basés sur des métaheuristiques et la précision des algorithmes exactes. Cet algorithme est basé sur l'idée de l'algorithme présenté dans le chapitre 5, pour la résolution du même problème dans un contexte mono-objectif. L'idée est de constuire un graphe intermédiaire en se basant sur le pricipe de la métaheuristique ACO. le graphe intermédiaire ne contient que les chemins physiques. Dans une deuxième phase, ce graphe intermédiaire est transformé en un "Relevant Graph", suite à des calculs additionnels issus de la prise en compte de la requête. Enfin, nous utilisons l'algorithme de Martins sur le graphe Relevant pour obtenir le front final.
La majeure différence entre la première méthode hybride pour le problème mono-objectif et celle présentée dans cette section, réside au niveau de la transformation des chemins physiques. Comme nous l'avons déjà mentionné dans le chapitre 5, la transformation d'un chemin physique dans le contexte multi-objectif est plus complexe et elle est éxécutée un grand nombre de fois.

6.5.2 Principe de l'algorithme hybride

L'algorithme hybride se compose des deux phases suivantes :

1. **Phase1 : Construction d'un graphe intermédiaire** : Dans la première phase de la méthode hybride, un algorithme basé sur l'idée des algorithmes de colonies de fourmis est appliqué à chaque composante mono-modale afin de créer une ensemble de chemins physiques reliant chaque paire de nœuds de transfert. A chaque fois qu'une fourmi arrive à un nœud de transfert, elle dépose une quantité de phéromone sur le chemin qu'elle a construit si ce dernier a des chances d'être solution. Au niveau de chaque nœud, le choix d'une arête parmi les différentes arrêtes sortantes dépend de la quantité de phéromone. Plus la quantité de phéromone d'une arête est importante, moins cette arête risque d'être sélectionnée. L'objectif est d'éviter les chemins déjà sélectionnés et de favoriser ainsi de nouveaux chemins non encore sélectionnés.

Algorithme 23 : Fonction createAbstractGraphs

```
1  pour tous les MonoModalGraph G faire
2      Initialization();
3      pour chaque TransferNode n ∈ G faire
4          répéter
5              PhysicalPath p;
6              Node current = n;
7              répéter
8                  pour chaque Noeud de Transfert succ Successeur de current faire
9                      PhysicalPath copy = p.copy();
10                     copy.add(succ);
11                     SavePath(copy);
12                 fin
13                 Array candidates;
14                 pour chaque Arête a sortante de current faire
15                     si a.to ∉ p AND a.to noeud local alors
16                         candidates.add(a);
17                     fin
18                 fin
19                 si candidates.size() alors Break;
20                 pour tous les Artea ∈ candidates faire
21                     ComputeProbability(a);
22                 fin
23                 Arête next = candidates.selectNext();
24                 p.add(next);
25                 current = next.to;
26             jusqu'à p.size < MaxSize ;
27             updatePheromone(p);
28         jusqu'à NbCycle not reached ;
29     fin
30 fin
```

Le calcul de l'ensemble des chemins physiques pour créer les graphes abstraits de

chaque composante mono-modale, est décrit par l'algorithme 23. La même procédure est exécutée par chaque composante mono-modale séparément, ce qui lui permet de s'exécuter en parallèle. Après une étape d'initialisation, la boucle principale est exécutée pour chaque nœud de transfert (ligne 3). La recherche des chemins physiques depuis le nœud sélectionné vers les nœuds de transfert, est répétée jusqu'à atteindre un nombre de cycles prédéfini. Lors de chaque cycle, un chemin physique vide est créé. La recherche commence par le nœud sélectionné (ligne 6) et se poursuit jusqu'à avoir un chemin physique de taille maximum prédéfini (ligne 7). Ensuite, les nœuds successeurs du nœud courant sont séléctionnés. Un nouveau chemin physique est créé en ajoutant chaque nœud au chemin courant (ligne 9-11). Chaque arête sortante du nœud courant, pouvant être ajoutée au chemin physique courant (Ligne 15) est enregistrée dans une liste de candidats (ligne 16). Enfin, la probabilité de chaque arête est calculée en fonction de sa quantité de phéromone, suite à quoi une arête est sélectionnée pour l'itération suivante. L'arête sélectionnée est ajoutée au chemin physique courant, et le nœud courant est mis à jour (Lignes 23-25). Si la liste de candidats est vide, la recherche est donc interrompue. Une fois que la recherche est terminée, la phéromone est mise à jour sur tout le chemin physique construit (ligne 27).

Comme nous pouvons bien le constater, dans cet algorithme, il existe des paramètres à prédéfinir par l'utilisateur qui peuvent influencer la qualité des chemins physiques trouvés et ainsi la solution finale. Ces paramètres sont :

- **NbCycle** : Ce paramètre représente le nombre cycles maximum à éxécuter par chaque noeud de transfert. Pour des résultats optimaux, ce nombre doit dépendre de la taille du réseau ainsi que de sa densité. En efet, plus le graphe est dense, plus y a des chemins physiques à découvrir et donc plus l'algorithme aura besoin de cycles pour trouver les bons chemins.

- **MaxSize** : C'est un paramètre très important qui influence à la fois la qualité de la solution et le temps d'exécution. La taille d'un chemin physique ne doit pas dépasser ce paramètre, auquel cas la recherche est interrompue (ligne 19). L'idée de ce paramètre est purement métier. En effet dans le domaine du transport public, plus un chemin est long et moins il est probable qu'il soit optimal.

Proposition 6.5.1

La complexité de calcul de la phase 1 est en $O(n(n+2m) = O(n^3)$ où n est le nombre de noeuds.

Preuve 6

Pour calculer la complexité de cet algorithme nous supposons que le graphe monomodal d'entrée est un graphe complet ($m = n(n-1)$). Dans ce cas, il est clair que la recherche d'un chemin physique pour un noeud de transfert donné se répète dans le pire des cas NbCycle fois. La boucle "répéter" ligne 7, est exécutée au maximum MaxSize fois. A chaque itération la boucle "pour chaque"(ligne 8) est exécutée au maximum n fois où n est le nombre de noeuds de transferts car dans le pire des cas tout noeud est aussi un noeud de transfert. Ensuite, la boucle "pour chaque" de la ligne 14 est répétée au maximum m fois, m étant le nombre d'arêtes. Cela donne une complexité

finale de l'ordre de $O(n(n+2m) = O(n^3)$.

Remarque 6.5.1

Il est vrai que cette complexité est polynomiale mais la qualité de la solution trouvée sera certainement très dépendante des deux paramètres présentés précédemment,une discution à ce sujet sera présentée à la section 6.5.3.

2. **Phase 2 : Résolution d'une requête** : Pour résoudre une requête avec ce modèle, trois étapes sont nécessaires :

   (a) **Calcul Additionnel** : Dans cette étape on calcule les chemins physiques de la source vers tous les noeuds de transfert si la source est un noeud local, sinon ce calcul est forcément réalisé dans la phase 1.

   (b) **Graphe Relevant abstrait** : Dans cette étape, le "Relevant Graph" abstrait est créé. Ce dernier est constitué de tous les noeuds de transfert, la source et la destination et des chemins physiques vus comme des hyper-arêtes.

   (c) **Martins** : Dans cette étape, l'algorithme de Martins pour le problème du plus court chemin dépendant du temps multi-objectif est utilisé. Ce dernier algorithme est utilisé modulo une modification mineure qui consiste à transformer les chemins physiques en chemins réels de façon dynamique. En effet, dans un "Relevant Graph" abstrait les hyper-arêtes encapsulent des chemins physiques, non encore transformés.

L'idée de Martins modifié pour le "Relevant Graph" abstrait est décrite par l'algorithme 24. La différence par rapport à l'algorithme originale pour le plus court chemin dépendant du temps, se situe au niveau de la boucle "pour tous" (ligne 12) qui transforme les chemins physiques d'une hyper-arête en des chemins dépendants du temps grâce à la fonction "Transform()" (Ligne 13). Cette fonction renvoie une liste de chemins non dominés, qui sont ensuite considérés comme des arêtes pour l'algorithme classique.

Une étape très importante de cet algorithme est la transformation d'un chemin physique en un ensemble de chemins réels non dominés. Cette transformation a fait l'objet de la section 6.2. Cette étape est importante car la méthode utilisée influence le nombre de chemins réels trouvés et ainsi la qualité de la solution finale, comme le souligne la discussion qui suit.

6.5.3 Implémentation et analyse des résultats

Dans cette section, nous discutons les caractéristiques de l'algorithme hybride relativement à différents paramètres de l'algorithme. Cette discussion concerne deux critères à savoir le temps d'exécution pour les deux phases de l'algorithme ainsi que la qualité de la solution finale d'autant plus que cette méthode est incomplète.

Algorithme 24 : L'algorithme Hybride pour le problème Multi-Objectifs

```
1  cnt ← 1;
2  h(cnt) ← s;
3  Z_s ← path(1);
4  X ← {cnt};
5  Time(cnt) ← t_0;
6  tant que X ≠ ∅ faire
7      x ← lexico(X);
8      X ← X\x;
9      i ← h(x);
10     pour tous les (i, j) ∈ E faire
11         time ← Time(x);
12         pour tous les PhysicalPath ph ∈ (i, j).paths() faire
13             List paths = Transform(ph,time);
14             pour chaque Path p ∈ paths faire
15                 Travel tr = TravelOf(p);
16                 path(cnt) ← path(cnt) ⊕(p, tr);
17                 si path(cnt) n'est pas dominé in Z_j alors
18                     cnt ← cnt + 1;
19                     ajouter (x, cnt) à ST ;
20                     h(cnt) ← j;
21                     Time(cnt) ← tr.arrival;
22                     X ← X ∪ cnt ;
23                     Z_j ← Z_j ∪ path(cnt);
24                     Supprimer tous les chemins dominés de Z_j;
25                     Supprimer les sous-arbres correspondants dans ST;
26                 fin
27             fin
28         fin
29     fin
30 fin
```

6.5.3.1 Rôle des paramètres NbCycle et MaxSize

Nb Cycle	CPU(s) Ph 1	CPU(s) Ph 2	Distance pareto
100	0.23	0.36	28.46
200	0.48	0.42	28.28
300	0.76	0.44	28.11
400	1.07	0.45	27.67
500	1.37	0.44	27.66
600	1.77	0.46	27.81
700	2.13	0.45	27.80
800	2.41	0.47	27.68
900	2.74	0.44	27.77
1000	3.01	0.46	27.68

TABLE 6.8 – Influence de NbCycle sur phase 1 et phase 2

Le tableau 6.9 présente les résultats de la discussion du paramètre "NBCycle", qui montre l'influence de l'augmentation du nombre de cycle de la première phase sur le temps de l'exécution et la qualité de la solution finale pour l'instance du problème (1k,2k) avec 2 modes et 50 travels/arcs. Nous avons fait varier le nombre de cycles de 100 à 1000. Nous observons aussi bien les temps d'exécution pour les deux phases que la qualité du front obtenue par rapport au font pareto complet. Ce calcul est effectué sur 100 requêtes et pour 100 graphes multimodaux, après avoir fixé le nombre de cycles. Ce tableau nous révèle deux résultats significatifs. D'abord nous constatons que le temps d'exécution de la deuxième phase croît avec l'augmentation du nombre de cycles et ce jusqu'à une limite. Au delà de cette limite (NbCycle = 500) il est pratiquement inutile d'augmenter le nombre de cycles. Cela s'explique par le fait que après cette limite les chemins physiques générés en plus, n'ont pas plus de chance participer à la solution finale car leurs qualités se détériorent.

Nous pouvons aussi constater que la qualité des fronts trouvés dépend de cette limite. En effet la qualité du front commence à s'affiner avec l'augmentation du nombre de cycles de la première phase jusqu'à atteindre une limite (aussi 500 cycles), au delà de laquelle la qualité de sera ne sera pas forcément améliorée. Par contre le temps d'exécution de la première phase augmente de façon linéaire par rapport à l'augmentation du nombre de cycles.

Nous synthétisons dans le tableau 6.9 l'impact du paramètre MaxSize de la première phase sur le temps d'exécution d'une requête ainsi que sur la qualité de la solution obtenue. Ce tableau présente le cas d'une instance du problème (1k, 2k) avec 2 modes et 50 travels/arcs. Nous avons fait varier la taille d'un chemin physique autorisé (MaxSize) de 10 à 100. Comme nous pouvons le constater, le temps d'exécution de la phase 1 augmente polynomialement avec l'augmentation du paramètre MaxSize. Il est clair que, plus les chemins physiques trouvés dans la phase 1 sont longs, plus l'algorithme de Martins exécuté dans la deuxième phase sera coûteux. Cette constation se confirme aussi par ce ce tableau. Le résultat le plus pertinent à retenir dans ce tableau est que la qualité de la solution finale s'améliore au début puis se dégrade assez vite.

MaxSize	CPU(s) Ph 1	CPU(s) Ph 2	Distance pareto
10	0.12	0.16	55.13
20	0.22	0.47	29.88
30	0.24	0.74	18.09
40	0.55	0.92	11.36
50	0.80	1.04	10.54
60	1.07	1.48	10.34
70	2.22	1.85	09.86
80	4.48	2.37	09.84
90	7.64	2.49	09.81
100	9.91	3.76	09.77

TABLE 6.9 – Influence de la taille maximale d'un chemin physique sur la qualité du résultat finale

La figure 6.13 présente la même synthèse mais pour 4 instances du problème de 1000 à 16000 noeuds. Le résultat obtenu pour la qualité de la solution finale dans le tableau 6.9 se confirme aussi. En effet, pour chaque problème considéré, l'amélioration de la qualité de la solution finale est bien nette jusqu'à une limite, au delà de laquelle, l'augmentation de la taille du chemin physique n'apporte plus aucun bénéfice. Cette limite n'est pas la même pour tous les instances traitées. en effet plus le graphe est grand plus la taille moyenne des chemins physiques est grande.

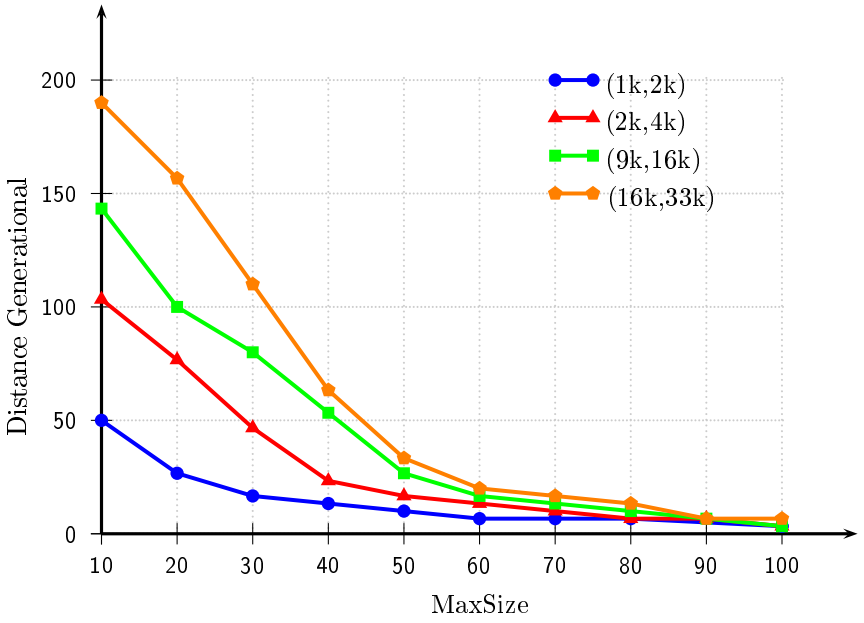

FIGURE 6.13 – Qualité du front obtenu Vs augmentation de MaxSize

Pour la suite des expérimentations nous fixerons 500 cycles pour la première phase quelque soit la taille du problème à résoudre et adapterons la taille du chemin physique à la taille de l'instance à résoudre (voir tableau 6.10).

Instance	MaxSize	NbCycle
(1k, 2k)	30	500
(2k, 4k)	40	500
(4k, 8k)	50	500
(9k, 16k)	60	500
(16k, 33k)	60	500
(32k, 66k)	70	500

TABLE 6.10 – NbCycle et MaxSize pour la suite des expérimentations

6.5.3.2 Transformation d'un chemin physique

Nous avons présenté dans la section 6.2 deux méthodes pour la transformation d'un chemin physique, à savoir l'approche exacte basée sur l'algorithme de martins et l'approche incomplète basée sur la technique d'agrégation. Nous montrons dans cette section leurs avantages et inconvénients sur l'algorithme hybride qui utilise la transformation dans la deuxième phase.

	Temps Exécution			Distance F.P		
	1000 Agg	5000 Agg	Martins	1000 Agg	5000 Agg	Martins
10 Travels/Arcs						
1k, 2k	00,27	01,01	00,01	00,38	00,21	$\simeq 0.0$
4k, 8k	00,66	03,32	00,03	00,55	00,32	$\simeq 0.0$
16k, 32k	03,06	06,01	00,21	01,03	00,87	$\simeq 0.0$
100 Travels/Arcs						
1k, 2k	07,60	35,04	00,22	02,87	02,85	02,16
4k, 8k	08,01	40,06	00,61	03,21	03,20	02,27
16k, 32k	10,57	50,32	01,05	04,57	04,22	02,98
200 Travels/Arcs						
1k, 2k	14,09	70,21	02,11	07,19	07,01	03,32
4k, 8k	18,31	91,01	02,55	09,98	09,22	04,07
16k, 32k	22,01	111,18	03,07	13,05	12,92	05,11

TABLE 6.11 – Influence de la tranformation des chemins physiques sur le resultat final avec 2 coûts

Les deux tableaux 6.11 et 6.12 présentent les résultats des méthodes utilisées pour la transformation de chemins physiques sur la qualité du résultat final obtenu ainsi que sur le temps d'exécution de l'algorithme. Les méthodes utilisées sont celle d'agrégation avec 1000 et 5000 fonctions et le front pareto. Trois types de réseaux sont considérés : (1k, 2k), (4k,8k) et (16k, 32k) avec une variation du nombre de travels de 10, 100 et 200. Le tableau 6.11 traite le cas de 2 coûts et le tableau 6.12 traite celui de 3 coûts. Pour le cas bi-objectif, nous pouvons tirer deux observations : d'abord l'utilisation de la méthode d'agrégation avec 5000 fonctions n'apporte aucune amélioration par rapport au temps d'exécution. De plus, l'application de Martins pour la transformation des chemins physiques est clairement de loin la meilleure méthode aussi bien en terme de temps d'exé-

cution qu'en qualité de la solution obtenue, et cela quelque soit la taille du problème traité.

	Temps Exécution			Distance F.P		
	1000 Agg	5000 Agg	Martins	1000 Agg	5000 Agg	Martins
10 Travels/Arcs						
1k, 2k	01,89	02,33	07,66	01,56	03,09	$\simeq 0.0$
4k, 8k	03,09	04,91	15,93	02,15	03,85	00.02
8k, 16k	04,11	07,35	29,11	03,84	04,22	00.03
100 Travels/Arcs						
1k, 2k	09,88	45,55	32,52	02,87	02,85	05,06
4k, 8k	11,76	55,73	65,72	03,21	03,20	06,80
8k, 16k	12,79	60,38	82,15	04,57	04,22	07,44

TABLE 6.12 – Influence de la tranformation de chemins physique sur le resultat final, avec 3 coûts

Dans le cas de tri-objectif, l'algorithme de Martins pour transformer les chemins physiques n'est plus efficace, car le temps d'exécution de cette méthode explose avec la taille du problème. Évidemment l'algorithme de Martins reste le meilleur en terme de qualité du front pareto. Par contre la méthode d'agrégation avec 1000 fonctions présente le meilleur compromis temps/qualité. Notons que pour le cas tri-objectif, nous n'avons pas considéré le test avec 200 travels/Arcs car l'exécution de l'algorithme de Martins pour le problème entier n'est pas possible et donc nous n'avons pas moyen de comparer la qualité de la solution trouvée par l'algorithme Hybride. Dans le reste des expérimentations nous utiliserons pour la transformation des chemins physiques l'algorithme de Martins dans le cas bi-objectif et la méthode d'agrégation avec 1000 fonctions pour le cas tri-objectif.

6.5.3.3 Comparaison des algorithmes de Martins et NSGA-II

Après avoir montré les limites de l'algorithme de Martins pour le problème de plus court chemin dépendant du temps et multi-objectif, nous discutons ici les performances et les limites de l'algorithme hybride ainsi que la comparaison avec l'algorithme NSGA-II présenté dans la section 6.4.

Le tableau 6.13 présente le temps d'exécution de la deuxième phase (réponse requête) de l'algorithme hybride. Dans ce tableau, nous avons testé 6 instances du problème en variant le nombre de noeuds de 1000 à 32.000, le nombre de coûts de 2 à 5 et le nombre de travels/arcs de 10 à 300. Ce même tableau a été présenté dans la section 6.3 pour montrer les limites de l'algorithme de Martins pour le MMTDSPP. Clairement cette méthode est beaucoup plus rapide que celle de Martins et pour des tailles des instances traitées beaucoup plus grandes.
La méthode hybride est incomplète à cause de sa phase 1 et par conséquent le résultat final obtenu n'est pas forcément de "bonne" qualité. Le tableau 6.14, présente une comparaison entre l'algorithme génétique (100, et 300 cycles) et l'algorithme hybride en terme de temps d'exécution et de qualité du front relativement au font complet. Ce tableau montre bien que l'algorithme hybride est bien meilleur en terme de qualité du solutions obtenues que l'algorithme génétique quelque soit le problème à résoudre. En ce qui concerne le temps d'exécution, il est a noter que pour l'algorithme génétique l'augmentation de la taille du

	(1k,2k)	(2k,4k)	(4k,8k)	(9k,16k)	(16k,33k)	(32k,66k)
Nombre de Coûts = 2						
10	00,01	00,02	00,03	00,12	00,21	00,37
100	00,22	00,43	00,61	00,87	01,05	02,49
200	02,11	02,47	02,55	02,79	03,07	05,86
300	04,62	04,97	06,07	07,23	10,43	15,66
Nombre de Coûts = 3						
10	01,89	02,22	03,09	03,88	04,11	05,62
100	09,88	10,05	11,76	12,08	12,79	13,55
200	14,60	14,99	16,28	17,17	19,55	22,71
300	23,28	24,83	28,88	30,07	31,13	32,89
Nombre de Coûts = 4						
10	2,07	04,61	06,08	08,43	12,03	24,01
100	11,54	13,79	14,66	17,98	24,04	33,78
200	15,43	19,51	22,90	30,19	40,01	55,90
300	27,77	30,39	33,87	35,99	47,92	69,09
Nombre de Coûts = 5						
10	5,13	07,82	11,55	14,05	21,72	36,36
100	24,66	33,88	48,06	61,08	$\succ$	$\succ$
200	33,01	51,19	$\succ$	$\succ$	$\succ$	$\succ$
300	42,12	$\succ$	$\succ$	$\succ$	$\succ$	$\succ$

TABLE 6.13 – Temps CPU de la phase 2 de l'algorithme

problème à résoudre influence légèrement le temps CPU puisque cet algorithme est plutôt sensible au nombre de cycles à exécuter. Par contre l'algorithme Hybride est très sensible à la taille de l'instance à traiter. En effet, quand l'instance du problème est facile (coût = 2), l'algorithme hybride est plus rapide que l'algorithme génétique et donne aussi de meilleurs résultats, cela même avec 300 cycles. Ensuite dès que le problème devient plus difficile à traiter (Coût >= 3) l'algorithme hybride devient, sans trop de surprise, plus lent que l'algorithme génétique.

	Temps Exécution			Distance F.P		
	A.G 100	A.G 300	Hybride	A.G 100	A.G 300	Hybride
Nombre de Coûts = 2						
1k, 2k	02,04	06,22	00,22	02,41	02,37	02,16
4k, 8k	02,88	07,19	00,61	02,66	02,44	02,27
16k, 32k	03,98	12,01	01,05	03,31	03,21	02,98
Nombre de Coûts = 3						
1k, 2k	04,34	12,91	09,88	07,12	07,10	02,87
4k, 8k	05,82	18,07	11,76	09,66	08,01	03,21
16k, 32k	06,66	19,05	12,79	12,83	10,39	04,57
Nombre de Coûts = 4						
1k, 2k	09,19	30,00	11,54	23,16	20,31	07,08
4k, 8k	10,12	32,91	14,66	27,88	22,94	12,09
16k, 32k	14,01	42,41	24,04	46,71	38,74	15,59

TABLE 6.14 – NSGA-II et Hybride : temps d'exécution et qualité des solutions avec 2 modes et 100 Travels/Arcs

6.6 Approche parallèle

6.6.1 Comment paralléliser dans le contexte multi-objectif ?

En algorithmique parallèle, il est important de distinguer la parallélisation d'un algorithme séquentiel, d'un algorithme inhéremment parallèle. Dans le cas de la parallélisation d'un algorithme séquentiel, l'objectif est d'exécuter des sous-tâches de l'algorithme, indépendantes ou non, en parallèle. Le degré de parallélisme dépend du nombre de tâches parallélisables. Dans un algorithme inhéremment parallèle, il s'agit d'un algorithme dont le fonctionnement est purement parallèle. Dans le domaine de l'optimisation, la programmation parallèle est généralement utilisée dans l'esprit de "coopérer pour élaguer au plus vite l'espace de recherche". Par conséquent, il ne s'agit pas uniquement d'exécuter un ensemble de tâches en parallèle mais aussi de rendre cette exécution plus bénéfique par le biais d'un bon principe de coopération et de compétition. Cette deuxième classe d'algorithme est beaucoup plus efficace qu'une simple exécution parallèle d'un ensemble de sous-tâches.

L'algorithme que nous avons proposé dans la section 5.3 du chapitre 5, fait partie de la catégorie d'algorithmes inhéremment parallèles, basé sur le principe de la coopération et d'élaguage de l'arbre de recherche. Rappelons que dans cet algorithme, la recherche dans une branche de l'arbre de recherche s'arrête dès que le chemin courant obtenu est moins bon que le meilleur chemin dèja obtenu.

Cette technique n'est plus applicable au cas multi-objectif, car la comparaison dans ce cas ne porte plus sur un seul objectif mais plutôt sur tous les objectifs simultanément, modulo la règle de dominance qui est beaucoup trop coûteuse pour pouvoir être effectuée à chaque étape de l'algorithme et par chaque processeur.

6.6.2 Une parallélisation naïve

Nous avons implémenté l'algorithme parallèle présenté dans la section 5.3 du chapitre 5, en considérant comme noyau l'algorithme de Martins pour supporter la multiobjectivité. Le tableau 6.15 présente une comparaison entre les résultats obtenus avec l'algorithme de Martins séquentiel et l'implémentation du l'algorithme parallèle sur le graphe de transfert.

	(1k,2k)	(2k,4k)	(4k,8k)	(9k,16k)	(16k,33k)	(32k,66k)
Martins Séquentiel (section 6.3)						
10	0.05	0.11	0.3	0.7	1.86	4.42
90	0.47	1.13	2.68	7.02	18.36	64.25
210	1.19	2.72	6.74	9.43	33.65	$\succ$
300	1.45	3.92	12.08	24.59	$\succ$	$\succ$
Martins Parallèle sur Graphe de Transfert						
10	0.12	0.23	0.93	1.2	4.09	12.47
90	1.53	6.01	14.41	38.55	61.06	$\succ$
210	17.92	29.98	49.02	$\succ$	$\succ$	$\succ$
300	44.63	$\succ$	$\succ$	$\succ$	$\succ$	$\succ$

TABLE 6.15 – Martins séquentiel et Martins parallèle sur graphe de transfert dans le cas bi-objectif

Les résultats de parallélisation obtenus sur deux processeurs sont clairement mauvais et confirment bien que ce type de parallélisation n'est pas efficace dans ce cas.

6.6.3 Parallélisation de NSGA-II

Une deuxième idée de parallélisation intéressante à développer consiterait à paralléliser l'algorithme de NSGA-II proposé dans la**section 6.4** . Cette idée a été largement explorée dans la littérature [14], pour d'autres applications que le transport. Les pincipes de parallélisation sont classiques. Nous envisageons de développer cette approche dans le cadre d'une formation post-doctorale portant sur la parallélisation des métaheuristiques de façon plus générale.

Conclusion

Dans ce chapitre nous nous sommes intéressés à la résolution du problème dans toute sa dimension, à savoir la résolution du plus chemin multimodal, multi-objectif et dépendant du temps. Comme nous l'avons déjà précisé dans l'état de l'art, une modification de l'algorithme de Martins peut permettre de résoudre ce problème. C'est pour cela que nous avons commencé par implémenter cette méthode. Nous avons pu mettre en évidence ses limites du point de vue pratique. En effet, l'algorithme de Martins s'est avéré trés performant dans le cas d'optimisation bi-objectif. Par contre au delà de 2 objectifs, il devient inefficace notamment en terme d'espace mémoire. Nous nous sommes ensuite orientés vers l'utilisation des métaheuristiques et avons proposé une modification de NSGA-II en adéquation avec le problème du transport multimodal dépendant du temps. Le comportement de cet algorithme est totalement opposé à celui de Martins. En effet nous avons pu

observer que cet algorithme présente de bonnes performances en terme de temps CPU, proportionnellement au nombre de cycles d'exécution et une complexité memoire réduite. Cependant, il ne peut proposer de front pareto de "bonne qualité" qui exige un espace de recherche trop grand pour exploiter une recherche stochastique guidée.

Comme dans le cas d'optimisation mono-objectif, la méthode hybride proposée pour le problème d'optimisation multi-objectif vise à combiner la rapidité des métaheuristiques et la complétude des méthodes exactes. Cette méthode basée sur le le graphe de transfert fait appel à un algorithme stochastique inspirée de l'idée des colonies de fourmis pour créer une base de données composés de chemins physiques indépendant des requêtes. Ensuite, pour chaque requête, un graphe intermédiaire ou graphe Relevant est créé pour enfin résoudre le problème en utilisant l'algorithme de Martins. Cet algorithme nous a permis d'améliorer les résultats obtenus par l'algorithme de Martins et l'algorithme NSGA-II. Une étude fine de cet algorithme selon diférents paramètres, nous a permis de conclure que ce dernier est plus sensible au nombre d'objectifs que que l'algorithme NSGA-II, relativement au temps d'exécution. Mais il offre une meilleure qualité de front pareto, qui dans le cas de NSGA-II se dégrade trop vite au delà de deux objectifs.

Chapitre 7

Conclusion générale et perspectives

Conclusion

Nous nous sommes intéressés dans cette thèse à la problématique de transport usager dans un contexte multimodal, multi-objectif et dépendant du temps. Dans le contexte de transport usager nous nous sommes particulièrement intéressés à la recherche d'itinéraires, qui satisfont dans la mesure du possible, les voeux des usagers. Du point de vue théorique, ce problème se ramène à celui du problème du plus court chemin, bien connu en théorie des graphes. Dans une première partie de cette thèse, nous avons fait état d'une étude de l'existant qui est suffisamment vaste en matière des trois mots résumant notre problématique, à savoir le transport multimodal, l'optimisation et le problème du plus court chemin.

Cet état de l'art relativement exhaustif nous a permis de constater que, bien que ce sujet ait attiré activement les chercheurs cette dernière décennie, les résultats les plus récents se limitent encore à des réseaux de taille moyenne. Nos principales contributions résident en la proposition de solutions algorithmiques génériques et adaptées au traitement de réseaux de transport très larges et sur une portée géographique très étendue.

Nous pouvons résumer nos contributions comme suit :

1. Nous avons proposé le modèle appelé graphe de transfert, comme modèle d'abstration des réseaux de transport mulimodaux ainsi que le modèle d'hypergraphe de transfert pour modéliser des réseaux de grande taille. Ces deux modèles n'imposent aucune restriction sur les déplacements possibles, ni sur les modes de transport qu'ils soient privés ou publics. De plus, ils prennent en compte les contraintes métier comme la non centralisation des données, en gardant les réseaux mono-modaux autonomes en facilitant ainsi leur gestion et leur mise à jour.

2. Nous nous sommes ensuite situés dans un contexte mono-objectif pour ne nous concentrer que sur la multimodalité et la notion de dépendance au temps. Nous avons à ce titre, proposé deux algorithmes séquentiels MTDSPP1 et MTDSPP2 ainsi qu'un algorithme parallèle PMTDSPP. L'algorithme MTDSPP1 est basé sur le graphe de transfert et sur une étape de pré-traitement indépendante de la requête usager. Pour cette étape cruciale de l'algorithme qui consiste à créer une base de données, nous avons proposé deux versions. La première version utilise un algorithme de Dijkstra dépendant du temps qui est très coûteux en terme d'espace

mémoire. La deuxième solution s'appuie sur la métaheuristique ACO qui est plus gourmande en temps CPU mais ne pose pas le problème de complexité spaciale, tout en ne garantissant pas l'obtention d'une solution même si elle existe. Pour pallier aux inconvénients des deux versions de l'algorithme MTDSPP1, nous avons proposé l'algorithme MTDSPP2, un algorithme hybride qui hérite des avantages des deux versions de l'algorithme PTDSPP1 tout en évitant leurs inconvénients. Bien que le temps de réponse aux requêtes soit très court avec cette approche, il n'en demeure pas moins que la construction de la base de données ou des structures intermédiaires limitent ces approches à des réseaux de taille moyenne. Le plus grand réseau traité se compose de 4.000 nœuds et de 15.000 arêtes. Pour faire face à des réseaux à l'échelle d'un pays voire de plusieurs pays, nous avons proposé l'algorithme parallèle PMTDSPP, en mémoire partagée basé sur l'hypergraphe de transfert. Cet algorithme a effectivement permis de résoudre le problème du plus court chemin sur des réseaux composés de 400.000 nœuds et 4.200.000 arêtes avec 16 processeurs en 13 secondes.

3. Une fois que les aspects multimodaux et dépendance au temps aient été bien maîtrisés, nous nous sommes intéressés à la résolution de ce même problème dans un contexte multi-objectif. Nous avons, dans un premier temps, implémenté l'algorithme exact de Martins dépendant du temps. Nous avons pu comprendre expérimentalement que cet algorithme est peu adapté à notre problématique. Les performances de cette approche se limitent au cas bi-objectif. Dans un deuxième temps, nous avons proposé une version modifiée de l'algorithme NSGA-II pour le résoudre. Les résultats expérimentaux obtenus sont meilleurs que ceux obtenus avec l'algorithme de Martins en terme de taille d'instances, bien entendu ceci au coût d'une qualité moins bonne du front pareto. Nous avons ensuite proposé une nouvelle approche plus originale ou approche hybride, pour pallier aux contraintes des deux premières solutions. Cette solution a permis de résoudre des instances du problème avec de meilleures performances en temps CPU que l'algorithme de Martins et une meilleure qualité du front pareto que celui obtenu par NSGA-II. La plus grande instance composée de 16.000 noeuds, de 32.000 arêtes et de 4 objectifs a été résolue en 42,21 secondes par cette approche.

Au terme de cette thèse, nous avons atteint la quasi totalité des objectifs que nous nous sommes fixés. Nous avons pu valider nos résultats sur des benchmarks académiques mais nous avons aussi pu les appliquer sur des cas réels. La parallélisation de l'algorithme du plus court chemin dans un contexte multi-objectif, pour lequel nous avons fait une première étude est le seul point non concluant fera l'objet d'un projet post-doctoral au niveau du CRP Henri Tudor.

L'ensemble de nos contributions ont fait l'objet des publications suivantes :

1. Hedi Ayed, Djamel Khadraoui, Zineb Habbas, Pascal Bouvry, & Jean François Merche : *Transfer Graph Approach for Multimodal Transport Problems*- MCO 2008 Luxembourg

2. Hedi Ayed, Djamel Khadraoui & Zineb Habbas. *Deployment of a new Optimization Technique based on Transfer Graph, Validation in Multimodal Transport.* Domain

Logistics and Transport 2009 Tunisie.

3. Ayed Hedi, Zineb Habbas, Carlos Galvez-Fernandez & Djamel Khadraoui. *Hybrid Algorithm for Solving a Multimodal Transport Problems Using a Transfer Graph Model* Ubiroads 2009 Tunisie.

4. Ayed Hedi, Zineb Habbas, & Djamel Khadraoui. *ACO for solving a Multimodal Transport Problems using a transfer graph model.* CIE39 Troyes 2009 France.

5. Carlos Galvez-Fernandez, Djamel Khadraoui, Hedi Ayed, Zineb Habbas, & Enrique Alba. *Distributed Approach for Solving Time-Dependent Problems in Multimodal Transport Networks.* Advances in Operations Research Journal (2009), Article ID 512613, ISSN :16879147

6. Hedi Ayed, Carlos Galvez-Fernandeza, Zineb Habbas, Djamel Khadraouia. *Solving Time-dependent Multimodal Transport Problems Using a Transfer Graph Model* Computers & Industrial Engineering Journal, Volume 61, September 2011. pp 391-401.

7. Ayed Hedi, Zineb Habbas, & Djamel Khadraoui. *A Parallel Time-dependent Multimodal Shoertest Path Algorithm based on geographical partitioning.* International Conference on Computer Design and Engineering. Malaysia 2011.

8. Ayed Hedi, Zineb Habbas, & Djamel Khadraoui. *A Parallel Algorithm for Solving Time Dependent Multimodal Transport Problem.* IEEE ITSC 2011 Washington DC, USA - October 5-7, 2011.

9. Ayed Hedi, Zineb Habbas, & Djamel Khadraoui. *A Parallel Time-dependent Multimodal Shortest Path Algorithm Based on Geographical Partitioning.* Third World Congress on Nature and Biologically Inspired Computing (NaBIC2011) October 19-21, 2011.

Perpectives

Sans vouloir être exhaustif, nous résumons quelques perspectives intéressantes comme suit :

- Une parallélisation en mémoire distribuée : Un de nos objectifs dans cette thèse est d'améliorer la résolution du plus court chemin multimodal, multi-objectif et dépendant du temps en faisant appel aux capacités de l'algorithmique parallèle et des machines parallèles. Notre contribution dans cette thèse a été axée sur le parallélisme en mémoire partagée. Pour mieux exploiter l'aspect inhéremment distribué des réseaux de transport et de la non centralisation des données, une étude de solutions parallèles en mémoire distribuée est à envisager.
- Une étude sur la robustesse : La robustesse est un bien grand mot qui cache un grand nombre de concepts. Cette perspective non étudiée dans cette thèse est fort

prometteuse non pas uniquement pour son intérêt scientifique, mais aussi pour une meilleure prise en compte des réseaux de transport du monde réel au sens large. Bien que nous n'ayons pas traité ce concept dans cette thèse, nous tenons toutefois à souligner que le modèle de graphe de transfert que nous avons proposé reste valable pour le traitement de celui-ci, à condition de prendre en compte les contraintes de robustesse au niveau des algorithmes proposés.

– L'intégration de modes spécifiques : Nous avons traité la multimodalité sans restriction particulière relative aux modes de transport. Pour prendre en considération les nombreuses solutions de transport non classiques qui s'imposent de plus en plus de nos jours, il serait intéressant d'envisager un déploiement de notre travail de thèse qui intégrerait les solution de co-voiturage, le Vlib, les taxis, voire de la mobilité électrique.

Bibliographie

[1] P. Adamson and E. Tick. Greedy partitionned algorithms for the shortest path problem. International Journal of Parallel Programming, 20 :271–298, 1991.

[2] R. Ahuja, J. Orlin, S. Pallottino, and M. G.Scutella. Dynamic shortest paths minimizing travel times and costs. Networks, 41 :205, 2001.

[3] R. Andrea and E. Matthias. A comparison of solution strategies for biobjective shortest path problems. Comput. Oper. Res., 36 :1299–1331, April 2009.

[4] G. Appa, L. Pitsoulis, and H. P.Williams. Handbook on modeling for discrete optimization. Springer, 2006.

[5] C. Appert and L. Santen. Modélisation du trafic routier par des automates cellulaires. Actes INRETS, 2004.

[6] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM, 45 :753–782, September 1998.

[7] H. Ayed, C. Galvez-Fernandez, Z. Habbas, and D. Khadraoui. Solving time-dependent multimodal transport problems using a transfer graph model. Computers & Industrial Engineering, 2010.

[8] J. Azevedo, S. Costa, M. Emilia, and S. Madeira. An algorithm for the ranking of shortest paths. European Journal of Operational Research, 69(1) :97–106, 1993.

[9] F. Baccelli, G. Cohen, G. Olsder, and J. Quadrat. Du grafcet aux réseaux de petri. Wiley, 1992.

[10] S. Balev, F. Guinand, and P. Yoann. Maintaining shortest paths in dynamic graphs. In NCP07 Nonconvex Programming Local & Global Approaches, pages 117–118, Rouen, France, 2007.

[11] R. Bartak. Constraint programming : In pursuit of the holy grail. in Proceedings of the Week of Doctoral Students (WDS99), 1999.

[12] V. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. Algorithmica, pages 97–105, 2009.

[13] R. Bellman. Dynamic programming. Princeton University Press, Princeton, NJ, 1957.

[14] J. Berger and M. Barkaoui. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research, 31(12) :2037–2053, 2004.

[15] D. Bertsekas. Nonlinear programming. Athena Scientific, Belmont, MA, 2004.

[16] D. Bertsekas, F. Guerriero, and R. Musmanno. Parallel asynchronous label-correcting methods for shortest paths. Journal of Optimization Theory and Applications, 88 :297–320, 1996.

[17] M. Bielli, A. Boulmakoul, and H. Mouncif. Object modeling and path computation for multimodal travel systems. European Journal Of Operational Research, 175(3) :1705–1730, December 2006.

[18] A. Bousquet. Optimisation d'itinéraires multimodaux fondée sur les temps de parcours à l'échelle d'une agglomération urbaine dense. PhD thesis, Ecole Doctorale : Mécanique, Energétique, Génie Civil et Acoustique, 2010.

[19] C. Brian. Shortest paths in fifo time-dependent networks : theory and algorithms. MIT, Cambridge, MA, 2004.

[20] G. Brodal and R. Jacob. Time-Dependent Networks as Models to Achieve Fast Exact Time-Table Queries, volume 92 of Electronic Notes in Theoretical Computer Science. Elsevier Science, 2003.

[21] G. S. Brodal and R. Jacob. Time-dependent networks as models to achieve fast exact time-table queries. Electronic Notes in Theoretical Computer Science. Elsevier,, 92, 2004.

[22] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. European Journal of Operational Research, 43(2) :216–224, 1989.

[23] I. Chabini. Discrete dynamic shortest path problems in transportation applications : Complexity and algorithms with optimal run time. Transportation Research Records, 1645 :170–175, 1998.

[24] J. Climaco, J. Craveirinha, and M. Pascoal. A bicriterion approach for routing problems in multimedia networks. Networks, 41(4) :206–220, 2003.

[25] J. Climaco and E. Martins. A bicriterion shortest path algorithm. European Journal of Operational Research, 11(4) :399–404, 1982.

[26] K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. Journal of Mathematical Analysis and Applications, 14 :493–498, 1966.

[27] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. MIT Press, Cambridge, MA, second edition, 2001.

[28] Q. L. D. Bolin, Y. Jeffrey. Finding time-dependent shortest paths over large graphs. In Proceedings of the 11th international conference on Extending database technology : Advances in database technology, EDBT 08, pages 205–216, New York, NY, USA, 2008. ACM.

[29] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Activity Analysis of Production and Allocation, 1951.

[30] R. David and H. Alla. Synchronization and linearity : an algebra for discrete event systems. Série Automatique, HERMES,, 1992.

[31] M. Dell'Amico, M. Iori, and D. Pretolani. Shortest paths in piecewise continuous time-dependent networks. Operations Research Letters, 36(6) :688–691, 2008.

[32] D. Delling. Time-dependent sharc-routing. Algorithmica, pages 1–35, 2008.

[33] P. Dell'Olmo, M. Gentili, and A. Scozzari. On finding dissimilar pareto optimal paths. European Journal of Operational Research, 162 :70–82, 2005.

[34] E. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1 :269–271, 1959.

[35] M. Dorigo, D. Caro, and L. Gambardella. Ant algorithms for discrete optimization. artif life. Spring, 5(2) :137–172, 1999.

[36] S. Dreyfus. An appraisal of some shortest-path algorithms. Operations Research, 17 :395–412, 1969.

[37] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum, pages 425–460, 2000.

[38] K. Evangelos, D. Yang, X. Tian, and Z. Donghui. Finding fastest paths on a road network with speed patterns. In Proceedings of the 22nd International Conference on Data Engineering, ICDE '06, page 10, Washington, DC, USA, 2006. IEEE Computer Society.

[39] C. Galvez-Fernandez, D. Khadraoui, H. Ayed, Z. Habbas, and E. Alba. Distributed approach for solving time-dependent problems in multimodal transport networks. Journal of Advances in Operations Research, pages 1–15, 2009.

[40] X. Gandibleux, F. Beugnies, and S. Randriamasy. Martins algorithm revisited for multi-objective shortest path problems with a maxmin cost function. A Quarterly Journal of Operations Research, pages 47–59, 2006.

[41] M. R. Garey and D. S. Johnson. Computers and Intractability ; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

[42] R. Garroppo, S. Giordano, and L. Tavanti. A survey on multi-constrained optimal path computation : Exact and approximate algorithms. Comput. Netw., 54 :3081–3107, December 2010.

[43] F. Glover. Improved linear integer programming formulations on nonlinear integer programs. Management Science, 22 :455–460, 1975.

[44] F. Glover. Animproved mip formulation for products of discrete and continuous variables. Journal of Information and Optimization Sciences, 5 :469–471, 1984.

[45] M. Gondran and M. Minoux. Graphes, dioides et semi-anneaux. TEC & DOC, Paris, 2002.

[46] T. Grabener. Calcul d'itinéraire multimodal et multiobjectif en milieu urbain. PhD thesis, Université de Toulouse 1 Capitole, 2010.

[47] T. Grabener. Calcul d'itinéraire multimodal et multiobjectif en milieu urbain. PhD thesis, Université Toulouse 1 Capitole, 2010.

[48] F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. Journal of Optimization Theory and Applications, 111 :589–613, 2001.

[49] M. Hannemann, Matthias, F. Schulz, W. Dorothea, and C. Zaroliagis. Timetable information : Models and algorithms. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis, editors, Algorithmic Methods for Railway Optimization, volume 4359 of Lecture Notes in Computer Science, pages 67–90. Springer Berlin / Heidelberg, 2007.

[50] P. Hansen. Multiple criteria decision making : theory and application. Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 177 :109–127, 1980.

[51] R. Hassin. Approximation schemes for the restricted shortest path problem. Mathematics of Operations Research, 17 :36–42, 1992.

[52] M. Hershel and B. Orlin. Fast approximation schemes for multi-criteria combinatorial optimization. In Sloan School of Management, Massachusetts Institute of Technology, Working Paper, pages 124–134, 1995.

[53] M. Hizem. Recherche de chemins dans un graphe à pondération dynamique. PhD thesis, École Centrale de Lille, 2008.

[54] D. Hochbaum. Approximation algorithms for np-hard problems. International Thomson Publishing, 1996.

[55] M. Hribrar, V. Taylor, and D. Boyce. Implementating parallel shortest path for parallel transportationapplications. Parallel Computing, 27 :1537–1568, 2001.

[56] B. Huang, Q. Wu, and F. Zhan. A shortest path algorithm with novel heuristics for dynamic transportation networks. Int. J. Geogr. Inf. Sci., 21 :625–644, January 2007.

[57] R. Impagliazzo. A personal view of average-case complexity. In Proceedings of the 10th Annual Structure in Complexity Theory Conference (SCT'95), Washington, DC, USA, 1995. IEEE Computer Society.

[58] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments—a survey. IEEE Transactions on Evolutionary Computation, 9 :303–317, 2005.

[59] Y. Jungkeun, N. Brian, and L. Mingyan. Surface street traffic estimation. In Proceedings of the 5th international conference on Mobile systems, applications and services, MobiSys 07, pages 220–232, New York, NY, USA, 2007. ACM.

[60] S. A. K. Deb, A. Pratap and T. Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. IEEE Trans. on Evol. Computation, 6, 2002.

[61] N. K. Kamarkar. A new polynomial-time algorithm for linear programming. Combinatorica, 4 :373–395, 1984.

[62] M. Kangas, M. Hippi, and J. Ruotsalainen. The fmi road weather model. HIRLAM Newsletter, 51 :117–123, 2006.

[63] R. Korf. Depth-first iterative-deepening : An optimal admissible tree search. Artificial Intelligence, page 97–109, 1985.

[64] M. M. Kostreva and M. M. Wiecek. Time dependency in multiple objective dynamic programming. Journal of Mathematical Analysis and Applications, 173 :289–307, 1993.

[65] J. Lemesre, C. Dhaenens, and E. Talbi. Parallel partitioning method (ppm) : A new exact method to solve bi-objective problems. Computers & Operations Research, 34(8) :2450–2462, 2007.

[66] A. Lozano and G. Storchi. Shortest viable path algorithm in multimodal network. Transportation Research Part A, 35 :225–241, 2001.

[67] E. Martins. On a multicriteria shortest path problem. European Journal of Operational Research, 16 :236–245, 1984.

[68] E. Martins, J. Paixao, M. Rosa, and J. Santos. Ranking multiobjective shortest paths. Pré-publicacoes do Departamento de Matemática 07-11, Universidade de Coimbra, 2007.

[69] E. Martins and J. Santos. The labeling algorithm for the multiobjective shortest path problem. Departamento de Matemática, Departamento de Matematica da Universidade de Coimbra, 1999.

[70] R. Mohring. Graphentheoretische modelle und algorithmen. Angewandte Mathematik – insbesondere, pages 192–220, 1999.

[71] J. Mote, I. Murthy, and D.Olson. A parametric approach to solving bicriterion shortest path problems. European Journal of Operational Research, 53(1) :81–92, 1991.

[72] M. Muller-Hannemann and M. Schnee. Finding all attractive train connections by multicriteria pareto search. Proceedings of the 4th Workshop in Algorithmic Methods and Models for Optimization of Railways (ATMOS 2004),, 2004.

[73] M. Muller-Hannemann and K. Weihe. Pareto shortest paths is often feasible in practice. Algorithm Engineering, Springer, 2141 :185–198, 2001.

[74] Nachtigal. Time depending shortest-path problems with applications to railway networks. European Journal of Operations Research, 83 :154–166, 1995.

[75] A. Nait-Sidi-Moh. Contribution à la Modélisation, à l'Analyse et à la Commande des Systèmes de Transport Public par les Réseaux de Petri et l'Algèbre (Max, Plus). PhD thesis, L'UNIVERSITE DE TECHNOLOGIE DE BELFORT-MONTBELIARD, 2003.

[76] G. Nannicini, D. Delling, L. Liberti, and D. Schultes. Bidirectional a^* search for time-dependent fast paths. Lecture Notes in Computer Science, 5038, 2008.

[77] G. Nemhauser and L. Wolsey. Integer and combinatorial optimization. Wiley, 1999.

[78] Y. Nie and W. Xing. Shortest path problem considering on-time arrival probability. Transportation Research Part B : Methodological, 43(6) :597–613, July 2009.

[79] A. Orda and R. Raphael. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J. ACM, 37 :607–625, July 1990.

[80] A. Orda and R. Raphael. Minimum weight paths in time-dependent networks. Networks : An International Journal, 21, 1991.

[81] A. Osvald and L. Stirn. A vehicle routing algorithm for the distribution of fresh vegetables and similar perishable food. Journal of Food Engineering, 85(2) :285–295, 2008.

[82] J. Paixao and J. Santos. Labelling methods for the general case of the multi-objective shortest path problem - a computational study. a computational study, Prépublicaçõesdo Departamento de Matemática, Universidade de Coimbra, pages 07–42, 2007.

[83] J. Paixao and J. Santos. A new ranking path algorithm for the multiobjective shortest path problem,. Pré-publicaoes do Departamento de Matemática, pages 08–27, 2008.

[84] S. Pallottino and M. Scutella. Shortest path algorithms in transportation models : classical and innovative aspects. Kluwer, 1997.

[85] S. Pallottino and M. Scutella. Shortest path algorithms in transportation models : Classical and innovative aspects, 1998.

[86] C. Papadimitriou and K. Steiglitz. Combinatorial optimization : Algorithms and complexity. Prentice-Hall, New York, 1982.

[87] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. Technical report, DELIS Project, Universitat Paderborn, Germany, 2004.

[88] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards realistic modeling of time-table information through the time-dependent approach. Electronic Notes in Theoretical Computer Science, 92 :85–103, 2004.

[89] E. Pyrga, D. Wagner, and C. Zaroliagis. Experimental comparison of shortest path approaches for timetable information. Proceedings of the Sixth Workshop on Algorithm Engineering and, pages 88–99, 2004.

[90] B. Raney, M. Balmer, K.Axhausen, and K. Nagel. Agent-based activities planning for an iterative traffic simulation of switzerland. Proc of 10th International Conference on Travel Behaviour Research, Lucerne, 2003.

[91] S. Russell and P. Norvig. Artificial intelligence : A modern approach. Prentice-Hall, 1995.

[92] S. Sahni. General techniques for combinatorial approximation. Operations Research, 25 :920–936, 1977.

[93] V. Sastry, T. Janakiraman, and S. Ismail. New polynomial time algorithms to compute a set of pareto optimal paths for multi-objective shortest path problems. International Journal of Computer Mathematics, 82(3) :289–300, 2005.

[94] T. Schoemaker, K. Koolstra, and P. Bovy. Traffic in the 21st century - a scenario analysis for the traffic market in 2030. Weijnen M.P.C., E.F. ten Heuvelhof, The infrastructure playing field, pages 174–194, 1999.

[95] F. Schulz. Timetable Information and Shortest Paths. PhD thesis, Universitat Karlsruhe (TH), Fakultat Informatik, 2005.

[96] F. Schulz, D. Wagner, and K. Weihe. An empirical case study from public railroad transport. PhD thesis, Journal of Experimental Algorithmics, 2000.

[97] J. Skriver and K. Andersen. A label correcting approach for solving bicriterion shortest-path problems. Computers & OR, pages 507–524, 2000.

[98] R. Sperb. Solving time-dependent shortest path problems in a database context. PhD thesis, INTERNATIONAL INSTITUTE FOR GEO-INFORMATION SCIENCE AND EARTH OBSERVATION ENSCHEDE, THE NETHERLANDS, March 2010.

[99] M. Tai-Yu. Modèle dynamique de transport basé sur les activités. PhD thesis, ECOLE NATIONALE DES PONTS ET CHAUSSEES, 2007.

[100] E. Talbi. Metaheurisrics From Design to Implementation. Wiley, first edition, 2009.

[101] Y. Tang, Y. Zhang, and H. Chen. A parallel shortest path algorithm based on graph partioning and iterative correcting. pages 155–161. The 10th IEEE International Conference on High Performance Computing and Communications, 2008.

[102] E. Tsang. Foundations of constraint satisfaction. Academic Press, London, 1995.

[103] R. Vaughan. Urban spatial traffic patterns. Pion Limited, London, 1987.

[104] V. Vazirani. Approximation algorithms. Springer, 2003.

[105] D. Veldhuizen and G. Lamont. Multiobjective evolutionary algorithm research : A history and analysis. Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, Tech. Rep, (TR-98-03), 1998.

[106] P. Vincke. Problémes multicritéres. Cahiers du Centre d'Etudes de Recherche Opérationelle, 16 :425–439, 1974.

[107] M. Visee, J. Teghem, M. Pirlot, and E. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. J. of Global Optimization, 12 :139–155, March 1998.

[108] L. Zhao and T. Ohshima. a^* algorithm for the time-dependent shortest path problem. WAAC08 : The 11th Japan-Korea Joint Workshop on Algorithms and Computation,, 2008.

[109] A. Ziliaskopoulos, D. Kotzinos, and H. Mahmassani. Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications. Transpn Res, 5(2) :95–107, 1997.

[110] A. Ziliaskopoulos and W. Wardell. An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. European Journal of Operational Research, 125 :486–502, 2000.

Résumé

Dans cette thèse, nous nous intéressons à la problématique de transport usager dans un contexte multimodal, multi-objectif et dépendant du temps. Dans ce contexte nous proposons particulièrement des solutions de calcul d'itinéraires, qui satisfont dans la mesure du possible, les vœux des usagers. D'un point de vue théorique, ce problème s'avoisine très fort au problème du plus court chemin, bien connu en théorie des graphes.

Notre première contribution porte sur la définition du graphe de transfert, un modèle de représentation des réseaux multimodaux qui est en adéquation avec la nature distribuée de tels réseaux. Sur base de ce modèle, cette thèse propose plusieurs algorithmes de calculs d'itinéraires multimodaux et dépendant du temps mais simplement mono-objectifs. Les deux premiers consistent en la construction préalable d'une base de donnée mémorisant toutes les étapes fixes du calcul d'itinéraire. Ces deux algorithmes diffèrent par la façon dont cette base de données est construite. Les résultats expérimentaux ont montré que le calcul de la base de donnée, via des méthodes exactes ou méta-heuristiques, se limite à des réseaux de petite taille, faute d'espace mémoire ou de temps d'exécution important. Pour faire face à ce problème, nous avons proposé un nouvel algorithme hybride qui, sans avoir recours à la construction de la base de donnée, combine les avantages des des méthodes exactes et méta-heuristiques. Cet algorithme améliore considérablement les deux premiers, mais pas encore de façon suffisante pour traiter des réseaux de très grande taille. Pour ce dernier point, notre principale contribution est la proposition d'un algorithme parallèle, approche tirant ses avantages de la décomposition naturelle des réseaux de transport réel en fonction de modes et des régions. Nous avons expérimenté cette approche sur une machine parallèle à mémoire partagée. Les résultats des expériences sur les graphes aléatoires théoriques ainsi que sur les réseaux de transport réel sont très prometteurs.

Toujours dans le soucis de faire face aux exigences des usagers, nous nous intéressons dans une deuxième partie de cette thèse au problème mult-objectif. Nous avons expérimenté dans un premier temps, la version dépendante du temps de l'algorithme exact de Martins, basé sur le calcul des fronts pareto. Nous avons pu mettre en évidence ses limites du point de vue pratique. Nous avons ensuite proposé une solution basée sur les algorithmes génétiques (algorithme NSGA-II). Comme pour le cas mono-objectif ces deux approches restent limitées faute de temps ou d'espace. L'algorithme hybride combinant la rapidité des méta-heuristiques et la complétude des méthodes exactes a donné de meilleurs résultats. Toutes nos solutions ont été validées théoriquement et expérimentalement, aussi bien sur des réseaux académiques que des réseaux de transport réel dans le cadre du projet Carlink.

Mots-clés: Transport multimodal dépendant du temps, Plus court chemin, Algorithme exact, Métaheuristique, Algorithme parallèle

Abstract

The main focus of this thesis is about multi-modal, multi-objective and time-dependent in passengers transport networks. In this context, we particularly propose

itineraries processing solutions and route planning services that satisfy the users needs, as much as possible. In the area of graph theory, this issue is similar to the well known shortest path problem.

The first part of our contributions begins with the definition of the Transfer-Graph model that is consistent with the distributed nature of multi-modal transport networks. Based on this model, we propose several itineraries processing mono-objective and time-dependent algorithms. The two first algorithms consist in building a database storing intermediate data required to the actual itinerary processing. They are distinguished since they use different ways of building the intermediate database. The experimental results of these two algorithms show that building the intermediate database using exact methods (based on Dijkstra) or meta-heuristics (based on ACO) are only be applicable to transport networks of small size. This is due to either the memory space problem and to the excessive response time. To solve this problem, we proposed a new hybrid algorithm, that does not require any intermediate database, which combines the advantages of exact methods and meta-heuristics. This algorithm significantly improves the first two ones, but this is not sufficient to deal with a very large transport networks. To deal this last question, our main contribution is to develop a new approach based on parallel algorithms, taking into account the advantages of the distributed nature of transport networks considering real transport modes and on the geographical regions where they are basically deployed. We experimented this approach on a shared memory parallel computer. The results of experiments based on random graphs as well as on real transport networks are very promising.

In order to satisfy the growing users needs we have been interested, in a second part of this thesis, in developing multi-objective solutions to satisfy more constraints at the same time. We first experimented the time-dependent version of an exact algorithm based on Martins which generates Pareto fronts and were able to show its practical limits. We then proposed a solution based on a genetic algorithm (NSGA-II). As with the mono-objective use case, both of these approaches are limited because of either excessive time response or memory space limit. The hybrid algorithm which combines the speed of meta-heuristics and completeness of exact methods, provide better results.

Our solutions have been validated in theory using simulations but also in real experiment considering OpenStreetMap data in the frame of the European Carlink project.

Keywords: Multi-modal and time-dependent transport, Shortest Path Problem , exact algorithm, Meta-heuristic, Parallel algorithm