Algorithmes pour les problèmes de tournées à la demande

Xiagang Zhao

N° d'ordre : D.U : 2125
EDSPIC : 521

Université Blaise pascal – Clermont-Ferrand II

Ecole doctorale
Sciences Pour l'Ingénieur de Clermont-Ferrand

Thèse

Présentée par

Xiagang ZHAO

Pour l'obtention du Grade de

Docteur d'Université

Spécialité : Informatique

Algorithmes pour les problèmes de tournées à la demande

Directeur de la thèse :
Philippe LACOMME Maître de conférences, HDR, Université Blaise Pascal
Alain QUILLIOT Professeur, Université Blaise Pascal
Membres du Jury :
Pierre CASTAGNA Professeur, Université de Nantes
Aziz MOUKRIM Professeur, Université de Technologie de Compiègne
Eric SANLAVILLE Professeur, Université du Havre
Hélène TOUSSAINT Ingénieur de Recherche CNRS, LIMOS

Thèse préparée au sein du laboratoire LIMOS (Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes) Unité Mixte de Recherche 6158 du CNRS .

LIMOS
Complexe scientifique des Cézeaux,
63173 AUBIERE cedex, FRANCE.

REMERCIEMENTS

Je tiens tout d'abord à remercier mes deux directeurs de thèse, Alain QUILLIOT et Philippe LACOMME, pour m'avoir encouragé dans cette voie. Ils sont des encadrants exceptionnels de par leurs compétences, leur patience et leur gentillesse.

Je remercie les membres du jury de ma thèse pour l'intérêt qu'ils ont porté à mon travail et pour leur disponibilité.

Pour réaliser ce document, j'ai largement bénéficié de l'aide d'Hélène TOUSSAINT. Je tiens à la remercier très sincèrement pour les heures qu'elle a passé à relire ce document et pour tous les conseils qu'elle m'a donné.

Merci à tous les permanents et thésards qui m'ont accompagné pendant ces années. Merci à tous pour votre aide, votre soutient.

Résumé

Dans le cadre de cette thèse, nous nous intéressons au problème du transport à la demande. Nous proposons des heuristiques pour résoudre ce problème de manière rapide et efficace. Dans cette thèse, nous traitons trois problèmes : le premier est le Dial-a-ride (DARP standard). Pour ce problème, nous proposons des heuristiques basées sur la technique d'insertion et une technique de propagation de contrainte. Nous proposons aussi la procédure SPLIT et des opérateurs classiques de recherche locale pour résoudre ce problème. Le second est le DARP multicritères pour laquelle nous proposons un schéma de type ELS. Le troisième est un problème de transport à la demande avec contraintes financières (DARPF), qui est une extension de DARP. Nous résolvons ce problème grâce à une heuristique d'insertion et une technique de propagation de contraintes. La fonction objectif détermine les caractéristiques des tournées. Des résultats expérimentaux montrent que nos (méta-) heuristiques donnent des résultats plus favorables aux clients (meilleure qualité de service)

Mots-clés : heuristique, méta-heuristique, transport, ordonnancement.

ABSTRACT

As part of this thesis, we investigate the vehicle routing problem. We propose heuristics to solve this problem quickly and efficiently. In this thesis, we deal with three problems: the first is the Dial-a-ride problem. For this problem, we propose heuristics based on the technique of insertion and a constraint propagation technique. We propose also the procedure SPLIT and some operators of local research to solve this problem. The second is the multi-criteria DARP for which we propose an ELS framework. The third is a DARP problem with financial constraints (DARPF), which is an extension of DARP. We solve this problem thanks to insertion heuristics using a constraint propagation technique. The objective function determines the characteristics of the tour. Experimental results show that our (meta-) heuristics give results more favorable to customers (better quality of service)

Keywords: heuristics, meta-heuristics, transportation, scheduling.

Sommaire

LISTE DES ALGORITHMES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

1. Contexte de mobilité

La mobilité tient une place importante dans notre société et dans nos vies quotidiennes. Il s'agit de modes de transport individuel (vélo, marche à pied,…), de modes de transport en communs (bus, train,…) ou de modes de transport flexibles qui s'adaptent à des zones moins densément peuplées (transport à la demande, covoiturage,…). Pour la plupart des personnes, les déplacements sont liés à l'automobile. En espace rural, où les déplacements se font sur des distances relativement longues et où la population est dispersée, la voiture est utilisée dans 90% des cas; il en va de même en espace périurbain, où les transports sont souvent peu structurés et les déplacements importants. Enfin, en espace urbain dense, bien que l'offre de transport soit plus forte, l'automobile est là encore un véhicule omniprésent.

Les problèmes liés à l'organisation de la mobilité sont étiquetés comme « **Problèmes de tournées** ». Les problèmes de tournées sont une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Nous nous intéressons ici à la perspective de nouveaux services de transport (fret, personnes ciblées, desserte de zones rurales, sites dédiés, véhicules autonomes), pilotés de façon à réagir à une demande fluctuante dans le temps, selon une logique qui combine service individualisé et mutualisation (minibus, navettes, ...). La cible applicative des services innovants pour la mobilité est un paysage de la mobilité en milieu urbain, périurbaine ou rural. A présent, la mobilité subit de fortes évolutions d'origine sociale, économique et technologique.

1.1. Evolution du contexte socio-économique

Les évolutions de la demande de transport sont dépendantes de l'environnement économique. La croissance économique est la principale variable explicative des modèles de transport. Nos

sociétés doivent faire face à un ensemble de défis dont chacun place l'organisation et les technologies de la mobilité au cœur des réflexions :

- Le renchérissement croissant des énergies fossiles et les fortes hausses du cout de ces ressources. On peut penser que ces énergies vont poser deux problèmes principaux. Le premier est que leur rareté va créer des tensions géopolitiques dans le monde. Le second problème est que les émissions de CO2, qu'elles engendrent, sont élevées et contribuent fortement au réchauffement climatique ;

- Le réchauffement climatique et le recyclage de certains déchets : le secteur des transports est aujourd'hui responsable de la quasi-totalité des émissions d'oxydes d'azote, des trois quarts du monoxyde de carbone, de la moitié des composés volatiles non méthaniques et d'environ 85% des particules ;

- Les phénomènes de congestion urbaine : en plus de réduire la vitesse de circulation, un embouteillage est une source importante de pollution atmosphérique, due à une consommation supplémentaire de carburant ;

- L'urbanisation et la densité de population : les concentrations urbaines sont de plus en plus importantes ;

- Le renouvellement des générations et le vieillissement de la population : 80% des ménages sont aujourd'hui équipés d'automobiles et le marché ne progresse plus que par la multi-motorisation.

L'importance prise par ces défis tend enfin à pousser les acteurs public à intervenir de plus en plus dans l'organisation de la mobilité. Ces défis sont à l'origine d'une recherche de nouveaux services innovants et de nouvelles technologies.

1.2. Evolution de la technologie

Parallèlement aux évolutions environnementales, les vingt dernières années ont été marquées par d'importantes évolutions technologiques, principalement liées à l'énergie et aux technologies des l'information. Ces évolutions concernent les technologies de la mobilité autonome, des moteurs électriques, des télécommunications mobiles et embarquées, et rendent possible une nouvelle organisation de la mobilité. L'évolution peut être divisée en deux classes :

La première classe est liée aux véhicules eux-mêmes :

- Emergence de nouvelles classes de véhicules électriques, autonomes ou semi-autonomes ;
- Rapidité des transports en forte croissance ;

- Emergence de nouvelles formes de tramway (à pneu, à guidage monorail, autonome).

La seconde classe est liée aux dispositifs de suivi et de supervision susceptibles de permettre la mise en œuvre et l'optimisation de nouveaux services :

- Progrès dans les techniques de communication mobile et embarquée ;
- Montée en puissance des technologies web et des services associés ;
- Progrès réalisés dans le domaine de la localisation GPS et de la cartographie ;
- Progrès réalisés dans le domaine des systèmes répartis.

En général, il existe plusieurs freins à la diffusion de ces nouvelles technologies dans la réalité économique : le changement de technologie coûte généralement plus cher, la diffusion des nouvelles technologies est donc très lente. Aujourd'hui, un phénomène peut aider ces nouvelles technologies à se diffuser plus rapidement, c'est la prise de conscience des problèmes environnementaux par les consommateurs. Par ailleurs, les personnes recherchent des services plus économiques, flexibles et proches de leurs besoins. Cependant, les transports publics ne répondent que partiellement à ces besoins. C'est pourquoi des services innovants ont été engendrés.

1.3. Evolution de la demande de transport

L'évolution de la demande de transport est déterminée par l'évolution des coûts du transport. Ces coûts comprennent aussi bien les coûts financiers que les coûts en temps, et sont influencés par les politiques mises en œuvre par les pouvoirs publics. Parallèlement à cela, le coût environnemental des transports est considéré :

- Le coût financier : il s'agit du coût de l'utilisation d'une voiture (coût de l'achat du véhicule, coût annuel de son utilisation et coût variable de l'utilisation de la voiture) ;
- Le coût en temps : il s'agit du temps de transport qui englobe les composantes suivantes : le temps passé dans le véhicule, le temps de marche et d'attente ;
- Le coût environnemental : il s'agit de l'impact sur la pollution atmosphérique et sur le changement climatique. Il faut effectuer une distinction entre les émissions directes (énergie et émissions de CO2 pendant l'usage du véhicule) et les émissions indirectes du transport (énergie dépensée et émissions de CO2 pour produire une unité d'énergie de carburant et la distribuer jusqu'au véhicule).

Dans la section suivante, nous évoquons ce que nous entendons par des nouveaux services de la mobilité.

2. Nouveaux services de la mobilité

Le contexte évoqué à ci-dessus induit le besoin pour l'émergence de nouveaux services d'aide à la mobilité des biens et des personnes. Des nouveaux services pour des nouvelles catégories de véhicules exploitent à la fois la notion de multi-modalité et correspondent à des ciblages « client ». Ils ont vocation à se situer entre la logique du transport individuel pur et celle du transport collectif, et ils doivent promouvoir la mutualisation tout en gardant un certain nombre de caractéristiques de flexibilité et de réactivité. De nouveaux comportements de mobilité se développent notamment dans les villes : la multi-modalité (l'usage de différents modes de transport sur un même parcours quand l'usager y trouve un avantage en temps ou financier) ou l'usage de voitures partagées dans les villes. Ces nouveaux services doivent se décliner selon des modes combinant :

- la flexibilité, la réactivité et l'inter-modalité;
- la sécurité, la sureté et la qualité de service;
- la viabilité économique et un niveau élevé de mutualisation.

Depuis une décennie, des nouveaux services et systèmes sont mis en œuvre en milieux urbain, périurbain ou rural. Nous en présentons quelques uns (on se référera à la liste bibliographique de la fin de cette section) :

- **TELEBUS** à BERLIN : Un système de transport à la demande pour les handicapés.
- **Lilas** à LILLE [**1**] : Un système de réservation de véhicules par abonnement en utilisant le téléphone ou internet.
- **Mobility** en SUISSE [**2**] : Un système de réservation de véhicules pour 55500 clients, les 1700 véhicules de 8 catégories différentes sont répartis pas sur 1000 emplacements dans toute la Suisse.
- **CityCarClub** en GRANDE BRETAGNE [**3**] : Un système de réservation de véhicules créé en 2000, avec plus de 350 véhicules et plus de 40 stations réparties.
- **Modulauto** à MONTPELLIER : Un système de location de voiture en libre service dans l'agglomération. Ce n'est pas un système de covoiturage mais de location simplifiée.
- **PRAXITELE** [**4**] : Un service public de transport, complémentaire des transports en commun et des taxis qui visent à offrir une alternative aux déplacements en voiture individuelle privée.
- **LISELEC** à LA ROCHELLE [**5**] : Un service par abonnement de location de voitures électriques. Créé en 1999 par PSA Peugeot Citroën, VIA GTI et Alcatel CGA.
- **Vélib, Autolib** [**6**] : un système de vélo (voitures propres électriques) en libre service dans certaines grandes villes de France.

Des nouveaux services existent dans autres pays hors Europe : **ZipCar** [**7**] aux USA, **Communauto** [**8**,**9**] en Amérique de Nord, etc.

Après des études sur ces nouveaux services et des observations effectuées en Hollande autour des services innovants basés sur la notion de « **véhicules partagés** », on peut conclure que ces systèmes induisent :

- une réduction importante (jusqu'à 50%) de leur taux de motorisation et du nombre de voyageur-kilomètres effectués en automobile ;
- une réduction de la consommation en énergie par client ;
- une réduction des émissions de gaz à effet de serre.

Ces nouveaux services jouent un rôle important dans l'organisation du transport. Ces services sont socialement profitables et sont capables de réduire les coûts. Ils sont plus attrayants pour les usagers parce qu'ils offrent un meilleur service. La consommation d'énergie est réduite par passager transporté.

Afin de répondre à ces nouveaux services, un certain nombre de véhicules intelligents a été élaboré. Le véhicule intelligent se dirige tout seul pourvu qu'il connaisse son itinéraire. C'est là qu'intervient le GPS, qui lui sert de guide de navigation. Des capteurs sont installés dans le véhicule intelligent pour mieux appréhender les conditions de circulation et adapter son comportement à son environnement. La Figure 1 présente un véhicule intelligent dit « Cycab » réalisé par l'INRIA. Il s'agit d'un véhicule en libre-service pour des quartiers interdits aux voitures particulières, proposant ainsi une alternative aux déplacements pédestres.

Figure 1 : Cycab à Clermont-Ferrand

3. Positionnement du Problème

Notre problème est lié à une problématique applicative de la mobilité réactive, et plus précisément au pilotage d'un système de transport à la demande (TAD). Ce système exige une

forte réactivité en particulier dans les zones faiblement desservies par les transports en commun (espace périurbain ou rural). On s'intéresse ici à un tel service structuré autour d'une flotte de véhicules, qui répond à des demandes en temps réel et qui fonctionne selon des délais de réaction face à la demande allant de quelques heures à quelques minutes. Les transports à la demande (TAD) sont un mode de transport particulier appartenant à une famille de services qui peut inclure aussi les taxis traditionnels ou collectifs, les programmes de bus scolaires, le covoiturage….

Les TAD se distinguent des services de transports collectifs car les véhicules n'empruntent pas d'itinéraire fixe et ne respectent pas un horaire précis, sauf pour satisfaire parfois un besoin particulier. Les TAD sont cependant organisés par des professionnels du transport, et à la différence des taxis, les voyages ne sont en général pas individuels.

Notre problème est un problème générique, dont nous pouvons faire différentes interprétations. Ce problème générique peut se présenter de la manière suivante : dans un espace donné (un réseau), on cherche à acheminer des demandes de clients à l'aide de moyens de transport pouvant combiner différents types de véhicules. Chaque demande du client est définie par un lieu d'origine, un lieu de destination, un nombre de passagers et des contraintes temporelles. Ces demandes peuvent être déterministes (connues à l'avance) ou stochastiques (non connues à l'avance). Plus précisément :

- à un instant donné t, un client fait une demande de transport à un véhicule afin qu'il vienne le chercher en un point de rencontre y à un instant t'. Le véhicule doit alors effectuer le transport des passagers en direction de la destination;

- à l'instant t', le véhicule arrive sur le point de rencontre y. Son itinéraire est affiché sur sa façade, ainsi que sur la borne de communication qui est localisée en y. Les passagers montent dans le véhicule. Le véhicule continue sa route, de façon à emmener ses passagers à leur destination finale, ou à un point de correspondance.

Un ou plusieurs critères de performances dont la qualité de service, le niveau de mutualisation, et les coûts de transport sont considérés. La gestion « intelligente » d'un tel système flexible induit alors un certain nombre de problèmes techniques, la plupart sont encore du domaine de la recherche.

3.1. Problème de conception de modèles et d'algorithmes

Ces problèmes concernent principalement l'algorithmique du routage et de l'allocation de charges, et renvoient aux modèles de tournées de véhicules, de transport à la demande (Dial-a-Ride : DARP), de Pick-up and Delivery (PDP), de Stacker-Crane. Ils peuvent être considérés selon deux modes :

- **Mode statique :** à l'instant t, on dispose des véhicules oisifs stationnés sur des « dépôts » ou sur des points de correspondance. Toutes les demandes des clients sont connues. Les routes sont alors calculées de façon à ce que le coût combiné des trajets soit minimal. Les véhicules appliquent le planning ainsi calculé. L'approche envisagée pour résoudre ce problème est une approche par optimisation combinatoire, telle que celle appliquée à des problèmes de tournées de véhicules. Nous traitons ces problèmes en introduisant une difficulté particulière : les véhicules peuvent procéder à des échanges de clients dans des points de correspondance, ou à des échanges de ressources.

- **Mode dynamique :** à un instant t', le système compte des véhicules actifs, en train d'effectuer des acheminements et des véhicules oisifs. Des nouvelles demandes de transport peuvent arriver à n'importe quel moment. Ces nouvelles demandes doivent alors être prises en compte en modifiant les tournées qui étaient prévues. De ce fait, le modèle doit être capable d'intégrer des imprévus : des tournées peuvent être replanifiées. Ce problème est un problème de type « Recherche Opérationnelle Embarquée », il concerne un problème de décision avec des contraintes sur des délais de réponse. La décision (routage, allocation) concerne deux points : le premier point de décision, qui concerne les différentes demandes, est essentiellement combinatoire et vise à concilier la qualité de service avec un impératif de mutualisation. La deuxième décision porte sur repositionnement des véhicules par rapport à une distribution de la demande dans le temps et dans l'espace. Il faut chercher à replacer les véhicules qui ne sont pas en trajectoire active et qui sont néanmoins en service de façon à les rapprocher de la demande prévisible.

3.2. L'enjeu de la réactivité

La **réactivité** ici est la capacité du système à répondre rapidement aux sollicitations de son environnement par la mise en œuvre de synergies ou par la flexibilité des ressources. La réactivité est donc un enjeu décisif pour l'attractivité des nouveaux systèmes de mobilité. Les échelles de réactivité sont classées en deux modes :

- **Mode faiblement réactif** (échelle de temps =1 jour) : le système organise les parcours de véhicules et le routage des clients, et tenant compte de l'existence des autres moyens de transport et de la possibilité d'échanges de clients entre véhicules. Le système cherche à optimiser le niveau de mutualisation, la fluidité, le confort d'acheminement, l'homogénéité et la pérennité des équipes ;

- **Mode fortement réactif :** (échelle de temps =1h, 30min, …) : dans ce mode, plusieurs cas doivent être pris en compte :
  - Une défaillance de l'offre chauffeur : il s'agit alors de rerouter les clients de la façon la plus efficace possible;

- Une défaillance du client : il s'agit de prévenir le chauffeur et de rerouter certains clients;
- Une offre pilote additionnelle : on peut envisager de rerouter certains clients;
- Une demande additionnelle du client : il faut raccrocher cette demande à un équipage existant et rerouter le véhicule correspondant.

3.3. L'identification des critères de qualité de service

En ce qui concerne les transports à la demande, les articles faisant référence à des critères d'optimisation de tournées de véhicules autre que le nombre de véhicules et/ou la distance totale parcourue sont très rares. En général, le TAD (le transport à la demande) est traité de façon purement « multicritère » par un somme pondérée à minimiser (maximiser) qui inclut l'évaluation de chaque critère ou l'agrégation d'un même critère pour un ensemble de passagers.

Nous considérons un TAD théorique pour lequel chaque client indique la date de chargement souhaitée, la date de déchargement souhaitée, le nombre de passagers transportés et une limitation de sa durée de transport. La société de transport dispose d'un nombre de véhicules disponibles au dépôt caractérisés par : une limitation de durée de service, un type et une capacité. Nous sélectionnons des critères que nous pensons représentatifs des problèmes posés. Les critères de qualité de service sont classés en trois dimensions :

Le premier concerne la personne qui peut être le passager. Les critères sélectionnés sont le temps de trajet, le temps d'attente pour commencer le service de chargement et déchargement, l'écart entre la date de chargement (déchargement) effective et la date de chargement souhaitée (déchargement), et le temps pour répondre à la demande du client.

Le second concerne les véhicules (les chauffeurs) ou la société de transport. Les critères sélectionnés sont la distance totale parcourue, le nombre de véhicules utilisés et le temps d'attente.

Un troisième critère commun à l'utilisateur et à l'opérateur, concerne la sécurité, c'est-à-dire la capacité du système à pallier les erreurs, de communication, de rendez-vous, etc.

3.4. Problèmes lié à la conception de l'architecture de la supervision et de la communication

La supervision est une technique de suivi et de pilotage informatique. Elle est la surveillance du bon fonctionnement d'un système de transport. Le fonctionnement principal est suivant : le client communique avec un superviseur, par le téléphone (signal parlé), internet ou via une

borne (signal tactile). Ce superviseur dispose d'une vision sur l'état du système. Il peut refuser ou accepter la demande. Dans ce dernier cas, il peut commander le déroutage d'un des véhicules, ce qui donne lieu à un calcul de synchronisation, et à une évaluation de la fenêtre du temps qui verra la transaction s'effectuer. Il transmet ensuite sa proposition aux véhicules concernés et au demandeur. L'autre fonctionnement à envisager est le suivant : le client communique directement avec les véhicules et sa demande est filtrée en direction de ceux qui se trouvent à un certain niveau de proximité de client. Ceux-ci se concertent alors avant de transmettre une réponse à client.

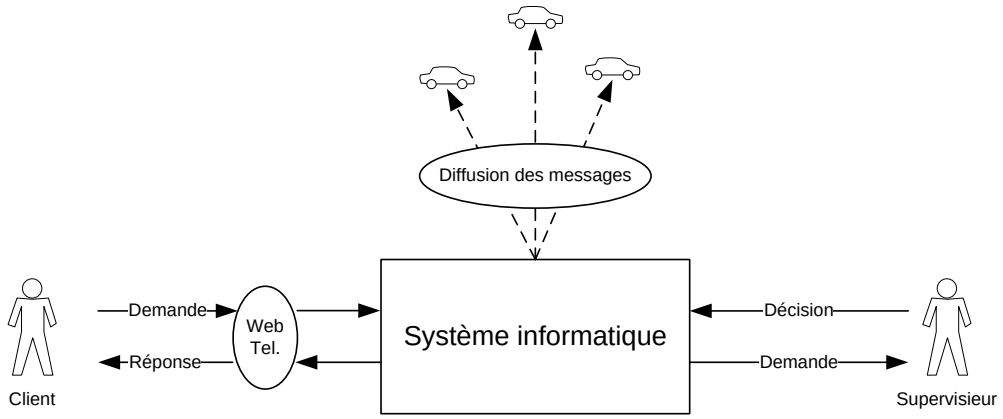

Figure 2 : Architecture de Supervision et de Communication

La communication porte ici sur la diffusion de messages entre superviseurs, véhicules et clients de plusieurs façons (téléphone, web, ad hoc,…, etc.). Une des principales difficultés porte ici sur l'articulation entre les dispositifs de communication réactive et les modules de planification.

Autour de l'architecture de supervision et de communication (voir la Figure 2), quatre problèmes doivent être traités :

- **Problème de communication entre superviseurs, véhicules et clients**

  La communication avec le client : identification de la demande qui donne lieu à un enregistrement, rejet ou acquittement de la demande. Cette communication prend également en compte l'identification du lieu de rendez-vous : le client doit pouvoir reconnaître son véhicule et celui-ci doit reconnaître son client, le client doit pouvoir comprendre le parcours qui lui est proposé dans le cas où une correspondance est envisagée.

Notons que certaines demandes peuvent être abandonnées, c'est-à-dire que le client ne donne pas suite, sans pour autant en informer le système.

Le traitement des embarquements « à la volée » ne sont pas intégrés dans une logique de réservation. Ceux-ci peuvent être prohibés, mais cela n'est sans doute pas souhaitable. Dans le cas où un tel embarquement est accepté, il doit pouvoir être identifié, et la destination souhaitée par le nouveau passager doit pouvoir être prise en compte de façon optimale. Egalement, dans le cas où cet embarquement à la volée rend impossible la prise en compte complète d'un autre embarquement programmé, cette situation doit être identifiée et la satisfaction effective de la demande ainsi retardée doit être assurée.

- **Problèmes de communication inter-véhicules**

Les problèmes de communication inter-véhicules sont principalement associés à la gestion des dépassements en temps au niveau des aires de dégagement, des opérations d'embarquement et de débarquement sur les aires de dégagement. Elles doivent permettre une gestion des correspondances pour les clients, les véhicules et le traitement en simultané sur une même aire de dégagement, de plusieurs opérations d'embarquement et débarquement.

Les principaux problèmes portent sur la fiabilité des correspondances, sur la nature du dialogue entre système et véhicules et entre véhicules qui effectuent des opérations de reroutage. Le système porte aussi sur la possibilité d'échanges de clients entre véhicules, sur la prise en compte de l'inter-modalité et sur sa synchronisation.

- **Problèmes de conception des véhicules**

Les véhicules doivent être conçus, tant au niveau de procédés de guidage qu'au niveau des dispositifs de sécurité mis en place afin de répondre à des défaillances où à des situations perturbées. De cette façon on permet leur intégration par rapport à un service réalisant les fonctionnalités décrites ci-dessus, dans les conditions de sécurité requises par le contexte d'utilisation.

- **Problèmes de traçabilité**

Un tel système doit par ailleurs permettre la traçabilité de l'opération de partage de véhicules, de façon à permettre à la fois la tarification, et l'anticipation de la probabilité de la demande.

4. Les objectifs de la thèse

Nous nous intéressons ici à la perspective de nouveaux services de transport (fret, personnes ciblées, desserte de zones rurales, sites dédiés, véhicules autonomes), pilotés de façon à réagir à une demande fluctuante dans le temps, selon une logique qui combine service individualisé et mutualisation. La thèse concerne l'aide logicielle au pilotage de systèmes intelligents de transport à la demande (TAD) sous forme de transport souple combiné à la technologie embarquée. Le problème étudié dans cette thèse est lié à un aspect particulier du transport à la demande qui est celui du routage des demandes, dit **DARP** (Dial-a-Ride Problem). On se concentrera ici sur les aspects algorithmiques. Globalement, ce problème devra être connecté à la conception d'outils informatiques pour le suivi et le contrôle en temps réel d'une flotte de véhicules, son mode de supervision et son mode de communication avec l'usager cible. Ces outils vont mettre en jeu des techniques de communication mobile, sur lesquelles s'appuiera un noyau décisionnel proprement dit, c'est-à-dire un ensemble de procédures qui effectueront le routage des véhicules et l'affectation des demandes sur ces véhicules en vue d'une bonne mutualisation de la ressource entre les demandes. Ce noyau constitue le cœur du sujet de ce travail.

Le problème DARP tel que l'on va le traiter ici, concerne un réseau urbain, rural où périurbain qui est représenté par un graphe complet. Des véhicules (oisifs ou actifs) avec la capacité limitée sont basés en certains des sommets du graphe (dépôts ou points de desserte). La demande de transport émanant d'un client contient un couple de lieu (l'origine et la destination), une quantité à charger, des fenêtres du temps sur l'embarquement et le débarquement et le temps maximal de parcours. Pour chaque client, une réponse reçue est : soit une réponse de refus, soit une réponse d'acceptation. Si la demande est acceptée, la réponse contient une date d'embarquement, un nom de véhicule et un temps de parcours. Une large variété de mesure de performance est utilisée dans ce problème. Il s'agit du coût économique, de la fiabilité et de la qualité de service. Ce problème est un problème de recherche opérationnelle.

Le problème DARP peut être vu selon en deux modes : statique ou dynamique. Pour le premier mode (DARP statique), toutes les demandes de transport sont prévues avant le début du service et on applique simplement le planning calculé. En revanche, dans le modèle dynamique, on travaille sur un système en constante évolution, qui génère un flot de données. Les routes et les ordonnancements doivent être déterminés en temps réel et en synchronisation avec le système en mouvement. Les solutions du DARP statique constituent souvent les bases pour le DARP dynamique : on en extrait des règles de décision adaptables au cas dynamique. Le problème DARP peut avoir plusieurs variantes qui concernent :

- les objets transportés et la capacité des véhicules ;
- la « préemption » : ces charges sont divisibles, et/ou on peut transporter une même charge avec 2 véhicules ;
- les contraintes globales : ressources globales (argent, …).

Le DARP est une généralisation du problème de tournées de véhicules, et fait partie de la classe des problèmes NP-difficiles. Comme pour la plupart des problèmes NP-difficiles, il est difficile de trouver une solution optimale parmi un grand nombre de solutions admissibles. On se contente alors de trouver des solutions de « bonne qualité ». On utilise soit des méthodes exactes (qui cherchent l'optimum), soit des méthodes approchées (heuristique) sont utilisées pour obtenir une solution qui est proche de l'optimum ou de « bonne qualité » dans des temps de calculs raisonnables.

Dans le premier chapitre, nous présentons un état de l'art pour le PDP (Pick-up and Delivery Problem) et pour le DARP. Le DARP est une généralisation du problème de PDP et VRP (Vehicle Routing Problem) où les charges sont des personnes transportées en groupe ou individuellement. Dans le DARP, le temps de trajet pour le client (Riding Time) est considéré comme un objectif. Le DARP est considéré comme un cas particulier du transport à la demande (TAD) au sens **transport en commun routier**. Dans le DARP, le client (passager) spécifie une demande pour une opération de transport. Une demande du client consiste en : le lieu d'origine pour le chargement des passagers et le lieu de destination pour le déchargement des passagers. Une fenêtre de temps indique à quel moment la demande doit être transportée. Le client propose aussi le nombre de passagers transportés, la date de départ et le date d'arrivée. Le but du DARP porte alors sur la conception de tournées de véhicules qui respectent les contraintes temporelles et qui satisfait toutes les demandes.

Dans le second chapitre, nous formalisons les problèmes de DARP que nous allons étudier. Nous discutons : des contraintes de temps (liées aux demandes de transport ou liées aux véhicules) et les contraintes de capacité. Nous discutons également de la fonction objective. Nous allons étudier le DARP standard: Il s'agit d'un modèle de DARP proposé par Cordeau et Laporte (2003). Nous proposons des méthodes heuristiques basées la technique d'insertion pour le DARP standard. Nous testons le schéma d'insertion :

1. le client est choisi au hasard (ou par un ordre) et on l'insère dans la tournée active (celle de durée minimale);
2. le client est choisi au hasard et on essaye de l'insérer dans toutes les tournées actives ;
3. si l'insertion échoue, alors une procédure de réparation est appliquée.

La meilleure position d'insertion est toujours choisie. Nous utilisons un procédé de diversification pour éviter le blocage d'insertion de client. Nous utilisons des règles pour accélérer la vitesse d'insertion.

Dans le troisième chapitre, nous proposons et testons une autre méthode heuristique d'insertion pour le DARP standard, impliquant de la propagation de contraintes. Des heuristiques ont été développées pour le DARP. Les heuristiques d'insertion sont une des méthodes populaires pour résoudre le problème du VRP, car elles sont rapides, elles peuvent produire des solutions de bonne qualité, elles sont faciles à mettre en œuvre et enfin, elles

peuvent facilement être étendues : De plus, elles peuvent être facilement randomisées et intégrées à des schémas de Monte-Carlo. Nous testons plusieurs schémas d'insertion :

1. le client est choisi au hasard et on l'insère dans la tournée active (celle de durée minimale);
2. le client est choisi au hasard et on essaye de l'insérer dans toutes les tournées actives ;
3. le client est choisi par application des principes de la « variable la plus contrainte », après application d'un procédé de propagation de contraintes sur les fenêtres de temps.

La meilleure position d'insertion est toujours choisie.

L'heuristique donne un ensemble de tournées qui sera utilisé pour initialiser les procédures de transformation locale, mises en œuvre dans les chapitres suivants. Les vingt instances qui sont proposées par Cordeau et Laporte (2003) sont testées par notre heuristique. Dans la dernière section, les résultats obtenus par les heuristiques sont présentés.

Dans le quatrième chapitre, une procédure SPLIT de transformation locale est proposée pour le DARP standard. La procédure SPLIT a été appliquée pour les autres problèmes de tournées, pour lesquelles elle s'avère très efficace. Nous avons réussi à adapter cette procédure au DARP. La procédure SPLIT travaille sur une permutation des nœuds (tour géant). Elle trouve d'abord les points de découpage dans la permutation des nœuds (tour géant) où le nombre de chargements du véhicule est égal à zéro. Un graphe auxiliaire est construit par ces points de découpage. Et chaque arc entre deux points de découpage représente une tournée. Cette procédure Split découpe une permutation des nœuds (tour géant) en un ensemble de tournées réalisables.

Le DARP est en soi un problème multicritère puisque on doit y concilier le point de vue de l'opérateur et celui de l'usager, mais, très souvent, les auteurs ramènent le critère multiple à un seul. Ici, nous parlerons de **DARP multicritère**, quand nous nous efforcerons de traiter l'ensemble de critères de façon indépendante. Pour le DARP multicritères, nous avons utilisé trois critères : TRT (Total Riding Time) est le temps total de trajet pour les clients, TD (Total Duration) est la durée totale des tournées, TWT (Total Waiting Time) est le temps total d'attente. À cause de l'existence de plusieurs critères, nous avons défini la relation « **dominance** » (se référer à la notion de Pareto optimalité) pour éviter un excès de génération des solutions. Dans la dernière section, les résultats obtenus par la procédure SPLIT sont présentés.

Dans le cinquième chapitre, nous proposons une méthode pour résoudre le DARP multicritère. Nous utilisons des heuristiques avec la procédure SPLIT pour générer un ensemble de solutions (le nuage de solutions). Nous cherchons les fronts de Pareto dans ce nuage de solutions. Une direction de recherche locale qui dépend de sa position dans le front est associée à chaque solution. Chaque solution est améliorée par une recherche locale de type ELS grâce à la direction de recherche locale (Evolutionary Local Search).

Nous constatons que la procédure SPLIT fait un découpage optimal avec multicritères pour un tour géant obtenu par l'heuristique et retourne des solutions non-dominées. Nous devons alors répondre aux questions suivantes :

- Comment construire les fonts de Pareto? Est-ce que nous utilisons toutes les solutions non-dominées ou pas ?
- Comment choisir une solution parmi les solutions non-dominées obtenues par la procédure SPLIT pour la recherche locale?
- Comment définir les directions de recherche locale?

Dans le dernier chapitre, nous présentons précisément le DARP avec contrainte cumulative « flux financier » et des méthodes de résolution. Dans ce problème, nous devons déterminer les tournées par une flotte de véhicules pour satisfaire un ensemble des demandes, et à chaque instance la différence entre le capital initial, les revenus et les dépenses doit être positive. Nous proposons pour ce problème des méthodes d'insertion basée sur des techniques de propagation de contraintes.

CHAPITRE 1
ETAT DE L'ART

1. Les problèmes de tournées : principaux composants

Le problème général de « chargement et déchargement » (GPDP : General Pickup and delivery problem) constitue une classe importante de problème de tournées de véhicules. Ce problème a été étudié depuis plus de 30 ans. Il se pose dans de nombreux contextes comme la logistique, la gestion de services de mobilité etc. Il peut être décrit comme un problème de conception de tournées mettant en jeu des lieux géographiquement dispersés (villes, entrepôts, etc.). Les tournées doivent être conçues de telle sorte que chaque lieu soit visité au moins une fois par un seul véhicule ou par plusieurs véhicules, et que certaines contraintes soient satisfaites.

Le GPDP est présenté ici afin de pouvoir faire face aux différentes caractéristiques trouvées dans de nombreux problèmes de chargement et déchargement. Dans la section suivante, certains des principaux composants du GPDP sont décrits :

- **Les véhicules :** Dans les problèmes de tournées avec opérations de chargement et déchargement, un ensemble de véhicules est disponible pour les opérations de transport. Chaque véhicule est accompagné d'une capacité de transport limitée (quantité, taille, poids, etc.), un dépôt de départ et un dépôt d'arrivée. Si tous les véhicules ont la même capacité, les véhicules sont homogènes. Dans le cas inverse, les véhicules sont hétérogènes.

- **Les demandes :** Dans le GPDP, un ensemble de tournées doit être construit pour satisfaire des demandes de transport. Chaque client émet une demande : une demande de chargement concerne un lieu où les objets doivent être collectés (les origines) et une

demande de déchargement concerne un lieu où les objets doivent être livrés (les destinations). Le client spécifie aussi le volume d'objets transportés pour le chargement ou pour le déchargement. Les objets peuvent être transportés de leurs origines à leurs destinations avec ou sans transbordement.

- **Les contraintes**

Les contraintes de capacité : la quantité d'objets transportés pour chaque demande doit être inférieure à la capacité du véhicule. La somme d'objets chargés à l'intérieur d'un véhicule à un instant donné ne doit pas excéder la capacité du véhicule.

Pour certaines variantes du GPDP, il existe une contrainte sur le type d'objet transporté ou sur le nombre des différents types d'objet transporté.

Les contraintes temporelles : on parlera d'« opération de service » pour designer les opérations de chargement et déchargement auxquelles se livrent les véhicules sur les lieux origine et destination de chaque client. Lorsque les opérations de service ne doivent avoir lieu que dans des périodes de temps données à l'avance, cette période de temps est appelée « fenêtre de temps », et représentée par un intervalle défini par une date au plus tôt et une date au plus tard de service. Quand des contraintes temporelles sont présentées, alors le problème combine une problématique « tournée de véhicules » et une problématique d'ordonnancement. Ces contraintes induisent un ordre parmi certaines demandes et réduisent le nombre de solutions réalisables. Les contraintes de temps peuvent se décomposer en deux grandes catégories:

- Les contraintes de temps liées aux véhicules : il s'agit du temps de travail pour les conducteurs et véhicules. Ces contraintes sont souvent modélisées par une fenêtre de temps ou un temps maximal (une constante).
- Les contraintes de temps liées aux clients (demandes) transportés : une telle contrainte se traduit souvent par une fenêtre de temps associée à un lieu de service. Les fenêtres de temps contraignent les temps d'arrivée et de départ pour chaque véhicule. Elles contraignent aussi les dates de début (fin) de service pour chaque demande qui peuvent ne pas coïncider avec les dates de départ (arrivée).

Dans certains cas particuliers, si le véhicule arrive plus tôt que sa fenêtre de temps, un temps d'attente est imposé avant de commencer le service. La qualité de service est enfin prise en compte dans certaines versions du GPDP telles que le DARP. La plupart du temps, cela implique une borne supérieur sur le temps passé entre le lieu du chargement et le lieu du déchargement, c'est-a-dire sur la durée pendant laquelle le client est dans le véhicule.

La présence de contraintes de temps complique considérablement le problème. S'il n'y a pas de contrainte de temps, une tournée faisable se trouve de façon générale assez

facilement : il suffit d'affecter arbitrairement les demandes de transport pour les véhicules en respectant les capacités, d'ordonner arbitrairement les demandes affectées à un véhicule, puis de traiter séparément chaque demande. En présence de contraintes de temps, le problème qui consiste à trouver une tournée est NP- difficile. En conséquence, il peut être difficile de construire une tournée faisable, particulièrement quand les contraintes de temps et de capacité sont toutes deux restrictives.

D'autres contraintes « physiques » sont toujours présentes dans le GPDP, tels que la contrainte de « précédence » (pour chaque demande, son lieu de chargement doit être visité avant son lieu de déchargement), la contrainte de « couplage » (un même véhicule visite le lieu de chargement et le lieu de déchargement indiqués par chaque demande).

- **Contexte dynamique et contexte statique :** Comme d'autres problèmes de routage, le GPDP peut être considéré selon deux contextes : statique et dynamique. Parafais [**10**] utilise la définition suivante pour le problème statique de routage : le problème est statique si la solution est un ensemble de routes planifiées à l'avance qui ne sont pas ré-optimisées et qui sont calculées à partir de données d'entrées qui n'évoluent pas en temps réel. Au contraire, ce problème est dynamique si la solution n'est pas un ensemble de routes, mais plutôt une politique qui prescrit la façon dont les routes devraient évoluer en fonction des données d'entrée qui évoluent en temps réel. Dans les définitions ci-dessus, la dimension temporelle joue un rôle essentiel pour la classification d'un problème de routage.

  Nous définissons ce que nous voulons dire lorsque nous parlons d'un problème statique de tournées :
  - on suppose que toute l'information concernant la planification des tournées est connue par l'opérateur avant que le processus commence.
  - l'information concernant le cheminement ne change pas après que les tournées aient été construites.
  - la solution est un ensemble des tournées qui sont planifiées et optimisées, ces tournées satisfont toutes les demandes des clients.

  L'information concernant les demandes inclut tous les attributs des clients tels que la localisation géographique des clients, le temps de service sur site, les fenêtres de temps et le volume de chaque demande. En outre, des informations système comme par exemple les temps de voyage du véhicule entre les clients doivent être connus.

  Nous définissons le problème du « Routage de Véhicules » dynamique comme suit:

  - Certaines informations concernant la planification des tournées sont connues par l'opérateur quand le processus de cheminement commence.

- D'autres informations, concernant notamment les demandes, apparaissent après que les tournées initiales aient été construites. On dispose a priori d'information sur la distribution probabiliste de ces demandes.
- A chaque instant, le résultat (solution) d'un calcul porte sur l'incorporation des demandes, dans l'ensemble des tournées existantes.
- Le processus de calcul (processus décisionnel de routage et d'affectation des demandes aux tournées), agit en constante interaction avec le processus physique de circulation des véhicules et avec les processus logiciels de dialogue avec l'utilisateur, d'acquisition des données, et de supervision du système.

Dans la Figure 3, un exemple simple du problème dynamique général de chargement et déchargement est montré. Dans l'exemple, deux véhicules doivent assurer le service à la fois des demandes connues à l'avance et des nouvelles demandes, sans fenêtre de temps. Les demandes connues à l'avance des clients sont représentées par des nœuds noirs. Les nouvelles demandes sont représentées par des nœuds blancs. Les lignes continues représentent les deux itinéraires prévus pour les véhicules quittant le dépôt. Les deux arcs épais indiquent les positions des véhicules lorsque les demandes dynamiques sont reçues. Dans le meilleur des cas, les nouveaux clients devraient être insérés dans les itinéraires déjà prévus, l'ordre des clients non encore visités ne devrait pas être modifié, et les nouvelles visites ne devraient induire qu'un minimum de retard. Toutefois, en pratique, l'insertion de nouvelles demandes est habituellement une tâche beaucoup plus complexe et elle peut impliquer un ré-ordonnancement des clients déjà planifiés et non encore visités.

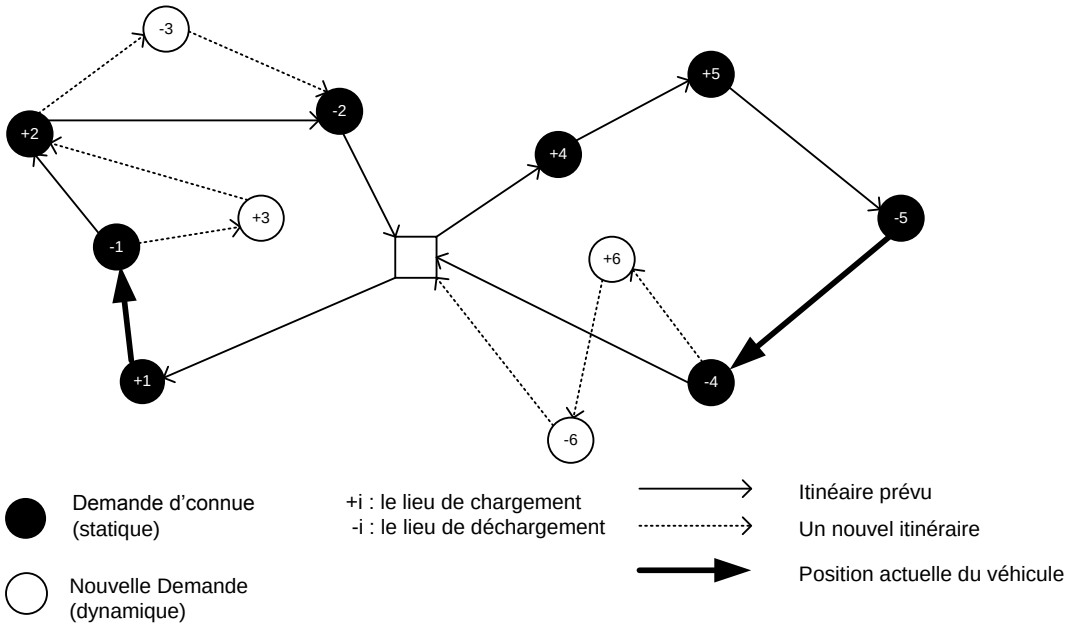

Figure 3 : Le GPDP dynamique avec 4 clients connus à et 1client à insérer dynamiquement

La présence de contraintes de fenêtres de temps implique que le système est en général amené à rejeter un nombre conséquent de demandes.

- **Le cas des parcours préemptifs :** lié à la possibilité d'opérations de transbordement : dans la plupart des variantes du GPDP, les objets transportés doivent être transportés sans aucune escale de leurs origines à leurs destinations. On dit alors que le problème est « non-préemptif ». Mais dans certaines variantes du problème GPDP, un ensemble de lieux de transbordement est donné : les objets transportés sont autorisés à abandonner temporairement leur véhicule dans ces lieux de transbordement puis à reprendre leur chemin plus tard, éventuellement avec un autre véhicule. Ces problèmes sont dits « préemptif ».

- **Les fonctions objectifs :** dans les problèmes de chargement et déchargement, plusieurs critères de performance sont utilisés. Les plus communs sont décrits ci-dessous :

  - **Minimisation de la durée maximale des tournées :** la durée d'une tournée est le temps nécessaire à un véhicule pour exécuter ses opérations. Cette durée inclut les temps d'attente, les temps de chargement et de déchargement.

  - **Minimisation de la somme des longueurs des tournées :** la longueur de la tournée est la distance entre le dépôt de départ et le dépôt d'arrivée.

  - **Minimisation des temps de voyage :** le temps de voyage d'un client réfère au temps passé par ce client sur son déplacement entre le lieu de chargement et le lieu de déchargement.

  - **Minimisation du désagrément du client :** dans certains problèmes, le désagrément du client est mesuré en termes de déviation, qui renvoie à :

    - la différence entre la date du chargement (déchargement) effectif et la date souhaitée de chargement (déchargement) ;
    - l'excès du temps de voyage, par rapport à un souhait exprimé.

  - **Minimisation du nombre de véhicules utilisés :** dans certaines variantes du GPDP, cette quantité se combine avec une des fonctions présentées ci-dessus.

  - **Minimisation des coûts économiques :** on agrège un ensemble de coûts, faisant apparaître coûts de conducteurs, coûts d'infrastructure et coût de fonctionnement.

  - **Maximisation du profit :** cette quantité peut faire appel à toutes les fonctions ci-dessus. Elle peut être utilisée dans un système où il y a la possibilité de rejeter une

demande de transport quand elle est défavorable et où le traitement de chaque demande engendre un profit connu à l'avance.

Dans les problèmes dynamiques de chargement et déchargement, on a une difficulté pour choisir une fonction objectif à optimiser. Une fonction objectif pour des problèmes dynamiques pourra en effet concerner seulement une décision ponctuelle et ses conséquences immédiates ou au contraire, un flux de décision et une évolution de la « trajectoire » du système.

2. Classification académique pour le GPDP statique

Le GPDP statique travail sur un graphe $G = (V, A)$ avec $V = (0,1, \dots, n)$. Le sommet 0 représente le dépôt (le dépôt de départ et le dépôt d'arrivée), et chaque sommet restant représente un lieu de demande de client. En chaque sommet (sauf le dépôt), on peut associer une opération de chargement ou de déchargement (ou les deux). L'ensemble $A = \{(i,j)|i,j \in V, i \neq j\}$ est un ensemble d'arcs. Chaque arc $(i,j) \in A$ a un coût (ou une longueur) non négatif $c_{i,j}$ (habituellement égal au temps de déplacement entre le sommet i et le sommet j). Un ensemble d'objets transportés est représenté sous la forme $X = \{1, \dots, O\}$. Chaque sommet (client virtuel) est fournisseur ou collecteur d'une quantité d'objets. Cette quantité peut être nulle. $D = (d_{i,o})$ dénote la matrice d'objets, où le nombre $d_{i,o}$ est positif si la quantité d'objets o est fournie par le sommet i, $d_{i,o}$ est négatif si la quantité d'objets est collectée par le sommet i. Pour chaque objet transporté $o \in X$, on suppose que $\sum_{i \in V} d_{i,o} = 0$. Un ensemble de véhicules $K = (1, \dots, m)$ est disponible au dépôt, chaque véhicule a une capacité Q. Un sous-ensemble $T \in V$ indique les sommets de transbordement possibles où les objets transportés peuvent être abandonnés temporairement et repris plus tard (probablement avec un autre véhicule). Le GPDP consiste en la construction de m tournées telles que :

- toutes les demandes de chargement et de déchargement sont satisfaites ;
- les transbordements d'objets n'ont pas lieu aux sommets $V \backslash T$;
- la quantité de charge d'un véhicule ne dépasse jamais sa capacité ;
- la somme des coûts des tournées est minimisée.

On peut utiliser des indications symboliques pour classer les différentes versions du GPDP :

- La première indication est dite « Structure », elle spécifie le nombre d'origines et de destinations pour les objets transportés. La notation « many-to-many » (M-M) indique que tout sommet du graphe G peut servir comme origine ou comme destination pour tous les objets transportés. La notation « one-to-many-to-one » (1-M-1) indique que les objets transportés sont d'abord disponibles au dépôt et sont destinés à des sommets,

ou réciproquement, que les objets transportés sont disponibles à des sommets et sont destinés au dépôt. Les ensembles de chargement et de déchargement des objets ne sont pas nécessairement disjoints (Röpke et Pisinger 2006b) [**11**] et coïncident souvent. Enfin, la notation « one-to-one » (1-1) indique que chaque objet transporté a une origine donnée et une destination donnée.

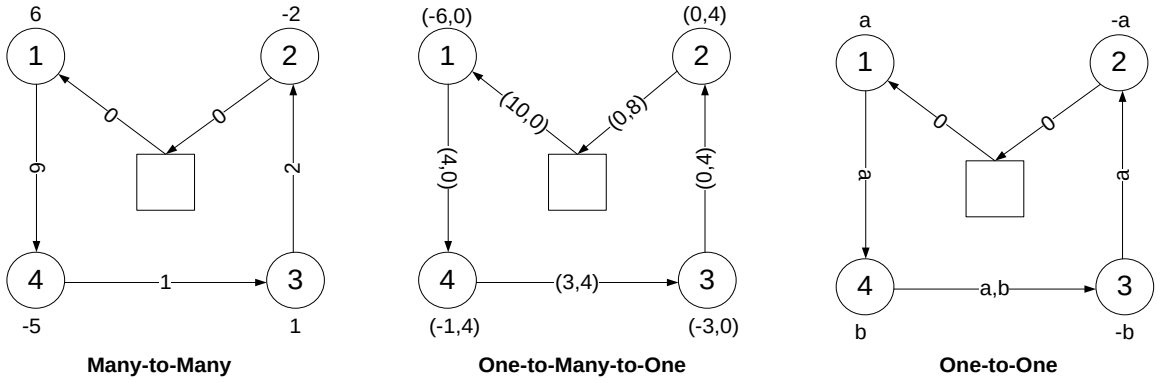

Figure 4 : Les types de structure

La Figure 4 représente un exemple pour chaque type de structure. La figure de gauche représente un exemple de la structure « many-to-many », un label positif x associé à chaque sommet décrit la quantité de chargement et un label négatif $-x$ décrit la quantité de déchargement. Le label z associé à un arc indique la quantité de charge d'un véhicule. La figure de milieu représente un exemple de la structure « one-to-many-to-one », le label $(-x, y)$ associé à chaque sommet représentant une opération de déchargement avec x unités et une opération de chargement avec y unités. Le label (p, q) qui est associé à l'arc représente p unités correspondant au déchargement et q unités correspondant au chargement. La figure de droite représente un exemple de la structure « one-to-one ». Le label x signifie que le sommet fournit le type d'objet x et le label $-x$ signifie que le sommet reçoit le type d'objet x. Le label y associé à chaque arc désigne les types d'objets transportés par le véhicule.

- La seconde indication est dite « Visite », et concerne des informations sur la façon dont les opérations de chargement et de déchargement sont effectuées au niveau des nœuds (clients). Nous utilisons la notation « PD » pour indiquer que chaque sommet est visité une seule fois pour une opération combinant le chargement et le déchargement. La notation « P-D » signifie que les opérations peuvent être exécutées simultanément ou séparément sur un même sommet. Enfin, la notation « P/D » renvoie au cas où chaque sommet est concerné soit par une demande de chargement soit par une demande de déchargement, mais pas les deux.

2.1. Le GPDP avec la structure « Many-to-Many »

De nombreux problèmes de GPDP avec la structure « many-to-many » sont caractérisés par plusieurs origines et plusieurs destinations pour chaque objet transporté. Trois problèmes seront décrits dans cette section: SP (Swapping Problem), 1-PDTSP (One-commodity Pickup-and-Delivery Travelling Salesman Problem) et Q-DTSP (The Q-delivery Traveling Salesman Problem).

2.1.1. SP (The swapping problem)

Le SP a été prouvé NP-difficile (Garey et johnson 1979) [**12**] et ce problème avec multi-objets à transporter a été introduit par Anily et Hassin (1992) [**13**]. Il traite de l'opération d'échange des objets entre les sommets autre que le dépôt avec un seul véhicule. Le SP contient les contraintes suivantes :

- Un seul véhicule de capacité unitaire est utilisé pour transporter les objets : $m = 1$, $Q_m = 1$;
- Pour tout $o \in X$, $d_{dépôt,o} = 0$: il n'y a pas d'opération de chargement ou de déchargement au dépôt ;
- Chaque demande de chargement ou de déchargement concerne au plus un type d'objet;
- Chaque sommet $i \in V\backslash\{0\}$ contient au plus une paire de types d'objets (x_1, x_2), $x_1, x_2 \in X$ correspondant au chargement et au déchargement. Les quantités de chargement d_{i,x_1} et de déchargement d_{i,x_2} sont choisies dans un ensemble $\{-1, 0, 1\}$, $d_{i,x_1} \neq d_{i,x_2}$. Donc le SP est un GPDP avec l'indication « P-D ».

Le SP peut être « non-préemptif » ou « préemptif ». Dans le cas « non-préemptif », il n'y a pas de sommet de transbordement, c.-à-d. $T \neq \emptyset$. Dans le cas « préemptif », si $T = V$ alors les objets peuvent être déchargés temporairement dans tous les sommets. Si $T \neq V$ et $T = \emptyset$ alors on parle de MSP (Mixed Swapping Problem). Le MSP consiste à repositionner les objets au sens du GPDP, à l'aide d'une tournée unique de coût minimal.

La Figure 5 représente un exemple simple de SP «non-préemptif» avec 6 objets transportés $\{a, b, c, d, e, f\}$ et 7 sommets : $\{0, 1, 2, 3, 4, 5, 6\}$. Le label $(x, -y)$ qui est associé à chaque sommet représente une opération de chargement sur l'objet x et une opération de déchargement sur l'objet y. Le label x qui est associé à chaque arc représente le type d'objet transporté.

Anily et Hassin (1992) [**13**] ont caractérisé la structure de la solution optimale et ont développé deux heuristiques dont la performance au pire des cas est au plus 2,5 fois celle de la solution optimale. Ces heuristiques sont basées sur la solution d'une relaxation du

problème, et sur l'extension de la solution obtenue en une solution réalisable de telle façon que le coût est augmenté par un facteur constant.

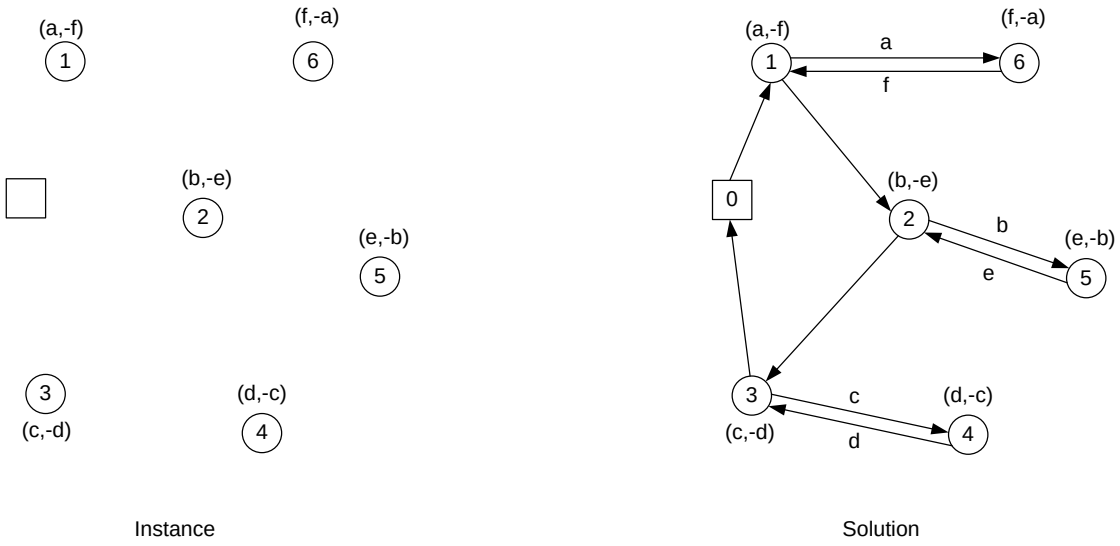

Figure 5 : Le SP « non-préemptif »

Chalasani et Motwani (1999) [**14**] ont développé un algorithme qui est basé sur une procédure « matroïde intersection » pour traiter le SP avec deux types d'objet ($|X| = 2$). Ce problème particulier est dit « Bipartite Traveling Salesman Problem ». Ils ont obtenu une meilleure borne avec une performance au pire des cas qui est au plus 2 fois celle de la solution optimale.

Anily et al. (1999) [**15**] ont développé une méthode exacte $O(n^2)$ pour le SP. Erdoğan et al. (2008) [**16**] ont développé une méthode basé sur une méthode de « Branch-and-cut » pour le SP « préemptif » et « non-préemptif ». Bordenave et al (2010) [**17**] ont présenté une heuristique constructive pour le MSP.

2.1.2. 1-PDTSP (One-commodity Pickup-and-Delivery Travelling Salesman Problem)

Dans le 1-PDTSP, l'ensemble des lieux « client » ($V\backslash\{0\}$) est partitionné en lieux de chargement et lieux de déchargement qui doivent être servis par un seul véhicule basé au dépôt. Le 1-PDTSP contient les contraintes suivantes :

- Un seul véhicule est utilisé pour transporter les objets : $m = 1$, et la capacité du véhicule n'est pas nécessairement unitaire $Q_m > 0$;
- Le nombre total de type d'objets transportés est $|X| = 1$(un seul type d'objet) ;
- Pour tout $o \in X$, $d_{dépôt,o} = 0$: il n'y a pas d'opération de chargement et de déchargement au dépôt ;

- $T = \emptyset$: pas de lieu de transbordement.

Le 1-PDTSP est étroitement lié à un cas particulier du SP. Considérons le SP « non préemptif » dans lequel on permet de charger et décharger des objets. Supposons qu'il y ait seulement un type d'objet chargé ou déchargé à chaque sommet. Le 1-PDTSP est alors équivalent à cette version du SP avec la contrainte supplémentaire que la tournée doit être hamiltonienne. Le 1-PDTSP est une version du GPDP avec l'indication « P/D »

La 1-PDTSP a été introduit par Hernandez-Pérez et Salazar-Gonzalez (2004a) [**18**], et il s'agit d'un problème NP-difficile. La Figure 6 représente un exemple de 1-PDTSP avec un seul objet transporté et 7 sommets : $\{0, 1, 2, 3, 4, 5, 6\}$. Un label l qui est associé à chaque sommet représente une opération de chargement ($l > 0$) ou déchargement ($l < 0$).

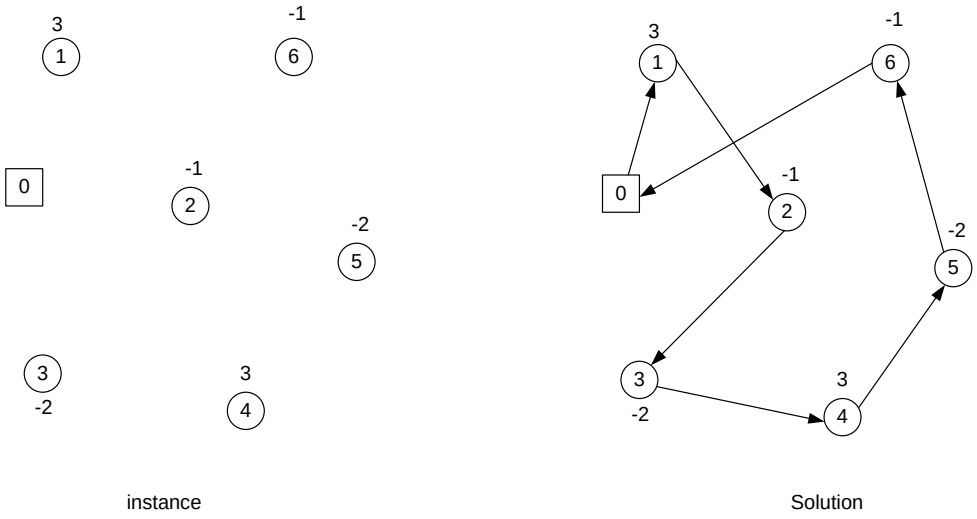

Figure 6 : 1-PDTSP

Hernandez-Perez et Salazar-Gonzalez (2004a) [**18**] ont développé un algorithme de « Branch-and-cut » pour le 1-PDTSP, capable de résoudre des instances jusqu'à 40 sommets. Ils ont proposé un modèle de programmation linéaire en nombres entiers (PLNE) qui est équivalent à un modèle classique de TSP avec un ensemble de coupes appelées « Benders cuts ». Cela implique que toutes les inégalités valides connues pour le TSP sont également valides pour le 1-PDTSP. Les auteurs ont proposé un algorithme pour le flot maximal et une heuristique pour identifier rapidement les « Benders cut » et ils ont utilisé les inégalités de cliques (voir Balas et Padberg [**19**]).

Hernandez-Perez et Salazar-Gonzalez (2004b) [**20**] ont développé deux heuristiques pour le 1-PDTSP et ont été capables de résoudre des instances de moins de 500 sommets. La première heuristique utilise une version de l'algorithme modifié dit du « plus proche voisin » pour le

TSP. Les coûts de transport sont redéfinis afin de favoriser des raccordements entre le chargement et le déchargement, de façon à aider à la génération d'une solution réalisable. L'autre point important de l'heuristique est l'adaptation des procédés de transformation locale de « 2-opt » et « 3-opt » [**21**,**22**], où certaine échanges sont envisagées. Ces échanges sont appliqués de façon à minimiser l'infaisabilité de la tournée sous certaines contraintes. La deuxième heuristique est un algorithme d'optimisation incomplète qui utilise les méthodes exactes développées par Hernández-Pérez et Salazar-González (2004a) [**18**] pour déterminer itérativement des solutions optimales dans des espaces faisables limités.

En 2009, Hernandez-Perez et al. [**23**] ont développé une méthode hybride combinant le GRASP et l'heuristique VND. ZHAO et al. (2009) [**24**] ont utilisé un algorithme génétique pour le 1-PDTSP jusqu'à 500 sommets.

2.1.3. Q-DTSP (The Q-delivery traveling salesman problem)

Le Q-DTSP a été introduit par Chalasani et Motwani (1999) [**14**]. Il consiste à charger et décharger un type d'objet unique de leurs origines vers leurs destinations. Le Q-DTSP contient les restrictions suivantes :

- Un seul véhicule est utilisé pour transporter les objets : $m = 1$; la capacité du véhicule est unitaire : $Q_m = 1$;
- Le nombre total de type d'objets transportés est égal à 1 ($|X| = 1$) ;
- Pour tout $o \in X$, $d_{dépôt,o} = 0$: pas d'opération de chargement et de déchargement au dépôt ;
- $T = \emptyset$: pas de lieu de transbordement (Q-DTSP est un tour hamiltonien) ;
- Chaque sommet $i \in V\backslash\{0\}$ contient un seul type d'objet correspondant à son chargement ou à son déchargement. La quantité de chargement où de déchargement $d_{i,x} \in [-1,1]$.

Le Q-DTSP est aussi un problème NP-difficile. Il est une version du GPDP avec l'indication « P/D ». La Figure 7 représente un exemple du Q-DTSP avec un seul objet transporté et 7 sommets : $\{0, 1, 2, 3, 4, 5, 6\}$. Un label l associé à chaque sommet représente une opération de chargement ($l > 0$) ou de déchargement($l < 0$).

Chalasani et Motwani (1999) [**14**] ont étudié le Q-DTSP ainsi qu'une nouvelle variante dynamique du problème qui est appelé « Dynamic K-collect TSP », et ils ont décrit un algorithme d'approximation polynomiale.

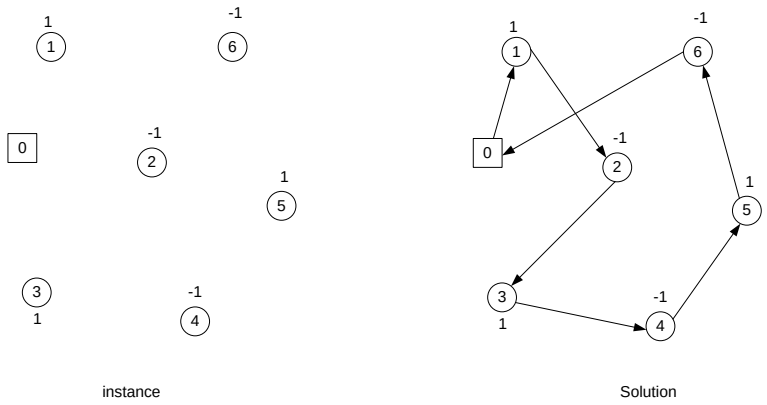

Figure 7 : Q-DTSP

Anily et Bramel (1999) [**25**] ont étudié le Q-DTSP sous le nom « Capacitated Traveling Salesman Problem with Pickups and Deliveries». Les auteurs ont légèrement modifié l'algorithme de Chalasani et Motwani (1999) [**14**] et amélioré le pire des cas, passant de 9,5 à 7 fois la valeur de la solution optimale. En outre, ils ont dérivé un nouvel algorithme en temps polynômial dans lequel le pire des cas dépend de la capacité des véhicules Q. Ce nouvel algorithme a une meilleure performance au pire des cas que le précédent pour toutes les instances de capacité $Q \leq 255$.

La Figure 8 représente le schéma de classification pour GPDP avec la structure « M-M » et les relations principales entre ces problèmes.

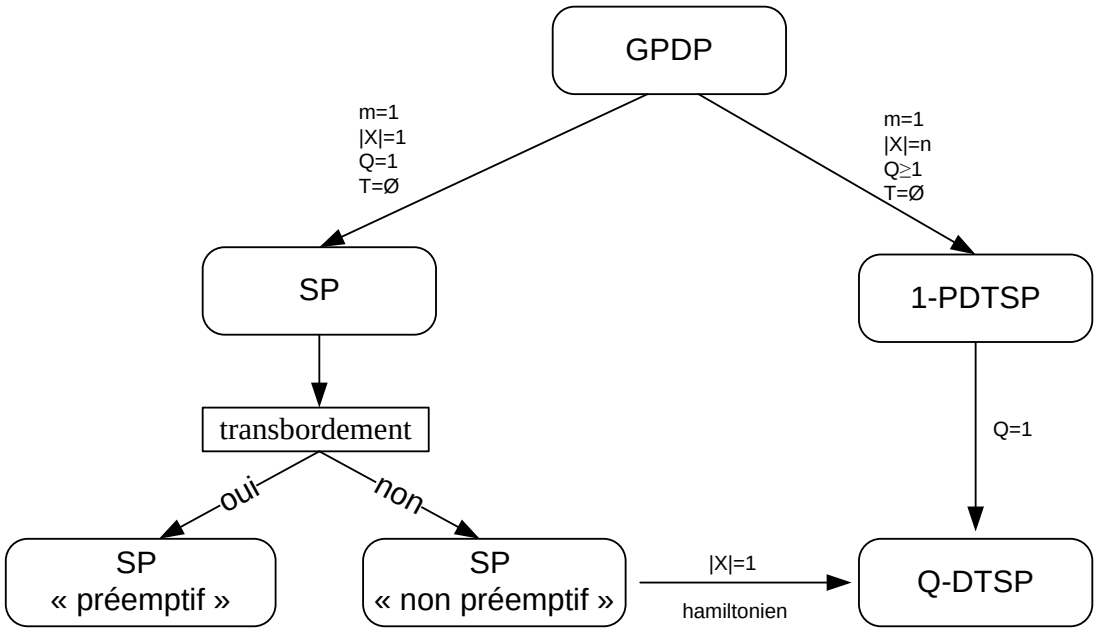

Figure 8 : La classification pour GPDP avec la structure « M-M »

2.2. Le GPDP avec la structure « 1-M-1 »

Dans les versions de problème GPDP avec la structure « 1-M-1 » dites « 1-M-1 PDPs », certains objets sont transportés du dépôt vers les sommets de déchargement, tous les autres objets étant chargés par les véhicules dans des sommets de chargement et ramenés au dépôt. Cette catégorie de problèmes contient les restrictions suivantes :

- Le nombre total de type d'objets transportés est $|X| = 2$. On pose $X = \{1, 2\}$;
- $\forall i \in V\backslash\{0\}$, $d_{i,1} \geq 0$ et $d_{i,2} \leq 0$. Chaque client peut émettre une quantité de chargement et une quantité de déchargement ;
- $T = \emptyset$.

Nous considérerons deux types principaux de demande pour le GPDP « 1-M-1» : les « demandes combinées » et les « demandes simples ». Dans les problèmes avec demandes combinées, au moins un client propose une demande de chargement et une demande de déchargement. Dans les problèmes avec demandes simples, chaque client propose soit une demande de chargement soit une demande de déchargement.

2.2.1. Le problème avec la structure 1-M-1 et les demandes combinées

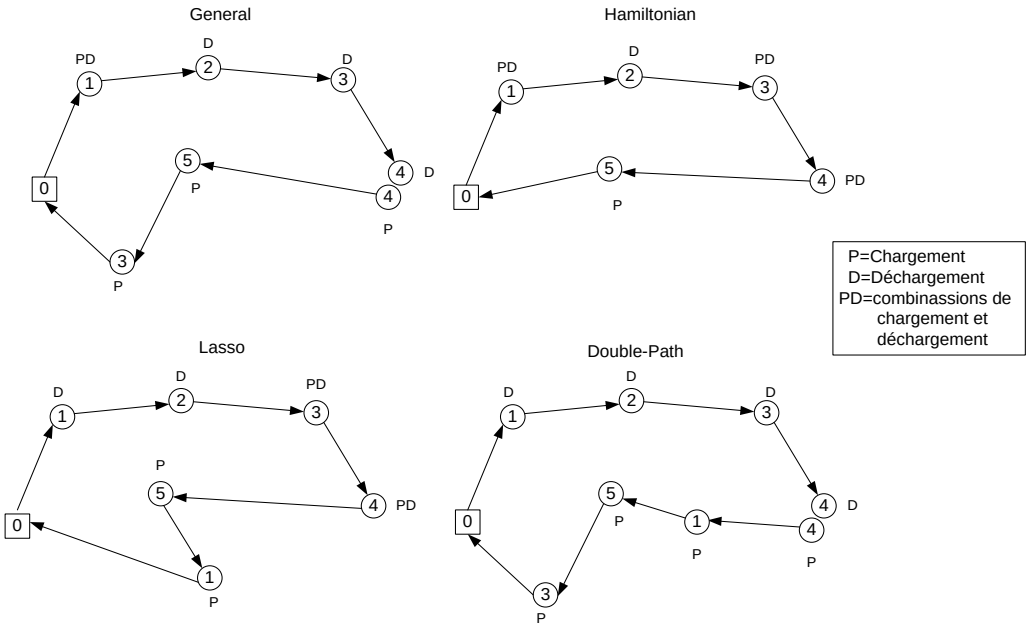

Figure 9 : Quatre types de solution avec 5 clients

Dans ce problème, il existe au moins un sommet $i(i \neq 0)$ avec $d_{i,1} \neq 0$ et $d_{i,2} \neq 0$. Gribkovskaia et al. (2007) [**26**] identifie quatre types de solution pour les problèmes GPDP avec la structure « 1-M-1», les demandes combinées et le véhicule unique : « General »,

« Hamiltonian », « Lasso » et « Double-path » (voir les exemples dans la Figure 9). Dans le cas « General », une solution n'a pas de forme prédéterminée : cela signifie que tout sommet peut être visité une ou deux fois. Une solution « Hamiltonienne » est une solution dans laquelle chaque sommet est visité une seule fois pour une opération qui combine un chargement et un déchargement. Dans le cas « Lasso », le véhicule effectue d'abord des déchargements en utilisant un sous-ensemble de sommets, puis le véhicule visite tous les autres sommets pour effectuer une opération combinant un déchargement et un chargement. Finalement le véhicule effectue des chargements aux sommets de chargement. Dans le cas « Double-Path », tous les sommets sont associés à presque deux chemins « hamiltoniens »: le premier chemin pour tous les sommets qui contiennent une opération de déchargement, et le deuxième pour tous les sommets qui contiennent une opération de chargement. Dans ce type de solution, un seul sommet est utilisé simultanément pour un chargement et un déchargement, alors que tous les autres sommets sont visités au plus deux fois.

Gribkovskaia et al. (2007) [**26**] décrivent plusieurs méthodes heuristiques produisant des solutions générales pour le problème avec la structure « 1-M-1 », un seul véhicule et des demandes combinées. Il s'agit notamment de procédures de construction gloutonne et d'amélioration locale ainsi que d'une heuristique de recherche taboue. Le principe des procédures de construction gloutonne est le suivant : d'abord, une solution « hamiltonienne » éventuellement infaisable est obtenue par une procédure du plus proche voisin, par une procédure « balayage » ou par une procédure « balayage modifié ». Ensuite, un circuit « hamiltonien » est utilisé pour générer une série de solutions en enlevant une des arêtes et en reliant les sommets d'une manière particulière. Enfin, une procédure de fusion essaye d'éliminer la seconde visite pour les sommets qui sont visités deux fois en transformant la première visite en une visite de chargement et de déchargement simultanés. Deux variantes de cette procédure de fusion sont proposées par les auteurs. Six procédures différentes de construction qui sont ainsi définies en utilisant les différentes combinaisons possibles. Une procédure d'amélioration locale tente d'obtenir une meilleure solution en supprimant itérativement un sommet visité une fois et en le réinsérant dans la tournée. L'heuristique taboue est basée sur l'algorithme unifié de recherche taboue de Cordeau et al. (2001) [**27**] pour le VRP. Les résultats obtenus sur 17 instances comprenant de 16 à 101 sommets indiquent que 62% des solutions sont « hamiltoniennes », 32% contiennent un sommet visité deux fois, et 6% contiennent deux sommets visités plus de deux fois.

2.2.2. VRPSPD (The Vehicle Routing Problem with Simultaneous Pickups and Deliveries)

Le **VRPSPD** est un problème avec la structure « 1-M-1 » (voir la figure Figure10), demandes combinées et multi-véhicules. Il consiste à construire un ensemble de tournées avec les hypothèses suivantes:

- $|X| = 2$;
- des demandes de chargement et de déchargement émanent simultanément de chaque lieu de demande;

- chaque sommet doit être visité au plus une fois par un seul véhicule. c.-à-d. chaque tournée passe au plus une fois en chaque sommet, et un même sommet n'est visité que par un véhicule ;
- la somme des coûts de tournées doit être minimale.

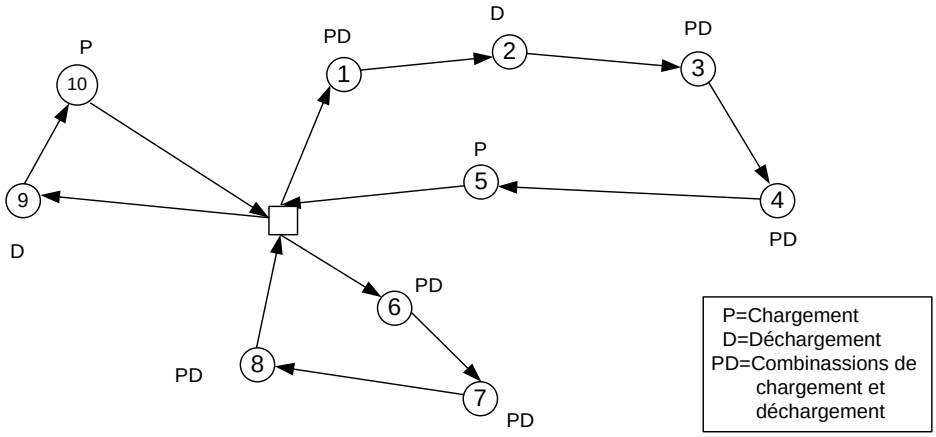

Figure10 : VRPSPD

Un résultat théorique concernant le VRPSPD est dû à Anily (1996) [**28**] qui a proposé un algorithme heuristique ($n^3 \log n$) pour le VRPSPD et prouvé que cet algorithme converge vers la solution optimale dès lors que l'on impose des conditions probabilistes. Une heuristique avec trois phases pour le VRPSPD a été développée par Min (1989) [**29**]. Dans la première phase, les groupes de clients sont formés en utilisant la méthode « average linkage » [**30**]. La distribution de clients est effectuée dans la deuxième phase. Dans la troisième phase, un problème de routage est résolu par un algorithme de TSP, en modifiant de façon itérative la matrice de distance pour éviter d'obtenir des solutions qui dépassent les capacités du véhicule. Une autre méthode heuristique a été développée par Nagy et Salhi (2005) [**31**] pour le VRPSPD avec contrainte sur la longueur maximum de trajet. Elle consiste à construire une solution initiale (qui est souvent irréalisable), et à appliquer une série de procédures d'amélioration et de réparation. Les heuristiques travaillent aussi sur la version multi-dépôt du problème. Ces heuristiques sont capables de résoudre des instances allant jusqu'à 249 clients en quelques secondes.

Chen et Wu (2006) [**32**] ont proposé une procédure basée sur l'insertion et une heuristique hybride pour le VRPSPD. L'heuristique hybride est basée sur le principe « record-to-record travel » (Dueck 1993) [**33**], les listes taboues et plusieurs procédures d'amélioration locale. Les algorithmes ont été testés sur des instances comprenant jusqu'à 199 clients et 19 véhicules. Bianchessi et Righini (2007) [**34**] ont développé et comparé des algorithmes constructifs (gloutons), un algorithme de recherche locale et un algorithme de recherche tabou pour le VRPSPD, en utilisant cinq voisinages. Plusieurs résultats ont permis aux auteurs de conclure que la recherche locale avec les voisinages variables sont robustes et peuvent donner

des bonnes solutions. Un algorithme de recherche taboue pour le VRPSPD a été présenté par Montané et Galvão (2006) [**35**]. Le voisinage est défini par des mouvements « 3-inter-route » et un opérateur « 2-opt intra-route ». A chaque itération, le meilleur voisin réalisable est sélectionné. La méta-heuristique est pilotée par des stratégies d'intensification et de diversification en utilisant un schéma de pénalité de fréquence. L'algorithme génère des solutions de bonne qualité sur les instances testées comprenant jusqu'à 400 clients dans un temps maximal de calcul de six minutes.

Il existe de nombreuses applications du VRPSPD par exemple dans des problèmes de logistique liés au courrier : les lettres et les colis peuvent être livrés et collectés sur la même tournée. Le VRPSPD est une version du GPDP avec l'indication « PD ».

2.2.3. Le problème avec la structure « 1-M-1 » et les demandes simples

Dans ce problème, chaque client émet une demande de chargement ou une demande de déchargement, mais jamais les deux à la fois. Deux types de solution ont été étudiés (voir la Figure 11). Dans le premier type, les solutions sont dites « mélangées » si les demandes de déchargement et les demandes de chargement peuvent être servies en utilisant n'importe quel ordre, c.-à-d. si on n'indique pas a priori d'ordre sur les opérations. Dans une version plus contrainte (VRPB), dite avec « Backhauls », les opérations de chargement doivent avoir lieu après les opérations de déchargement.

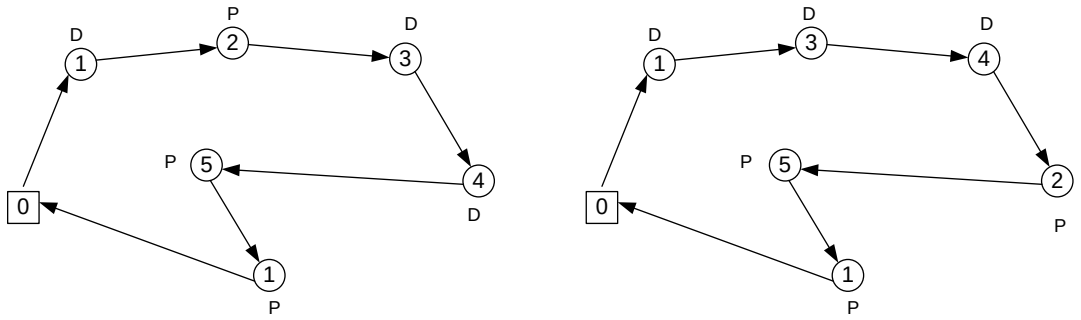

Figure 11 : Type de solution

La première heuristique pour le VRPB a été présentée par Deif et Bodin (1984) [**36**]. Il a résolu des instances symétriques à l'aide d'une extension de l'algorithme de Clarke et Wright (1964) [**37**] pour le VRP. Une heuristique de type « cluster-first route-second » pour le VRPB symétrique et asymétrique, a été proposée par Toth et Vigo (1999) [**38**]. Dans cette heuristique, les clients « Linehaul » et les clients « Backhaul » sont d'abord déterminés en résolvant une relaxation lagrangienne augmentée du problème [**39**]. Le couplage entre les clients « Linehaul » et les clients « Backhaul » est calculé. Les tournées sont obtenues par une heuristique modifiée de « Farthest-Insertion » et sont améliorées par des procédures « intra-

route » et « inter-route post-optimization ». L'heuristique est relativement rapide et on obtient presque toujours des solutions meilleures que l'heuristique précédente.

Osman et Wassan (2002) [**40**] ont développé une heuristique de recherche taboue et obtenu des solutions avec des qualités semblables, mais nécessitant des temps de calcul plus élevés. Brandão (2006) [**41**] a proposé un algorithme de recherche tabou qui est plus rapide et utilise une relaxation de VRPB, appelée le « m-arbre », pour obtenir une solution initiale. L'algorithme a donné les meilleurs résultats connus pour le VRPB pour des temps de calcul assez faibles. Gelinas et al. (1995) [**42**] ont proposé un algorithme exact pour le VRPPD avec fenêtres de temps (VRPPDTW) qu'ils ont étendu au VRPB avec fenêtres de temps (VRPBTW). Des algorithmes heuristiques pour le VRPBTW ont été décrits dans trois articles.

Potvin et al. (1996) [**43**] ont développé un algorithme génétique conçu pour trouver un bon ordonnancement des clients. Un algorithme de recherche taboue a été développé par Duhamel et al. (1997) [**44**], dans lequel le voisinage est choisi au hasard parmi le 2-opt, l'Or-opt, et une procédure de permutation. Thangiah et al. (1996) [**45**] ont proposé un heuristique de construction (gloutonne) qui insère des clients un par un dans les tournées en utilisant une des deux procédures de recherche locale suivantes. La première, appelé « $\lambda - interchange$», est basée sur l'échange des clients entre les tournées. La deuxième est une procédure 2-opt qui est une extension du VRPTW avec 2-opt. Les trois algorithmes ont produit des résultats proches de l'optimum théorique sur des instances comprenant jusqu'à 100 clients et 21 véhicules.

Le problème GPDP avec la structure « 1-M-1 », un seul véhicule, demande simple et « backhauls » est habituellement appelé le **TSPB** (Traveling Salesman Problem with Backhauls). Il s'agit d'un cas particulier du « Clustered Traveling Salesman Problem » dans lequel $V\backslash\{0\}$ est divisée en p groupes au lieu de deux (Chisman 1975 [**46**]). Dans ce problème, la contrainte de capacité ne joue aucun rôle, la capacité Q du véhicule doit seulement satisfaire à $Q \geq \sum_{i=1}^{n} d_{i,1}$ et $Q \geq \sum_{i=1}^{n}(-d_{i,1})$ pour que une solution existe.

Gendreau et al. (1996) [**47**] ont développé six heuristiques pour le TSPB, basé sur GENIUS (Gendreau et al. 1992) [**48**]. Ils les ont testés sur des instances comprenant de 100 à 1000 sommets. Sur des instances impliquant jusqu'à 200 sommets, toutes les heuristiques génèrent des solutions qui sont à moins de 8% de la solution optimale. En 1997, Mlade-Nović et Hansen [**49**] ont introduit la recherche à voisinage variable (VNS), cette méta-heuristique consiste à changer systématiquement le voisinage dans une procédure de recherche locale. Ces auteurs ont amélioré les résultats de [**47**] pour le TSPB en intégrant le GENIUS dans un cadre de VNS. Ghaziri et Osman (2003) [**50**] ont développé un algorithme de réseau de neurones pour le TSPB basé sur la carte de Kohonen à auto-organisation (Kohonen 1995 [**51**]). Les auteurs ont conçu une architecture de réseau qui se transforme en un tour réalisable de TSPB. La solution obtenue est ensuite améliorée par la procédure « 2-opt ».

2.2.4. TSPPD (Travelling Salesman Problem with Pickup and Delivery)

Le problème GPDP avec la structure « 1-M-1 », un seul véhicule, demande simple et solution « mélangée », est habituellement appelé le **TSPPD.**

Gendreau et al. (1999) [**52**] ont proposé deux heuristiques pour le TSPPD. Le premier est un algorithme linéaire exact pour le cas particulier où le graphe est un cycle. Les auteurs ont présenté une méthode heuristique pour le cas général, qui dérive de cet algorithme exact. L'heuristique, appelé « H », consiste à déterminer une solution du TSP en utilisant d'abord l'heuristique de Christofides [**53**], puis en appliquant l'algorithme exact TSPPD sur la solution du TSP, et en dérivant enfin une solution « hamiltonienne » à l'aide de raccourcis.

Le chemin du TSPPD est deux fois plus long que la longueur de la solution de TSP, et l'heuristique de Christofides a une performance au pire des cas qui est de 1.5 fois la valeur de la solution optimale. « H » a une performance au pire des cas qui est de 3 fois la valeur de la solution optimale. Le deuxième algorithme proposé par [**52**] est une heuristique de recherche taboue qui effectue des échanges de deux arcs sur la solution générée par l'heuristique « H ». Des précautions particulières doivent être prises tout en effectuant ces échanges pour éviter de générer des solutions infaisables. Les auteurs ont été en mesure de vérifier la faisabilité des solutions en temps constant en utilisant des structures de données ad hoc. Les solutions heuristiques ont améliorées les solutions obtenues par heuristique « H », et elles ont de meilleures performances que toutes les heuristiques précédentes pour le TSPPD, mais cela au prix de durées de calcul plus longues. Baldacci et al. (2003) [**54**] ont développé un algorithme exact de « branch-and-cut » pour le TSPPD, en utilisant une nouvelle formulation de flux sur deux produits. Les auteurs ont été capables de résoudre de façon optimale des instances comprenant jusqu'à 200 sommets dans une durée de calcul limitée à une heure.

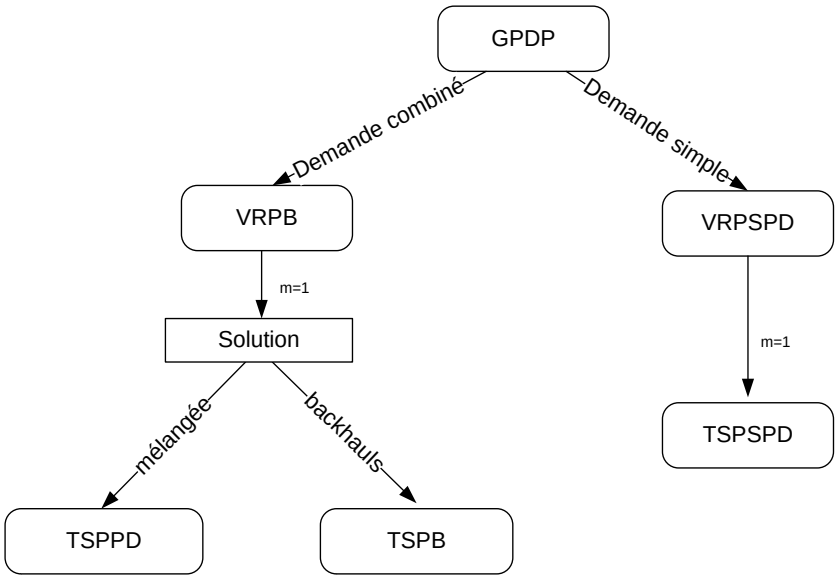

Figure 12 : La classification pour GPDP avec la structure « 1-M-1 »

La Figure 12 représente le schéma de classification pour le GPDP avec la structure « 1-M-1 » et les relations principales qui existent entre les problèmes.

2.3. Le GPDP avec la structure « 1- 1 »

Dans ce problème, à chaque type d'objet transporté correspond exactement un sommet de chargement et un sommet de déchargement. Deux problèmes importants appartenant à cette catégorie sont le **VRPPD** (Vehicle Routing Problem with Pickups and Deliveries) et le **DARP** (Dial-a-Ride Problem). Pour le VRPPD, l'objet transporté est en général un lot de marchandises. Lorsque les véhicules sont autorisés à déposer des objets temporairement, le problème est appelé VRPPD avec transbordements (VRPPDT). Pour le DARP, l'objet transporté est un groupe de personnes et la fonction objectif prend en compte certains éléments de qualité de service. Pour bien présenter ces deux problèmes, nous présentons d'abord le SCP (Stacker Crane Problem). Le SCP est un cas particulier du SP et du VRPPD.

Le problème GPDP avec la structure « 1-1 » diffère du problème GPDP avec la structure « 1-M-1 » dans lequel chaque client reçoit un déchargement provenant d'un dépôt commun et donne une quantité de chargement au dépôt. Il diffère aussi du problème avec la structure « M-M » dans lequel un objet peut être chargé ou déchargé à un nœud quelconque du réseau.

2.3.1. SCP (Stacker Crane problem)

Le SCP a été introduit par Frederickson et al. (1978) [**55**]. Dans le SCP, les objets doivent être transportés de leur origine à leur destination en utilisant un véhicule avec une capacité unitaire. Chaque objet $o \in X$ est exactement sur deux sommets $x(o)$ et $y(o)$, une fois avec la quantité $d_{x(o),o} = 1$ (l'objet doit partir de $x(o)$) et une fois avec la quantité $d_{y(o),o} = -1$ (il doit arriver en $y(o)$). Les demandes sont donc des demandes d'acheminement de chaque objet o depuis une origine $x(o)$ vers une destination $y(o)$. Le SCP contient les contraintes suivantes :

- Un seul véhicule avec une capacité unitaire est utilisé pour transporter les objets : $m = 1, Q_m = 1$;
- $\forall o \in X, d_{dépôt,o} = 0$, c.-à-d. qu'il n'y a pas d'opération de chargement ou de déchargement au dépôt;
- Pour chaque sommet $i \in V\backslash\{0\}$, il existe exactement un objet $o \in X$ tel que $d_{i,o} \neq 0$, et dans ce cas $d_{i,o} = \pm 1$;
- Pour chaque objet $o \in X$, il y a exactement deux sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = 1$ et $d_{j,o} = -1$. Pour tous les autres sommets $w \in V - \{i, j, dépôt\}$, on a $d_{w,o} = 0$.

Le SCP est une version du GPDP avec l'indication « P/D ». Après la comparaison des contraintes entre SP et SCP, on voit que le SCP est un cas particulier du SP. Comme le SP, il peut être « non-préemptif » ou « préemptif », (voir la Figure 13). Celle-ci représente un exemple simple de SCP «non-préemptif» avec 3 objets transportés, 6 sommets : $\{+1, -1, +2, -2, +3, -3\}$ et un dépôt 0. Le couple de sommet $(+i, -i)$ représente un couple de demande de client i : $+i$ symbolise la demande de chargement, $-i$ symbolise la demande de déchargement.

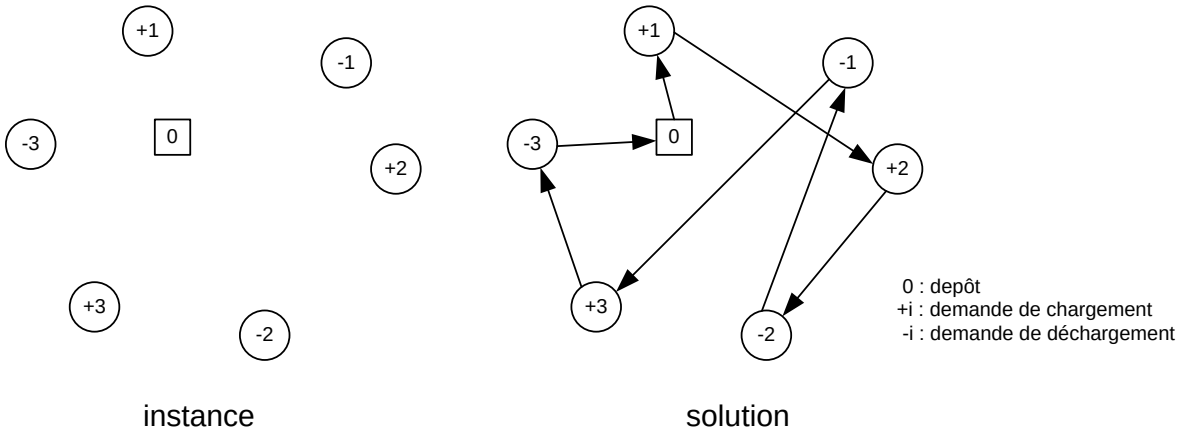

Figure 13 : Le SCP « non-préemptif »

Frederickson et al. (1978) [**55**] ont développé une heuristique dont la performance au pire des cas est de $9/5$ pour le SCP « non-préemptif » sur les graphes généraux. Le SCP « préemptif » sur un arbre a été étudié par Frederickson et Guan (1993) [**56**], qui ont montré que ce problème peut être résolu en temps polynomial. Ces auteurs ont présenté deux algorithmes en$O(k + qn)$ et $O(k + n \log n)$ où k est le nombre d'objets à transporter, n est le nombre de sommets, et $q \leq min\{k, n\}$. Frederickson [**56**] ont ensuite étudié le SCP « non-préemptif » sur un arbre. Ils ont montré que le problème est NP-difficile, en prouvant que le problème de l'**arbre de Steiner** peut être réduit au SCP. Ils ont aussi présenté plusieurs algorithmes d'approximation dont la performance au pire des cas est entre 1,5 et 1,21. L'algorithme dont la performance au pire des cas est 1,5 est basé sur une approximation du problème de l'arbre de Steiner, et s'exécute en temps linéaire.

2.3.2. VRPPD (Vehicle Routing Problem with Pickups and Deliveries)

Le VRPPD concerne le routage d'une flotte de véhicules afin de satisfaire à une série de demandes des clients. Le client spécifie la quantité d'objet à transporter ainsi que le lieu de chargement et le lieu de déchargement. Chaque objet doit être transporté par un véhicule et le

lieu de chargement doit être visité avant le lieu de déchargement. Ce problème a été étudié depuis plus de 30 ans. Le VRPPD est NP-difficile car il généralise le VRP.

- $\forall o \in X, d_{dépôt,o} = 0$;
- Pour chaque sommet $i \in V\backslash\{0\}$, il existe exactement un objet $o \in X$ tel que $d_{i,o} \neq 0$;
- Pour chaque objet $o \in X$, il y a exactement deux sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = -d_{j,o}$, et pour tous les autres sommets $w \in V\backslash\{i, j, dépôt\}$, on a $d_{w,o} = 0$;
- $T = \emptyset$.

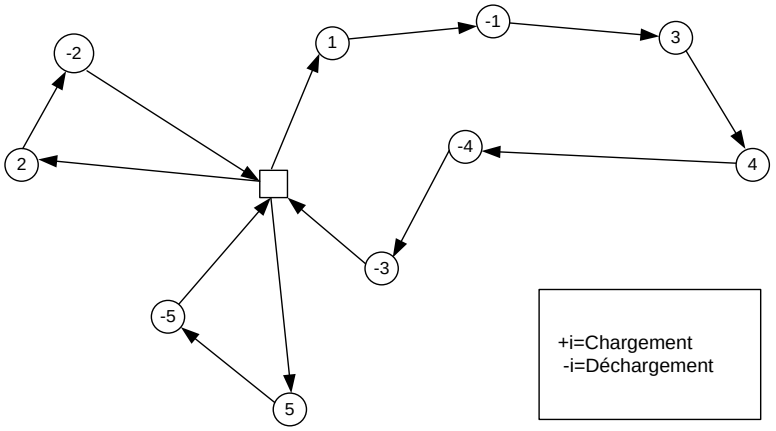

Figure 14 : VRPPD

La Figure 14 représente un exemple simple de VRPPD avec 5 clients, 10 sommets : $\{+1, -1, +2, -2, +3, -3, +4, -4, +5, -5\}$, et un dépôt 0. Le VRPPD est une version du GPDP avec l'indication « P/D ».

Le VRPPDTW (Vehicle Routing Problem with Pickups and Deliveries Time Window) est une extension du VRPPD. Dans ce problème, si un véhicule arrive à un sommet avant le début de sa fenêtre de temps il doit attendre pour commencer son service. Décider si une instance de VRPPDTW est faisable, est NP-Complet [**57**]. Ce problème contient les mêmes contraintes que le VRPPD, plus la contrainte de fenêtre de temps. Pour chaque sommet $i \in V$, la fenêtre de temps $[e_i, l_i]$, indique que e_i est la date au plus tôt et l_i est la date au plus tard pour le service. B_i est le date de début de service à chaque sommet $i \in V$. Les principales contraintes du VRPPDTW viennent comme suit :

- $\forall o \in X, d_{dépôt,o} = 0$;
- Pour chaque sommet $i \in V\backslash\{0\}$, il existe exactement un objet $o \in X$ tel que $d_{i,o} \neq 0$;
- Pour chaque objet $o \in X$, il y a exactement deux sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = -d_{j,o}$, et pour tous les autres sommets $w \in V\backslash\{i, j, dépôt\}$, on a $d_{w,o} = 0$;

- $T = \emptyset$;
- Pour chaque sommet $i \in V$, $e_i \leq B_i \leq l_i$, où B_i est la date de service en sommet i.

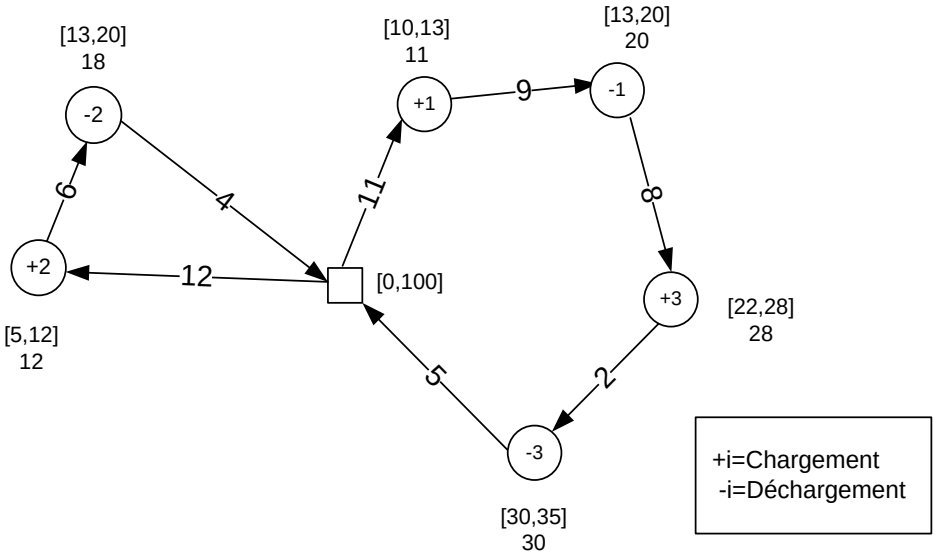

Figure 15 : VRPPDTW

La Figure 15 représente un exemple simple du VRPPDTW avec 2 véhicules, 3 objets et 7 sommets $\{0, +1, -1, +2, -2, +3, -3\}$. Deux labels sont associés à chaque sommet (hors dépôt). Le label $[x, y]$ représente la fenêtre de temps de chaque sommet, et le label z représente la date de début de service à chaque sommet. Un label associé à une arête représente la longueur ou le temps nécessaire pour traverser cette arête.

Un algorithme de génération de colonnes pour le VRPPD avec fenêtres de temps a été proposé par Dumas et al. (1991) [**58**]. L'algorithme peut également traiter les cas avec multiples dépôts et différents types de véhicules. Il peut résoudre des instances comprenant jusqu'à 55 demandes. Savelsbergh et Sol (1998) [**59**] ont développé un autre algorithme exact pour le VRPPD avec fenêtres de temps. L'algorithme a été conçu et intégré dans le système d'aide à la décision pour une société de distribution, il est capable d'insérer de nouvelles demandes de façon dynamique. Xu et al. (2003) [**60**] ont proposé un autre algorithme de génération de colonnes pour un VRPPD possédant une série de restrictions adaptées à la logistique réelle. Les instances incluent des fenêtres de temps multiples, des règles de travail pour le conducteur, des contraintes de compatibilité pour les transporteurs, etc. Le problème principal est résolu par un solveur LP, tandis que les sous-problèmes sont résolus par une méthode heuristique. L'algorithme est capable de générer des solutions qui sont proches de l'optimale pour des instances générées aléatoirement, comprenant jusqu'à 200 demandes dans un temps raisonnable. Récemment, un algorithme de « branch-and-cut » pour le VRPPDTW a été proposé par Röpke et Cordeau (2009) [**61**]. Le VRPTW est formulé comme un ensemble de problème de « Set Partitioning », et la résolution de deux problèmes du plus court chemin

(Shortest-Path) permet d'obtenir une évaluation pour ce problème. Les auteurs ont prouvé que la relaxation de la formulation linéaire des problèmes « Set Partitioning » implique des inégalités valides utilisées par [**61**], et Cordeau (2006) [**62**] dans des algorithmes de « branch-and-cut ». L'algorithme est capable de résoudre de façon optimale quelques instances comprenant jusqu'à 500 demandes.

Une heuristique de recherche taboue réactive [**63**] pour le VRPPDTW a été développée par Nanry et Barnes [**64**]. Une solution est d'abord obtenue en utilisant une procédure gloutonne. Trois types de voisinages ont été considérés. Le premier consiste à déplacer une paire de sommets (chargement et déchargement) d'une tournée à une autre. Le deuxième consiste à échanger une paire de sommets (chargement et déchargement) entre les tournées. Le troisième consiste à réorganiser des clients dans une tournée. Pendant la recherche, les solutions doivent respecter les contraintes de capacité du véhicule et les fenêtres de temps. La fonction objectif de l'heuristique est modifiée pour inclure une pénalité. Un mécanisme de recherche hiérarchique a été développé afin de choisir parmi les trois voisinages selon l'état des contraintes de fenêtres de temps par rapport à la solution courante. Des résultats optimaux ou proche-optimaux ont été obtenus en moins de cinq minutes sur une série d'instances aléatoires de 100 clients.

2.3.3. SVRPPD (Single Vehicle Routing Problem with Pickups and Deliveries)

Le SVRPPD est un cas particulier du VRPPD, dans laquelle il n'y a qu'un seul véhicule. L'une des premières études sur les SVRPPD (sans la contrainte de la capacité) a été faite par Stein (1978) [**65**]. Les contraintes du SVRPPD sont :

- $m = 1$;
- $\forall o \in X, d_{dépôt,o} = 0$;
- Pour chaque sommet $i \in V\backslash\{0\}$, il existe exactement un objet $o \in X$ tel que $d_{i,o} \neq 0$;
- Pour chaque objet $o \in X$, il y a exactement deux sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = -d_{j,o}$, et pour tous les autres sommets $w \in V\backslash\{i, j, dépôt\}$, on a $d_{w,o} = 0$;
- $T = \emptyset$.

Une heuristique pour le SVRPPD sans contrainte de capacité a été proposée par Psaraftis [**66**]. Elle construit d'abord une tournée du TSP à travers tous les sommets de chargement et de déchargement. Puis une solution du SVRPPD est obtenue en traversant le TSP dans le sens des aiguilles d'une montre jusqu'à ce que tous les sommets soient visités. Les sommets déjà visités ou les sommets de déchargement, pour lesquelles le sommet de chargement correspondant n'est pas encore visité, sont ignorés. L'auteur a prouvé que si l'heuristique de Christofides [**53**] pour le TSP est utilisée dans la première étape, l'heuristique est en $O(n^2)$ et a une performance au pire des cas qui est 3 fois la solution optimale. Une des méthodes de

recherche locale pour le SVRPPD est une procédure « $k - interchange$ » présenté par Psaraftis [**67**], qui étend l'idée proposée par [**21**] pour le TSP

Desrosiers et al. (1986) [**68**] ont développé l'algorithme de programmation dynamique pour le SVRPPD avec fenêtres de temps et sans contrainte de capacité. Cet algorithme a résolu des instances comprenant jusqu'à 40 sommets dans un temps très court. Kalantari et al. (1985) [**69**] ont développé des procédures de « branch-and-bound » pour le SVRPPD avec (ou sans) contrainte de capacité. Leurs procédures sont des extensions de l'algorithme de Little et al. (1963) [**70**] pour le TSP. Ruland et Rodin (1997) [**71**] ont présenté un algorithme de « branch-and-cut » en utilisant quatre classes d'inégalités.

Plusieurs méthodes heuristiques ont été proposées pour plusieurs versions du SVRPPD. La version avec fenêtre de temps, a été résolue par recherche locale [**72**], algorithme Tabou [**73**], algorithme génétique (Pankratz, 2005), et méthode d'insertion (Lu et Dessouky, 2006). Une heuristique de perturbation a aussi été développée par Renaud, Boctor et Laporte (2002) pour le SVRPPD sans fenêtre de temps.

Le cas particulier du SVRPPD, appelé TSPPD avec la contrainte « LIFO » (TSPPDL), qui se pose lorsque les déchargements doivent respecter une règle «last-in first-out», a été étudié par Pacheco (1997) [**74**], Carrabs et al. (2006) [**75**], et Cordeau et al. (2010) [**76**]. Le service LIFO signifie que le véhicule ne peut effectuer de déchargement que si celui-ci est associé à la dernière opération de chargement.

2.3.4. DARP (Dial-a-Ride Problem)

Le DARP est un cas spécial du VRPPDTW dans lequel les « objets » sont des clients devant être transportés de leurs origines à leurs destinations. Le DARP peut avoir plusieurs contraintes concernant des fenêtres de temps, une borne sur le temps maximal de trajet pour chaque objet o à transporter (Riding time), noté L_o , et la qualité de service [**77**,**62**]. L'application principale du DARP est le service « porte-à-porte » de transport offert aux personnes âgées et aux personnes à mobilité réduite dans de nombreuses villes. Pour le DARP, les objets transportés sont un groupe de personnes. Les contraintes du DARP sont :

- $\forall o \in X, d_{dépôt,o} = 0$;
- Pour chaque sommet $i \in V\backslash\{0\}$, il existe exactement un objet $o \in X$ tel que $d_{i,o} \neq 0$;
- Pour chaque objet $o \in X$, il y a exactement deux sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = -d_{j,o}$, et pour tous les autres sommets $w \in V\backslash\{i, j, dépôt\}$, on a $d_{w,o} = 0$;
- $T = \emptyset$;
- Pour chaque sommet $i \in V$, la date de début de service doit être dans la fenêtre de temps $[e_i, l_i]$;

- Pour chaque objet $o \in X$, il y a exactement deux sommets sommets $i, j \in V\backslash\{0\}$ tel que $d_{i,o} = -d_{j,o}$, et l'écart entre les instants de passage en ces deux sommets ne doit pas excéder une borne L_o.

L'une des premières études pour le DARP avec un seul véhicule a été réalisée par Psaraftis (1980). qui a examiné le cas dans lequel les clients souhaitent être desservis aussitôt que possible. Ce problème a été résolu de manière exacte par programmation dynamique. L'objectif est de minimiser une combinaison du temps total de service et de l'insatisfaction des clients. Le niveau d'insatisfaction est une fonction linéaire dépendant du temps d'attente pour le chargement et du temps de trajet pour le client. Sexton et Bodin (1985) [**78**] [**79**] ont proposé un algorithme heuristique basé sur la décomposition de Benders pour le DARP avec un seul véhicule. Dans leur problème, les clients spécifient une date souhaitée de déchargement et le retard n'est pas permis. L'objectif est de minimiser les insatisfactions des clients.

Dans la version multi-véhicules du DARP, une attribution des demandes aux véhicules doit être effectuée en plus de la conception et l'ordonnancement de chaque trajet. En conséquence, l'espace des solutions est beaucoup plus grand que dans le DARP avec un seul véhicule, et le problème devient beaucoup plus difficile à résoudre.

Jaw et al. (1986) [**80**] ont développé une heuristique pour le DARP avec multi-véhicules, où les clients spécifient soit une date souhaitée de chargement soit une date souhaitée de déchargement, mais pas les deux. Les véhicules ne sont pas autorisés à être vides tout en allant chercher les clients, et l'objectif consiste à trouver un compromis entre la minimisation des coûts et la minimisation des temps de trajet, où le temps de service est mesuré par la durée du temps de parcours effectué entre une opération de chargement et l'opération associée de déchargement.

L'algorithme insère des clients, en utilisant un ordre prédéterminé, dans les tournées. Chaque étape de l'algorithme tient compte de tous les moyens possibles d'insérer un client dans toutes les tournées. Le mouvement de coût minimal est mis en application. Les auteurs ont aussi développé une version de leur algorithme dans laquelle plusieurs clients peuvent être insérés en même temps. Des instances avec un maximum de 2617 clients et 28 véhicules ont été testées.

Une modification de l'heuristique de Jaw et al. (1986) [**80**] a été proposée par Potvin et Rousseau (1992), qui ont modifié la procédure d'insertion en gardant les meilleures solutions partielles à chaque itération. Cette méthode réduit le comportement myope de l'heuristique précédente. En outre, ils ont proposé deux nouvelles phases. La première est une phase d'initialisation qui groupe quelques clients pour créer des tournées partielles. L'autre est une procédure de post-optimisation basée sur des échanges de clients. Cette heuristique génère de meilleurs résultats que celle de [**80**]. mais nécessite plus de temps de calcul.

Cullen et al. (1981) [**81**] ont développé un routage interactif heuristique pour le DARP et les autres problèmes de routage sans fenêtres de temps en utilisant un technique « cluster-first route-second ». Cette catégorie d'algorithmes groupe d'abord des clients, puis conçoit une tournée pour chaque groupe. Dans l'heuristique, Cullen, Jarvis et Ratliff ont résolu le groupement et les sous-problèmes de routage par la génération de colonnes.

Une autre méthode heuristique pour DARP multi-véhicules a été proposée par Roy et al (1983) [**82**] qui ont étudié une version complexe du DARP, incluant les fonctionnalités suivantes : les demandes peuvent inclure une ou plusieurs personnes handicapées, avec ou sans fauteuil roulant, et éventuellement une personne accompagnante. Des mesures de qualité de service peuvent être calculées, la vitesse du véhicule peut varier avec le temps, etc. Le client indique une date souhaitée de chargement ou une date souhaitée de déchargement. L'objectif est d'abord de minimiser les coûts d'opération en assurant une qualité de service minimale. Dans une phase de post-optimisation, l'algorithme essaye de maximiser la qualité du service sans augmenter les coûts d'opération atteints dans la première phase. Cette heuristique est également une méthode de « cluster-first route-second ». Des groupes de clients sont créés selon une relation de proximité. Une mesure d'affinité est utilisée pour guider l'insertion parallèle des groupes des demandes dans les tournées. Cet algorithme a été testé avec succès sur des instances réels comprenant jusqu'à 578 demandes. Bodin et Sexton [**83**] ont développé une autre heuristique « cluster-first route-second » pour le DARP dans lequel le client spécifie la date souhaitée de déchargement. L'objectif est de minimiser la somme de l'écart entre date de déchargement réel et date de déchargement souhaité, ainsi que le temps de trajet en excès pour chaque client. La première heuristique divise les clients en groupes, et construit un tour sur chaque groupe en utilisant l'algorithme de Sexton et Bodin [**78**,**79**] pour le DARP avec un seul véhicule. Enfin, une méthode appelée « Swapper » tente de déplacer des clients entre les tournées et effectue la re-optimisation de tournée.

Un algorithme de « Branch-and-cut » a été développée par Cordeau [**62**] pour le même problème. Cet algorithme utilise plusieurs nouvelles inégalités valables pour le DARP ainsi que les inégalités connues pour des problèmes de tournées de véhicules. L'algorithme peut résoudre des instances incluant jusqu'à 32 demandes et quatre véhicules dans la limite d'un temps de calcul de quatre heures. Un autre algorithme de « branch-and-cut » pour le DARP et le VRPPDTW a été développé dans [**84**]. Les auteurs ont adapté des méthodes de propagation de contraintes et ils ont introduit deux nouvelles familles d'inégalités. Les résultats expérimentaux ont montré que ces approches sont plus rapides que celle utilisée dans [**62**].

2.3.5. VRPPDT (Vehicle Routing Problem with Pickups and Deliveries and transshipments)

Dans le VRPPDT, les objets peuvent être transportés par différents véhicules depuis leur origine jusqu'à leur destination. Il faut alors laisser les objets en des points spécifiques appelés

points de transbordement. Le VRPPDT contient les mêmes contraintes que le VRPPD sauf que l'ensemble de points de transbordement n'est pas vide ($T \neq \emptyset$). En raison de l'existence de points de transbordement, le nombre de solutions du VRPPDT est beaucoup plus grand que dans le VRPPD standard. Cette flexibilité permet de satisfaire un plus grand nombre de demandes, mais il pose aussi le défi de création de méthodes efficaces pour trouver les solutions optimales dans un espace de solutions qui est très grand.

Cortés et al. [**85**] ont proposé un modèle de programmation linéaire en nombres entiers et mixtes pour le DARP. Les auteurs ont implémenté un algorithme de « branch-and-cut », basé sur une méthode de décomposition, et ont résolu des instances très petites avec jusqu'à huit clients et un point de transbordement. Enfin, Mitrovic-Minic et Laporte [**86**] ont développé un algorithme en deux-phases pour le VRPPDT avec fenêtres de temps. La première phase utilise des insertions de mauvaise qualité, et la deuxième phase est une phase d'amélioration basée sur la réinsertion des demandes. Les demandes peuvent être réparties dans les deux phases pour permettre le transbordement. L'heuristique a permis de résoudre des instances générées de façon aléatoire avec un maximum de 100 demandes et quatre points de transbordement. Les auteurs ont conclu que des gains significatifs peuvent être obtenus en permettant des transferts lorsque les demandes sont groupées.

La Figure 16 représente le schéma de classification pour le GPDP avec la structure « 1-1 » ainsi que les relations principales entre les problèmes de type GPDP.

Nous avons présenté une vue d'ensemble des problèmes statiques de chargement et de déchargement. Les principales relations entre ces problèmes sont représentées dans la Figure **17**.

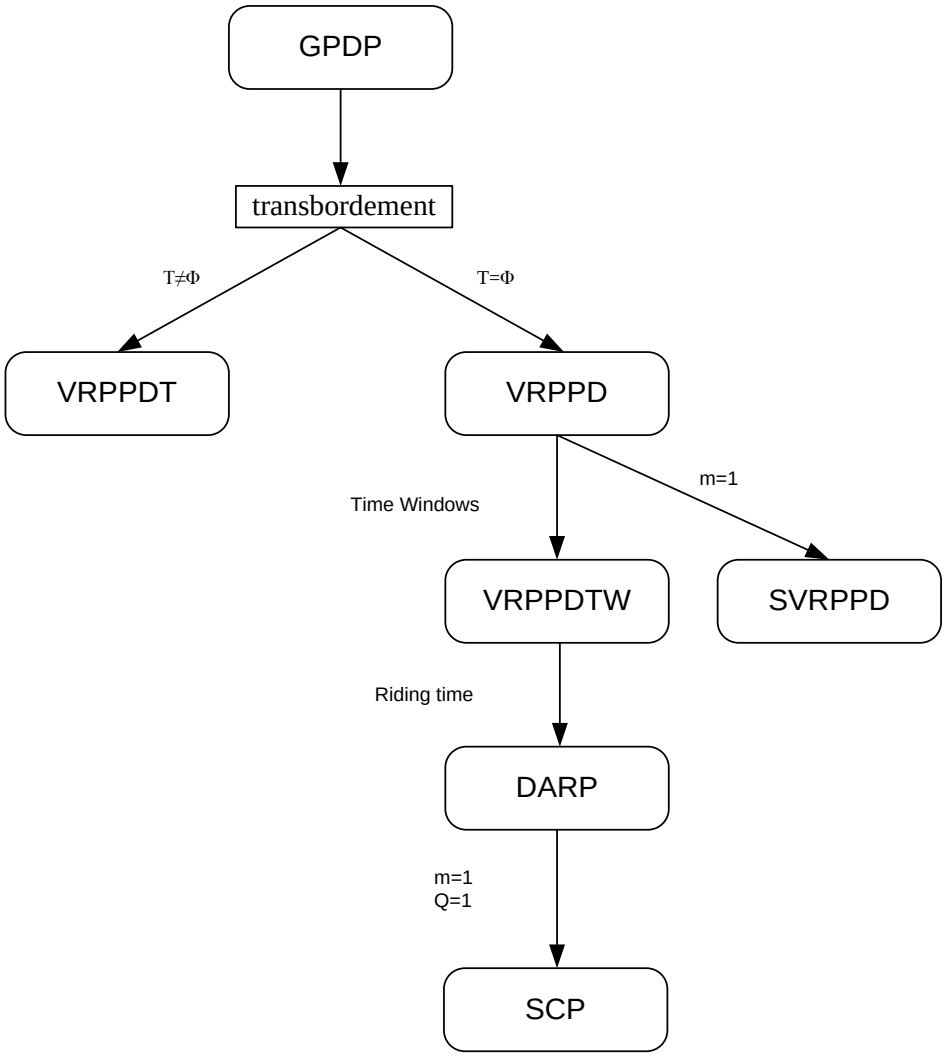

Figure 16 : La classification pour le GPDP avec la structure « 1-1 »

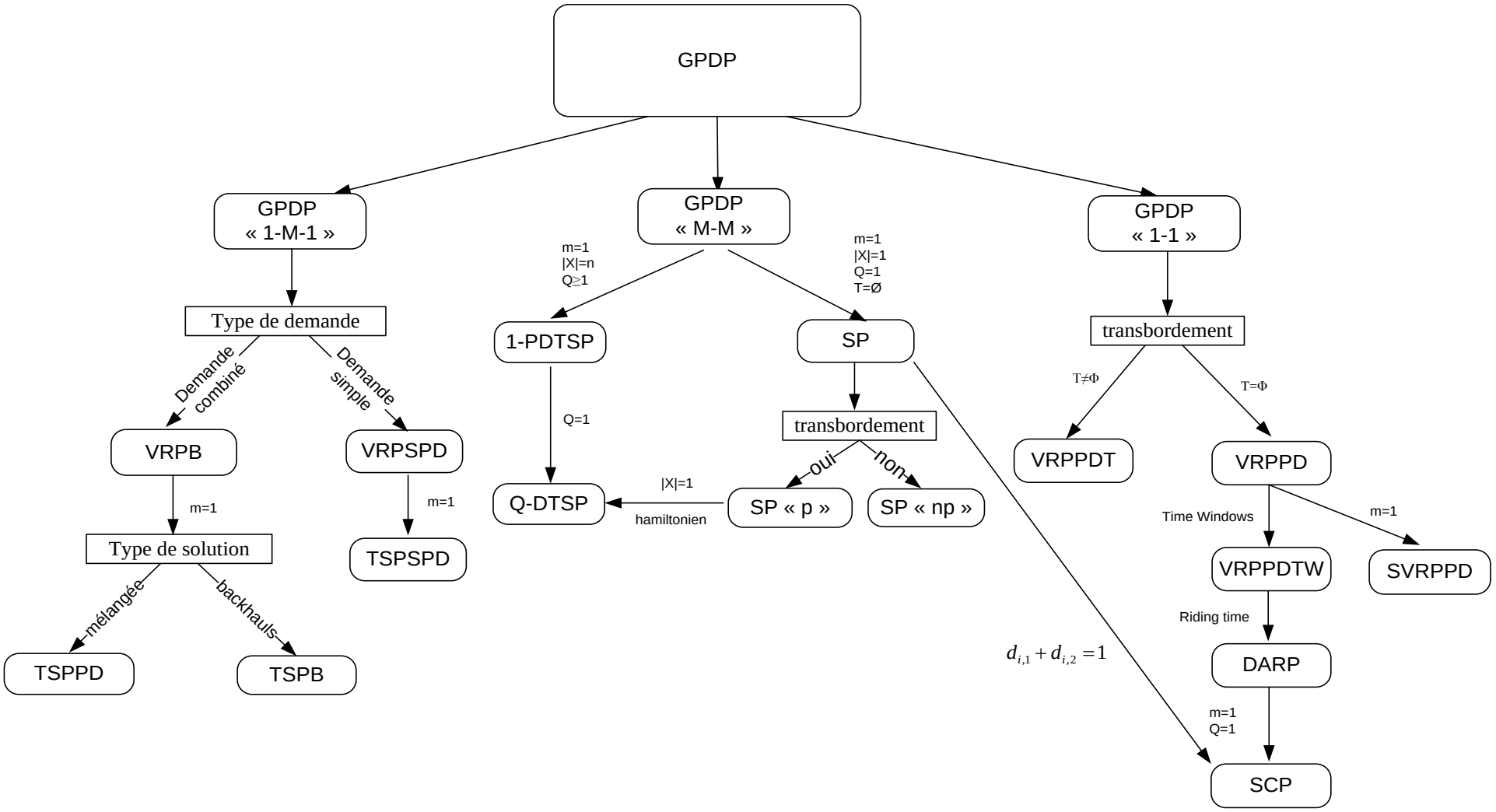

Figure 17 **: Classification du GPDP**

3. Classification académique pour le GPDP dynamique

Un problème de routage est dit « statique » lorsque toutes les données d'entrée du problème sont connues a priori. Dans un problème de routage dynamique, certaines des données d'entrée sont indiquées ou mises à jour pendant que les opérations ont lieu. Dans le problème dynamique de chargement et de déchargement, les données d'entrée qui sont indiquées au fil du temps sont en général les demandes des utilisateurs. Contrairement à ce qui se passe dans un problème « statique », l'horizon de planification d'un problème dynamique peut être illimité. Par conséquent, une solution à un problème dynamique ne peut pas être un résultat statique, mais plutôt une stratégie de gestion des véhicules qui, en utilisant les informations indiquées, spécifie quelles actions doivent être effectuées au fur et à mesure que les opérations se déroulent.

La plupart des études sur les problèmes de chargement et déchargement se sont concentrées sur le cas statique et peu de travail a été fait sur les cas dynamiques. Un exemple d'un problème dynamique apparaît dans le transport des personnes handicapées et âgées dans les zones urbaines, dès lors que l'on se donne comme objectif de rendre le système le plus réactif possible. Dans cette section, nous fournissons un cadre général pour le GPDP avec la structure « 1-1 », nous décrivons les concepts de solution et nous présentons la littérature sur ces problèmes.

Dans le GPDP dynamique avec la structure « 1-1 », la demande de client se compose d'une charge pour un service de transport d'un sommet origine vers un sommet destination. Le sommet origine (destination) de la demande i est dénoté par $i^+(i^-)$, et la charge est dénoté par d_i. R est un ensemble de demandes. Le GPDP dynamique travaille sur un graphe complet et orienté $G = (V, A)$, avec un ensemble de sommets $V = \{0\} \cup \{i^+ | i \in R\} \cup \{i^- | i \in R\}$. Le sommet 0 est le dépôt. $A = \{(i, j) | i, j \in V, i \neq j\}$ est un ensemble d'arcs. Chaque arc $(i, j) \in A$ a un coût (ou une longueur) non négatif $c_{i,j}$ (habituellement égale au temps de déplacement entre le sommet i et le sommet j). Une tournée est un circuit, démarrant et finissant au dépôt. Le sommet de chargement précède naturellement le sommet de déchargement.

A chaque instant t, chaque véhicule sert un sommet, attend à un sommet, ou se déplace vers un sommet. Dans chacun de ces cas, la demande liée au sommet est connue au temps t. Les décisions qui doivent être prises peuvent être divisées en deux classes de décision :

- attendre ou rouler ;
- accepter ou rejeter.

Une décision de type « attendre » ou « rouler » se produit quand un véhicule achève de servir un sommet. En supposant que le temps actuel est t, le système doit choisir entre attendre à

l'emplacement actuel du véhicule ou se déplacer vers un sommet dont la demande est connue au temps t. Si la décision « attendre » a été choisie, une série de décision de type « attendre » à chaque pas de temps jusqu'à ce que la décision « rouler » ait été choisie en précisant le sommet suivant à visiter. Enfin, tout au long de l'horizon temporel, des demandes de services sont reçues. Ces évènements déclenchent des décisions d'acceptation ou de rejet à chacune des périodes de décision auxquelles le système doit décider, s'il accepte ou non la demande.

Un algorithme du GPDP dynamique avec la structure « 1-1 » consiste en un algorithme qui décide, quelles sont les actions à effectuer, à chaque fois qu'il est appelé pour traiter un ensemble de demandes. Le système (les véhicules) étant dans un état donné caractérisé par les parcours en cours, il s'agit de déterminer les demandes qu'il va accepter et avec quels véhicules. Toutes les demandes connues à l'avance et toutes les demandes acceptées doivent être satisfaites. La qualité d'un algorithme par le GPDP dynamique peut être mesurée par le nombre ou la proportion de demandes acceptées, la distance totale parcourue par les véhicules, et le temps nécessaire pour décider s'il accepte ou non une demande. Des hypothèses supplémentaires et des contraintes sur les tournées des véhicules, telles que des fenêtres de temps ou des temps maximum de trajet pour les clients peuvent être à prendre en compte selon l'application considérée.

La littérature sur le GPDP dynamique avec la structure « 1-1 » peut être divisée en trois catégories de problèmes. Dans le cas où les demandes sont pour le transport des objets et les véhicules peuvent servir plus d'une demande à la fois, le problème s'appelle le VRPPD dynamique (Dynamic Vehicle Routing Problem with Pickups and Deliveries). Lorsque les véhicules peuvent seulement traiter une demande à la fois (par exemple, quand toutes les demandes ont une charge égale à la capacité du véhicule), le problème s'appelle le SCP dynamique (Dynamic Stacker Crane Problem). Enfin, dans le cas où les demandes de transport se composent de passagers, le problème est appelé DARP dynamique (Dynamic Dial-a-Ride Problem). Ce problème ressemble au VRPPD dynamique. Toutefois, le DARP dynamique diffère du VRPPD dynamique en raison des fenêtres de temps serrées et des contraintes sur le temps maximal de trajet du client (Ride time) qui sont généralement imposées pour assurer une bonne qualité de service aux clients.

3.1. Le SCP dynamique

Dans le SCP dynamique, chaque demande doit être transportée directement à partir de son lieu d'origine vers son lieu de destination. La restriction selon laquelle une seule demande peut être traitée à la fois par un véhicule est généralement due à la capacité. L'application principale du SCP dynamique concerne la gestion d'une flotte de véhicule pour déplacer des chargements entre une origine et une destination. Pour cette raison le SCP dynamique s'appelle « Dynamic Full Truckload Pickup and Delivery Problem ». Le problème est dynamique parce que les demandes des clients arrivent sans interruption pendant les

opérations. Le problème concerne un horizon de planification de plusieurs jours. En général, environ de la moitié des demandes reçues sont pour le jour même. Powell et al. [**87**] a rédigé une revue des modèles dynamiques pour ces problèmes.

Powell [**88**] a proposé, une méthodologie pour modéliser et résoudre les SCP dynamiques rencontrés dans l'industrie du chargement de camion. Dans son modèle, la zone de planification est divisée en régions. Plusieurs données sont supposées être disponibles, y compris la distribution de probabilité des demandes entre chaque paire de régions pour chacune des journées de l'horizon de planification ainsi que le revenu net moyen par demande. Le problème est représenté comme un problème de flot dans un réseau où chaque sommet représente une région servie un jour spécifique. L'ensemble d'arc se compose de deux types d'arcs. Le premier type d'arcs représente des informations sur les demandes déterministes connues alors que le second type d'arcs, dénommés « arcs stochastiques », est utilisé pour estimer la valeur de l'envoi d'un véhicule supplémentaire à une région spécifique à un instant spécifique. Dans Power et al. (1988) [**89**], une extension de cette représentation du réseau est appliquée à la maximisation des profits d'un transporteur commercial aux Etats-Unis, qui possède avec plus de 5800 remorques.

Un autre problème en temps réel de chargement et déchargement multi-véhicules dans lequel aucune information stochastique n'est considérée a été présenté par Yang et al. (1998) [**90**]. Dans ce problème, chaque demande a une fenêtre de temps pendant laquelle un chargement doit avoir lieu si la demande est acceptée. Les auteurs ont modélisé le problème statique comme un programme linéaire en nombres entiers. Pour gérer la composante dynamique, le problème statique est résolu plusieurs fois dans un cadre à horizon roulant. Dans leur article, les auteurs présentent une comparaison entre la stratégie consistant à résoudre le MIP statique à chaque fois qu'une nouvelle demande arrive et d'autres stratégies plus rapides qui limitent la taille de l'espace des solutions. Les résultats sur des instances avec quatre véhicules ont prouvé que si la stratégie par résolution du MIP statique surpassait tous les autres, en termes de quantité de résultats, quelques stratégies simples ont produit des résultats similaires tout en exigeant beaucoup moins d'effort de calcul.

Une étude visant à quantifier la valeur de connaissance des demandes à l'avance a été effectuée par Tjokroamidjojo et al. (2006) [**91**]. Les auteurs ont considéré un problème de chargement avec un horizon de 20 jours, des fenêtres de temps flexibles et la possibilité de rejeter les demandes. Dans cette étude, aucune information stochastique sur des demandes futures n'est considérée. La stratégie de résolution consiste à résoudre une MIP statique à chaque fois qu'une nouvelle demande arrive.

Une approche à base d'agents pour une version du SCP dynamique a été développée par Mes et al. (2007) [**92**]. Ces auteurs ont étudié un problème, inspiré d'un réseau automatisé de transport AGVs, avec des contraintes de fenêtres de temps flexibles et dans lesquelles les demandes ne peuvent pas être rejetées. Un modèle à base d'agents essaye de modéliser des

phénomènes complexes utilisant un ensemble de différents agents individuels avec des tâches et des objectifs spécifiques. Les auteurs ont conçu quatre types d'agents : les agents du véhicule, les agents de la flotte, les agents du travail et les agents de l'expéditeur. L'offre se compose d'un prix, d'une heure de départ prévue et d'une heure d'arrivée prévue. Enfin l'agent de travail choisit une offre en utilisant le mécanisme de Vickrey, dans lequel la demande est assignée au plus haut soumissionnaire mais au prix du deuxième plus haut soumissionnaire. Les expérimentations de simulation sur des instances avec 20 AGVs et 20 lieux ont prouvé que l'approche à base d'agents est généralement plus performante que les heuristiques d'ordonnancement simple.

Un SCP complexe dynamique découlant d'un problème en mode réel de répartition d'un système de monorail électrique a été étudié par Gutenschwager et al. (2004) [**93**]. Les auteurs ont développé une recherche taboue et une heuristique de recuit simulé qui sont capables de répondre à des conditions en temps réel. La stratégie de résolution consiste à résoudre un problème statique plusieurs fois, sans l'aide des informations stochastiques sur les demandes futures. L'objectif du problème statique a été modifié afin d'améliorer la performance globale du problème en ligne. L'application des algorithmes proposés a réduit le nombre de charges électriques monorails de 24 à 22 ainsi que le nombre moyen et maximum d'opérations de transport simultanées effectuées.

3.2. Le VRPPD dynamique

Dans le VRPPD dynamique, les véhicules peuvent servir plus d'une demande en même temps. Dans ce type de problème, les demandes concernent généralement le transport d'objets (par exemple, des lettres, des charges, des colis, etc.). En général, lorsque les fenêtres de temps sont présentes, elles ne sont pas serrées. Dans plusieurs de ces problèmes, il n'y a aucune contrainte de capacité. Un exemple d'un VRPPD dynamique est produit par les entreprises de courrier express situé dans de nombreuses villes. Ces entreprises reçoivent habituellement des centaines ou des milliers de demandes dynamiques dans le même jour. Les demandes portent sur le transport des lettres ou des colis d'un point de collecte à un point de livraison. La première partie de cette section est consacrée à un examen des études de l'analyse concurrentielle des PDP. Le principal problème étudié dans ce domaine est un PDP simple appelé le problème à la demande en ligne (OlDARP). Malgré sa ressemblance étroite avec le DARP dynamique, l'OlDARP ne considère pas de contraintes de minimisation des inconvénients pour les utilisateurs qui sont présents dans le DARP tels que les contraintes de fenêtres de temps et de temps maximum de trajet. Pour cette raison, il semble raisonnable de passer en revue l'OlDARP dans cette section plutôt que dans la section sur le DARP dynamique. La deuxième partie de cette section évoque des algorithmes heuristiques mis en œuvre pour des problèmes plus complexes qui améliorent le modèle au monde réel.

Une des quelques études analytiques pour un VRPPD dynamique et stochastique a été effectuée par Swihart et Papastavrou (1999) [**94**]. Les auteurs ont considéré le problème de routage avec un seul véhicule pour servir des demandes de transport formulées selon un processus de Poisson. Les lieux de chargement et de déchargement sont indépendants et uniformément réparties sur une région convexe avec une métrique euclidienne. L'objectif est de minimiser le temps prévu où les demandes demeurent dans le système. Des bornes sur les performances de certaines politiques de routage proposées sont présentées à la fois pour les véhicules avec capacité unitaire et pour les véhicules avec capacités différentes. Waisanen et al. (2008) [**95**] ont étudié deux versions de ce problème avec plusieurs véhicules, et sans la contrainte de capacité. Dans la première version, lorsqu'une demande arrive elle contient seulement le lieu de chargement, le lieu de déchargement n'étant connu que lorsque le véhicule effectue le chargement. Dans la deuxième version, qui est la plus commune, les endroits de chargement et de déchargement sont disponibles lorsque la demande est reçue. Pour les deux versions, les auteurs ont imposé une limite inférieure à la performance de toute politique de contrôle et ont présenté une politique de contrôle de la réalisation de la borne inférieure à un facteur constant. Waisanen et al. (2009) [**96**] ont étendu certains de ces résultats pour les problèmes de tournées de véhicules dynamiques avec « multi-stage » dans lesquels une demande nécessite un service en plus de deux lieux.

Savelsbergh et Sol (1998) [**59**] ont proposé un algorithme pour le VRPPD dynamique multi-véhicules. Il prend en compte des contraintes standard, ainsi que des caractéristiques spécifiques du problème tels que la présence des véhicules avec des caractéristiques spéciales et des capacités différentes, des fenêtres de temps, et des contraintes de période. L'optimisation doit être réalisée sur un horizon de plusieurs jours et l'objectif est de réduire au minimum le coût de transport pour le conducteur, qui dépend de nombreux facteurs tels que la distance parcourue et le nombre de nuit hors du domicile. Les auteurs décomposent le problème en une série de problèmes statiques avec un sous-ensemble des demandes les plus récurrentes dans un cadre à horizon roulant. Chaque problème statique est résolu au moyen d'une heuristique de « branch-and-price ». La fonction objectif pour le problème statique a été modifiée afin d'améliorer la performance globale de l'algorithme dans l'environnement dynamique. L'algorithme a été testé sur une instance réelle de 10 jours avec plus de 200 demandes actives à tout moment. Quand il y avait un grand nombre de demandes, la solution donnée par l'algorithme a surpassé celle produite par les planificateurs.

Un algorithme basé sur l'horizon roulant pour un VRPPD dynamique sans la contrainte de capacité, et avec des fenêtres de temps a été développé par Mitrovic-Minic et Laporte (2004) [**97**]. Le problème étudié par ces auteurs a été motivé par le problème de routage et d'ordonnancement qui est rencontré par les entreprises de courrier pour faire le chargement et le déchargement de lettres et petits colis dans la même journée. L'objectif est de servir toutes les demandes, tout en minimisant la distance totale parcourue. Chacun des problèmes statiques résolus est optimisé avec une heuristique en deux phases. La première phase consiste en une procédure d'insertion, tandis que la seconde se compose d'un algorithme de recherche

taboue similaire à celui de Gendreau et al. (2006) [**98**]. Même si aucune distribution de probabilité sur les demandes entrantes n'est considérée, différentes stratégies d'attente ont été développées. Grâce à une série de tests de calcul, les auteurs ont montré que la distance totale parcourue a été réduite lorsqu'une stratégie d'attente avancée est utilisée. Des résultats similaires sur les avantages des stratégies d'attente ont été analytiquement prouvés par Branke et al. (2005) [**99**] pour un problème simplifié.

Saez et al. (2008) [**100**] ont développé un algorithme adaptatif de contrôle prédictif pour le VRPPD dynamique avec contrainte de capacité, basé sur des algorithmes génétiques et sur une technique de classification floue. Dans leur problème, les fenêtres de temps ne sont pas présentes, et la fonction objectif prend en compte l'attente et le temps d'acheminement pour les demandes, ainsi que le temps de déplacement des véhicules à vide. Les auteurs ont utilisé un contrôle adaptatif prédictif hybride (HAPC), basé sur l'utilisation de variables d'état pour modéliser le problème. L'objectif prend en compte les demandes futures grâce à l'utilisation des probabilités calculées à partir des données historiques, en utilisant une méthode de classification floue. Le HAPC est optimisé au moyen d'un algorithme génétique. Les tests sur une instance avec 120 demandes au titre d'une distribution hétérogène ont prouvé que l'algorithme obtient des résultats jusqu'à 22% meilleurs quand il prend en compte les demandes futures, au lieu de la stratégie à court terme qui ne tient compte que des demandes reçues.

Ghiani et al. (2009) [**101**] ont proposé des algorithmes pour un VRPPD dynamique sans la contrainte de capacité qui se présente dans la gestion de services de courrier au même jour. Dans leur modèle, à chaque demande i est associée une fonction non décroissante et convexe $f_i(t)$, qui exprime le coût inconvénient pour effectuer le déchargement à l'instant t. Suivant les idées de Van Hentenryck et Bent (2004) [**102**], les auteurs évaluent les solutions possibles en utilisant une fonction objectif qui prend en compte les demandes futures grâce à un échantillonnage. Afin de réduire les temps de calcul, l'échantillonnage se fait seulement sur un horizon à court terme. Dans le premier algorithme, appelé « procédure d'insertion d'anticipation », des solutions intégrant les requêtes entrantes sont construites en utilisant une méthode d'insertion simple. Une recherche locale basée sur des mouvements simples est utilisée pour le second algorithme, appelé algorithme de recherche locale d'anticipation. Dans une série de tests avec un maximum de 600 demandes, les deux algorithmes ont surpassé de 11% à 69% l'algorithme réactif correspondant qui n'utilise pas d'échantillonnage. Les résultats ont également montré une amélioration peu significative des résultats produits par la recherche locale d'anticipation grâce à la procédure d'insertion d'anticipation.

3.3. Le DARP dynamique

Dans le DARP dynamique, les demandes se composent des clients qui doivent être transportés de leurs origines vers leurs destinations. En général, ces problèmes contiennent plusieurs

contraintes qui visent à limiter l'insatisfaction des clients. Des exemples de telles contraintes sont des fenêtres de temps serrées et des temps maximum de trajet pour chaque client. L'application principale du DARP est le transport de personnes âgées et handicapées dans les villes. Des modèles et des algorithmes pour les versions statiques et dynamiques du DARP ont été étudiés par Cordeau et Laporte (2007) [**103**].

Une des premières études sur le DARP dynamique a été effectuée par Psaraftis (1980) [**104**] qui a considéré le cas avec un seul véhicule. Dans ce problème, les clients souhaitent être desservis aussitôt que possible et l'objectif est de minimiser une combinaison pondérée du temps total de service et de l'insatisfaction des clients. D'abord, un algorithme exact de programmation dynamique $\left(O(n^2 3^n)\right)$ a été présenté pour le cas statique. Pour le cas dynamique, quand une nouvelle demande arrive, l'instance statique est mis à jour et ré-optimisée en fixant la tournée partielle déjà construite. En raison de la complexité de calcul de l'algorithme statique, seules les petites instances peuvent être résolues.

Madsen et al. (1995) [**105**] ont présenté un algorithme pour un cas réel du DARP dynamique multi-véhicules qui concerne jusqu'à 300 demandes par jour pour le transport des personnes âgées et handicapées, à Copenhague. Le problème a beaucoup de contraintes telles que des fenêtres de temps, des contraintes de capacité multidimensionnelle, des priorités sur les clients et une flotte de véhicules hétérogènes. Beaucoup de critères de qualité, tenant compte de la satisfaction des clients et des coûts de service ont été considérés et mélangés par l'utilisation de paramètres de poids. Quand une nouvelle demande arrive, elle est insérée dans l'itinéraire courant en utilisant un algorithme efficace d'insertion basé sur celui de [**80**]. Les résultats sur des instances réels avec un maximum de 300 demandes et 24 véhicules ont prouvé que l'algorithme est assez rapide pour être utilisé dans un contexte dynamique et qu'il est capable de produire des solutions de bonne qualité.

Horn (2002) [**106**] a développé un logiciel pour des services de transports à la demande tels que les taxis et les autobus à tournées variables. Son problème inclut des contraintes de fenêtres de temps pour les demandes dynamiques, des contraintes de capacité et des annulations de réservation. L'insertion de nouvelles demandes est faite par un schéma d'insertion à coût minimal qui est complété par une procédure de recherche locale exécutée périodiquement. L'auteur a développé une heuristique pour déplacer stratégiquement les véhicules à vide en tenant compte des tendances des futures demandes. Des expérimentations ont été effectuées et basées sur des données d'une société de taxis, en Australie. Les résultats obtenus sur les instances testées, qui concernaient environ 4200 demandes et 220 taxis dans une période de 24 heures, ont prouvé que le logiciel proposé se comporte bien sur des instances dynamiques de grande taille.

Un algorithme parallèle pour le DARP dynamique a été développé par Attanasio et al. (2004) [**107**]. Au début de l'horizon de planification, une solution statique est construite à partir des demandes connues en utilisant un algorithme de recherche taboue. Quand une nouvelle

demande arrive, chacun des threads parallèles insère la demande de façon aléatoire dans la solution courante et exécute la recherche taboue pour l'obtention d'une solution réalisable. Si une solution faisable est trouvée, la demande est acceptée et une phase de post-optimisation est exécutée. Des résultats expérimentaux correspondant à différentes stratégies de parallélisassions sont décrits pour des exemples comprenant jusqu'à 144 demandes.

Un algorithme pour un problème de type DARP dynamique a été développé par Coslovich et al. (2006) [**108**]. Dans ce problème, un conducteur peut recevoir une demande de voyage d'une personne située à un arrêt et doit décider rapidement s'il l'accepte ou pas. L'algorithme maintient un stock de solutions réalisables qui sont compatibles avec les tournées partielles déjà effectués. Quand une nouvelle demande arrive, un algorithme efficace d'insertion s'efforce d'insérer cette demande dans au moins une des solutions. La demande acceptée seulement si l'algorithme d'insertion réussit. Une fois la nouvelle demande est acceptée, le stock des solutions faisables est mis à jour. Ce procédé est exécuté tandis que le véhicule se déplace entre deux endroits. Les temps de réponse ne sont pas pris en compte, ni les contraintes de capacité. Des résultats sur des instances artificielles comprenant jusqu'à 50 demandes sont décrits.

Beaudry et al. (2008) [**109**] ont développé un algorithme en deux phases pour résoudre un DARP dynamique complexe qui se présente dans le cas du transport des patients dans les hôpitaux. Ce problème comporte plusieurs contraintes portant sur la prise en compte de demandes avec différents degrés d'urgence et des besoins en équipement, des modes multiples de transport des patients, et des fenêtres de temps. De nouvelles demandes sont insérées en utilisant un schéma d'insertion rapide, capable de répondre aux exigences en temps réel. La deuxième phase consiste en une recherche taboue qui essaye d'améliorer la solution courante. L'algorithme a réussi à réduire les délais d'attente pour les patients tout en utilisant moins de véhicules sur les données réelles d'un hôpital allemand.

Xiang et al. (2008) [**110**] ont étudié un DARP dynamique sophistiqué dans lequel les temps de transport et de service ont un composant stochastique. En outre, il peut y avoir des non-présentations ou des annulations et les véhicules peuvent être bloqués dans des embouteillages ou des pannes. De nouvelles demandes sont insérées dans les tournées établies au moyen d'une procédure de recherche locale basée sur des mouvements simples entre tournées. La recherche locale a été complétée avec une stratégie de diversification qui utilise un objectif secondaire en mesurant la période des véhicules à vide. L'algorithme a été évalué par simulation sur plusieurs instances impliquant jusqu'à 610 demandes.

Cordeau et al. (2007) [**111**] ont présenté le problème « Dial-a-Flight » dans lequel l'objectif est de modéliser et d'optimiser les services d'accès à des vols sans réservation qui sont proposés par près de 3000 sociétés dans les États-Unis. Dans ce problème, les demandes se composent généralement de chargements multiples et de paires de déchargements sur lesquels

des règles strictes concernant le service sont imposées. Les auteurs discutent des caractéristiques du problème et présentent quelques approches de résolution.

CHAPITRE 2
HEURISTIQUES D'INSERTION POUR LE DARP

Dans ce chapitre, nous présentons des heuristiques d'insertion pour le DARP. Nous présentons aussi des procédures de réparation pour le cas tel qu'il existe des demandes ne peuvent pas être traitées par nos heuristiques.

1. Introduction

Les heuristiques déjà utilisées pour le DARP multi-véhicules peuvent être classées en trois catégories générales : le cluster-first route-second, l'amélioration locale et la procédure d'insertion.

Le « cluster-first route-second » est une technique utilisée dans le VRP (Vehicle Routing Problem) et a été essayé pour le DARP. Cette méthode décompose le problème de tournée en 2 sous-problèmes. Le premier est un problème de partitionnement : l'ensemble des clients est divisé en groupes de sorte que chaque groupe de clients sera traité par le même véhicule. Le deuxième consiste à confectionner les tournées. Sexton et Bodin [**78**,**79**] proposent une formulation non linéaire pour le DARP statique avec un seul véhicule. Ils utilisent la décomposition de Benders et traitent les 2 sous problèmes de manière indépendante. Bodin et Sexton (1986) [**83**] ont développé une heuristique basée sur le principe du cluster-first route-second et y intègre un mécanisme d'échange pour les problèmes multi-véhicules. Desrosiers et al. (1988) [**112**] proposent également une heuristique pour le DARP multi-véhicules basée sur le principe cluster-first route-second. Baugh et al. (1998) [**113**] résolvent ce problème en utilisant une méta-heuristique de type recuit simulé pour résoudre le problème de partitionnement et une heuristique basée sur la recherche des plus proches voisins pour le problème de confection des tournées.

Plusieurs méta-heuristiques ont déjà été testées pour le DARP. Cordeau et Laporte (2003) [**77**] utilisent une recherche taboue pour le DARP statique. Attanasio et al. (2004) [**107**] résolvent la version dynamique grâce à une technique de calcul parallèle. Toth et Vigo (1997) [**114**] ont développé également une méthode taboue. Hart (1996) [**115**] utilise un recuit simulé. On doit aussi mentionner le travail de Van Der Bruggen et al. (1993) [**72**] qui développent une méthode de recherche locale pour le problème de chargement et de déchargement avec un seul véhicule et des fenêtres de temps. Le travail proposé par Toth et Vigo (1996) [**116**] décrit les procédures qui peuvent être utilisées pour améliorer les solutions obtenues par une heuristique d'insertion.

Les méthodes d'insertion incluent le travail de Jaw et al. (1986) [**80**], Kikuchi et Rhee (1989) [**117**], Madsen et al. (1995) [**105**], Potvin et Rousseau (1992) [**118**], Toth et Vigo (1997) [**114**] et Diana et Dessouky (2004) [**119**]. L'idée fondamentale de la méthode d'insertion a été appliquée aux autres problèmes avec des objectifs spécifiques [**120**,**121**,**122**]. Les heuristiques d'insertion se sont révélées être des méthodes très pratiques pour résoudre une variété de problème de tournées de véhicules et d'ordonnancement car elles sont rapides, peuvent produire des solutions de bonne qualité, sont faciles à mettre en œuvre, et peuvent facilement être étendues pour traiter des contraintes additionnelles [**123**]. De plus, ces méthodes peuvent être facilement rendues non déterministes, ce qui permet de les intégrer dans des schémas de type « MONTE-CARLO ». Une étude comparative réalisée par Solomon (1987) [**124**] indique que la méthode d'insertion est une heuristique efficace pour le problème de tournées de véhicules avec fenêtres de temps (surtout pour les problèmes fortement contraints). Les méta-heuristiques telles que la recherche taboue sont très couteuses et leur performance est directement liée au temps de fonctionnement et à l'étalonnage des paramètres de l'algorithme. En outre, pour un problème fortement contraint, tel que le DARP, il est très difficile de maintenir la faisabilité de la solution après chaque mouvement de transformation locale. Il est difficile de converger vers une solution finale réalisable si on relâche des contraintes lors des transformations locales.

Dans ce chapitre, nous décrivons des heuristiques d'insertion. Elles ont l'avantage d'être rapides.

2. Notations générales sur les séquences et les algorithmes

Pour toute séquence (ou liste) Γ dont les éléments appartiennent à un ensemble Z d'éléments, nous définissons :

- $Debut(\Gamma)$: le premier élément de Γ;
- $Fin(\Gamma)$: le dernier élément de Γ;
- $Queue(\Gamma)$: la sous-séquence obtenue à partir de Γ par la suppression de $Debut(\Gamma)$;

- Pour tout élément z dans Γ :
  - $succ(\Gamma, z)$= le successeur de z dans Γ;
  - $prev(\Gamma, z)$= le prédécesseur de z dans Γ.
- Pour tout z, z' dans Γ :
  - $z \ll_\Gamma z'$ si z est situé avant z';
  - $z \ll_\Gamma^= z'$ si $z \ll_\Gamma z'$ où $z = z'$;
  - $Segment(\Gamma, z, z')$: la séquence définie par tout z" dans Γ tel que $z \ll_\Gamma^= z" \ll_\Gamma^= z'$. si $z = NULL$, $Segment(\Gamma, NULL, z')$ désigne alors la séquence définie par tout z" dans Γ tel que $z" \ll_\Gamma^= z'$.

Dans toute description algorithmique, nous utilisons le symbole « ← » pour désigner l'opérateur d'affectation de valeur : $x \leftarrow a$, signifie que la variable x reçoit la valeur a. Ainsi, nous utilisons uniquement le symbole « = » comme opérateur de comparaison.

3. Description du DARP

Le problème de Dial-a-Ride est essentiellement défini par :

- un réseau de transport $G = (X, A)$, où X est l'ensemble des sommets. Il contient au moins le sommet spécial « Dépôt ». A est l'ensemble des arcs, chaque arc (x, y) est associé à une distance.
- une flotte de véhicules, notée $VH = \{1, \dots, K\}$;
- un ensemble de demandes D. Soit I l'ensemble des indices des demandes. Chaque demande D_i, $i \in I$ est définie par le 6-uple $D_i = \{o_i, d_i, c_i, F(o_i), F(d_i), \Delta_i\}$ avec la signification suivante :
  - o_i est le sommet origine de la demande D_i dans le réseau de transport : le chargement de la demande en o_i doit être traité par un unique véhicule dans la fenêtre de temps $F(o_i) = [\alpha(o_i), \beta(o_i)]$;
  - d_i est le sommet destination de la demande D_i dans le réseau de transport : la demande D_i doit être déchargée en d_i dans la fenêtre de temps $F(d_i) = [\alpha(d_i), \beta(d_i)]$;
  - c_i est la charge de la demande D_i;
  - Δ_i est la durée maximale autorisée entre le chargement et le déchargement de la demande D_i (autrement dit, il s'agit de la durée de transport de la demande D_i);

Le but de ce problème est de planifier les demandes $D_i, i \in I$, à l'aide de la flotte de véhicules VH, en tenant compte des contraintes qui proviennent des caractéristiques du réseau G, et en optimisant un critère de performance qui se compose habituellement d'un coût économique et de critères de qualité de service.

Ce problème général peut être spécialisé de plusieurs façons :

Véhicule et charge

- la flotte de véhicules VH peut être hétérogène, ou homogène. Dans le premier cas, une partie du problème consiste à affecter les demandes aux différentes classes de véhicules;
- la charge c_i peut être décrite comme un ensemble d'objets dotés de leurs propres caractéristiques (poids, volume, mobilité autonome…), ou plus simplement, caractérisée par un nombre, qui identifie la charge comme le volume ou comme le poids des objets qui la composent. Dans le premier cas, une partie du problème consiste à identifier les combinaisons d'objets qui peuvent être simultanément transportés par un véhicule donné. Il faut alors prendre en compte les chargements et les déchargements ainsi que le type du véhicule impliqué.

Préemption

- la « préemption », c'est-à-dire le fait d'autoriser le traitement d'une demande par plusieurs véhicules, peut être interprétée de deux manières :
  - la charge c_i peut être fractionnée en morceaux traités par des véhicules différents (« préemption sur les charges ») ;
  - la charge c_i peut être acheminée en plusieurs étapes par des véhicules différents (« préemption sur les véhicules »).

Contraintes statiques, dynamiques et serrées

- les contraintes temporelles liées aux fenêtres de temps pour $F(o_i), F(d_i), i \in I$, et liées aux Δ_i, $i \in I$ peuvent être plus ou moins serrées;
- le problème peut être traité selon un contexte dynamique, certaines demandes ne sont pas connues à l'avance et doivent être traitées « à la volée » : dans ce cas, il faut tenir compte de la façon dont le système est supervisé et de la façon dont ses différentes composantes communiquent avec les utilisateurs. Le problème peut aussi être traité de manière statique : toutes les données sont connues à l'avance, la planification est calculée et gérée par le système. Dans ce cas, les divergences éventuelles entre les données qui ont été utilisées durant les phases de planification, et les données réelles du système doivent être prises en compte. Ce concept s'appelle la robustesse du système.

Contraintes supplémentaires

Des contraintes supplémentaires peuvent être abordées, telles que des contraintes sociales, techniques, financières ou encore la prise en compte de ressources renouvelables ou non renouvelables. Dans ce dernier cas, on peut lier ce problème avec le **RCPSP** (Resource Constrained Project Scheduling Problem).

Tout au long de notre travail, nous considérons une flotte de véhicules homogènes, et nous nous limitons au point de vue statique. Nous accordons une attention particulière au cas où les contraintes temporelles sont serrées, et nous traitons les différents exemples de préemption, ainsi que certaines contraintes additionnelles.

3.1. Le cas standard

Nous considérons ici qu'aucune préemption n'est autorisée, et qu'aucune contrainte supplémentaire ne doit être prise en compte. Dans ce cas, nous n'avons pas besoin de considérer le réseau entier de transport G, et nous pouvons nous limiter aux sommets qui sont soit l'origine soit la destination de la demande. Nous considérons que l'ensemble des sommets $\{Depot, o_i, d_i, i \in I\}$ est composé de sommets distincts. Pour tout couple (x, y) dans $\{Depot, o_i, d_i, i \in I\}$, la fonction $Dist$ fait correspondre la distance du plus court chemin de x à y dans le réseau de transport G.

Nous dédoublons également le sommet dépôt selon son statut (arrivée ou départ) et selon les véhicules considérés : $DepotD(k)$ est le dépôt de départ pour le véhicule k et $DepotA(k)$ est le dépôt d'arrivée pour le véhicule k. Les données d'entrée d'une instance du DARP standard sont définies par :

- l'ensemble des véhicules homogènes $VH = \{1, \dots, K\}$;
- la capacité CAP des véhicules;
- l'ensemble des sommets $X = \{o_i, d_i | i \in I\} \cup \{DepotD(k), DepotA(k) | \ k \in VH\}$;
- on dispose alors, entre les sommets de ce réseau, d'une matrice de distance $Dist$: pour tout $x, y \in X$, $Dist[x, y]$ présente le temps minimal pour aller de x vers y ; $Dist$ vérifie les inégalités triangulaires;
- à tout sommet x dans X, on fait correspondre :
    - son type $Type(x)$:
        - $Origine, Destination$ et $Depot = \{DepotD, DepotA\}$;
    - le numéro de demande Dem associée :
        - $Dem(x) = \begin{cases} i & \text{si } x = o_i \text{ ou } d_i \\ 0 & \text{sinon} \end{cases}$
    - le numéro de véhicule VI associé :
        - $VI(DepotD(k)) = VI(DepotA(k)) = k$, et $VI(x) = Indéfini$ pour autre sommet $x \in X$;
    - la charge $c(x)$ associée :
        - si $Type(x) \in Depot$ alors $c(x) = 0$;
        - si $Type(x) = Origine$ alors $c(x) = c_i$;
        - si $Type(x) = Destination$ alors $c(x) = -c_i$.
    - pour chaque sommet $x \in X$:
        - si $x = o_i (\ respectivement\ d_i)$ alors

on pose $Hom(x) = d_i(respectivement\ o_i)$;
- si $x = DepotD(k)$ $(respectivement\ DepotA(k))$ alors
  on pose $Hom(x) = DepotA(k)(respectivement\ DepotD(k))$;

- pour Δ_i :
  - si $x = o_i$ ou $x = d_i$ on pose $\Delta(x) = \Delta_i$; sinon on pose $\Delta(x) = \Delta$ où Δ est la durée maximale d'une tournée;

Selon cette construction, nous comprenons que le système fonctionne comme suit : le véhicule $k \in VH$ commence sa tournée à $DepotD(k)$ à un instant $t(DepotD(k))$ et se termine en $DepotA(k)$ à un instant $t(DepotA(k))$. Durant sa tournée, il transporte un sous-ensemble de demandes $D(k) = \{D_i, i \in I\}$: cela signifie que pour tout D_i, le véhicule k arrive à o_i à un instant $t(o_i)$, charge la quantité c_i, et la transporte jusqu'à d_i à un instant $t(d_i)$. La durée de transport de D_i ne doit pas dépasser Δ_i : $t(d_i) - t(o_i) \leq \Delta_i$. Clairement, la solution de l'instance du DARP standard liées aux données $(X, Dist, K, CAP)$ est donnée par les dates $t(x), x \in X$ et par les itinéraires suivis par les véhicules $k, k \in VH$.

Remarque 1

De nombreux auteurs [**77**] [**103**] [**125**] ajoutent une durée de service dans leurs modèles. Ils supposent que le chargement et le déchargement liés aux différents sommets $x \in X$ nécessitent une quantité de temps $s(x)$. Ils distinguent donc la date de début du service $t(x)$ et la date de fin du service $t(x) + s(x)$ pour le sommet $x, x \in X$. De la même manière, certains auteurs supposent que les véhicules fonctionnent toujours à leur vitesse maximale, et font une différence entre le date d'arrive $t'(x)$, au sommet $x, x \in X$, et la date de début du service $t(x)$ au sommet x. Notons que ce temps de service $s(x)$, qui tend à augmenter la taille des variables du modèle et à le rendre plus complexe, n'a vraiment de sens que si l'on suppose qu'il dépend de l'état actuel du véhicule (charge courante). De même, faire apparaître explicitement le temps d'attente $t(x) - t'(x)$ est vraiment utile si on souhaite inclure la vitesse des véhicules dans les critères de performance. La connaissance des itinéraires des véhicules, la valeur des dates $t(x), x \in X$, suffit à vérifier la validité d'une solution donnée et à évaluer sa performance. De plus, assurer la compatibilité du modèle avec des données qui concernent des temps de service et des temps d'attente, revient à adapter la fenêtre de temps $F(x)$, la durée du transport $\Delta(x), x \in X$, et la matrice des distances $Dist$.

3.1.1. Tournée

Une **tournée** servie par un véhicule $k \in VH$ est ici un circuit Γ dans le graphe orienté complet $G = (X, A)$ défini sur l'ensemble des sommets X, tel que :

- Γ débute en un sommet de type « $DepotD$ » et finit en un sommet de type «$DepotA$», ces deux sommets étant associés au même véhicule $k \in VH$:
  - $Type\big(Debut(\Gamma)\big) = DepotD, Type\big(Fin(\Gamma)\big) = DepotA$;

- $VI\big(Debut(\Gamma)\big) = VI(Fin(\Gamma))$;

- Pour tout sommet x dans Γ qui n'est pas de type «$DepotD$» ou «$DepotA$», on a la contrainte de précédence :
  - Si x est de type « $Origine$ » (x s'écrit o_i) alors le sommet de type «$Destination$» associé $\big(Hom(x)\big)$, est dans Γ, et il est situé après x dans Γ;
  - Si x est le type « $Destination$ » (x s'écrit d_i) alors le sommet de type «$Origine$» associé $\big(Hom(x)\big)$, est dans Γ, et il est situé avant x;
- un même sommet $x \in X$ apparaît au plus une fois dans Γ.

3.1.2. Tournée « Charge-Admissible »

Une tournée Γ donnée est dite « Charge-Admissible » si et seulement si pour tout y dans Γ, on a : $Charge(y) \leq CAP$ où la quantité $Charge(y)$ est définie par :

$$Charge(y) = \sum_{x \ll_{\Gamma}^{=} y} c(x)$$

3.1.3. Tournée « Temps-Admissible »

Une tournée Γ est dite « Temps-Admissible », si :

- Pour tout $x \in \Gamma$, $t(x) \in F(x)$;
- Pour tout $x \in \Gamma$, de type $Origine$ ou $DepotD$, on a : $t\big(Hom(x)\big) - t(x) \leq \Delta(x)$;
- Pour tout $x \in \Gamma$,$Type(x) \neq DepotA$, on a :
  $t\big(succ(\Gamma, x)\big) - t(x) \geq Dist\big(x, succ(\Gamma, x)\big)$.

3.1.4. Tournée « Admissible »

La tournée est dite « Valide » ou « Admissible » si et seulement si elle est « Temps-Admissible » et « Charge-Admissible ».

Pour mesurer la performance d'une tournée « admissible » Γ, on pose :

- $T_{Global}(\Gamma) = t\big(Fin(\Gamma)\big) - t(Debut(\Gamma))$;
  Cette quantité désigne la durée totale de la tournée Γ;
- $T_{Ride}(\Gamma) = \sum_{x \in \Gamma, Type(x) \in \{DepotD, Origine\}} t\big(Hom(x)\big) - t(x)$;
  Cette quantité peut être considérée comme un critère de qualité de service, et dénote la somme des durées de transport des demandes individuelles qui sont prises en charge par la tournée Γ;
- $T_{Wait}(\Gamma) = T_{Global}(\Gamma) - \sum_{x \in \Gamma, x \neq Fin(\Gamma)} Dist(x, succ(\Gamma, x))$;
  Cette quantité dénote le temps d'attente du véhicule impliqué dans la tournée Γ, le temps d'attente lié à un sommet x est le moment où le véhicule attend le chargement

ou le déchargement au sommet x, en supposant qu'il a roulé à vitesse maximale sur la route qui relie le sommet $prev(\Gamma, x)$ au sommet x;

Soient $\gamma, \delta, \varepsilon$, trois coefficients. On pose :

$$Cout_{\gamma,\delta,\varepsilon}(\Gamma) = \gamma \times T_{Global}(\Gamma) + \delta \times T_{Ride}(\Gamma) + \varepsilon \times T_{Wait}(\Gamma)$$

Nous supposons que nous avons déduit à partir des données $G = (X, A)$, $VH = \{1, \dots, K\}$ $D = (o_i, d_i, c_i, F(o_i), F(d_i), \Delta_i, i \in I)$, un 4-uplet $(X, Dist, K, CAP)$, et que nous avons également proposé 3 coefficients multicritères γ, δ et $\varepsilon \geq 0$. Nous résolvons une instance du DARP standard en calculant :

- un ensemble de tournées admissibles : $T(1), \dots, T(K)$;
- un vecteur de dates : $t = \{t(x), x \in X\}$.

de telle sorte que :

- chaque tournée $T(k)$, $k \in \{1, \dots, K\}$ est une tournée à la fois « Charge-Admissible » et « Temps-Admissible »;
- les tournées $T(k), k \in \{1, \dots, K\}$ constituent une partition de X;
- le coût global $Perf_{\gamma,\delta,\varepsilon}(T) = \sum_{k=1,\dots,K} Cout_{\gamma,\delta,\varepsilon}(T(k))$ est le plus petit possible.

Remarque 2

$Cout_{\gamma,\delta,\varepsilon}(\Gamma)$ peut aussi s'écrire : $Cout_{\gamma,\delta,\varepsilon}(\Gamma) = \sum_{x \in \Gamma} \theta_x \cdot t(x)$ tel que :

- si $x = Debut(\Gamma)$ alors $\theta_x = -\gamma - \delta - \varepsilon$;
- si $x = Fin(\Gamma)$ alors $\theta_x = \gamma + \delta + \varepsilon$;
- si $x \in \Gamma$ est de type $Origine$ alors $\theta_x = -\delta$;
- si $x \in \Gamma$ est de type $Destination$ alors $\theta_x = \delta$.

Remarque 3

Supposons que la durée du service $s(x)$ fait partie des données, et que pour tout $x \in X$, nous distinguons :

- la date d'arrivée $t'(x)$ du véhicule;
- la date $t(x)$, qui identifie la date de début de service (chargement ou déchargement) au sommet x;
- la date $t"(x) = t(x) + s(x)$, qui identifie le moment où le service (chargement ou déchargement) est fini en x.

Donc, si nous supposons, comme dans [**77**], que

- l'ensemble des fenêtres de temps $F(x)$ est lié à $t(x)$, $x \in X$ pour l'origine et la destination;
- la durée de transport de demande Δ_i est liée à la différence $t(d_i) - t"(o_i), i \in I$;
- la quantité $T_{Wait}(\Gamma)$ est définie comme la somme $\sum_{x \in \Gamma}(t(x) - t'(x))$;

- la matrice $Dist$ est telle que $Dist[x, succ(\Gamma, x)] = t'\big(succ(\Gamma, x)\big) - \big(t(x) + s(x)\big)$ (pour un véhicule roule à la vitesse maximale de x vers $succ(\Gamma, x)$).

Alors nous voyons que nous pouvons passer de ce modèle au modèle décrit précédemment, en posant :

- $\forall x, y \in X, Dist[x, y] = Dist[x, y] + s(x)$;
- $\forall i \in I, \Delta_i = t(d_i) - t(o_i) - s(d_i)$.

Remarque 4

On peut remarquer que si la durée de service doit être prise en compte, par exemple, deux sommets $x = o_i$ et $y = d_i$ sont liés au même sommet v dans le réseau de transport $G = (X, A)$, alors les durées de service $s(x)$ et $s(y)$ doivent être cumulées si le même véhicule k traite en même temps les deux demandes. Une telle chose va sembler surprenante dans la plupart des situations pratiques.

3.2. Evaluation de la tournée du DARP

Cordeau et Laporte [**77**] ont proposé une procédure efficace d'évaluation pour la tournée du DARP. Cette procédure peut retarder autant que possible les dates de début de service afin de réduire la durée de la tournée et les durées de transport. Cette procédure n'induit pas de violations temporelles. Ici, nous nous intéressons à cette procédure qui peut déterminer soit une tournée Γ « Charge-Admissible » soit une tournée « Temps-Admissible ». C'est-a-dire, nous utilisons cette procédure pour trouver une fonction date t, qui associe une date $t(x) \geq 0$ à tout sommet $x \in \Gamma$ telle que Γ est une tournée « Charge-Admissible ».

Soit une tournée « Charge-Admissible » Γ, on suppose d'abord que le véhicule quitte son dépôt de départ $DepotD(\Gamma)$ le plus tôt possible, la date de départ $t\big(DepotD(\Gamma)\big)$ est attribuée par la valeur de la borne inférieure $\alpha\big(DepotD(\Gamma)\big)$ de la fenêtre du temps $F(Depot(\Gamma))$. La date de début de service est activée le plus tôt possible à tout autre sommet $x \in \Gamma, x \neq DepotD(\Gamma)$ tel que $t(x) = max\{\alpha(x), t\big(prev(\Gamma, x)\big) + Dist[prev(\Gamma, x), x]\}$ pour satisfaire sa fenêtre de temps. Si $\exists x, t(x) \notin F(x)$ alors on peut conclure que la tournée Γ n'est pas « Temps-Admissible ». Si $\forall x \in \Gamma, t(x) \in F(x)$, et $\exists y \in \Gamma, t\big(Hom(x)\big) - t(x) \geq \Delta(x)$, alors on ne peut pas déterminer cette tournée est « Temps-Admissible » ou pas. Toutefois, nous avons constaté que si des dates $t(x)$ $x \in \Gamma$ peuvent être retardées encore, alors la tournée Γ peut devenir faisable.

Une simple méthode réduit la durée de la tournée en utilisant la règle : $t\big(Debut(\Gamma)\big) = max\{\alpha\big(Debut(\Gamma)\big), \alpha(x) - Dist[Debut(\Gamma), x]\}$ où x est le premier sommet visité dans la tournée Γ par le véhicule après avoir quitté son dépôt. Pour les autres sommets $x \in \Gamma$,

$t\big(succ(\Gamma,x)\big) = \max\{\alpha\big(succ(\Gamma,x)\big), t(x) + Dist[x, succ(\Gamma,x)]\}$(voir la Figure 18). Après la mise à jour des dates $t(x), x \in \Gamma$, on peut rencontrer des cas tels que :

1. Si $\forall x, t(x) \in F(x)$, $t\big(Hom(x)\big) - t(x) \leq \Delta(x)$ alors la tournée Γ est « Temps-Admissible » ;
2. Si $\exists x, t(x) \notin F(x)$, alors la tournée Γ n'est pas « Temps-admissible » ;
3. Sinon $\forall x, t(x) \in F(x)$ mais $\exists x$, $t\big(Hom(x)\big) - t(x) \geq \Delta(x)$, alors nous ne pouvons pas encore déterminer si la tournée Γ est « Temps-Admissible » ou pas.

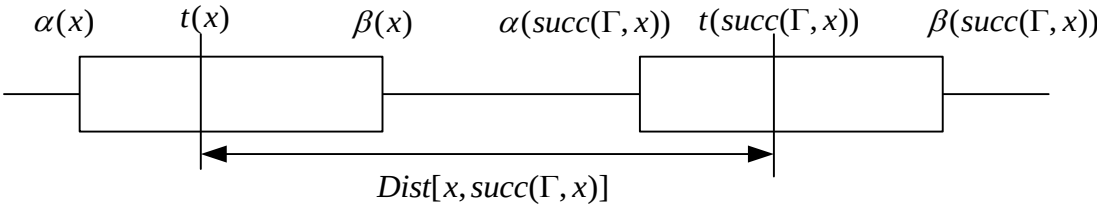

Figure 18 : Le début de service au sommet $t(succ(\Gamma, x))$

3.2.1. Forward Time Slack

Dans le premier et le troisième cas ci-dessous, certaines fenêtres de temps sont larges, si la date de début du service $t(x)$, $x \in \Gamma$ peut être encore retardée, la tournée peut être alors devenir faisable. Afin de retarder autant que possible la date $t(x)$ pour tout sommet x dans la tournée Γ, Cordeau et Laporte [**77**] utilisent la notion « **Forward Time Slack** » (FTS) introduite par Savelsbergh [**126**]. Pour un sommet x, on définit son « **Forward Time Slack** » $FTS(x)$ par :

$$FTS(x) = \min_{x \leq y < Fin(\Gamma)} \left\{ \sum_{x < p \leq y} W(p) + \big(\beta(y) - t(y)\big) \right\} \quad (1)$$

Où $W(p) = t(p) - t\big(prev(\Gamma,p)\big) - Dist[prev(\Gamma,p), p]$ est le temps d'attente à chaque sommet $p \in \{o_i, d_i | i \in I\}$.

Si $FTS(x)$ est calculé pour chaque sommet $x \neq Debut(\Gamma)$ et $x \in \Gamma$, la durée maximale de transport de la demande D_i doit être vérifiée. En effet lorsque la date de début de service $t(x)$ est retardée, la durée de transport pour la demande D_i telle que $o_i \ll_\Gamma x \ll_\Gamma^{=} d_i$ peut être augmentée. Afin d'éviter la violation de la durée de transport, on calcule FTS de la manière suivante :

$$FTS(x) = \min_{x \leq y < Fin(\Gamma)} \left\{ \sum_{x < p \leq y} W(p) + (min\{\beta(y) - t(x), \Delta(y) - r(y)\})^+ \right\} \quad (2)$$

$$\text{Où } r(y) = \begin{cases} t(y) - t(hom(y)), \text{si } \forall\, y \in \{d_i | i \in I\} \text{ et } y \in \Gamma; \\ 0\,, \text{sinon;} \end{cases}$$

Si la date de départ au dépôt est $t\big(Debut(\Gamma)\big) = \alpha\big(Debut(\Gamma)\big) + FTS\big(Debut(\Gamma)\big)$ au lieu de $t\big(Debut(\Gamma)\big) = \alpha\big(Debut(\Gamma)\big)$, la durée de la tournée est alors minimisée.

Cette procédure permet de déterminer la faisabilité d'une tournée. Elle peut aussi améliorer la qualité de service (minimisation des durées de transport, temps d'attentes …). Ici, nous somme intéressés par la tournée admissible, la procédure s'est arrêtée lorsqu'elle trouve des violations temporelles (l'étape 9 et 18).

Algorithme 1 : Procédure Evaluation_Cordeau_Laporte

Entrées : Γ : une tournée « Charge-Admissible » ;
Sorties : Res : booléen, (Γ, t) : une tournée admissible ;

1 $Res \leftarrow true$; Mettre $t\big(Debut(\Gamma)\big) \leftarrow \alpha(Debut(\Gamma))$;
2 Calcule $t(x)$ pour le sommet $x \in \Gamma$;
3 **Si** $\exists x, t(x) \notin F(x)$ **alors** return $Res \leftarrow false$;
4 **Sinon**
5 | Calcule $FTS(Debut(\Gamma))$;
6 | Mettre $t\big(Debut(\Gamma)\big) \leftarrow \alpha\big(Debut(\Gamma)\big) + min\big\{FTS\big(Debut(\Gamma)\big), \sum_{Debut(\Gamma)<x<Fin(\Gamma)} W(x)\big\}$;
7 | Mettre à jour $W(x), t(x)$ pour le sommet $x \in \Gamma$;
8 | **Si** $\exists x, t(x) \notin F(x)$ **alors**
9 | | $Res \leftarrow false$;
10 | **Sinon**
11 | | Calculer $t(Hom(x)) - t(x), x \in \Gamma \; et \; x \in \{o_i | i \in I\}$;
12 | | **Pour** le sommet $x \in \Gamma, x \in \{o_i | i \in I\}$ et $Res = true$ **faire**
13 | | | Calculer $FTS(x)$;
14 | | | Mettre $t(x) \leftarrow t(x) + min\big\{FTS(x), \sum_{x<y<Fin(\Gamma)} W(y)\big\}$;
15 | | | Mettre à jour $W(y), t(y)$ pour le sommet y est situé après le sommet x ;
16 | | | Calcule $t(y) - t(Hom(y))$ pour le sommet $y \in \Gamma, y \in \{d_i | i \in I\} \; et \; x \ll_\Gamma y$;
17 | | | **Si** $\exists x \in \Gamma \; et \; x \in \{o_i | i \in I\}, t(Hom(x)) - t(x) > \Delta(x)$ **alors**
18 | | | | $Res \leftarrow false$;
19 | | | **Fin**
20 | | **Fin**
21 | **Fin**
22 **Fin**

Nous utilisons la procédure « Evaluation_Cordeau_Laporte » (voir l'Algorithme 1) pour déterminer si la tournée « Charge-Admissible » Γ est « Temps-Admissible » ou pas. Si la tournée Γ admissible est déterminée par la procédure, les trois critères de la tournée : $T_{Global}(\Gamma)$, $T_{Ride}(\Gamma)$ et $T_{Wait}(\Gamma)$ sont alors obtenues en même temps.

4. Prétraitement

Cette section décrit la réduction des fenêtres de temps et l'élimination d'arcs du graphe qui peuvent être appliqués avant l'algorithme. Ces prétraitements permettent d'accélérer les calculs effectués par les heuristiques.

4.1. La réduction des fenêtres de temps

Ce prétraitement réduit au maximum les fenêtres de temps avant l'exécution de l'algorithme d'insertion. Afin de réduire les fenêtres de temps, on utilise les règles suivantes :

- Les fenêtres de temps liées au dépôt $DépotD$ et au dépôt $DépotA$ peuvent être réduites par :
  - $\alpha(DépotD) = min_{\forall i \in I}\{\alpha(o_i) - Dist[DepotD, o_i]\}$;
  - $\beta(DépotA) = max_{\forall i \in I}\{\beta(d_i) + Dist[d_i, DepotA]\}$.
- La fenêtre de temps $F(o_i)$ liée au sommet origine o_i, $i \in I$ peut être réduite par :
  - $\alpha(o_i) = max\{\alpha(o_i), Dist[DepotD, o_i]\}$;
  - $\alpha(o_i) = max\{\alpha(o_i), \alpha(d_i) - \Delta_i\}$;
  - $\beta(o_i) = min\{\beta(d_i), \beta(d_i) - Dist[o_i, d_i]\}$.
- La fenêtre de temps $F(d_i)$ liée au sommet destinataire $d_i, i \in I$ peut être réduite par :
  - $\alpha(d_i) = max\{\alpha(d_i), Dist[DepotD, o_i] + Dist[o_i, d_i]\}$;
  - $\alpha(d_i) = max\{\alpha(d_i), \alpha(o_i) + Dist[o_i, d_i]\}$;
  - $\beta(d_i) = min\{\beta(d_i), \beta(o_i) + \Delta_i\}$.

4.2. L'élimination d'arcs

Soient deux demandes D_i et D_j, $i, j \in I, i \neq j$. Dans le cadre du prétraitement, nous pouvons vérifier la validité des tournées suivantes :

$$\begin{cases} DepotD(1), o_i, d_i, o_j, d_j, DepotA(1) & : \sigma_1 \\ DepotD(1), o_i, o_j, d_i, d_j, DepotA(1) & : \sigma_2 \\ DepotD(1), o_i, o_j, d_j, d_i, DepotA(1) & : \sigma_3 \\ DepotD(1), o_j, o_i, d_j, d_i, DepotA(1) & : \sigma_4 \\ DepotD(1), o_j, o_i, d_i, d_j, DepotA(1) & : \sigma_5 \\ DepotD(1), o_j, d_j, o_i, d_i, DepotA(1) & : \sigma_6 \end{cases}$$

Dans le cas où le processus de vérification conclut qu'aucune des tournées ci-dessus n'est admissible, on dit que D_i et D_j ne sont pas compatibles, et nous notons $non\_compatible(i, j)$. En raison des contraintes des fenêtres de temps et des durées maximum de transport, plusieurs arcs du graphe G peuvent en effet être retirés car ils ne peuvent appartenir à une solution réalisable. Une analyse simple conduit aux observations suivantes :

- $\forall x \in \{o_i | i \in I\}, \forall k \in \{1, \dots, K\}$, les arcs $\big(DépotD(k), Hom(x)\big)$, $\big(x, DépotA(k)\big)$ et $(Hom(x), x)$ sont infaisables;

- $\forall x, y \in X$, les arcs (x, y) sont infaisables si $\alpha(x) + Dist[x, y] > \beta(y)$;

- $\forall x \in \{o_i | i \in I\}, y \in X$, les deux arcs (x, y) et $\big(y, Hom(x)\big)$ sont infaisables si $Dist[x, y] + Dist[y, Hom(x)] > \Delta(x)$.

Les autres règles de l'élimination d'arc ont été proposées par Dumas et al. (1991) [**58**], combinant des fenêtres de temps et des contrainte d'appariement. On obtient des règles d'élimination encore plus forte :

- $\forall x, y \in \{o_i | i \in I\}$, si le chemin $C = \{y, x, Hom(y), Hom(x)\}$ est infaisable, alors l'arc $(x, Hom(y))$ est infaisable;

- $\forall x, y \in \{o_i | i \in I\}$, si le chemin $C = \{x, Hom(x), y, Hom(y)\}$ est infaisable, alors l'arc $(Hom(x), y)$ est infaisable;

- $\forall x, y \in \{o_i | i \in I\}$, si deux chemins $C1 = \{x, y, Hom(x), Hom(y)\}$ et $C2 = \{x, y, Hom(y), Hom(x)\}$ sont infaisables, alors l'arc (x, y) est infaisable ;

- $\forall x, y \in \{o_i | i \in I\}$, si deux chemins $C1 = \{x, y, Hom(x), Hom(y)\}$ et $C2 = \{y, x, Hom(x), Hom(y)\}$ sont infaisables, alors l'arc $(Hom(x), Hom(y))$ est infaisable.

Dans le cas où l'arc (x, y) ci-dessus n'est pas infaisable, on dit que les sommets x et y ne sont pas compatibles, c'est-à-dire, dans la tournée Γ, le sommet y ne se situe pas après le sommet x, et nous notons $conflit\_sommet(x, y)$.

5. La boucle principale

Dans la boucle principale, l'opération d'insertion est la composante la plus importante. Nous essayons deux stratégies différentes d'insertion. Ces deux stratégies sont conformes au schéma qui est présenté par la Figure 19 ci-dessous. Nous essayons de joindre une fonction de réparation et une fonction de réparation forte pour traiter le cas où une demande ne peut plus être insérée dans l'ensemble des tournées. Le but est de minimiser le coût total des tournées.

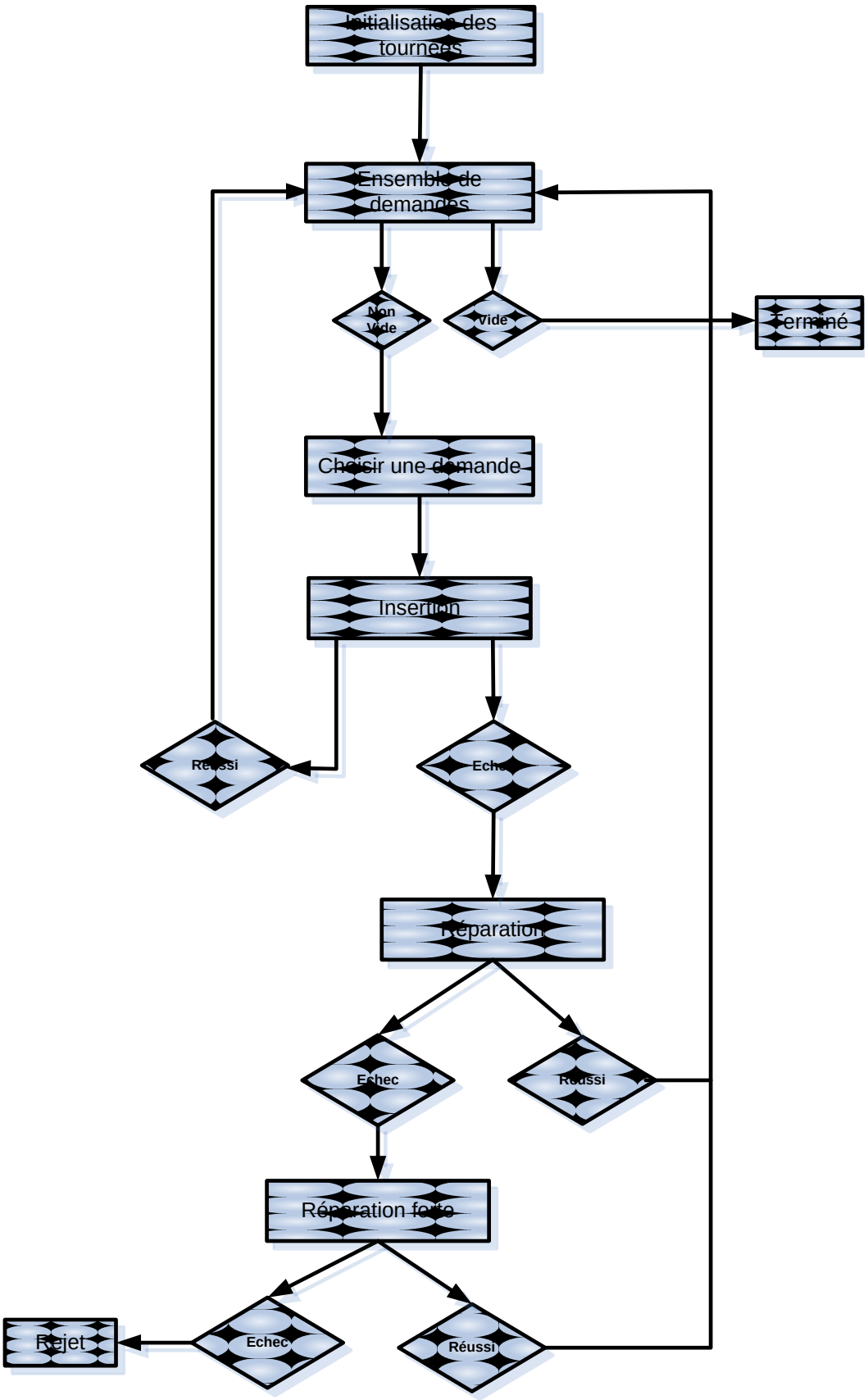

Figure 19 : Schéma d'insertion générale

5.1. Le mécanisme d'insertion

Etant donnés une tournée admissible Γ, une demande $D_i = \{o_i, d_i, c_i, F_{o_i}, F_{d_i}, \Delta_i\}$ et deux sommets $x, y \in \Gamma$ et $x \ll_{\Gamma}^{=} y$, l'opération $Inserer(\Gamma, x, y, D_i)$ insère la demande D_i dans la tournée Γ selon les sommets (positions d'insertion) x et y. Cet opération construit une nouvelle tournée en plaçant o_i entre x et $succ(\Gamma, x)$ et d_i entre y et $succ(\Gamma, y)$ (x et y peuvent être identiques). L'application de cette opération donne donc un résultat constitué de :

- un booléen Res, qui indique si la tournée est « admissible » ou pas ;
- une valeur VAL de coût de la nouvelle tournée admissible ;
- la nouvelle tournée admissible ΓR ainsi construite.

La nouvelle tournée est évaluée en utilisant la procédure d'évaluation proposée par Cordeau et Laporte [**77**] décrite par l'Algorithme 1 dans la section précédente.

5.2. Processus de recherche d'insertion

À un moment donné en cours de processus, on considère une demande D_{i_0} à insérer. On doit en principe regarder toutes les tournées $T(k)$, $k \in \{1, \dots, K\}$. Pour chacune des tournées, tous les couples de sommet (x, y) et x, $y \in T(k)$ sont testés par l'application d'opérateur $Inserer(T(k), x, y, D_{i_0})$. Dans tous les cas, on souhaite retenir l'insertion meilleure possible, c'est-à-dire celle qui correspond à la plus petite valeur VAL. Si la tournée contient q sommets (avec le dépôt de départ et le dépôt d'arrivée), on a $q(q-1)/2$ positions différentes d'insertion pour la demande D_{i_0}. Afin de diminuer le nombre des tests d'insertion et d'accélérer ce processus, on utilise $conflit\_sommet$ et $non\_compatible$ qui permettent :

- d'éviter d'une insertion invalide de la demande D_{i_0} dans une tournée $T(k)$ qui contient une demande D_j telle que $non\_compatible(i_0, j)$;
- d'éviter d'une opération $Inserer(T(k), x, y, D_{i_0})$ invalide.

5.2.1. L'intervalle d'insertion

Afin de diminuer le nombre d'opérations invalides, nous essayons de trouver deux intervalles d'insertion dans la tournée $T(k), k = \{1, \dots, K\}$ pour le sommet origine et le sommet destination de la demande D_{i_0} :

- **la borne inférieure** est donnée par le dépôt de départ $Debut(T(k))$ de la tournée $T(k)$.
- **la borne supérieure** est donnée par le premier sommet x dans la tournée $T(k)$ où il existe un conflit temporel avec le sommet y de la demande D_{i_0} tel que : $\alpha(x) +$

$Dist[x, \mathrm{y}] > \beta(\mathrm{y})$. Soit $z1$ ($z2$) la borne supérieure pour le sommet o_{i_0} (d_{i_0}), les bornes supérieures sont définies comme suit :

- Si $z2 \ll_{T(k)} z1$, alors les sommet o_{i_0} et d_{i_0} sont insérés uniquement après le sommet x, $x \in \left[Debut\left(T(k)\right), prev(T(k), z2)\right]$ (contrainte de précédence) ;

- Si $z1 \ll^{=}_{T(k)} z2$, alors le sommet o_{i_0} est inséré seulement après le sommet $x, x \in \left[Debut\left(T(k)\right), prev(T(k), z1)\right]$ et le sommet d_{i_0} est inséré seulement après le sommet $x, x \in \left[Debut\left(T(k)\right), prev(T(k), z2)\right]$;

- Si le sommet $z1$ (ou $z2$) n'est pas trouvé dans la tournée, alors le sommet o_{i_0} (ou d_{i_0}) peut être inséré à n'importe quelle position dans la tournée $T(k)$ et $z1 = z2 = prev(T(k), Fin(T(k)))$.

La Figure 20 illustre la borne supérieure pour le sommet x. Dans la tournée, o_3 est le premier sommet où il existe un conflit temporel avec x : $\alpha(o_3) + Dist[o_3, x] = 14 + 5 = 19 > 18 = \beta(x)$. L'intervalle d'insertion pour le sommet x est $[0, d_2]$.

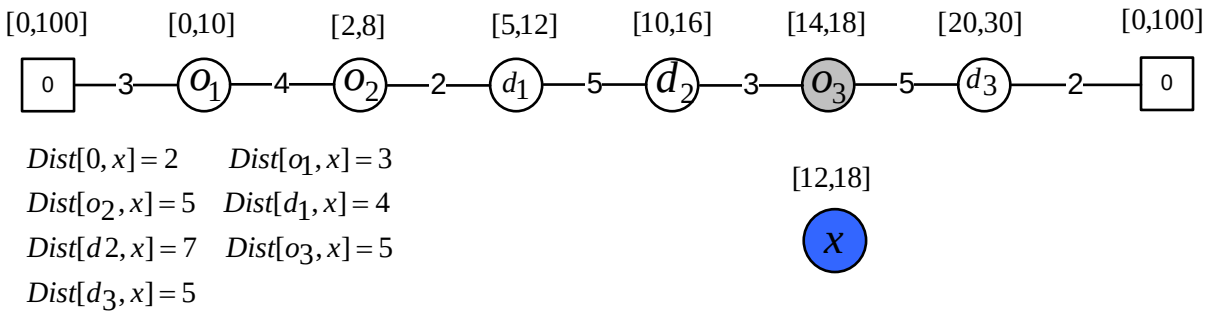

Figure 20 : La borne supérieure d'insertion pour un sommet x

Lorsque deux intervalles $\left[Debut\left(T(k)\right), z1\right]$ et $\left[Debut\left(T(k)\right), z2\right]$ sont trouvés pour la demande D_{i_0}, le sommet origine o_{i_0} peut être inséré après un sommet $x \in \left[Debut\left(T(k)\right), z1\right]$ tel qu'il n'existe pas $conflit\_sommet(x, o_{i_0})$ ni $conflit\_sommet\left(o_{i_0}, succ(T(k), x)\right)$. Le sommet destination d_{i_0} peut être inséré après un sommet $x \in \left[Debut\left(T(k)\right), z2\right]$ tel qu'il n'existe pas $conflit\_sommet(x, d_{i_0})$ ni $conflit\_sommet\left(d_{i_0}, succ(T(k), x)\right)$.

Pendant les tests d'insertion, la borne inférieure de l'intervalle d'insertion pour le sommet destination dépend de la borne inférieure du sommet origine : le sommet destination doit être situé après le sommet origine correspondant.

La Figure 21 montre l'intervalle d'insertion pour la demande D_i dans la tournée $\Gamma = \{0, o_1, o_2, d_1, o_3, o_4, o_5, \dots, 0\}$. Supposons que $\alpha(o_4) + Dist[o_4, o_i] > \beta(o_i)$ et $\alpha(o_3) + Dist[o_3, d_i] > \beta(d_i)$, la borne supérieure d'intervalle d'insertion pour le sommet origine o_i est o_4 et la borne supérieure pour le sommet d_i est o_3. En raison de la contrainte de précédence, la borne supérieure du sommet o_i est modifiée par o_3. Nous supposons qu'il existe trois conflits temporels tels que $conflit\_sommet(o_1, o_i)$, $conflit\_sommet(o_2, o_i)$ et $conflit\_sommet(o_2, d_i)$. Pour le sommet origine o_i, il reste encore 2 positions possibles (la position entre 0 et o_1 et la position entre d_1 et o_3). Si o_i est inséré entre 0 et o_1, il y a 3 positions possibles dans la tournée $\{0, o_i, o_1, o_2, d_1, o_3, o_4, o_5, \dots, 0\}$ pour d_i. Si o_i est inséré entre d_1 et o_3, alors il n'y a qu'une seule position (entre o_i et o_3) dans la tournée $\{0, o_1, o_2, d_1, o_i, o_3, o_4, o_5, \dots, 0\}$ pour d_i.

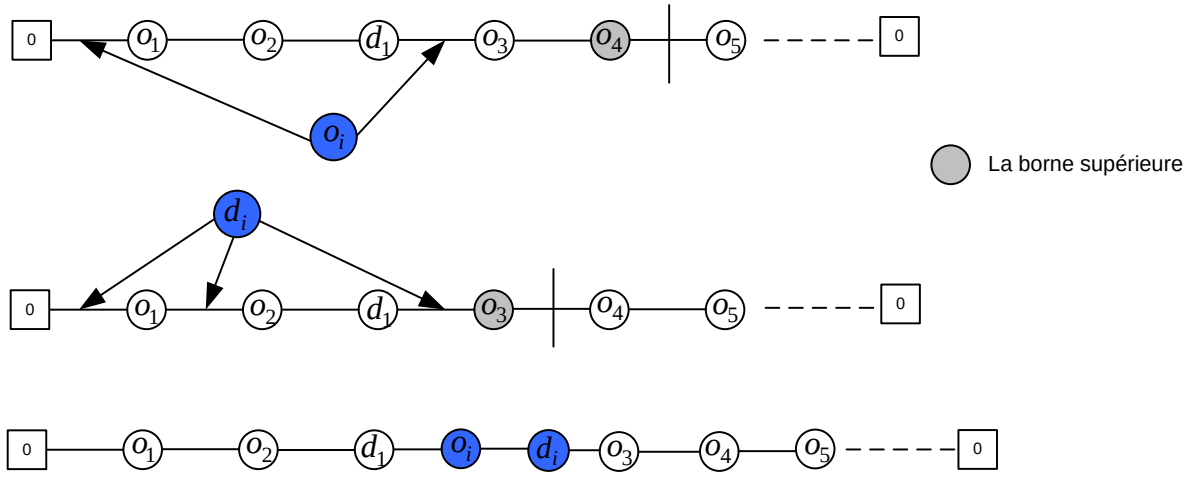

Figure 21 : L'intervalle de position insérée

5.3. Stratégie de recherche

Nous cherchons à priori la meilleure insertion possible pour la demande D_{i_0}. Le temps de calcul nécessaire pour réaliser une recherche exhaustive étant long, on envisage de ne faire qu'une recherche partielle. Nous testons donc deux stratégies :

- **Stratégie exhaustive** (voir l'Algorithme 2) : la demande D_{i_0} étant donnée, on teste toutes les insertions $Inserer(T(k), x, y, D_{i_0})$ compatibles avec $non\_compatible$ et $conflit\_sommet$;

- **Stratégie partielle** (voir l'Algorithme 3) : on trie les valeurs k selon l'ordre croissant des valeurs $T_{Global}(T(k)), k \in \{1, \dots, K\}$, et on arrête la recherche dès qu'une insertion $Inserer(T(k), x, y, D_i)$ a été trouvée et que $Inserer \cdot Res = 1$. (c'est-à-dire la tournée résultante est « Charge-Admissible » et « Temps-Admissible »)

Algorithme 2 : Insertion_Exhaustive (Stratégie1)

Entrées : D : l'ensemble des demandes ;
Sorties : T : l'ensemble des tournées ;
$Reject$: l'ensemble des demandes rejetées ;

$Reject \leftarrow Null$;
Tant que $D \neq NULL$ **faire**
| Choisir D_{i_0} dans D ; $D \leftarrow D - \{D_{i_0}\}$;
| Enumérer toutes les insertions possibles : $Inserer\left(T(k), x, y, D_{i_0}\right)$ qui soit telles que $Inserer \cdot Res = 1$;
| **Si** Echec_Enumérer **alors**
| | $Reject \leftarrow Reject + \{D_{i_0}\}$;
| **Sinon**
| | Choisir k_o, x_0, y_0 qui correspond à la meilleure valeur $Inserer \cdot VAL$;
| | $T(k_0) \leftarrow Inserer(T(k_o), x_0, y_0, D_{i_0})$;
| **Fin**
Fin

Algorithme 3 : Insertion_Partielle (Stratégie2)

Entrées : D : l'ensemble des demandes ;
Sorties : T : l'ensemble des tournées ;
$Reject$: l'ensemble des demandes rejetées.

$Reject \leftarrow Null$;
Tant que $D \neq NULL$ **faire**
| Choisir i dans D ; $D \leftarrow D - \{D_{i_0}\}$;
| Trier les valeurs k selon l'ordre croissant des valeurs $T_{Global}\left(T(k)\right),\ k = 1, \dots, K$;
| $stop \leftarrow false$;
| **Tant que** $(k \leq K)$ && $(stop = false)$ **faire**
| | Enumérer tous les couples $x, y \in T(k)$ tels que $Inserer\left(T(k), x, y, D_{i_0}\right) \cdot Res = 1$;
| | **Si** la liste résultat de cette énumération est non vide **alors**
| | | $stop \leftarrow true$;
| | | Récupérer x_0, y_0 correspondant à la meilleure valeur T_{Global};
| | **Fin**
| **Fin**
| **Si** $stop = true$ **alors**
| | $T(k_0) \leftarrow Inserer(T(k_o), x_0, y_0, D_{i_0})$;
| **Sinon**
| | $Reject \leftarrow Reject + \{D_{i_0}\}$;
| **Fin**
Fin

Algorithme 4 : Réparer

Entrée : $\boldsymbol{\Gamma_1, \Gamma_2}$: tournées ;
$\boldsymbol{D_{i_0}}$: la demande ;
Sorties : $\boldsymbol{Res}$: booléen ;

Construire la liste L_C en utilisant Γ_1 et D_{i_0};
$Res \leftarrow false$;
Tant que $Res = false$ && $L\_C \neq NULL$ **faire**
    $D_i \leftarrow Debut(L\_C)$;
    $L\_C \leftarrow Queue(L\_C)$;
    $\Gamma \leftarrow \Gamma_1$;
    Retirer D_i dans la tournée Γ ;
    Enumérer tous les couples $x, y \in \Gamma_2$ tels que $Inserer(\Gamma_2, x, y, D_i) \cdot Res = 1$;
    Si la liste résultat de cette énumération est non vide **alors**
        Recupérer x_2, y_2 correspondant à la meilleure valeur ;
        Enumérer tous les couples $x, y \in \Gamma$ tels que $Inserer(\Gamma, x, y, D_{i_0}) \cdot Res = 1$;
        Si la liste résultat de cette énumération est non vide **alors**
            Recupérer x_1, y_1 correspondant à la meilleure valeur ;
            $\Gamma_1 \leftarrow Inserer(\Gamma, x_1, y_1, D_{i_0})$; $\Gamma_2 \leftarrow Inserer(\Gamma_2, x_2, y_2, D_i)$;
            $Res \leftarrow true$
        Sinon
            $\Gamma_1 \leftarrow \Gamma$; $\Gamma_2 \leftarrow Inserer(\Gamma_2, x_2, y_2, D_i)$;
        Fin
    Fin
Fin

5.4. Processus de réparation

Les expérimentations des algorithmes d'insertion exhaustive (Algorithme 2) et d'insertion partielle (Algorithme 3) font apparaître de façon fréquente des demandes rejetées ($Reject \neq Null$). Afin de diminuer ce taux d'échec, on introduit une procédure de réparation que l'on applique si l'insertion d'une demande D_{i_0} échoue. Cette procédure de réparation utilise un triplet $(\Gamma_1, \Gamma_2, D_{i_0})$. Elle fonctionne selon les étapes suivantes :

1 Construire une liste L_C de demandes en utilisant la tournée Γ_1. La demande D_i dans la liste L_C est sélectionnée selon :
   - Soit $non\_compatible(i, i_0)$, $D_i \in \Gamma_1$;
   - Soit $conflit\_sommet(x, y)$ ou $conflit\_sommet(y, x)$, $\forall x \in \{o_i, d_i | D_i \in \Gamma_1\}$ et $y \in \{o_{i_0}, d_{i_0}\}$.

2 Sélectionner la demande $D_i \in L\_C$ dans l'ordre de la liste ;

3 Retirer la demande D_i dans Γ_1 ;
4 Essayer d'insérer D_i dans Γ_2 ;
5 Si l'insertion de la demande D_i réussit, on essaye d'insérer D_{i_0} dans Γ_1. Sinon retourner à l'étape 2.
6 Si l'insertion de la demande D_{i_0} échoue, accepter l'insertion de la demande D_i et retourner à l'étape 2. Sinon la procédure s'arrête.

Une application de « Réparer » peut échouer. L'algorithme « Insertion_Exhaustive » peut être alors modifié comme décrit par l'Algorithme 5.

Algorithme 5 : Insertion_Exhaustive_R

Entrées : D : l'ensemble des demandes ;
Sorties : T : l'ensemble des tournées ;
$Reject$: l'ensemble des demandes rejetées.

1. $Reject \leftarrow NULL$
2. **Tant que** $D \neq NULL$ **faire**
3. | Choisir D_{i_0} dans D ;
4. | $D \leftarrow D - \{D_{i_0}\}$
5. | Enumérer toutes les insertions possibles $Inserer\left(T(k), x, y, D_{i_0}\right)$ $k \in VH$ qui soit telles que $INSERT \cdot Res = 1$;
6. | **Si** Echec_Enumérer **alors**
7. | | $succe = false$;
8. | | $k \leftarrow 1$;
9. | | $k' \leftarrow 1$;
10. | | **Tant que** $k \leq K$ && $succe = false$ **faire**
11. | | | **Tant que** $k' \leq K$ && $k \neq k'$ && $succe = false$ **faire**
12. | | | | $succe \leftarrow Reparer(k, k', i_0)$;
13. | | | **Fin**
14. | | **Fin**
15. | | **Si** $succe = false$ **alors**
16. | | | $Rejet \leftarrow Reject + \{i_0\}$;
17. | | **Fin**
18. | **Sinon**
12 | | Choisir k_o, x_0, y_0 qui correspond à la meilleure valeur $Inserer \cdot VAL$;
13 | | $T(k_0) \leftarrow Inserer(T(k_o), x_0, y_0, D_{i_0})$;
19. | **Fin**
20. Fin

Clairement, on peut modifier l'algorithme « Insertion_Partielle » de la même façon (Algorithme 6).

Algorithme 6 : Insertion_Partielle_R

Entrées : D : l'ensemble des demandes ;
Sorties : T : l'ensemble des tournées ;
$Reject$: l'ensemble des demandes rejetées.

$Rejet \leftarrow Null$;
Tant que $D \neq NULL$ **faire**
  Choisir D_{i_0} dans D ;
  $D \leftarrow D - \{D_{i_0}\}$;
  Pour k **de** 1 **à** K **faire**
    $marque(k) = 0$;
  Fin
  $succe \leftarrow false$;
  Tant que $(k \leq K)\&\&(succe \leftarrow false)$ **faire**
    Choisir k_0 tel que $marque(k_0) = 0$ Et $T_{Global}(T(k_0))$ est minimale ;
    $marque(k_0) = 1$;
    Enumérer tous les couples $x, y \in T(k_0)$ tels que $Inserer\big(T(k_0), x, y, D_{i_0}\big) \cdot Res = 1$;
    Si la liste résultat de cette énumération est non vide **alors**
      $succe \leftarrow true$;
      $Recupérer\ x_0, y_0$ correspondant à la meilleure valeur $Inserer \cdot VAL$;
      $Inserer(T(k_o), x_0, y_0, D_{i_0})$;
    Sinon
      $k \leftarrow 1$; $k' \leftarrow 1$;
      Tant que $k \leq K$ && $succe = false$ **faire**
        Tant que $k' \leq K$ && $k \neq k'$ && $succe = false$ **faire**
          $succe \leftarrow Reparer(k, k', i_0,)$;
        Fin
      Fin
      Si $succe = false$ **alors**
        $Rejet \leftarrow Reject + \{i_0\}$;
      Fin
    Fin
  Fin
Fin

5.5. Second Processus de Réparation

Afin de renforcer le procédé précédent, on met en œuvre un deuxième procédé. Lorsque l'insertion et la réparation d'une demande D_{i_0} a échoué, on considère tous les couples des tournées, on tente d'échanger aléatoirement deux demandes entre les tournées, puis on tente l'insertion de D_{i_0} . Cette procédure utilise un opérateur $Echang(\Gamma_1, \Gamma_2, D_{i_1}, D_{i_2})$ (Algorithme 7)

Algorithme 7 : Echange

Entrée : Γ_1, Γ_2 : tournées ; D_{i_1}, D_{i_2} : demandes
Sortie : res : booléen ; $x1, y1, x2, y2$: les sommets

Retirer D_{i_1} de Γ_1 ; Retirer D_{i_2} de Γ_2 ;
$res \leftarrow false$;
Chercher $(x1, y1)$ dans Γ_1, $(x2, y2)$ dans Γ_2 tels que :
    $res \leftarrow true$;
    $Inserer(\Gamma_1, x1, y1, D_{i_2}) \cdot Res = 1$;
    $Inserer(\Gamma_2, x2, y2, D_{i_1}) \cdot Res = 1$;
Si cherche Echec **alors**
| $Echange \leftarrow \{Res, *\}$;
Sinon
| $Echange \leftarrow \{Res, x1, y1, x2, y2\}$;
Fin

La procédure $Reparer\_Fort$ s'écrit alors :

Algorithme 8 : Reparer_Fort

Entrée : $T(k), k \in [1, \dots, K]$: l'ensemble des tournées ; D_{i_0} : la demande ;
Sortie : res : booléen

1. **Pour** $k1, k2 = 1, \dots, K$ et $k1 < k2$ **faire**
2. | Choisi au hasard D_{i_1} dans $T(k1)$;
3. | Choisi au hasard D_{i_2} dans $T(k2)$;
4. | $(R, x1, y1, x2, y2) \leftarrow Echange(k1, k2, D_{i_1}, D_{i_2})$;
5. | **Si** $R = true$ **alors**
6. | | Retire D_{i_1} de $T(k1)$; Retire D_{i_2} de $T(k2)$;
7. | **Fin**
8. | $T(k1) \leftarrow Inserer(T(k1), x1, y1, D_{i_2})$;
9. | $T(k2) \leftarrow Inserer(T(k2), x2, y2, D_{i_1})$;
10. **Fin**
11. $res \leftarrow false$;
12. **Pour** $k = 1, \dots, K$ && $res = false$ **faire**
13. | Enumérer toutes les insertions possibles $Inserer(T(k), x, y, D_{i_0})$ qui soit telles que $INSERT \cdot Res = 1$;
14. | **Si** réussi_Enumérer **alors**
14 | | $res \leftarrow true$;
15 | | Choisir k_o, x_0, y_0 qui correspond à la meilleure valeur $Inserer \cdot VAL$;
16 | | $T(k) \leftarrow Inserer(T(k), x_0, y_0, D_{i_0})$;
15. | **Fin**
16. **Fin**

Algorithme 9 : Insertion_Exhaustive_RF

Entrées : D : l'ensemble des demandes ;
Sorties : T : l'ensemble des tournées ;
$Reject$: l'ensemble des demandes rejetées.

$Reject \leftarrow NULL$
Tant que $D \neq NULL$ **faire**
| Choisir D_{i_0} dans D ; $D \leftarrow D - \{D_{i_0}\}$;
| Enumérer toutes les insertions possibles $Inserer\left(T(k), x, y, D_{i_0}\right)$ qui soit telles que $INSERT \cdot Res = 1$;
| **Si** Echec_Enumérer **alors**
| | $succe = false$; $k \leftarrow 1$; $k' \leftarrow 1$;
| | **Tant que** $k \leq K$ && $succe = false$ **faire**
| | | **Tant que** $k' \leq K$ && $k \neq k'$ && $succe = false$ **faire**
| | | | $succe \leftarrow Reparer(k, k', i_0)$;
| | | **Fin**
| | **Fin**
| | **Si** $succe = false$ **alors**
| | | $res \leftarrow Reparer\_Fort(T, D_{i_0})$;
| | | **Si** $res = false$ **faire**
| | | | $Rejet \leftarrow Reject + \{i_0\}$;
| | | **fin**
| | **Fin**
| **Sinon**
| | Choisir k_o, x_0, y_0 qui correspond à la meilleure valeur $Inserer \cdot VAL$;
| | $T(k_0) \leftarrow Inserer(T(k_o), x_0, y_0, D_{i_0})$;
| **Fin**
Fin

L'algorithme « Insertion_Exhaustive_RF » peut être alors modifié comme décrit par l'Algorithme 9. Bien sûr, la même procédure peut être appliquée à « Insertion_Partielle_RF ».

6. Evaluations Numériques

Toutes les procédures ont été codées en Borland C++ 6.0 et exécutées sur un PC 2.4GHz sous Windows Xp avec 4 Go de mémoire. Toutes les instances construites par Cordeau et Laporte [**77**] sont disponibles sur le site : http://neumann.hec.ca/chairedistributique/data/darp. Ces instances sont composées de 24 et 144 demandes. Dans chaque instance, chaque demande de possède une fenêtre de temps associée, soit à son origine, soit à sa destination. Un temps de service (de durée 10) est imposé pour le chargement et le déchargement. Chaque demande est associée à une charge unitaire. La durée maximale d'une tournée est fixée à 480, la durée de transport maximale pour une demande est 90 et la capacité du véhicule est égale à 6. Les

instances sont divisées en deux groupes : (Pr01…Pr10) et (Pr11…Pr20). Dans le premier groupe, les fenêtres du temps sont plus serrées. Dans chaque groupe, le nombre de véhicules disponibles pour les instances 1 à 6 est tel que les tournées ne sont pas trop chargées. Le nombre de véhicules disponibles pour les instances 7 à 10 est supposé être le nombre minimal de véhicules nécessaires.

6.1. Résultats des heuristiques

Le Tableau 1 présente les résultats obtenus par la méthode heuristique « Insertion_Exhaustive » (Algorithme 2) sans utiliser la procédure « Réparer » ni la procédure « Reparer_fort ». Les demandes sont triées par ordre croissant selon $\beta(x)$, $x \in \{o_i, d_i | i \in I\}$, et on choisit la demande D_{i_0} à insérer dans cette liste. Dans le Tableau 1, la première colonne présente l'instance. La seconde colonne présente le nombre de véhicules utilisés K' et le nombre de véhicules disponibles K. La troisième colonne $|I'|$ présente le nombre de demandes traitées par l'heuristique. La quatrième colonne présente le temps d'exécution en secondes pour obtenir une solution. Les colonnes 5 à 7 représentent les trois critères T_{Global}, T_{Ride}, T_{Wait}. L'objectif est de minimiser $\sum_{k=1,\dots,K}(\gamma \times T_{Global}(k) + \delta \times T_{Ride} + \varepsilon \times T_{Wait})$, les trois coefficients sont définis par : $\gamma = 2, \delta = 1$ et $\varepsilon = 1$. Parmi les instances testées, les instances pr09, pr10, pr19 et pr20 ont des demandes rejetées. Le nombre de véhicules utilisés pour les instances pr13, pr14, pr16 est inférieur au nombre de véhicules disponibles.

Tableau 1 : Résultats obtenus par « Insertion_Exhaustive »

Instance	K	K'	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}
pr01	3	3	24	24	0,094	746,65	749,64	11,84
pr02	5	5	48	48	0,422	1701,39	1645,59	254,64
pr03	7	7	72	72	0,765	2465,66	2723,07	170,73
pr04	9	9	96	96	1,844	3036,41	3785,54	137,52
pr05	11	11	120	120	3,156	3455,83	4700,50	0,16
pr06	13	13	144	144	3,875	4376,44	6143,41	179,17
pr07	4	4	36	36	0,234	1216,13	1086,46	74,14
pr08	6	6	72	72	0,938	2165,05	2413,68	1,27
pr09	8	8	108	**105**	2,313	3093.59	3864.41	0
pr10	10	10	144	**140**	4.719	4403,81	6023,70	236,96
pr11	3	3	24	24	0,187	711,83	761,84	0,11
pr12	5	5	48	48	0,766	1383,08	1634,03	0
pr13	7	6	72	72	1,938	2159,88	2753,44	0
pr14	9	8	96	96	3,703	2825,70	3803,12	6,75
pr15	11	11	120	120	6,266	3402,88	4112,06	0
pr16	13	12	144	144	8,422	4166,18	5425,18	0
pr17	4	4	36	36	0,546	1065,07	1282,56	5,19
pr18	6	6	72	72	1,531	2233,47	3137,12	19,95
pr19	8	8	108	**107**	4.547	3206,36	3755,46	0,32
pr20	10	10	144	**140**	9,110	4151.76	5395,33	0,42

Tableau 2 : Résultats obtenus par « Insertion_Partielle »

Instance	K	K'	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}
pr01	3	3	24	24	0,110	1116,52	401,97	311,50
pr02	5	5	48	48	0,125	2023,40	1187,70	504,87
pr03	7	7	72	72	0,219	2889,46	1519,56	390,94
pr04	9	9	96	96	0,312	3782,98	2061,97	593,61
pr05	11	11	120	120	0,390	4319,15	2836,63	569,77
pr06	13	13	144	144	0,672	5374,75	2743,09	680,15
pr07	4	4	36	36	0,094	1492,39	600,88	297,75
pr08	6	6	72	72	0,234	2423,04	1850,26	48,23
pr09	8	8	108	**100**	0.219	3355,66	2398,86	89,98
pr10	10	10	144	**132**	0.344	4479,47	3263,39	177,21
pr11	3	3	24	24	0,062	1068,33	655,70	285,52
pr12	5	5	48	48	0,156	1858,92	1080,68	358,24
pr13	7	7	72	72	0,234	2584,35	1819,02	165,88
pr14	9	9	96	96	0,328	3271,37	2586,96	259,18
pr15	11	11	120	120	0,453	4224,8	2932,67	527,53
pr16	13	13	144	144	0,500	5108,16	3397,48	552,49
pr17	4	4	36	36	0,125	1412,06	974,40	209,78
pr18	6	6	72	72	0,236	2546,30	2018,1	187,46
pr19	8	8	108	108	0,546	3554,36	3266,13	130,83
pr20	10	10	144	144	0,921	4485,33	4597,48	18,29

Tableau 3 : « Insertion_Exhaustive » sans trier les demandes

Instance	K	K'	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}
pr01	3	3	24	**20**	0.063	696,26	629 ,24	64,49
pr02	5	5	48	**37**	0,406	1284,91	1149,58	196,29
pr03	7	7	72	**67**	0.641	2300,13	2458,36	71,15
pr04	9	9	96	**88**	1,594	2780,52	3267,43	27 ,58
pr05	11	11	120	**102**	2,89	2983,58	3351,35	12,48
pr06	13	13	144	**136**	3,609	4184,96	4452,53	59,98
pr07	4	4	36	**35**	0,172	1266,77	891,48	112,99
pr08	6	6	72	**63**	0,938	2070,35	2182,75	29,59
pr09	8	8	108	**84**	2,485	2675,14	2397,16	19,44
pr10	10	10	144	**127**	4,688	4082,94	4292,9	144,23
pr11	3	3	24	24	0,125	732,27	795,19	0
pr12	5	5	48	**47**	0,625	1384,11	1362,65	2,93
pr13	7	7	72	**67**	1,89	2157,3	2471,67	3,14
pr14	9	9	96	**95**	4,36	2865,91	3141,60	1,30
pr15	11	11	120	**114**	6,094	3284,78	4108,33	0
pr16	13	13	144	**140**	9.141	4124 ,31	5269,08	0,59
pr17	4	4	36	36	0.42	1215,13	1153,98	58,61
pr18	6	6	72	**70**	1,75	2322,69	2493,92	51,59
pr19	8	8	108	**100**	5.296	3285,03	3137,72	172,55
pr20	10	10	144	**136**	10.254	4140,52	5114,04	20,80

Tableau 4 : « Insertion_Partielle » sans trier les demandes

Instance	K	K'	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}
pr01	3	3	24	24	0,047	1034,46	759,65	269,72
pr02	5	5	48	**32**	0,203	1138,27	895,25	181,00
pr03	7	7	72	**67**	0,359	2571,35	2365,74	259,38
pr04	9	9	96	**90**	0,781	3225,71	3101,39	354,71
pr05	11	11	120	**119**	0,766	4353,25	4201,91	657,23
pr06	13	13	144	**128**	2,172	4643,83	4290,48	534,73
pr07	4	4	36	**35**	0,062	1346,56	1242,24	192,78
pr08	6	6	72	**59**	0,64	2413,9	1764,41	442,67
pr09	8	8	108	**94**	1,516	3141,75	2952,16	50,85
pr10	10	10	144	**122**	3,078	4584,59	4537,58	567,91
pr11	3	3	24	24	0,062	972,97	833,14	228,08
pr12	5	5	48	48	0,188	1594,82	1673,52	122,82
pr13	7	7	72	**71**	0,484	2587,33	2761,24	241,88
pr14	9	9	96	96	0,563	3425,60	3526,01	333,03
pr15	11	11	120	120	1,687	4034,04	4092,72	407,67
pr16	13	13	144	144	1,609	4910,94	5175,20	419,06
pr17	4	4	36	36	0,36	1316,04	1535,49	131,14
pr18	6	6	72	**69**	0,89	2429,53	2431,83	178,76
pr19	8	8	108	**103**	2,485	3580,70	3636,06	263,66
pr20	10	10	144	**139**	4,062	4488,52	5271,56	66,09

Comme le Tableau 1, le Tableau 2 présente les résultats obtenus par l'heuristique « Insertion_Partielle » (Algorithme 3). Les instances pr09, pr10 ont des demandes rejetées.

Remarque 5

Si l'on choisit la demande insérée D_{i_0} sans utiliser la liste triée, on constate alors que beaucoup d'instances ont des demandes rejetées (voir Tableau 3 et Tableau 4). Pour la méthode d'insertion, l'ordre des demandes à insérer est très important.

6.2. Résultats des heuristiques avec la procédure de réparation

Lorsque certaines demandes ne sont pas traitées par la méthode « Insertion_Exhaustive » et « Insertion_Partielle », on utilise la procédure « Réparer ». Dans le Tableau 5 et le Tableau 6 la colonne « Nb_R » présente le nombre d'utilisation de réparations, la colonne « Nb_E » présente le nombre de réparations échouées.

Dans le Tableau 5, trois instances (pr09, pr19, pr20) sur quatre sont bien réparées. Pour l'instance pr10, la procédure « réparer » est appelée 8 fois, et échoue 2 fois. Dans le Tableau 6, l'instance pr10 est bien réparée. Pour l'instance pr09, 2 demandes sont encore rejetées.

Tableau 5 : Résultats obtenus par « Insertion_Exhaustive_R »

Instance	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}	Nb_R	Nb_E
pr09	108	108	9,344	3258,90	4385,68	0	6	0
pr10	144	**142**	51,781	4468,55	6370,57	94,24	8	2
pr19	108	108	8,187	3254,15	3821,21	2,99	2	0
pr20	144	144	66,829	4433,83	5471,67	3,29	7	0

Tableau 6 : Résultats obtenus par « Insertion_Partielle_R »

Instance	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}	Nb_R	Nb_E
pr09	108	106	18,734	3616,21	4015,23	0	8	2
pr10	144	144	5,735	4682,83	4292,75	21,49	2	0

6.3. Résultats des heuristiques avec la procédure de réparation forte

On teste les instances qui ont des demandes rejetées en utilisant la procédure « $Reparer\_Fort$ ». Les résultats obtenus par la méthode « Insertion_Exhaustive_RF » et « Insertion_Partielle_RF » sont présentés par le Tableau 7 et le Tableau 8. La colonne « Nb_R » représente le nombre d'utilisations de réparation, la colonne « Nb_RF » présente le nombre d'utilisations de réparation forte. Dans le Tableau 7, pour l'instance pr10, On trouve que toutes les demandes sont traitées par la méthode « Insertion_Exhaustive_RF » et que le nombre nécessaire de réparations est inférieur à celui requis par la méthode « Insertion_Exhaustive_R ». Dans le Tableau 8, l'instance pr09 n'est pas réparée par la méthode « Insertion_Partielle_RF ».

Tableau 7 : Résultats obtenus par « Insertion_Exhaustive_RF »

Instance	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}	Nb_R	Nb_RF
pr10	144	144	19,281	4487,2	6069,45	140,66	6	1

Tableau 8 : Résultats obtenus par « Insertion_Partielle_RF »

Instance	$\|I\|$	$\|I'\|$	CPU (s)	T_{Global}	T_{Ride}	T_{Wait}	Nb_R	NB_RF
pr09	108	**106**	17,844	3630,98	4022,17	16,49	7	4

6.4. Comparaison avec les résultats de la recherche taboue [77]

Le Tableau 9 compare les résultats de nos heuristiques avec deux procédures de réparation à la recherche taboue de Cordeau et Laporte [**77**]. Cordeau et Laporte [**77**] minimisent les

distances parcourues alors que nos heuristiques minimisent une combinaison des différentes durées. Nous mettrons les coefficients $\gamma = 10,\ \delta = 0,\ \varepsilon = 1$ afin de donner plus de poids à la durée totale des tournées.

Les résultats donnés par [**77**] sont obtenues avec 10^5 itérations. Nous remarquons que nos heuristiques sont très rapide par rapport à la recherche taboue [**77**]. Nous constatons par contre que la recherche taboue donne des distances deux fois meilleures en moyenne que nos heuristiques, l'heuristique « Insertion_Exhaustive_RF » donne des durées des tournées et des temps d'attente plus faibles. L'heuristique « Insertion_Partielle_RF » donne des temps de transport plus faibles. La fonction objectif détermine les caractéristiques des tournées. Les résultats obtenus par nos heuristiques sont plus favorables aux clients (meilleure qualité de service).

6.5. Comparaison avec les résultats de l'algorithme génétique [127]

Le Tableau 10 compare les résultats de nos heuristiques avec la procédure de réparation à l'algorithme génétique de Jorgensen et al. [**127**]. Jorgensen et al. minimisent une combinaison linéaire des critères suivants :

- la distance totale des tournées $\sum_{k=1,\dots,K} T_{Global}\big(T(k)\big) - T_{Wait}(T(k))$;
- le temps de transport supplémentaire par rapport au temps de transport pour chaque demande $(t(d_i) - t(o_i) - s(o_i) - Dist[o_i, d_i])$;
- le temps total d'attente des passagers (temps d'attente en chaque sommet multiplié par le nombre de passagers dans le véhicule) ;
- la durée totale des tournées : $T_{Global}\big(T(k)\big), k \in \{1, \dots, K\}$;
- la pénalité de la violation des fenêtres de temps : $max\{0, t(x) - \beta(x)\}$;
- la pénalité de la violation du temps de transport maximal des demandes : $max\{0, \Delta_i - (t(d_i) - t(o_i))\}$;
- la pénalité de la violation de la durée maximale des tournées.

Pour ces 7 critères, ils proposent 7 coefficients ω_i, $i = 1, \dots, 7$ qui sont fixés à $\omega_1 = 8$, $\omega_2 = 3$, $\omega_3 = 1$, $\omega_4 = 1$ et $\omega_5 = \omega_6 = \omega_7 = n$ où n est le nombre des demandes. Pour effectuer la comparaison entre nos heuristiques et l'heuristique de Jorgensen [**127**]. Nous mettrons les coefficients : $\gamma = 8,\ \delta = 3,\ \varepsilon = 1$.

Jorgensen et al. [**127**] ne donnent que des résultats pour 13 instances sur 20. En moyenne sur les 13 instances, nous constatons que l'heuristique « Insertion_Exhaustive_RF » donne des durées totales de transport T_{Ride} trois fois plus faibles que l'algorithme génétique, et des temps d'attente 47% plus faibles. L'heuristique « Insertion_Partielle_RF » donne des durées totales de transport T_{Ride} 51.01% plus faibles. Il y a peu de différences pour les autres critères.

7. Conclusion

Ce chapitre présente deux heuristiques gloutonnes basées sur la technique d'insertion pour le problème de Dial-A-ride (DARP). Le but de ces méthodes est de fournir rapidement des solutions du DARP. Les instances proposées par Cordeau et Laporte [**77**] sont été testées par nos heuristiques. Les expérimentations montrent que nos heuristiques sont capables de donner des résultats de bonne qualité, que nos heuristiques proposent des solutions qui sont plus favorable aux clients (meilleure qualité de service).

Les travaux présentés dans ce chapitre ont fait l'objet d'une publication, présentée à la conférence EU/ME'08 2008.

Tableau 9 : Comparaison des résultats entre nos heuristiques et la recherche taboue

	Insertion_Partielle_RF					**Insertion_Exhaustive_RF**					**Recherche taboue** [77]				
Instance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance
pr01	0,03	1063,27	851,12	269,19	314,08	0,03	796,52	819,97	38,99	277,53	1140	881,20	1095,00	211,20	190,00
pr02	0,03	2043,09	1521,66	509,09	574,00	0,30	1696,00	1951,10	264,50	471,50	4836	1985,90	1976,70	723,90	302,10
pr03	0,08	2858,92	2261,11	331,40	1087,52	0,50	2375,01	2706,46	80,34	854,67	10308	2579,40	3586,70	607,30	532,10
pr04	0,08	3684,72	2546,87	439,05	1325,67	1,06	2908,84	3466,21	3,28	985,56	17262	3583,20	5021,30	1090,40	572,80
pr05	0,14	4352,90	3430,17	612,16	1340,74	1,97	3508,49	4474,11	63,90	1044,59	27744	3870,00	6156,50	833,00	637,00
pr06	0,27	5412,75	3826,77	734,61	1798,14	2,82	4295,13	5815,29	4,68	1410,45	32322	5056,80	7273,50	1375,40	801,40
pr07	0,02	1495,64	808,71	268,30	507,34	0,13	1205,35	1094,88	98,33	387,02	2634	1452,10	1508,90	440,30	291,70
pr08	0,09	2457,28	1987,36	62,95	954,33	0,62	2250,26	2788,70	66,19	744,07	12264	2345,20	3691,50	410,30	494,90
pr09	5,12	3542,68	4388,95	33,78	1348,90	4,18	3365,41	3654,12	42,57	1162,84	30306	3155,50	5621,80	323,10	672,40
pr10	27,92	4702,47	6103,67	127,21	1695,26	6,60	4545,99	6092,70	175,22	1490,77	52518	4480,10	7163,70	721,30	878,80
pr11	0,03	1059,41	741,14	253,50	325,91	0,09	745,52	1026,91	8,52	257,00	1158	965,10	1041,50	320,60	164,50
pr12	0,06	1804,75	1647,17	241,75	603,00	0,59	1401,42	1296,37	0,00	441,42	4974	1564,70	2393,80	308,70	296,10
pr13	0,09	2576,99	2214,78	148,44	988,55	1,36	2173,95	2757,13	0,00	733,95	11124	2263,70	3788,80	330,40	493,30
pr14	0,16	3274,35	2737,16	217,23	1137,12	3,25	2833,99	3757,38	0,00	913,99	18708	2882,20	4632,80	426,30	535,90
pr15	0,23	4329,83	3595,01	579,35	1350,48	4,87	3417,53	4669,48	0,00	1017,53	32598	3595,60	6104,70	605,90	589,70
pr16	0,25	5084,89	4278,89	418,00	1786,89	7,11	4160,34	5507,95	0,00	1280,34	44220	4072,50	7347,40	448,90	743,60
pr17	0,08	1376,77	1025,92	163,99	492,78	0,36	1107,53	1252,68	0,00	387,53	2538	1097,30	1762,00	129,00	248,20
pr18	0,09	2607,00	2181,65	239,09	927,91	1,28	2358,40	2663,34	146,07	772,33	13716	2446,50	3659,00	523,40	462,70
pr19	0,36	3637,44	3385,56	171,16	1306,28	3,51	3353,59	3816,04	85,52	1108,07	30768	3249,30	5581,00	487,30	602,00
pr20	1,03	4617,89	4273,79	65,13	1672,76	36,83	4463,77	5279,68	8,26	1575,51	55446	4041,00	7072,30	362,40	798,60
Moyenne	1,81	3099,15	2690,37	294,27	1076,88	3,87	2648,15	3244,53	54,32	865,83	20329,20	2778,37	4323,95	533,96	515,39

Tableau 10 : Comparaison des résultats entre nos heuristiques et l'algorithme génétique

	Insertion_Partielle_RF					**Insertion_Exhaustive_RF**					**Algorithme génétique** [**127**]				
Instance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Distance
pr01	0,02	1091,80	219,11	300,99	310,81	0,03	1078,20	276,13	302,64	295,56	334,20	1041,00	477,00	252,00	309,00
pr02	0,05	2063,43	542,05	541,93	561,50	0,39	2035,87	626,37	553,98	521,89	685,80	1969,00	1367,00	470,00	539,00
pr03	0,06	2871,21	1046,00	305,01	1126,20	0,39	2598,65	1021,23	179,42	979,23	3094,80	2779,00	3081,00	292,00	1047,00
pr04	0,08	3736,11	881,88	554,81	1261,30	0,98	3693,33	1749,92	679,11	1094,22	-	-	-	-	-
pr05	0,11	4326,81	1117,24	490,44	1436,37	1,53	4165,68	1705,62	527,77	1237,91	3493,80	4250,00	5099,00	500,00	1350,00
pr06	0,13	5349,90	1387,09	564,64	1905,26	1,97	4627,72	1718,31	226,27	1521,45	-	-	-	-	-
pr07	0,02	1499,58	365,42	285,92	493,66	0,11	1327,00	636 ,00	142,38	464,62	-	-	-	-	-
pr08	0,13	2441,01	1401,38	32,53	968,48	0,51	2319,39	1261,12	5,91	873,48	-	-	-	-	-
pr09	4,82	3517,09	4204,05	8,90	1348,19	3,90	3455,13	2607,91	9,54	1285,59	2446,80	3597,00	6251,00	94,00	1343,00
pr10	35,77	4703,04	7434,57	109,06	1713,98	14,26	4670,48	3361,43	126,91	1663,57	3958,80	5006,00	8413,00	315,00	1811,00
pr11	0,03	1100,65	179,44	316,77	303,88	0,13	805,29	391,72	61,13	264,16	327,60	907,00	630,00	143,00	284,00
pr12	0,08	1837,97	405,54	304,32	573,65	0,50	1430,58	416,34	0,00	470,58	703,20	1719,00	1214,00	198,00	561,00
pr13	0,09	2622,04	626,64	93,29	1088,75	1,08	2337,39	751,03	0,00	897,39	-	-	-	-	-
pr14	0,14	3371,99	930,72	228,35	1223,64	2,32	2973,27	922,21	0,00	1053,27	-	-	-	-	-
pr15	0,22	4318,89	996,02	528,36	1390,53	3,82	3526,28	990,06	0,00	1126,28	3535,80	4296,00	4615,00	552,00	1344,00
pr16	0,20	5073,13	1072,06	294,12	1899,01	4,87	4333,56	1354,89	3,77	1449,79	4873,80	5309,00	6134,00	630,00	1799,00
pr17	0,05	1378,20	291,61	135,02	523,18	0,31	1321,00	421,12	190,16	410,84	497,40	1299,00	990,00	102,00	478,00
pr18	0,09	2580,83	790,05	166,50	974,33	1,06	2646,00	1193,98	329,51	876,49	-	-	-	-	-
pr19	0,75	3703,02	1200,10	98,06	1444,96	3,14	3429,13	1524,72	28,23	1240,90	2679,60	3679,00	5362,00	147,00	1372,00
pr20	63,65	4643,46	6573,72	13,47	1749,99	27,14	4562,02	3175,07	43,56	1638,46	3984,60	4733,00	7969,00	113,00	1740,00
Moyenne sur 13 instances	8,14	3125,28	1944,73	265,11	1106,33	4,65	2877,84	1374,82	155,93	968,06	2355.09	3121,85	3969,38	292,92	1075,15
Moyenne sur 20instances	5,32	3111,51	1583,23	268,62	1114,88	3,42	2866,80	1305,26	170,51	968,28					

CHAPITRE 3
PROPAGATION DE CONTRAINTES POUR LA RESOLUTION DU DARP

1. Introduction

Dans le chapitre précédent, nous avons présenté des heuristiques gloutonnes basées sur la technique d'insertion. Nous utilisons la fonction d'évaluation proposée par Cordeau et Laporte [**77**]. Dans ce chapitre, nous proposons les outils pour vérifier rapidement les contraintes temporelles et évaluer le coût de la tournée. Ces outils nous permettent de concevoir des algorithmes d'insertion.

2. Manipulation des contraintes temporelles serrées : outils de base

Soit Γ une tournée, l'algorithme que nous décrivons dans la section suivante, est essentiellement basé sur des techniques d'insertion. Nous devons donc vérifier de façon rapide si l'insertion d'une demande D_i dans la tournée Γ garde valide de la tournée Γ, et obtenir une évaluation de la qualité de cette insertion. Nous portons une attention particulière au cas où les contraintes temporelles sont serrées. Nous présentons d'abord les manipulations concernant les contraintes.

2.1. La vérification de la charge-admissibilité

La vérification de la « Charge-Admissibilité » pour la tournée Γ est facile. Afin d'être en mesure de tester l'impact de l'insertion d'une demande dans une tournée Γ sur la « Charge-Admissibilité », nous associons à la tournée Γ, la quantité de charge $c(\Gamma, x), x \in X$, définie par :

$$\forall x \in \Gamma, c(\Gamma, x) = \sum_{y \ll_{\Gamma}^{=} x} c(y)$$

Alors Γ est une tournée « charge-admissible » si et seulement si :

$$\forall x \in \Gamma, c(\Gamma, x) \leq CAP$$

2.2. La vérification de la temps-admissibilité

La vérification de la « Temps-Admissibilité », selon un ensemble des fenêtres de temps actuelles $FS = \{F(x) = [\alpha(x), \beta(x)], x \in \Gamma\}$ peut être réalisée par les règles de propagation $R_i, i = 1, \dots, 5$:

- $R_1: x \in \Gamma; y = succ(\Gamma, x);\ t(y)$ ne peut pas commencer avant $FS.\alpha(x) + Dist[x, y]$:
$$FS.\alpha(y) < FS.\alpha(x) + Dist[x, y] \vDash FS.\alpha(y) \leftarrow FS.\alpha(x) + Dist[x, y]$$
- $R_2: x \in \Gamma; y = succ(\Gamma, x);\ t(x)$ ne peut pas commencer après $FS.\beta(y) - Dist[x, y]$:
$$FS.\beta(x) > FS.\beta(y) - Dist[x, y] \vDash FS.\beta(x) \leftarrow FS.\beta(y) - Dist[x, y]$$
- $R_3: x \in \Gamma; Type(x) = Destination$ ou $DepotA$, $y = Hom(x)$, $t(x)$ ne peut pas commencer après $FS.\beta(x) - \Delta(x)$:
$$FS.\beta(x) > FS.\beta(y) + \Delta(x) \vDash FS.\beta(x) \leftarrow FS.\beta(y) + \Delta(x)$$
- $R_4: x \in \Gamma; Type(x) = Origine$ ou $DepotD, y = Hom(x),\ t(x)$ ne peut pas commencer avant $FS.\alpha(y) - \Delta(x)$:
$$FS.\alpha(x) < FS.\alpha(y) - \Delta(x) \vDash FS.\alpha(x) \leftarrow FS.\alpha(y) - \Delta(x)$$
- $R_5:\ x \in \Gamma$:
$$FS.\alpha(x) > FS.\beta(x) \vDash Echec$$

Nous pouvons alors définir une fonction « $\boldsymbol{Propage}$ » de propagation de contraintes par l'Algorithme 10**.**

La tournée Γ est « Temps-Admissible » selon un ensemble de fenêtres de temps $FS(\Gamma)$ si et seulement si la composante Res du résultat de la procédure $Propage(\Gamma, FS)$ est égal à vrai. Dans un tel cas, le vecteur de dates t lié à la Γ et FS est tel que : $\forall x \in \Gamma, t(x) \in FS(x)$.

Notons que l'ensemble des fenêtres de temps réduites $FR(\Gamma)$ qui résultent de la procédure $Propage(\Gamma, FS).FR$, peuvent être considérées comme les plus grandes fenêtres de temps qui sont inclues dans FS et qui sont stables selon les règles R_i, $i = 1, \dots, 5$.

Algorithme 10 : Propage

Entrées : Γ : une tournée ;
FS : l'ensemble des fenêtres de temps lié à Γ ;
Sorties : Res : booléen ;
FR : l'ensemble des fenêtres de temps réduites lié à Γ ;

```
1  L ←Liste des sommets de Γ ;
2  Res ← vrai ;
3  Tant que (L ≠ NULL) && (Reussite = vrai) faire
4  |   x ← Debut(L) ; L ← Queue(L) ;
5  |   Pour i de 1 à 5 faire
6  |   |   Chercher si R_i s'applique à partir de x ;
7  |   |   Si R_i s'applique et i = 5 alors
8  |   |   |   Res ← faux
9  |   |   Sinon Si R_i s'applique et i ≤ 4 alors
10 |   |   |   Appliquer R_i ;
11 |   |   |   Si le sommet résultant n'est pas dans L alors
12 |   |   |   |   Insérer ce sommet dans L
13 |   |   |   Fin
14 |   |   Fin
15 |   Fin
16 Fin
17 Propage ← (Res, FS)
```

3. Evaluation de la tournée

Considérons maintenant la tournée Γ, associée à l'ensemble des fenêtres de temps $FR(\Gamma)$. Nous voulons obtenir rapidement des estimations sur la meilleure valeur possible $Cout_{\gamma,\delta,\varepsilon}(\Gamma) = \gamma \times T_{Global}(\Gamma) + \delta \times T_{Ride}(\Gamma) + \varepsilon \times T_{Wait}(\Gamma)$ définie dans la section précédente. Nous voulons effectuer ce processus d'évaluation d'une manière rapide, nous concevons donc deux procédures EVAL1 et EVAL2.

La procédure EVAL1 fonctionne d'une manière gloutonne, d'abord en attribuant au premier sommet $Debut(\Gamma)$ de la tournée Γ sa plus grande valeur possible pour la date $t(x)$, et en suivant l'exécution d'un processus de « Bellman » pour attribuer à chaque sommet x de Γ sa plus petite valeur possible.

La procédure EVAL2 commence à partir d'une solution produite par la procédure EVAL1, elle l'améliore en réalisant une séquence de mouvements locaux, chaque mouvement implique une valeur $t(x)$.

Algorithme 11 : Procédure EVAL1

Entrées : Γ : une tournée ;
Sorties : Val : réel ; t : l'ensemble des dates ;

Pour chaque sommet x dans Γ **faire**
| $[\alpha(x), \beta(x)] \leftarrow F(x)$;
Fin
$x \leftarrow Debut(\Gamma)$;
$t(x) \leftarrow \beta(x)$;
Tant que $x \neq Fin(\Gamma)$ **faire**
| $y \leftarrow Succ(\Gamma, x)$;
| $t(y) \leftarrow Max(\alpha(y), t(x) + Dist[x, y])$;
| $x \leftarrow y$;
Fin
$t \leftarrow \{t(x), x \in \Gamma\}$;
$Val \leftarrow Cout_{\gamma,\delta,\varepsilon}(\Gamma)$;

Algorithme 12 : Procédure EVAL2

Entrées : Γ : une tournée ;
Sorties : Val : réel ; t : l'ensemble des dates ;

Pour chaque sommet x **faire**
| $t(x) \leftarrow EVAL1(\Gamma).t$;
Fin
Tant que $Non\ Stop$ **faire**
| Recherche x dans Γ tels que l'une des deux contraintes ci-dessous est vrai :
| $(E2)\ Type(x) \in \{Origine, Depot1\}$ && $t(x) \neq Min(\beta(x), t\big(succ(\Gamma, x)\big) - Dist[x, succ(\Gamma, x)]))$;
|
$(E3)\ Type(x) \in \{Destination, Depot2\}$ && $t(x) \neq Max(\alpha(x), t\big(prev(\Gamma, x)\big) - Dist[prev(\Gamma, x), x]))$;
| **Si** $Echec(Recherche)$ **alors**
| | $Stop$
| **else**
| | **Si** $(E2)$ **alors** $t(x) = Min(\beta(x), t\big(succ(\Gamma, x)\big) - Dist[x, succ(\Gamma, x)]))$;
| | **Si** $(E3)$ **alors** $t(x) = Max(\alpha(x), t\big(prev(\Gamma, x)\big) - Dist[prev(\Gamma, x), x]))$;
| **Fin**
Fin
$t \leftarrow \{t(x), x \in \Gamma\}$;
$Val \leftarrow Cout_{\gamma,\delta,\varepsilon}(\Gamma)$;

Le vecteur de dates t obtenu par les procédures EVAL1 et EVAL2, est compatible avec la tournée Γ et l'ensemble des fenêtres de temps F. En outre, si $\delta = \varepsilon = 0$, la procédure EVAL1 génère alors une valeur optimal Val, c'est-à-dire la plus petite valeur possible pour $Cout_{\gamma,\delta,\varepsilon}(\Gamma)$.

Γ étant une tournée admissible, on désigne par VAL1(Γ), VAL2(Γ), les valeurs produites respectivement par les procédures EVAL1 et EVAL2.

4. Graphe non compatible sur les demandes

Soient deux demandes D_i et D_j, $i, j \in I, i \neq j$. Dans le cadre d'un prétraitement, nous pouvons vérifier la validité des tournées suivantes:

$$\begin{cases} DepotD(1), o_i, d_i, o_j, d_j, DepotA(1) & : \sigma_1 \\ DepotD(1), o_i, o_j, d_i, d_j, DepotA(1) & : \sigma_2 \\ DepotD(1), o_i, o_j, d_j, d_i, DepotA(1) & : \sigma_3 \\ DepotD(1), o_j, o_i, d_j, d_i, DepotA(1) & : \sigma_4 \\ DepotD(1), o_j, o_i, d_i, d_j, DepotA(1) & : \sigma_5 \\ DepotD(1), o_j, d_j, o_i, d_i, DepotA(1) & : \sigma_6 \end{cases}$$

Dans le cas où le processus de vérification conclut qu'aucune des tournées ci-dessus n'est admissible, on dit que D_i et D_j ne sont pas compatibles, et nous notons $non\_compatible(i,j)$. Nous définissons alors le graphe non compatible $G\_No\_Comp(I, non\_Compatible)$ pour l'ensemble des demandes.

Clairement, toute solution réalisable d'une instance liée au DARP va induire une coloration des sommets pour le graphe G_No_Comp avec K couleurs $1, \dots, K$. Supposons maintenant que les tournées $T(k), 1, \dots, K$ ont été partiellement construites, ce signifie que certains sous-ensembles $J \subset I$ sont tels que les demandes $\{D_j, j \in J\}$ ont déjà été insérées dans l'ensemble des tournées $T(k), 1, \dots, K$, et que l'ensemble complémentaire $H = I - J$ reste à traiter. La position d'insertion pour la demande non insérée D_j par rapport aux véhicules ou aux autres demandes insérées, peut être définie grâce à une coloration des sommets du sous graphe G_No_Comp qui est induite par H, de telle sorte que :

- $\forall\, i \in H$, la couleur $r(i)$ de la demande D_i est une partie du domaine autorisé $Dom(i) = \{k = 1, \dots, K\}$ tels qu'il n'existe pas une demande D_j telle que :
    - la demande D_j est traitée par le véhicule k ;
    - $non\_compatible(i,j)$.

Le test de l'existence d'une telle coloration des sommets peut être réalisé par la procédure $COULEUR$ en utilisant une recherche arborescente : la procédure prend H et $Dom = \{Dom(i), i \in H\}$ comme données d'entrée, et effectue une recherche arborescente partielle, filtrée par des règles de propagation liées au domaine fini. La procédure produit un résultat booléen Res qui exprime la réussite ou l'échec de la recherche, un ensemble r de sommets coloriés si une telle coloration existe, et une liste de $clauses$ qui exprime que certains

sommets ne sont pas coloriés. Le point fondamental ici est que nous voulons que ce processus soit associé au processus d'insertion globale proposé dans la section 6. La procédure $COULEUR$ est seulement heuristique, ce qui signifie que nous n'avons pas l'intention de procéder à la recherche d'arbres entiers. Elle est appelée plusieurs fois pendant le processus d'insertion, et les demandes sont essayées dans le même ordre dans le processus $COULEUR$ et dans le processus d'insertion globale.

Algorithme 13 : Procédure COULEUR

Entrées : H : l'ensemble des demandes non insérées ;
Dom : l'ensemble des véhicules autorisés ;
Sorties : Res : booléen ;
r : couleur ;
Cl : la liste de clauses ;

Si $H = NULL$ **alors**
  $COULEUR \leftarrow (1, NULL, NULL)$;
Sinon
  Sélectionner $i_0 \in H$;
  $Not\ Stop$;
  Sélectionner $L \subset Dom(i_0)$;
  $L\_Tabou \leftarrow NULL$;
  Tant que $Not\ Stop$ et $L \neq NULL$ **faire**
    $k_0 \leftarrow Debut(L); L \leftarrow Queue(L)$;
    $Dom1 \leftarrow Dom; J_0 \leftarrow \{i_0\}$;
    $(success, Dom1) \leftarrow$ Résultat de la propagation de la couleur affectée $r(i_0) \leftarrow k_0$
      en utilisant les règles d'inférence suivantes :
    -- R_1^*: $i \in H$ tel que $|Dom1(i)| = 1$ et $r(i)$ est libre $\vDash r(i) = k, k \in Dom1(i)$ et $J_0 = J_0 \cup \{i\}$
    -- R_2^*: $i, i' \in H, k \in \{1, \dots, K\}$ tels que $r(i) = k, k \in Dom1(i)$ et $Non\_Compatible(i, j) \vDash$
      supprimer k dans $Dom1(i')$
    -- R_3^*: $i \in H, k \in \{1, \dots, K\}$ sont tels que $k \in Dom1(i)$
      et k appartient à aucun sous ensemble $Dom1(j)$,
      $j \in H - J_0 \vDash r(i) = k, k \in Dom1(i)$ et $J_0 = J_0 \cup \{i\}$
    -- R_4^*: $i \in H, Dom1(i) = NULL \vDash succès = faux$
    Si $succès = faux$ **alors**
      Insérer k_0 dans L_Tabou
    Sinon
      $(Res\_aux, r\_aux, Cl\_aux) \leftarrow COULEUR(H - J_0, Dom1)$;
      Si $Res\_aux = 0$ **alors**
        Insérer k_0 dans L_Tabou;
      Sinon
        $Stop$;
        $Res \leftarrow 1; r \leftarrow r\_aux \cup \{r(i_0) = k_0\}$;
        $Cl \leftarrow \{r(i_0) \neq k, k \in L\_Tabou\} \cup \{r(i_0) = k_0 \rightarrow c, c \in Cl\_aux\}$;
      Fin
    Fin
  Fin
Fin

Remarque 6

Cette procédure $COULEUR$ est très simple, et de toute évidence, il est possible de procéder de manière plus efficace si nous souhaitons assurer l'existence d'une coloration pour l'ensemble H avec les couleurs $1, \dots, K$. En outre, elle est heuristique. Elle peut être exacte si la liste L est choisie comme l'ensemble $Dom(i_0)$. De plus, nous pouvons penser que, dans la plupart des cas, une coloration existe et peut être obtenue grâce nombre assez faible de mouvement de retours en arrière.

Remarque 7 : le sens de la composante clause $COULOUR.Cl$

Cette composante du résultat (produit par un appel à la procédure $COULOUR$) apporte une information supplémentaire qui permet de réduire les domaines $Dom(i), i \in H - J$ par la relation $non\_compatible$ lorsque des demandes sont insérées dans la tournée (k) $k \in \{1, \dots, K\}$.

5. Un simple algorithme d'insertion pour le DARP avec contraintes serrées

5.1. Le mécanisme d'insertion

L'algorithme fonctionne d'une manière très naturelle. Soit Γ une tournée admissible, D_i une demande dont le sommet origine o_i et le sommet destination d_i ne sont pas dans la tournée Γ. Soit deux sommets $x, y \in \Gamma$, tel que $x \ll^{=}_{\Gamma} y$. La nouvelle tournée (après insertion de D_i) est obtenue par $INSERT(\Gamma, x, y, i)$:

- o_i est situé entre x et $succ(\Gamma, x)$;
- d_i est situé entre x et $succ(\Gamma, y)$;

Nous disons que $Insert(\Gamma, x, y, i)$ donne le résultat de l'insertion de la demande D_i dans la tournée Γ selon les sommets (ou les positions) d'insertion x et y. Avant toute autre chose, nous détaillons les méthodes pour tester la validité d'une tournée.

5.1.1. Test de la « Charge-Admissibilité »

Pour la vérification de la « Charge-Admissible », Nous vérifions $c(\Gamma, z) + c_i \leq CAP$ pour tout z dans $Segment(\Gamma, x, y)$. Nous proposons la procédure suivante :

- Procédure Test_Load(Γ, x, y, i) :
  $Test\_load \leftarrow \{\forall\ z\ tel\ que\ x \ll^{=}_{\Gamma} z \ll^{=}_{\Gamma} y, c(\Gamma, z) + c_i \leq CAP\}$

5.1.2. Test du « Temps-Admissibilité »

La procédure $Propage(\Gamma \cup \{o_i, d_i\}, FR(\Gamma))$ est suffisante pour la vérification de la « Temps-Admissibilité ». Mais cette procédure est très couteuse en temps de calcul. Afin de rendre le processus de test plus rapide, nous proposons plusieurs tests intermédiaires, qui visent à interrompre le processus de test dans le cas où la tournée ne serait pas temps-admissible :

- Le premier test $Test\_Node1(\Gamma, u, z, z')$ vise à vérifier la faisabilité de l'insertion d'un sommet u lié à une quantité de charge Q entre deux sommets consécutifs z et z'. Nous donnons un ensemble $EC1$ des conditions nécessaires pour tester la faisabilité de cette insertion :

  On pose :
  $\forall x \in \Gamma, [a(x), b(x)] = FR(\Gamma)(x)$;
  $[\alpha(u), \beta(u)] = F(u)$;

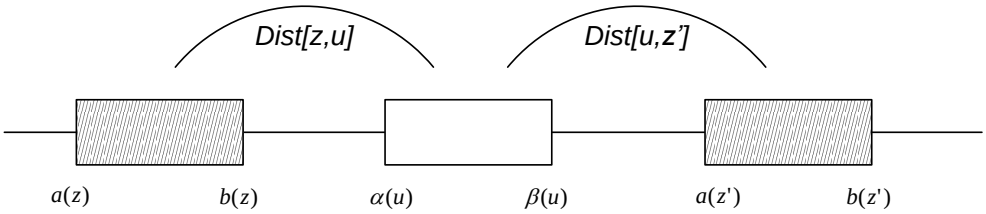

Figure 22 : Illustration pour $EC1$

  On peut alors définir $EC1$ par :

$$EC1 \leftarrow \begin{cases} a(z) + Dist[z,u] \leq \beta(u) \\ \alpha(u) + Dsit[u,z'] \leq b(z') \\ a(z) + Dist[z,u] + Dist[u,z'] \leq b(z') \\ c(\Gamma, z) + Q \leq CAP \\ c(\Gamma, z') + Q \leq CAP \end{cases}$$

- Le second test $Test\_Node2(\Gamma, z, z', o_i, d_i, c_i)$ vise à vérifier la faisabilité de l'insertion d'une demande D_i, $i \in I$ entre deux sommets consécutifs z et z' dans une tournée donnée Γ. Nous donnons un ensemble $EC2$ des conditions nécessaires pour tester la faisabilité de cette insertion :

  On pose :
  $\forall x \in \Gamma, [a(x), b(x)] = FR(\Gamma)(x)$;
  $\forall x \in \{o_i, d_i\}, [\alpha(x), \beta(x)] = F(x)$;

On peut alors définir $EC2$ par :

$$EC2 \leftarrow \begin{cases} a(z) + Dist[z, o_i] \leq \beta(o_i) \\ \alpha(o_i) + Dsit[o_i, d_i] \leq \beta(d_i) \\ a(d_i) + Dist[d_i, z'] \leq b(z') \\ a(z) + Dist[z, o_i] + Dist[o_i, d_i] \leq \beta(d_i) \\ a(z) + Dist[z, o_i] + Dist[o_i, d_i] + Dist[d_i, z'] \leq b(z') \\ a(o_i) + Dist[o_i, d_i] + Dist[d_i, z'] \leq b(z') \\ c(\Gamma, z') + c_i \leq CAP \end{cases}$$

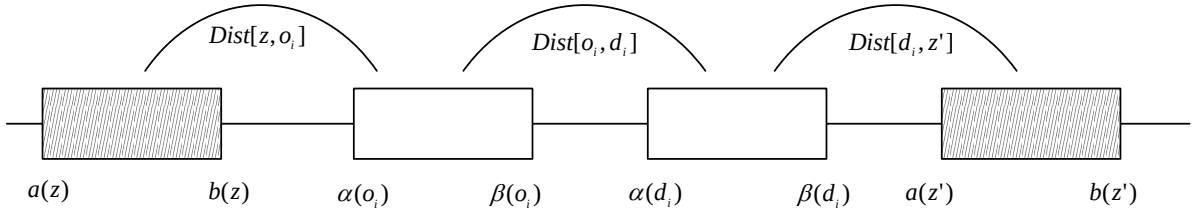

Figure 23 : Illustration pour $EC2$

Les tests préliminaires pour vérifier la « Temps-Admissibilité » d'une tournée obtenue par la procédure $INSERT(\Gamma, x, y, i)$ peuvent être réalisés grâce à la procédure suivante :

Algorithme 14 : Procédure Test_Insert

Entrées : Γ : une tournée ;
x, y : le sommet (position) ;
i : l'indice de la demande ;
Sorties : $Test$: booléen ;
Val : réel ;

1 **Si** $x \neq y$ **alors**
2 | $Test \leftarrow Test\_Node1(\Gamma, x, succ(\Gamma, x), o_i, c_i) \wedge Test\_Node1(\Gamma, y, succ(\Gamma, y), d_i, c_i)$;
3 **Sinon**
4 | $Test \leftarrow Test\_Node2(\Gamma, x, succ(\Gamma, x), o_i, d_i, c_i)$;
5 **Fin**
6 **Si** $Test = 1$ **alors** $Test \leftarrow Test\_Load(\Gamma, x, y, i)$;
7 **Si** $Test = 1$ **alors** $(Test, F1) \leftarrow Propage(\Gamma, \{o_i, d_i\})$;
8 **Si** $Test = 1$ **alors**
9 | $Val \leftarrow EVAL1(INSERT(\Gamma, x, y, i), F1).Val$;
10 **Sinon**
11 | $Val = INDEFINI$;
12 **Fin**
13 $Test\_Insert \leftarrow (Test, Val - Val1(\Gamma))$;

5.2. Le processus d'insertion

Cette procédure travaille sur l'ensemble D des demandes, le 4-uple $(X, Dist, K, CAP)$ que nous avons définie à la section précédente, et 3 coefficients $\gamma, \delta, \varepsilon \geq 0$. Elle fonctionne d'une manière gloutonne par l'insertion successive des différentes demandes D_i . Le point fondamental est que nous envisageons le cas de contraintes temporelles serrées pour les fenêtres de temps et la durée maximale de transport. Plusieurs règles de propagations de contrainte sont utilisées pour la conception de l'algorithme d'insertion. L'algorithme tient à jour les variables suivantes :

- L'ensemble $I_1 \subset I$ des demandes qui ont été déjà insérés dans une des tournées $T(k), k \in \{1, \dots, K\}$;
- L'ensemble des fenêtres de temps $FR(T(k))$ pour les tournées $T(k), k \in \{1, \dots, K\}$, ainsi que les valeurs de charge $c(T(k), x), x \in T(k)$ et les valeurs $VAL1(T(k))$ et $VAL2(T(k))$;
- Pour l'ensemble $FREE(i)$, $\forall i \in J = (I - I_1)$ qui contient des 4-uples (k, x, y, v) tels que $k \in \{1, \dots, K\}, x, y \in T(k)$, où v est une composante du résultat $(1, v)$ de la procédure $Test\_Insert(T(k), x, y, i)$. $N\_FREE(i)$ est le nombre de véhicules qui peut encore traiter la demande D_i.

Ensuite, l'algorithme d'insertion fonctionne selon le schéma suivant:

a. Tout d'abord, on choisit une demande D_{i_0} parmi les indices $i \in J$ des demandes plus contraintes, c'est-à-dire tels que $N\_FREE(i)$ et $Card\big(FREE(i)\big)$ soient petits. Plus précisément, s'il existe i tel que $N\_FREE(i) = 1$, alors i_0 est choisi d'une manière aléatoire parmi les indices de demande dans J tels que $N\_FREE(i) = 1$; sinon, nous choisissons i_0 aléatoirement parmi les deux plus petites valeurs de $N\_FREE(i)$ (**E4**)

b. L'algorithme choisit ensuite un 4-uple (k_0, x_0, y_0, v_0) dans $FREE(i_0)$ qui correspond simultanément à une des plus petites valeurs $(v = VAL2(T(k) - VAL1(T(k)))$, et à une des plus petites valeurs $EVAL2\big(INSERT(T(k), x, y, i_0)\big) - VAL2(T(k))$. Plus précisément, il établit d'abord une liste $L\_Candidate$ de 4-uple (k, x, y, v) dans $FREE(i_0)$ avec la meilleure (plus petite) valeur v , le nombre d'éléments de $L\_Candidate$ est noté $N_1 (N_1 \leq 5)$. La liste $L\_Candidate$ est triée selon les valeurs croissantes de v. Un 4-uple (k_0, x_0, y_0, v_0) est choisit au hasard parmi les N_2 $(N_2 \leq N_1)$ premiers 4-uples dans la liste $L\_Candidate$. Les nombres N_1 et N_2 sont donc deux paramètres de la procédure d'insertion. (**E5**)

c. La demande D_{i_0} est insérée dans la tournée $T(k_0)$ selon les sommets (positions) d'insertion x_0 et y_0.

d. $\forall i \in J$, nous cherchons un ensemble $\pi(i)$ des couples (x, y) tels qu'il existe une 4-uplet (k_0, x', y', v) dans $FREE(i)$, qui satisfait:
   a. $(x' = x) ou \big((x = x_0) et\ x' = prev(T(k_0), x)\big)$;
   b. $(y' = y) ou \big((y = y_0) et\ y' = prev(T(k_0), y)\big)$;

e. Enfin, la procédure $Test\_Insert$ est effectuée pour tout couple (x, y) dans $\pi(i)$, et met à jour $FREE(i)$ et $N\_FREE(i)$ en conséquence.

Algorithme 15 : Procédure INSERTION

Entrées : N_1, N_2: entier ;
Sorties : T: l'ensemble des tournées; t: l'ensemble des dates;
$Perf$: la performance des tournées; $Reject$:l'ensemble des demandes rejetées;

Pour $\forall k \in K$ **faire**
  $T(k) \leftarrow \{DepotD(k), DepotA(k)\}$;
  $t\big(DepotD(k)\big) = t\big(DepotA(k)\big) \leftarrow 0$;
Fin
$I_1 \leftarrow NULL; J \leftarrow I; Reject \leftarrow NULL$
Pour $\forall i \in J$ **faire**
  $N - FREE(i) \leftarrow K$;
  $FREE(i) \leftarrow Tous\ 4 - uple(k, x, y, v)\ les\ possibles$ où $k = 1 \dots K$, $x, y \in \{DepotD(k), DepotA(k)\}$, et $x \ll^{=}_{T(k)} y$, $v = EVAL2\big(T(k) \leftarrow \{DepotD(k), o_i, d_i, DepotA(k)\}\big).Val$;
Fin
Tant que $J \neq NULL$ **Faire**
  Choisir la demande i_0 dans J comme décrit par (E4) ;
  Supprimer i_0 dans J ;
  Si $FREE(i) = NULL$ **alors** $Reject \leftarrow Reject \cup \{i_0\}$;
  Sinon
    Générer $L\_Candidate$ en utilisant $FREE(i_0)$;
    Choisir (k_0, x_0, y_0, v_0) dans $L\_Candidate$ comme décrit par (E5) ;
    $T(k_0) \leftarrow INSERT(T(k_0), x_0, y_0, i_0)$;
    $t \leftarrow EVAL2\big(T(k_0)\big).t$; Insérer i_0 dans I_1 ;
    Pour $\forall i \in J$ **faire**
      Chercher l'ensemble $\pi(i)$;
      Pour tout couple (x, y) dans $\pi(i)$ **faire**
        $(Test, Val) \leftarrow Test\_Insert(T(k_0), x, y, i)$;
        Supprimer (k_0, x, y, v) dans $FREE(i)$;
        Mises à jour $N\_FREE(i)$ en conséquence ;
        Si $Test = 1$ **alors**
          Insérer (k_0, x, y, Val) dans $FREE(i)$ et Mettre à jour $N\_FREE(i)$;
        Fin
      Fin
    Fin
  Fin
Fin
$Perf \leftarrow Perf_{\gamma,\delta,\varepsilon}(T, t)$;
$INSERTION \leftarrow (T, t, Perf, Reject)$;

Les instructions a et b du schéma algorithmique qui sont représentés ci-dessus peuvent être écrites de manière non déterministe. L'algorithme $INSERTION$ (voir l'Algorithme 15) devient alors non déterministe et peut être utilisé dans le cadre d'un schéma de MONTE-CARLO (voir l'Algorithme 16).

Algorithme 16 : RANDOM-INSERTION

Entrées : N_1, N_2, P: entier ;
Sorties : T: l'ensemble des tournées; t: l'ensemble des dates;
$Perf$: la performance des tournées;
$Reject$:l'ensemble des demandes rejetées;

1 **Pour** p de 1 à P **faire**
2 | $INSERTION(N_1, N_2)$;
3 | Garder le meilleur résultat (la paire (T, t) tel que $|Reject|$ est le plus petit possible ;
4 **Fin**

5.3. Renforcement de la procédure d'insertion : manipuler des contraintes temporelles serrées

Nous remarquons que la procédure d'insertion peut partiellement échouer dans le traitement de l'ensemble des demandes, ce signifie que la composante $INSERTION(N_1, N_2).Reject$ du résultat de la procédure d'insertion peut être non vide. Cependant il est possible de faire en sorte que l'ensemble $Reject$ soit vide. Pour cela, plusieurs approches sont envisageables.

5.3.1. Etudes sur le choix de la demande D_{i_0} dans J (E4)

Le dispositif suivant peut être ajouté à l'algorithme de MONTE-CARLO RANDOM-INSERTION :

a. tout au long du processus $RANDOM\_INSERTION$, on utilise un tableau $Reject\_Tab$ pour mémoriser le nombre de rejet sur chaque demande ;

b. on utilise une liste L_Reject qui, à tout moment du processus, contient jusqu'à N_3 demandes qui sont les plus souvent rejetées (celles dont les valeurs dans $Reject\_Tab$ sont les plus élevés). Le nombre N_3 devient un paramètre du processus RANDOM_INSERTION.

La procédure $RANDOM\_INSERTION$ fonctionne alors en deux étapes:

- la demande i_0 est choisie parmi les demandes de L_Reject jusqu'à ce que L_Reject est vide ;
- la procédure $INSERTION$ fonctionne ensuite comme d'habitude ;

La procédure qui résulte de cette modification du choix de la demande D_{i_0} dans J (E4). Ensuite, la procédure RANDOM_INSERTION peut être reformulée comme suit:

Algorithme 17 : RANDOM_INSERTION_MEMOIRE

Entrées : N_1, N_2, N_3, P: entier ;
Sorties : T: l'ensemble des tournées; t: l'ensemble des dates;
$Perf$: la performance des tournées; $Reject$:l'ensemble des demandes rejetées;

1 $L\_Reject \leftarrow NULL$; $Reject\_Tab \leftarrow 0$;
2 **Pour** p de 1 à P **faire**
3 | $(T, t, Perf, Reject) \leftarrow INSERTION(L\_Reject, N_1, N_2, N_3)$;
4 | **Pour** $\forall i \in Reject$ **faire**
5 | | $Reject\_Tab[i] \leftarrow Reject\_Tab[i] + 1$;
6 | **Fin**
7 | $L_Reject \leftarrow$les N_3 indices des demandes qui ont les plus grands dans $Reject\_Tab$
8 | Garder le meilleur résultat (la paire (T, t) tel que $|Reject|$ est le plus petite possible, on obtient la meilleure valeur$Perf_{\delta,\varepsilon,\epsilon}(T, t)$) ;
9 **Fin**

5.4. Présentation du graphe non compatible dans le système de propagation de contraintes.

La procédure $INSERTION$ que nous avons décrite à la section précédente ne comporte pas les informations contenues dans le graphe non compatible**.** Donc, la procédure n'a pas de capacité à prévoir les échecs.

La procédure $COULEUR\_INSERTION$ que nous allons décrire maintenant, consiste en une mise à jour de la procédure d'insertion, qui applique la procédure $COULEUR$ de la section 4 à chaque itération de la boucle principale, afin de tester l'existence d'une coloration des sommets du sous graphe $G\_No\_Comp = (I, No\_Compatible)$ qui est induit par l'ensemble J des indices des demandes non insérées.

Plus précisément, la procédure $COULEUR\_INSERTION$ implique :

- une famille $\boldsymbol{Clause}$ des clauses décrites par la remarque 7 dans la section 4 ;
- une coloration des sommets du sous graphe $G\_No\_Comp = (J, No\_Compatible)$ induit par le sous ensemble des demandes non insérés $J \subset I$: cette coloration des sommets est exprimée par une séquence « chronologique » : $Couleur = \{(j_0, l_0), \dots, (j_m, l_m)\}$ ce qui signifie que, selon la couleur (le véhicule) k_s doit être attribuée à la demande D_{j_s}, $s = 0, \dots, m, j = \{j_0, \dots, j_m\}$, et que le processus $COULEUR$ a travaillé de sorte que, pour toute paire $(s, s'), avec\ s < s'$, la couleur l_s a

été attribuée à la demande D_{j_s} avant que la couleur $l_{s'}$ ne soit attribuée à la demande $D_{j_{s'}}$.

Une fois que l'indice i_0 de la demande a été choisi dans l'ensemble J des demandes non insérées, l'algorithme $COULEUR\_INSERTION$ choisit le véhicule k_0 s'il existe une coloration des sommets du sous graphe $G\_No\_Comp = (I, No\_Compatible)$ tel que $r(i_0) = k_0$. La vérification de l'existence d'une telle coloration des sommets est très coûteuse en temps de calcul. En effet les deux procédures conçues $COULEUR\_INSERTION$ et $COULOUR$ effectue une recherche arborescente, ce signifie qu'ils ont tendance à essayer tous les indices $i \in I$ des demandes, avec tous les véhicules $k \in K$. La famille $Clause$ est ensuite utilisée pour aider à couper l'arbre de recherche dans le processus de $COULEUR$ et éliminer les mauvais 4-uples dans $FREE(i)$, $i \in I$. La procédure $COULEUR\_INSERTION$ procède alors de manière itérative :

- Sélection de l'indice i_0 de la demande dans J: **(E6)**
  i_0 est choisi comme étant égal à j_0 comme (E4), sauf dans le cas où:
  - $N\_FREE(j_0) \geq 2$ et il existe i tel que $N\_FREE(i) = 1$: dans ce cas, la priorité est donnée à i tel que $N\_FREE(i) = 1$;
  - $N\_FREE(j_0) \geq 3$ et il existe i tel que $N\_FREE(i) \leq 2$: dans ce cas, la priorité est donnée à i tel que $N\_FREE(i) \leq 2$.

- Construction d'une liste $L\_Candidate$ de 4-uples (k, x, y, v) : **(E7)**
  - Si $j_0 \neq i_0$, nous construisons alors (comme dans (E5)) la liste $L\_Candidate$ avec N_1 4-uples (k, x, y, v) dans $FREE(i_0)$ avec la meilleure valeur v. Ensuite, pour tous les 4-uples, nous calculons la valeur $w = EVAL2\big(INSERT(T(K_0), x_0, y_0, i_0)\big).Val - VAL2(T(k))$ et enfin nous ordonnes la liste $L\_Candidate$ selon les valeurs de w croissantes ;
  - Si $j_0 = i_0$, alors nous construisons d'abord $L\_Candidate$ comme décrit ci-dessus, et nous définissons
    - $w_0 =$ la valeur v liée au premier 4-uple (k, x, y, w) de $L\_Candidate$;
    - $w'_0 =$ la plus petite valeur v liée au 4-uple (k', x, y, w) de $L\_Candidate$;

    Si $w'_0 - w_0 \leq V_1$ (où V_1est un seuil donné) alors on déplace le 4-uple lié w'_0 à la tête de la liste $L\_Candidate$; V_1 devient un paramètre de la procédure $COULEUR\_INSERTION$.

- Choisir (k_0, x_0, y_0, v_0) dans $FREE(i_0)$: **(E8)**
  - Nous appliquons récursivement la procédure $COULEUR$ pour l'ensemble J et pour l'ensemble Dom_Free : $\forall i \in J$, $Dom\_Free(i) = \{k = 1, \dots, K$ tel qu'il existe un 4-uple (k, x, y, v) dans $FREE(i)\}$;

- Nous adaptons la procédure $COULEUR$, de telle sorte que, chaque fois qu'un appel récursif est exécuté:
  - la valeur i_0 qui est impliquée dans le plus haut niveau d'appel récursif de la procédure $COULEUR$ (l'appel principale est au niveau 0) est obtenue par (E6);
  - si l'appel actuel récursif est lié à $H = \{j_s, \dots, j_m\}$, et si la coloration actuelle r est définie par la séquence $\{(j_0, k_0), \dots, (j_{s-1}, k_{s-1})\}$, la valeur i_0 est égale à une valeur d'indice i tel que la valeur $|Dom\_Free(i)|$ est minimale;
  - la liste $L \subset Dom(i_0)$ impliquée dans le plus haut niveau d'appel récursif de la procédure $COULEUR$ découle de la liste ci-dessus $L\_Candidate$ en réduisant tous les 4-uples (k, x, y, v) à sa couleur k;
  - le mécanisme de propagation de contraintes qui est utilisé dans le processus de $COULEUR$ (règles d'inférence, R_m^*, m = 1…4) est complété avec une autre règle d'inférence R_5^*:
    - R_5^*: il existe une clause $cl = \{(r(i_1) = l_1), \dots, (r(i_p) = l_p) \rightarrow Non(r(i_{p+1}) = l_{p+1}), \dots\}$ telle que les indices des demandes $i_1, \dots, i_p$ ont été respectivement associées à des couleurs $l_1, \dots, l_p$ ⊨la couleur l_{p+1} est retirée de l'ensemble $Dem(i_{p+1})$;

- La procédure $COULEUR$ est alors réécrite comme $COULEURBIS$ et prend paramètres : H : sous ensemble des indices des demandes non insérées, Dom : l'ensemble des couleurs (véhicules) disponibles, Col : la liste de couleurs affectées, Cl : la liste de clauses, L : la liste de couleurs : Res : Booléen, r : colorier, Cl: la liste des clauses, k: couleur) ;

- (k_0, x_0, y_0, v_0) le premier 4-uple (k_0, x, y, v) dans $L\_Candidate$ est sélectionné.

- Mise à jour de l'ensemble $FREE(i), i \in J$, de la famille Clause **(E9)**
  - utiliser les clauses dans ***Clause,*** qui réduisent à $(r(i_0) = k_0)$ afin de couper l'ensemble $FREE(i), i \in J$, supprimer ces clauses dans ***Clause***;
  - les clauses qui contiennent une affectation $r(i_0) = k$ avec $k \neq k_0$ sont retirées dans ***Clause***, pour tous les autres clause cl**,** supprimer $(r(i_0) = k_0)$ dans cl s'il existe.

Algorithme 18 : Procédure COULEUR_INSERTION

Entrées : N_1: entier ; V_1: réel ;
Sorties : T: l'ensemble des tournées; t: l'ensemble des dates;
$Perf$: la performance des tournées; $Reject$:l'ensemble des demandes rejetées;

Pour k de 1 à K **faire**
  $T(k) \leftarrow \{DepotD(k), DepotA(k)\}$;
  $t\big(DepotD(k)\big) = t\big(DepotA(k)\big) \leftarrow 0$;
Fin
$l_1 \leftarrow NULL; J \leftarrow I; Reject \leftarrow NULL$
Pour $i \in J$ **faire**
  $N\_FREE(i) \leftarrow K$;
  $FREE(i) \leftarrow Tout\ les\ 4-uple(k,x,y,v)\ possible$
    où $k = 1 \dots K,\ x, y \in \{DepotD(k), DepotA(k)\}$,
    $x \ll^{=}_{T(k)} y,\ v = EVAL2\big(T(k) \leftarrow \{DepotD(k), o_i, d_i, DepotA(k)\}\big).Val$
  $Dom\_Free(i) \leftarrow \{k = 1, \dots, K\ telle\ qu'il\ exsite\ un\ 4-uplet(k,x,y,v) dans\ Free(i)\}$
Fin
$(Res, Couleur, Clause) \leftarrow COULEUR(J, Dom\_Free)$;
Tant que $J \neq NULL$ **Faire**
  Choisir la demande i_0 dans J comme décrit par (E6) ;
  Supprimer i_0 dans J ;
  Si $FREE(i) = NULL$ **alors** $Reject \leftarrow Reject \cup \{i_0\}$
  Sinon
    Construire $L\_Candidate$ comme décrit par (E7) ;
    $(Res, Couleur, Clause, k_0) \leftarrow COULEURBIS(J, Dom\_Free, Couleur, Clause, L\_Candidate)$;
    Si $Res = 0$ **alors** $Reject \leftarrow Reject \cup \{i_0\}$;
    Sinon
      Choisir (k_0, x_0, y_0, v_0) dans $L\_Candidate$ comme décrit par (E8) ;
      $T(k_0) \leftarrow INSERT(T(k_0), x_0, y_0, i_0)$;
      $t \leftarrow EVAL2\big(T(k_0)\big).t$; Insérer i_0 dans l_1 ;
      Pour $\forall i \in J$ **faire**
        Chercher l'ensemble $\pi(i)$
        Pour tout couple (x, y) dans $\pi(i)$ **faire**
          $(Test, Val) \leftarrow Test\_Insert(T(k_0), x, y, i)$;
          Si $Test = 0$ **alors**
            Supprimer (k_0, x, y, v) dans $FREE(i)$
            Mettre à jour $N\_FREE(i)$ en conséquence.
          Sinon
            $v = Val$;
          Fin
        Fin
        Mettre à jour $FREE(i)$ comme décrit par (E9) ;
      Fin
      Mettre à jour $Clause$;
    Fin
  Fin
Fin
$Perf \leftarrow Perf_{\gamma,\delta,\varepsilon}(T, t)$;
$COULEUR\_INSERTION \leftarrow (T, t, Perf, Reject)$;

5.5. L'extension de la procédure insertion gloutonne dans un schéma d'arbre de recherche

Nous adaptons l'algorithme glouton $INSERTION$ qui permet de revenir en arrière au cas où l'ensemble $Reject$ n'est pas vide, plus précisément:

- l'instruction (E4) liée au choix de la demande D_{i_0} qui va être insérée par l'algorithme $INSERTION$ peut induire la création d'un nœud précédent (retour arrière, ou backtrack) si $N\_FREE(i_0) \leq 4$. La création effective d'un tel nœud est décidée selon un tri aléatoire : la probabilité P_1 de création, devient un paramètre du procédé TREE-INSERTION ;
- le « retour arrière » est exécuté si et seulement si le $Reject$ est non vide (cas d'échec) : dans ce cas, on essaie deux 4-uples (k, x, y, v) et (k', x', y', v') avec $k \neq k'$ dans la liste $L\_Candidate$ qui est construite comme dans (E5) ;
- le nombre des nœuds précédents est délimité par un nombre N_4, qui sera également un paramètre de la procédure $TREE\_INSERTION$;

Clairement, la procédure $INSERTION$ peut être intégrée dans la procédure $TREE\_INSERTION$, et les instances qui sont traitées avec succès par la procédure $INSERTION$ ($Reject = NULL$) n'impliquent aucune activation du processus « retour arrière ». De plus, $TREE\_INSERTION$ est une heuristique, on n'a donc aucune garantie de réussite. Puisque la procédure $TREE\_INSERTION$ est non déterministe et elle peut être utilisée dans un schéma de MONTE-CARLO, comme dans la section 5.

Algorithme 19 : Procédure TREE-INSERTION

1. **Entrées** : N_1, N_2, N_4: entier ; P_1: réel ;
2. **Sorties** : T: l'ensemble des tournées; t: l'ensemble des dates;
   a. $Perf$: la performance des tournées; $Reject$:l'ensemble des demandes rejetées;
3. **Pour** $k \in K$ **faire**
4. | $T(k) = \{DepotD(k), DepotA(k)\}$; $t\big(DepotD(k)\big) = t(DepotA(k)) \leftarrow 0$;
5. **Fin**
6. $I_1 \leftarrow NULL$; $J \leftarrow I$; $Reject \leftarrow NULL$; $Backtrack\_Weight \leftarrow 0$;
7. **Pour** $i \in J$ **faire**
8. | $N\_FREE(i) \leftarrow K$;
9. | $FREE(i) \leftarrow Tout\ les\ 4-uple(k, x, y, v)\ possible$
10. où $k = 1 \dots K, x, y \in \{DepotD(k), DepotA(k)\}$,
    a. $x \ll^{=}_{T(k)} y, v = EVAL2\big(T(k) \leftarrow \{DepotD(k), o_i, d_i, DepotA(k)\}\big).Val$
11. $Current\_Reject \leftarrow Indefini$; $Current\_T \leftarrow Indefini$; $Current\_t \leftarrow Indefini$;
12. $Current\_Pref \leftarrow Indefini$;
13. **Fin**
14. **Tant que** $Current\_Reject \neq NULL$ **faire**
15. | **Tant que** $J \neq NULL$ **faire**
16. | | Choisir la demande i_0 dans J comme decrit par (E4)
17. | | Supprime i_0 dans J
18. | | **Si** $FREE(i_0) = NULL$ **alors**
19. | | | $Current\_Reject \leftarrow Current\_Reject \cup \{i_0\}$

```
20. | | Sinon
21. | | | Si N_FREE(i_0) ≤ 4 alors
22. | | | | Sélectionner aléatoirement un nombre rand dans [0,1] ;
23. | | | | Si rand ≤ P_1 && Backtrack_Weight ≤ N_4 alors
24. | | | | | Backtrack_Status(i_0) ← 1;
25. | | | | | Backtrack_Weight ← Backtrack_Weight + N_FREE(i_0) − 1;
26. | | | | Sinon
27. | | | | | Backtrack_Status(i_0) ← 1
28. | | | | Fin
29. | | | Fin
30. | | | Calculer la liste L_Candidate des 4 − uple (k, x, y, v) dans FREE(i_0) comme (E5)
31. | | | Raccourcir la liste L_Candidate afin qu'une même valeur k ne soit pas impliquée deux fois dans la liste.
32. | | | Choisir (k_0, x_0, y_0, v_0) comme le premier élément de la liste
33. | | | Insérer D_{i_0} dans la tournée T(k_0), à la position x_0, y_0 non encore testée
34. | | | T(k_0) ← INSERT(T(k_0), x_0, y_0, i_0) ;
35. | | | t' = EVAL2(T(k_0)). t ; Insérer i_0 dans I_1 ;
36. | | | Pour tout x dans T(k_0) faire t(x) = t'(x) ;
37. | | | Pour i ∈ J faire
38. | | | | Chercher l'ensemble π(i)
39. | | | | Pour tout couple (x, y) dans π(i)faire
40. | | | | | (Test, Val) ← Test_Insert(T(k_0), x, y, i);
41. | | | | | Supprimer (k_0, x, y, v) dans FREE(i) ;
42. | | | | | Mettre à jour N_FREE(i) en conséquence.
43. | | | | | Si Test = 1 alors
44. | | | | | | Insérer (k_0, x, y, Val) dans FREE(i)
45. | | | | | | Mettre à jour N_FREE(i) en conséquence.
46. | | | | | Fin
47. | | | | Fin
48. | | | Fin
49. | | Fin
50. | Perf_aux ← Perf_{γ,δ,ε}(T, t) ;
51. | Si (|Reject| < |Current_Reject| ou ( |Reject| = |Current_Reject| et Perf_aux < Current_Pref))
52. | | Current_Reject ← Reject ; Current_Pref ← Perf_aux ; Current_T ← T ; Current_t ← t ;
53. | Fin
54. | INSERTION(Current_T, Current_t, Perf_aux, Current_Reject);
55. Fin
```

On peut intégrer $TREE\_INSERTION$ dans un schéma de MONTE-CARLO comme suit :

Algorithme 20 : RANDOM_TREE_INSERTION

Entrées : N_1, N_2, N_4, P: entier ; P: réel ;
Sorties : T: l'ensemble des tournées; t: l'ensemble des dates;
$Perf$: la performance des tournées; $Reject$:l'ensemble des demandes rejetées;

```
1  Pour p = 1, … , P faire
2  | Appliquer TREE_INSERTION(N_1, N_2, N_4, P_1)
3  | Garder le meilleur résultat (la paire (T, t) tel que |Reject| est le plus petit possible, et qui minimise
4  | Perf_{δ,ε,ϵ}(T, t)).
5  Fin
```

6. Expérimentations numériques

Nous utilisons les instances proposées par Cordeau et Laporte [**77**] proposent 20 instances comportant de 24 à 144 demandes (voir http://www.hec.ca/chairedistributique/data/darp/) .

6.1. Comparaison des trois heuristiques proposée

Le Tableau 11 donne le taux de réussie des trois heuristiques sur 100 exécutions. Les colonnes ont la signification suivante :

- Instance : nom de l'instance
- Basique : résultats pour l'heuristique RANDOM _INSERTION
- Mémoire : résultats pour l'heuristique RANDOM _INSERTION_MEMOIRE
- Arbre : résultats pour l'heuristique RANDOM_TREE_INSERTION
- $|D|$: nombre de demandes
- K : le nombre de véhicules disponibles
- CPU : temps cpu en secondes pour 100 exécutions
- Réussite : nombre d'exécution qui obtient une solution

Tableau 11 : Comparaison des trois heuristiques

			Basique		**Mémoire**		**Arbre**	
Instance	K	$\|D\|$	Réussite	CPU	Réussite	CPU	Réussite	CPU
pr01	3	24	100	0.01	100	0.29	100	0.42
pr02	5	48	99	0.99	100	1.08	100	1.29
pr03	7	72	93	2.14	100	2.01	100	2.52
pr04	9	96	99	4.58	100	3.75	100	4.24
pr05	11	120	95	7.1	100	6.25	100	6.79
pr06	13	144	98	10.11	100	7.97	100	9.41
pr07	4	36	93	0.02	100	0.60	100	0.71
pr08	6	72	37	2.39	100	4.62	100	293.23
pr09	8	108	1	5.53	51	33.44	30	441.02
pr10	10	144	0	12.07	1	76.02	25	25371
pr11	3	24	100	0.08	100	0.58	100	0.82
pr12	5	48	100	2.14	100	1.79	100	1.98
pr13	7	72	96	4.36	100	3.31	100	3.45
pr14	9	96	100	8.63	100	6.50	100	6.60
pr15	11	120	100	14.12	100	9.72	100	10.09
pr16	13	144	99	17.29	100	12.39	100	13.56
pr17	4	36	93	1.09	100	1.12	100	1.23
pr18	6	72	84	5.11	100	4.20	100	4.45
pr19	8	108	13	11.7	99	29.85	99	1215.92
pr20	10	144	0	17.69	90	71.18	45	1345.80
Moyenne			75	6.36	92.05	13.83	89.95	1436.73

Nous constatons que l'heuristique « basique » ne peut pas obtenir une solution à chaque exécution. Elle donne en moyenne 75% de réussite. L'heuristique « mémoire » augment le pourcentage de réussite jusqu'à 92.05%, mais elle augment aussi le temps de calcule (13.83 secondes en moyenne) par rapport à l'heuristique « basique ». L'heuristique a toujours des difficultés pour les instances pr09, pr10, pr19 et pr20. L'heuristique avec recherche arborescente a 89.95% de réussite, mais le temps de calcul en moyenne est élevé beaucoup. Elle est plus efficace pour l'instance pr10 (25% de réussite).

6.2. Comparaison avec les résultats de Cordeau et Laporte [77]

Nous comparons les résultats de l'heuristique « mémoire » à la recherche taboue de Cordeau et Laporte [**77**]. La comparaison entre deux méthodes est difficile, parce que Cordeau et Laporte minimisent les distances parcourues par les véhicules alors que notre heuristique minimise une combinaison des différentes durées. Pour effectuer la comparaison entre deux méthodes, nous mettrons $\gamma = 10, \delta = 0, \varepsilon = 1$ afin de donner plus de poids à la durée totale des tournées car minimiser les distances a pour effet de minimiser également la durée totale des tournées.

Tableau 12 : Comparaison entre l'heuristique « mémoire » et la recherche taboue

	Heuristique « mémoire »				**La recherche taboue** [77]			
Instance	T_{Global}	T_{Ride}	T_{Wait}	Distance	T_{Global}	T_{Ride}	T_{Wait}	Distance
pr01	840,3	653,6	60,4	299,9	881,2	1095,0	211,2	190,0
pr02	1622,1	1860,2	138,3	523,8	1985,9	1976,7	723,9	302,1
pr03	2593,0	2073,0	135,5	1017,5	2579,4	3586,7	607,3	532,1
pr04	3414,3	2872,9	241,5	1252,9	3583,2	5021,3	1090,4	572,8
pr05	3824,3	3900,3	115,6	1308,8	3870,0	6156,5	833,0	637,0
pr06	4632,0	4369,2	142,5	1609,6	5056,8	7273,5	1375,4	801,4
pr07	1221,6	1054,5	30,3	471,3	1452,1	1508,9	440,3	291,7
pr08	2477,9	2236,9	85,3	952,6	2345,2	3691,5	410,3	494,9
pr09	3470,2	3065,6	25,8	1284,4	3155,5	5621,8	323,1	672,4
pr10	4673,0	5152,7	118,8	1674,1	4480,1	7163,7	721,3	878,8
pr11	783,3	664,7	10,9	292,4	965,1	1041,5	320,6	164,5
pr12	1480,9	1351,8	25,3	495,6	1564,7	2393,8	308,7	296,1
pr13	2367,4	2110,6	1,5	925,9	2263,7	3788,8	330,4	493,3
pr14	2995,9	2598,6	18,0	1057,9	2882,2	4632,8	426,3	535,9
pr15	3573,1	3142,8	8,1	1164,9	3595,6	6104,7	605,9	589,7
pr16	4321,7	4116,7	8,7	1424,0	4072,5	7347,4	448,9	743,6
pr17	1229,8	1007,4	58,6	451,2	1097,3	1762,0	129,0	248,2
pr18	2452,0	2032,7	52,5	959,5	2446,5	3659,0	523,4	462,7
pr19	3574,1	3335,0	85,3	1328,8	3249,3	5581,0	487,3	602,0
pr20	4659,0	3839,4	118,2	1660,8	4041,0	7072,3	362,4	798,6
Moyenne	2810,3	2571,9	74,1	1007,8	2778,4	4323,9	534,0	515,4

Le Tableau 12 montre la comparaison entre deux méthodes, Les colonnes ont la signification suivante :

- Instance : nom de l'instance ;
- $T_{Globale}$: la durée totale des tournées ;
- T_{Ride} : la durée totale de transport de demande ;
- T_{Wait} : le temps total d'attente pour les véhicules ;
- Distance : distance parcourue par les véhicules.

Les temps de calcul ne sont pas donnés dans le Tableau 12, car Cordeau et Laporte [**77**] donne les résultats obtenus avec 10^5 itérations mais les temps de calcul sont donnes pour 10^4 itérations. Voir le Tableau 12, nous comparons d'abord les résultats entre notre heuristique et la recherche taboue [**77**]. La durée totale des tournées sont comparables entre deux heuristiques. Le temps de transport des demandes (T_{Ride}) obtenu par notre heuristique est deux fois plus faible en moyenne que l'heuristique de [**77**]. Le temps totale d'attente (T_{Wait}) est sept fois plus faible en moyenne que l'heuristique de [**77**]. La distance obtenue par notre heuristique est deux fois plus longe en moyenne que l'heuristique de [**77**].

6.3. Comparaison avec les résultats de Jorgensen et al. [127]

L'algorithme génétique proposé par Jorgensen et al. [**127**] utilise une combinaison linéaire des critères comme la fonction objectif telle que :

$$f(s) = \omega_1 c_1(s) + \omega_2 c_2(s) + \omega_3 c_3(s) + \omega_4 c_4(s) + \omega_5 c_5(s) + \omega_6 c_6(s) + \omega_7 c_7(s)$$

où $c_1(s)$ est la durée totale des tournées pour les clients (sans le temps d'attente du véhicule), $c_2(s)$ est le temps total de transport des clients, $c_3(s)$ est le temps d'attente total pour les clients, $c_4(s)$ est le temps total de tournées pour les véhicules (avec le temps d'attente), $c_5(s)$, $c_6(s)$ et $c_7(s)$ sont les violations sur les fenêtres de temps, le temps de transport des clients et la durée totale de tournée. Les poids sont définis : $\omega_1 = 8, \omega_2 = 3, \omega_3 = 1, \omega_4 = 1, \omega_5 = n, \omega_6 = \mathrm{n}, \omega_7 = n$ où n est le nombre de clients. Pour effectuer la comparaison entre notre heuristique et l'algorithme génétique, nous mettrons les coefficients $\gamma = 1, \delta = 8, \varepsilon = 1$. [**127**] ne donne que des résultats pour 13 instances parmi les 20. La moyenne est calculée uniquement en utilisant ces 13 instances.

Nous comparons les résultats de l'heuristique « mémoire » à l'algorithme génétique [**127**] (voir le Tableau 13). En moyenne sur les 13 instances, nous trouvons que l'heuristique « mémoire » est plus efficace que [**127**]: le temps de transport T_{Ride} obtenu par notre heuristique est 72.7% plus faible que [**127**] et le temps d'attente T_{Wait} obtenu par notre heuristique est 65.5% plus faible que [**127**]. La durée totale T_{Global} de tournées et la distances totale de tournées sont comparables entre deux heuristiques. Pour les instances pr02, pr10 et pr16, les trois critères et la distance [**127**] obtenues par l'heuristique « mémoire » sont

meilleurs que [**127**]. Pour les 12 instances (sauf pr20), les trois critères obtenues par l'heuristique « mémoire » sont meilleurs que [**127**].

Tableau 13 : Comparaison entre l'heuristique « mémoire » et l'algorithme génétique

	Heuristique « mémoire »					l'algorithme génétique [**127**]				
Instance	CPU	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU	T_{Global}	T_{Ride}	T_{Wait}	Distance
pr01	0,3	922,4	301,6	128,7	313,7	334,2	1041	477	252	309
pr02	1,3	1870,3	749,7	392,5	517,8	685,8	1969	1367	470	539
pr03	2,8	2719,6	860,2	205,0	1074,6	3094,8	2779	3081	292	1047
pr04	5,3	3300,1	1112,2	165,4	1214,7	-	-	-	-	-
pr05	8,6	4009,1	1106,6	253,1	1356,0	3493,8	4250	5099	500	1350
pr06	12,0	4941,3	1550,6	200,8	1860,5	-	-	-	-	-
pr07	0,7	1274,1	465,5	85,4	468,7	-	-	-	-	-
pr08	9,0	2462,9	1171,7	99,5	923,4	-	-	-	-	-
pr09	51,6	3588,4	2758,0	40,8	1387,6	2446,8	3597	6251	94	1343
pr10	105,4	4633,5	3162,4	57,7	1695,8	3958,8	5006	8413	315	1811
pr11	0,7	845,2	234,6	66,4	298,8	327,6	907	630	143	284
pr12	2,4	1567,8	323,9	17,6	590,2	703,2	1719	1214	198	561
pr13	4,8	2596,2	685,0	18,5	1137,7	-	-	-	-	-
pr14	9,0	3151,2	913,1	11,4	1219,8	-	-	-	-	-
pr15	13,2	3812,4	940,9	41,3	1371,1	3535,8	4296	4615	552	1344
pr16	16,9	4689,6	1097,5	22,2	1787,4	4873,8	5309	6134	630	1799
pr17	1,5	1235,0	343,9	22,3	492,7	497,4	1299	990	102	478
pr18	5,2	2473,5	860,1	54,4	979,1	-	-	-	-	-
pr19	31,8	3682,0	1192,4	84,8	1437,2	2679,6	3679	5362	147	1372
pr20	133,8	4773,5	1861,4	50,5	1843,0	3984,6	4733	7969	113	1740
Moyenne sur 13 instances	28,5	2949,9	1148,7	106,4	1089,7	2355.09	3121,85	3969,38	292,92	1075,15
Moyenne sur 20 instances	20,8	2927,4	1084,6	100,9	1098.5					

6.4. Comparaison avec les résultats de autres heuristiques

Nous comparons les résultats de l'heuristique « mémoire » à l'heuristique avec la procédure de réparation qui est présenté dans le chapitre 2 (voir le Tableau 15). Nous utilisons la même fonction objectif et nous mettrons les coefficients $\gamma = 1, \delta = 8, \varepsilon = 1$. L'heuristique « Insertion_Parteille_RF » donne des temps d'exécution. L'heuristique « mémoire » donne des temps de transport et temps d'attente plus faibles. L'heuristique « Insertion_Exhaustive_RF » donne des durées des tournées et des distances totales plus faibles.

Nous ne comparons que les résultats de l'heuristique « Insertion_Partielle_RF » à l'heuristique « mémoire». Pour les 11 instances sur 20 (pr01, pr03, pr04, pr05, pr06, pr07, pr10, pr14, pr15, pr16 et pr19), les trois critères obtenus par l'heuristique « mémoire» sont meilleures que ceux obtenus par l'heuristique « Insertion_Partielle_RF ». L'heuristique

« mémoire» donne aussi les distances totales plus faibles pour 9 instances (pr01, pr03, pr04, pr07, pr10, pr14, pr15, pr16, pr19). L'heuristique « Insertion_Partielle_RF » donne les temps d'exécution plus faibles pour les 20 instances.

Nous comparons également les résultats de l'heuristique « Insertion_Exhaustive_RF » à l'heuristique « mémoire». Pour les 10 instances sur 20 (pr03, pr04, pr05, pr08, pr11, pr12, pr13, pr14, pr17, pr18), l'heuristique « Insertion_Exhaustive_RF » donne les trois critères plus faibles et elle donne aussi les distances totales plus faibles pour ces 10 instances. L'heuristique « Insertion_Exhaustive_RF » donne des temps d'exécution plus faibles pour les 20 instances.

6.5. Comparaison entre l'heuristique « mémoire » et l'heuristique « COULEUR_INSERTION »

Les 20 instances sont aussi testées par l'heuristique « COULEUR_INSERTION » en utilisant les coefficients $\gamma = 1, \delta = 8, \varepsilon = 1$. Il n'y a que 15 instances prouvent être résolues par cette heuristique (voir le Tableau 14). Comme prévu, la vérification de l'existence d'une coloration des sommets est très coûteuse en temps de calcul. En moyenne sur les 15 instances, l'heuristique « mémoire» est 12 fois plus rapide que l'heuristique « COULEUR_INSERTION » en temps d'exécution. L'heuristique « mémoire» donne la durée totale de tournées et le temps d'attente plus faibles. L'heuristique « COULEUR_INSERTION » donne des temps de transport plus faibles.

Tableau 14 : Comparaison avec l'heuristique « COULEUR_INSERTION »

	Heuristique « COULEUR_INSERTION »				**Heuristique « mémoire »**			
Instance	CPU	T_{Global}	T_{Ride}	T_{Wait}	CPU	T_{Global}	T_{Ride}	T_{Wait}
pr01	4,1	1092,6	264,9	265,0	0,3	922,4	301,6	128,7
pr02	15,5	2053,4	785,7	502,4	1,3	1870,3	749,7	392,5
pr03	37,5	2897,8	729,3	278,2	2,8	2719,6	860,2	205,0
pr04	69,5	3741,2	989,3	477,2	5,3	3300,1	1112,2	165,4
pr05	116,6	4112,5	1074,6	237,7	8,6	4009,1	1106,6	253,1
pr06	178,7	5363,5	1224,6	494,1	12,0	4941,3	1550,6	200,8
pr07	8,8	1458,9	490,7	198,4	0,7	1274,1	465,5	85,4
pr11	4,8	886,5	174,1	77,8	0,7	845,2	234,6	66,4
pr12	17,0	1623,0	334,8	42,5	2,4	1567,8	323,9	17,6
pr13	38,1	2580,6	596,2	0,0	4,8	2596,2	685,0	18,5
pr14	74,1	3376,7	793,9	89,0	9,0	3151,2	913,1	11,4
pr15	120,8	4265,8	819,5	318,2	13,2	3812,4	940,9	41,3
pr16	194,6	5149,6	1035,9	279,6	16,9	4689,6	1097,5	22,2
pr17	9,9	1280,4	256,4	20,0	1,5	1235,0	343,9	22,3
pr18	39,8	2582,1	660,4	53,4	5,2	2473,5	860,1	54,4
Moyenne	62,0	2831,0	682,0	222,2	5,6	2627,2	769,7	112,3

Tableau 15 : Comparaison entre trois heuristiques

	Insertion_Partielle_RF					**Insertion_Exhaustive_RF**					**L'heuristique « mémoire »**				
Instance	CPU	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU	T_{Global}	T_{Ride}	T_{Wait}	Distance	CPU	T_{Global}	T_{Ride}	T_{Wait}	Distance
pr01	0,02	1111,61	311,58	300,54	331,07	0,05	1117,37	182,49	306,38	330,99	0,3	922,4	301,6	128,7	313,7
pr02	0,09	2094,32	604,80	563,31	571,01	0,20	2054,95	533,25	569,32	525,63	1,3	1870,3	749,7	392,5	517,8
pr03	0,22	2899,40	1019,35	347,46	1111,94	0,37	2590,63	606,21	78,22	1072,41	2,8	2719,6	860,2	205,0	1074,6
pr04	0,25	3779,16	1207,01	579,95	1279,21	0,86	3208,49	848,90	132,64	1155,85	5,3	3300,1	1112,2	165,4	1214,7
pr05	0,44	4357,32	1843,99	607,60	1349,72	1,37	3912,61	1028,51	235,12	1277,49	8,6	4009,1	1106,6	253,1	1356,0
pr06	0,47	5475,87	1871,02	776,85	1819,02	1,86	5084,87	1314,69	510,07	1694,80	12,0	4941,3	1550,6	200,8	1860,5
pr07	0,03	1527,94	502,20	285,15	522,79	0,09	1396,14	327,22	217,59	458,55	0,7	1274,1	465,5	85,4	468,7
pr08	0,50	2498,16	1696,82	75,55	982,61	0,50	2375,56	947,73	40,28	895,28	9,0	2462,9	1171,7	99,5	923,4
pr09	18,70	3494,39	4750,50	10,31	1324,08	14,95	3534,33	4726,10	39,14	1335,19	51,6	3588,4	2758,0	40,8	1387,6
pr10	53,71	4695,78	7700,32	100,06	1715,72	27,13	4740,42	7563,79	127,79	1732,63	105,4	4633,5	3162,4	57,7	1695,8
pr11	0,09	1097,06	214,54	308,17	308,89	0,09	767,48	151,52	0,00	287,48	0,7	845,2	234,6	66,4	298,8
pr12	0,06	1794,98	268,68	275,45	559,53	0,45	1495,36	268,68	0,00	535,36	2,4	1567,8	323,9	17,6	590,2
pr13	0,13	2602,05	662,59	77,99	1084,06	0,98	2426,01	574,93	0,00	986,01	4,8	2596,2	685,0	18,5	1137,7
pr14	0,45	3505,50	1090,94	341,14	1244,36	2,26	3039,89	652,10	0,00	1119,89	9,0	3151,2	913,1	11,4	1219,8
pr15	0,52	4454,16	984,18	683,44	1370,72	3,81	3740,23	791,24	83,13	1257,10	13,2	3812,4	940,9	41,3	1371,1
pr16	1,30	5317,03	1867,00	613,73	1823,30	4,45	4496,59	977,94	46,40	1570,19	16,9	4689,6	1097,5	22,2	1787,4
pr17	0,13	1400,95	250,53	147,10	533,85	0,30	1173,16	250,53	0,00	453,16	1,5	1235,0	343,9	22,3	492,7
pr18	0,11	2639,21	629,59	179,07	1020,14	0,97	2395,78	644,79	38,18	917,60	5,2	2473,5	860,1	54,4	979,1
pr19	4,87	3760,65	1573,48	127,26	1473,39	3,90	3675,92	1416,68	145,59	1370,33	31,8	3682,0	1192,4	84,8	1437,2
pr20	37,57	4733,23	3243,27	37,79	1815,44	56,43	4674,93	2727,06	1,77	1793,16	133,8	4773,5	1861,4	50,5	1843,0
Moyenne	5,98	3161,94	1614,62	321,90	1112,04	6,05	2895,04	1326,72	128,58	1038,46	20,8	2927,4	1084,6	100,9	1098.5

7. Conclusion

Dans ce chapitre, nous proposons des heuristiques basées sur la technique d'insertion et la propagation de contraint pour le problème de Dial-A-ride (DARP). Le but de ces méthodes est de fournir rapidement des solutions du DARP. Les heuristiques peuvent être utilisées dans le cadre d'un schéma de MONTE-CARLO

Les instances proposées par Cordeau et Laporte [**77**] sont été testées par nos heuristiques. Les expérimentations montrent que nos heuristiques sont capables de donner des résultats de bonne qualité, que nos heuristiques proposent des solutions qui sont plus favorable aux clients (meilleure qualité de service).

CHAPITRE 4
TRANSFORMATION LOCALE

1. Introduction

Nous avons présenté dans les chapitres précédents des procédés de constructions gloutonnes. Nous allons maintenant proposer des mécanismes de transformation locale. L'objet de base de la transformation locale est de former un couple (T, t) où :

- $T = \{T(k) | k \in \{1, \dots, K\}\}$ est une famille de tournées admissibles, telle que les ensembles $X(T(k)), k \in \{1, \dots, K\}$, de sommets couverts par les tournées $T(k)$, $k \in \{1, \dots, K\}$, constituent une partition de l'ensemble des sommets X ;
- t est un ensemble de dates, qui fait de chaque tournée $T(k)$, $k \in \{1, \dots, K\}$, une tournée temps-admissible : $\forall x \in X$, $t(x)$ représente la date de début de service au sommet x.

On doit, afin de pouvoir parler de transformation locale, définir des opérations qui agissent sur ce couple (T, t) et le transforment. Le chapitre commence par une présentation des procédures SPLIT classiques adaptées au DARP. Nous proposons aussi une extension de la procédure SPLIT. Les tournées obtenues par les procédures SPLIT vont être améliorées par les opérateurs (déplacement, échange) de recherche locale. La Figure 24 présente le schéma général de transformation locale.

Nous rappelons d'abord ici les notations utilisées dans la suite de ce chapitre : l'ensemble des véhicules disponibles est noté $\{1, \dots, K\}$; leur capacité commune est égal à CAP; l'ensemble d'indices des demandes est noté I ; chaque demande $D_i, i \in I$ est définie par un 6-uple $\{o_i, d_i, c_i, F_{o_i}, F_{d_i}, \Delta_i\}$, où

- o_i est l'origine de la demande D_i, la demande doit être chargée dans la fenêtre de temps $F(o_i) = [\alpha(o_i), \beta(o_i)]$;

- d_i est la destination de la demande D_i, la demande doit être déchargée dans la fenêtre de temps $F(d_i) = [\alpha(d_i), \beta(d_i)]$;
- c_i est la charge de la demande D_i;
- Δ_i est la durée maximale de transport pour la demande D_i.

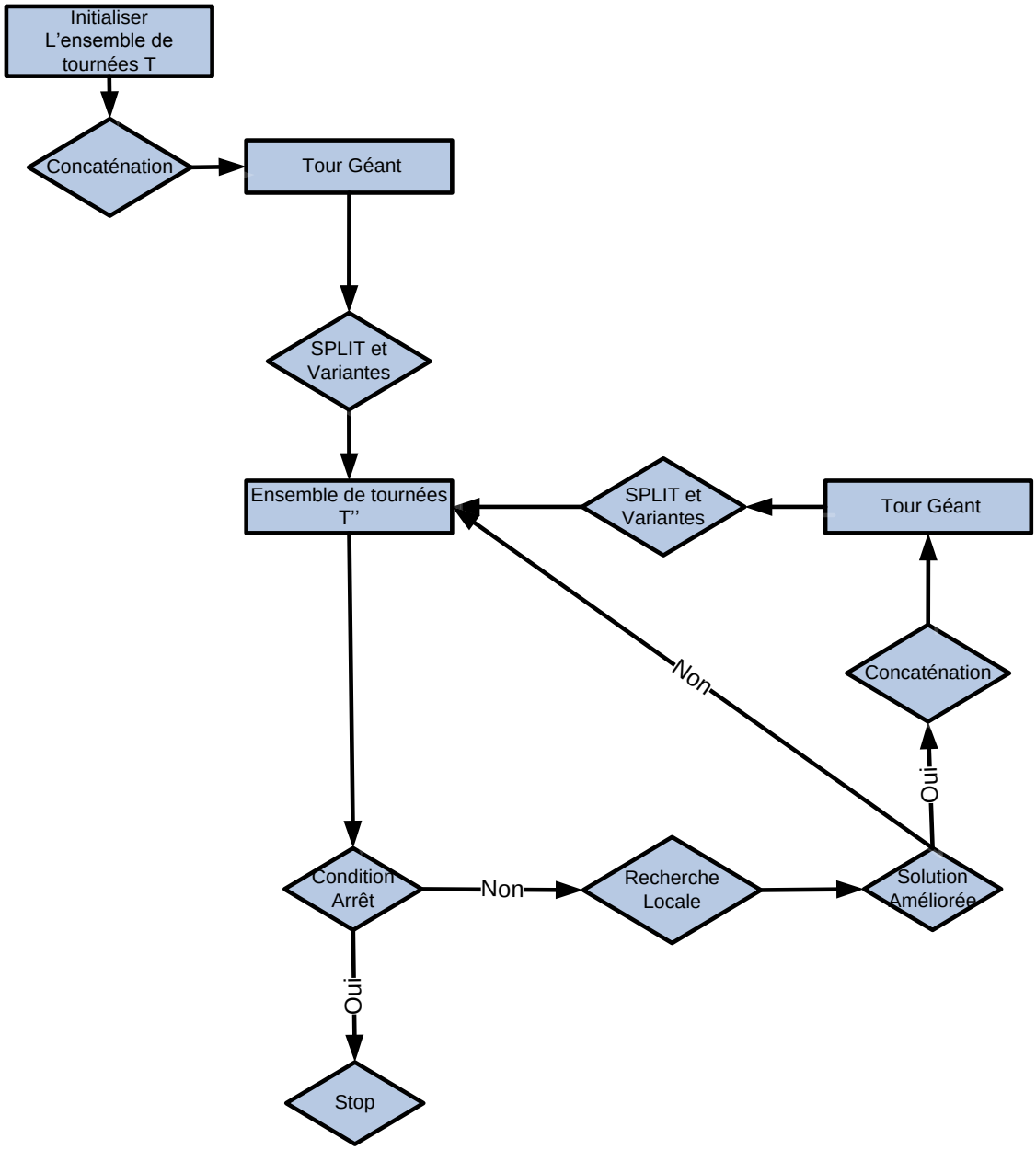

Figure 24 : Schéma général de transformation locale

L'ensemble des sommets sur lesquels sont construites les tournées est noté X ; Pour chaque $x \in X$, nous disposons :

- de son type $Type(x)$, qui peut être $DepotD, DepotA, Origine, Destination$ où $DepotD$ est le dépôt de départ, $DepotA$ est le dépôt d'arrivée;

- d'une quantité $c(x)$ qui est
  - Si $Type(x) = DepotD$ ou $DepotA$, alors $c(x) = 0$;
  - Si $Type(x) = Origine$, alors $c(x) = c_i$;
  - Si $Type(x) = Destination$, alors $c(x) = -c_i$;
- d'un homologue $Hom(x)$ qui est :
  - Si $x = o_i$, alors $Hom(x) = o_i$;
  - Si $x = d_i$, alors $Hom(x) = d_i$;
  - Si $x = DepotD(k)$, alors $Hom(x) = DepotA(k)$;
  - Si $x = DepotA(k)$, alors $Hom(x) = DepotD(k)$;
- d'une fenêtre du temps $F(x) = [\alpha(x), \beta(x)]$.

Une tournée Γ est une liste de sommets, qui débute en un sommet de type $DepotD$ et finit en un sommet de type $DepotA$, associés au même véhicule $k \in \{1, \dots, K\}$.

- pour tout sommet $x \in X$ de type $DepotD$ ou $Origine$, qui est dans Γ, son homologue $Hom(x)$ est aussi dans Γ, et situé après x dans Γ et réciproquement;
- un sommet figure au plus une fois dans Γ;
- Γ ne contient aucun sommet de type $DepotD$ ou $DepotA$, hormis les sommets $Début(\Gamma)$ et $Fin(\Gamma)$;
- une tournée Γ est « **charge-admissible** » si pour tout x dans Γ, on a :

$$\sum_{y\ avant\ x\ dans\ \Gamma} c(y) \leq CAP$$

- le couple formé de la tournée Γ et du vecteur $t = t(x),\ x \in \Gamma$ est « **temps-admissible** » si :
  - Pour tout $x \in \Gamma$, $t(x) \in F(x)$: fenêtre du temps autorisé à x;
  - Pour tout $x \in \Gamma$, de type $Origine$ ou $DepotD$, on a : $t\big(Hom(x)\big) - t(x)) \leq \Delta(x)$;
  - Pour tout $x \in \Gamma$,$Type(x) \neq DepotA$, on a : $t\big(succ(\Gamma, x)\big) - t(x) \geq Dist\big(x, succ(\Gamma, x)\big)$.

Une solution réalisable du problème du DARP est alors formée d'une famille $T(1), \dots, T(K)$ des tournées « Charge-Admissibles » et « Temps-Admissibles », qui partitionne l'ensemble de sommets X, et d'un vecteur date $t = t(x),\ x \in X$, associé. On suppose donc maintenant que l'on dispose d'une telle solution réalisable (T, t).

2. Procédure SPLIT

La première procédure, nommée **SPLIT**, est un algorithme de découpage. Le principe de SPLIT est de faire un découpage optimal d'un tour géant (tour qui contient tous les sommets de X) pour obtenir les tournées admissibles. L'idée de SPLIT a été proposée par Beasley

[**128**] en 1983 sous la forme de la méthode « *route-first and cluster-second* », mais il n'a pas donné de résultats numériques. Prins [**129**] a utilisé cette idée pour mettre au point un algorithme génétique pour le VRP. Cette procédure a été aussi appliquée avec succès pour le problème de tournées de véhicules sur arcs (CARP : Arc Routing Problems) par Lacomme et al. [**130**] [**131**] [**132**], et pour le LRP (Capacitated Location Routing Problem) par Lacomme et al. [**133**].

2.1. Tour géant

Les tournées $T(1), \dots, T(K)$ sont tout d'abord concaténées en un tour géant T_G (les sommets de type $Depot$A et $Depot$D ne sont pris en compte, voir la Figure 25). On cherche ensuite à découper « au mieux » ce tour T_G en K segments $S(1), \dots, S(K)$, tels que :

- pour chaque $k \in \{1, \dots, K\}$, la séquence $\{DepotD(k), S(k), DepotA(k)\}$ est une tournée admissible;
- l'ensemble des tournées $T' = \{DepotD(k), S(k), DepotA(k) | \, k \in \{1, \dots, K\}\}$ peut être daté par un vecteur date t' de tels sorte que (T', t') soit meilleur que (T, t) ;

On suppose toujours que la performance $PERF$ de référence, soit telle que : $PERF = \sum_{k=1,..,K} Cout\big(T(k)\big)$ où

$$Cout\big(T(k)\big) = \gamma \times T_{Global}\big(T(k)\big) + \delta \times T_{Ride}\big(T(k)\big) + \varepsilon \times T_{Wait}\big(T(k)\big)$$

γ, δ et ε sont coefficients.

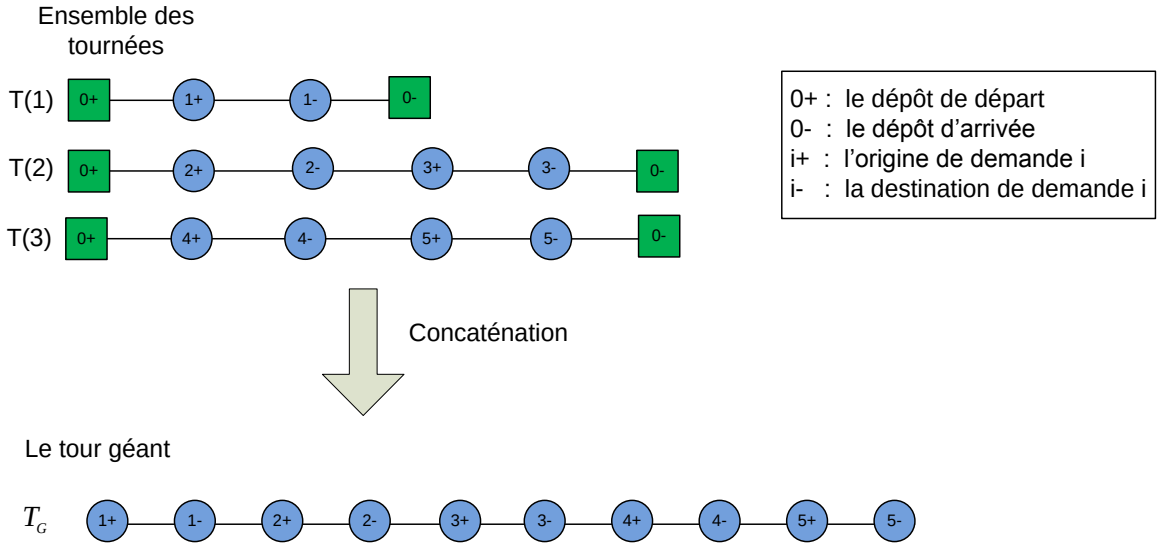

Figure 25 : Construction du tour géant

2.2. Le graphe auxiliaire

Afin de découper au mieux le tour géant la procédure SPLIT utilise un graphe auxiliaire. Le problème à résoudre est alors un problème de recherche de chemin (le plus court chemin) dans un graphe sans circuit. Pour construire ce graphe auxiliaire, nous posons la définition suivante:

➢ **Définition 1 : Point de découpage :**
Un sommet $x \in T_G$ est un point de découpage si $\sum_{y \ll_{T_G}^{=} x} c(y) = 0$, et si pour tout y avant ou égal à x dans T_G, on a $Hom(y)$ également avant ou égal à x dans T_G. On note alors ω_G est un ensemble de points de découpage dans T_G. Pour toute paire de sommets x, y et $x, y \in \omega_G, x \ll_{T_G} y$, la séquence $S = \{DepotA, z, DepotD | x \ll_{T_G} z \ll_{T_G}^{=} y\}$ représente une tournée. Dans la Figure 26, nous trouvons 5 points de découpage.

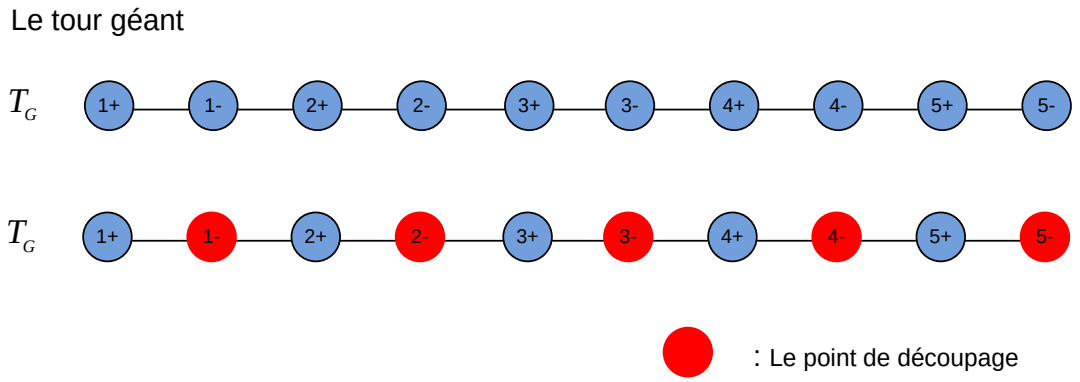

Figure 26 : Les points de découpage

2.2.1. Description du graphe auxiliaire

Pour la construction du graphe auxiliaire, nous n'utilisons que l'ensemble de points de découpage $\omega'_G = \omega_G - \{Fin(T_G)\}$. Le graphe auxiliaire $H = (\Psi, A, Z)$ est défini comme suit :

- L'ensemble Ψ des sommets du graphe H : $\{debut, x_k, fin \mid x \in \omega'_G, k \in \{1, \dots, K-1\}\}$ où les sommets $debut, fin$ sont des sommets fictifs, et x_k signifie que le sommet x est visité par le véhicule k (voir la Figure 27);
- L'ensemble A des arcs du graphe H est défini en posant d'abord pour tout couple x, y dans ω'_G, tels que $x \ll_{T_G} y$: $S_{x,y} = \{z \in T_G \ tq \ x \ll_{T_G} z \ll_{T_G}^{=} y\}$ (avec $\ll_{T_G}$: avant ; $\ll_{T_G}^{=}$: avant ou égal). Si $x = Null$, alors $S_{x,y}$ désigne l'ensemble de sommets z tels que $z \ll_{T_G}^{=} y$; si $y = Null$, alors $S_{x,y}$ désigne l'ensemble de sommets z tels que $x \ll_{T_G}^{=} z$; On définit alors A par :
  - $(debut, x_1), x \in \omega_G$ tel que $\Gamma_{Null,x}$ est une tournée admissible, cette tournée peut être notée $T(1)$;

- $(x_{K-1}, fin), x \in \omega_G$ tel que $\Gamma_{x,Null}$ est une tournée admissible, cette tournée peut être notée $T(K)$;
- $(x_k, y_{k+1}), k \in \{1, \dots, K-2\}, x, y \in \omega_G$, tel que $\Gamma_{x,y}$ est une tournée admissible, cette tournée peut être notée $T(k')$ et $k' \in \{2, \dots, K-1\}$.

- On associe à chaque arc de A sa longueur de façon naturelle : chaque arc de A est défini à partir d'une tournée admissible $\Gamma_{x,y}$, $x, y \in \{Null\} \cup \omega_G$ et la longueur de cet arc est alors la meilleure mesure de performance $PERF$ que l'on peut associer à $\Gamma_{x,y}$: $l_{x,y} = Inf\left(PERF\left(\Gamma_{x,y}, t\right)\right)$, t est un ensemble de date associé à $\Gamma_{x,y}$.

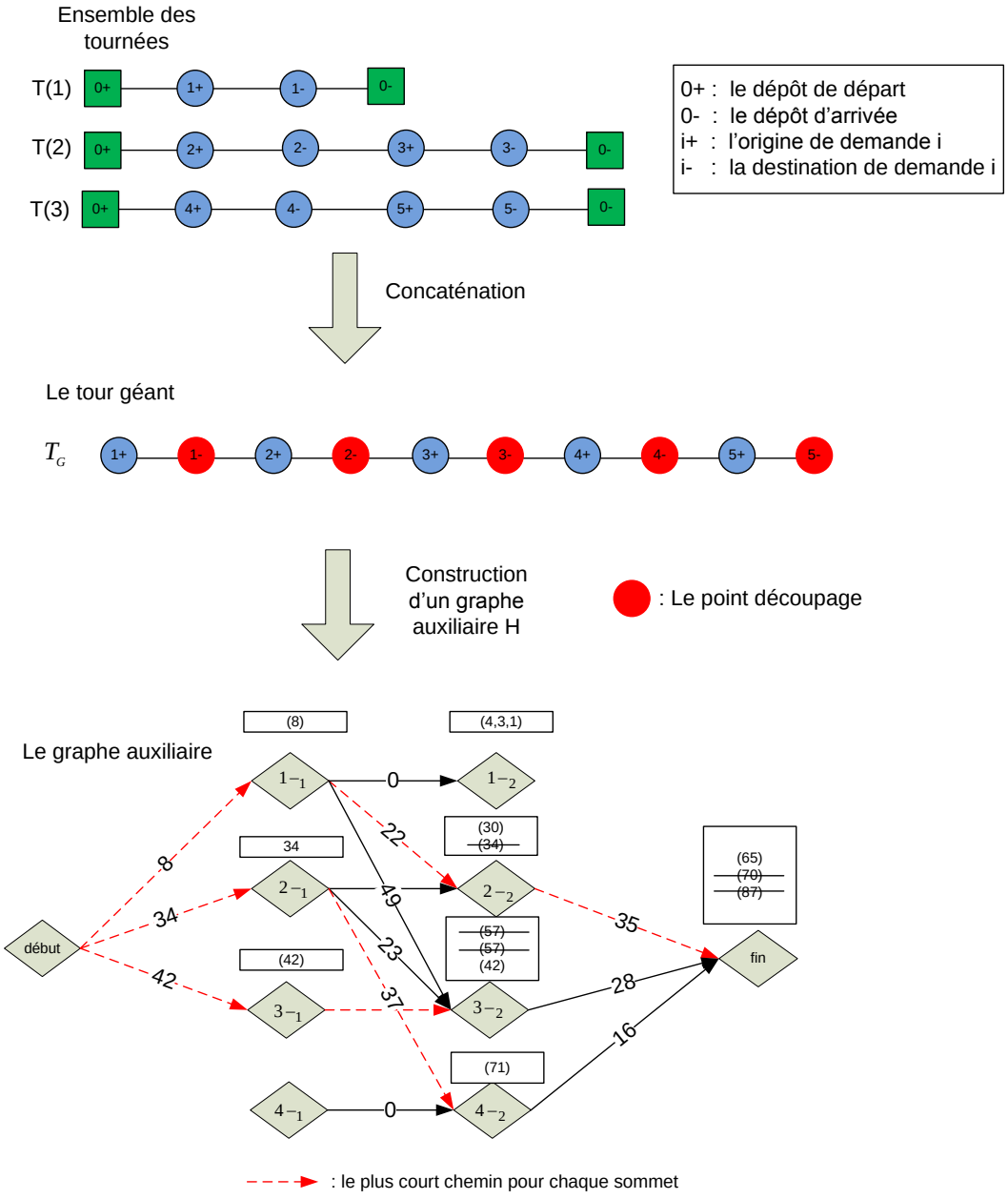

Figure 27 : Construction du graphe auxiliaire

La Figure 27 montre un exemple de la construction du graphe auxiliaire H. Le coût total de l'ensemble de tournées initiales est 85. Nous construirons d'abord le tour géant en utilisant trois tournées $T_G = \{1+,1-,2+,2-,3+,3-,4+,4-,5+,5-\}$. L'ensemble de points de découpage est $\omega_G = \{1-,2-,3-,4-,5-\}$. Un graphe est construit par ces points de découpage et deux sommets fictifs (début et fin). Chaque arc du graphe représente une tournée admissible et les valeurs sur les arcs représentent le coût de la tournée associée. Dans la Figure 27, l'arc $(début, 1-_1)$ représente la tournée $\{0+,1+,1-,0-\}$, et le coût de cette tournée est 8.

Nous constatons que le graphe H est un graphe orienté et valué. Afin de trouver les meilleures tournées (i.e. le meilleur découpage du tour géant), on cherche un (des) plus courts chemins depuis la source $debut$ vers le sommet fin. Ici, nous utilisons un algorithme de programmation dynamique (basé sur l'algorithme de « Bellman-Ford ») qui permet de trouver les plus courts chemins dans le graphe H. La procédure SPLIT est décrite par l'Algorithme 21.

Algorithme 21 : SPLIT

Entrée : H : graphe auxiliaire
Sortie : T ensemble de tournées
Paramètre : Z : *tableau des PERF depuis debut*

$Pred$: *tableau des prédécesseurs sur un plus court chemin depuis debut*

Pour $k = 1, \dots, K$ **faire**
  Notons A_k l'ensemble des arcs de H de la forme (x_{k-1}, y_k)
  Si $k = 1$ **alors**
    $A_1 \leftarrow$l'ensemble des arcs $(debut, x_1)$;
  Fin
  Si $k = K$ **alors**
    $A_K \leftarrow$l'ensemble des arcs de la forme (x_{K-1}, fin) ;
  Fin
Fin
Pour $k = 1, \dots, K$ **faire**
  Pour chaque arc $(u, v) \in A_k$ **faire**
    Si $Z(v) > Z(u) + l_{u,v}$ **alors**
      $Z(v) \leftarrow Z(u) + l_{u,v}$;
      $Pred(v) \leftarrow u$;
    Fin
  Fin
Fin

Lorsque l'opération du découpage pour le tour T_G est terminée, les tournées ont obtenues en remontant à partir du sommet fin du graphe H vers le sommet $Debut$.

Dans la Figure 27, la procédure SPLIT trouve le plus court chemin à partir du sommet $début$ vers le sommet fin qui a un coût meilleur que la solution initiale $(65 < 85)$. le nouvel ensemble des tournées optimales est représenté par les arcs $(début, 1-_1)$, $(1-_1, 2-_2)$ et $(2-_2, fin)$, qui correspond aux tournées $\{0+, 1+, 1-, 0-\}$, $\{0+, 2+, 2-, 0-\}$ et $\{0+, 3+, 3-, 4+, 4-, 5+, 5-, 0-\}$.

2.3. La procédure SPLIT-SHIFT

Dans la section précédente, nous avons présenté la procédure SPLIT classique qui consiste à découper de manière optimale un tour géant donné T_G. Dans cette section, nous présentons une autre version de SPLIT appelée SPLIT-SHIFT, qui manipule le tour géant et génère un nouveau tour.

Supposons que le tour géant T_G est divisé en un ensemble de segments $S = \{s(1), \dots, s(|\omega_G|)\}$ par les points de découpage ω_G. Chaque segment $s(v), v = 1, \dots, |\omega_G|$ est renversé afin d'obtenir un nouveau segment $s'(v)$ par la procédure $Invers(s(v))$. Comme le sommet origine d'une demande doit être visité avant son sommet destination, pour chaque demandeD_i, $i \in I$ incluse dans le segment inversé $s'(v), v = 1, \dots, |\omega_G|$, l'origine o_i et la destination d_i sont échangées. Si la tournée $\Gamma' = \{Depot1_k,\ s'(v),\ Depot2_k\}$ n'est pas une tournée admissible, alors on décide de conserver le segment $s(v)$ et on pose $s'(v) = s(v)$.

Le nouveau tour inversé, noté T_G^{inv} est obtenu en concaténant les segments inversés $s'(v), v = 1, \dots, |\omega_G|$. Le tour géant inversé T_G^{inv} a le même nombre de points de découpage que le tour géant T_G. La Figure 28 montre un exemple de la construction d'un tour inversé. Nous constatons que le segment $s(1)$ est un segment symétrique, son segment inversé $s'(1)$ est alors le même que $s(1)$. Pour le segment $s(2) = \{2+, 3+, 2-, 3-\}$, nous renversons d'abord ce segment $s'(2) = \{3-, 2-, 3+, 2+\}$, ensuite nous échangeons l'origine o_i et la destination d_i de la demande $D_i \in s'(2) : s'(2) = \{3+, 2+, 3-, 2-\}$.

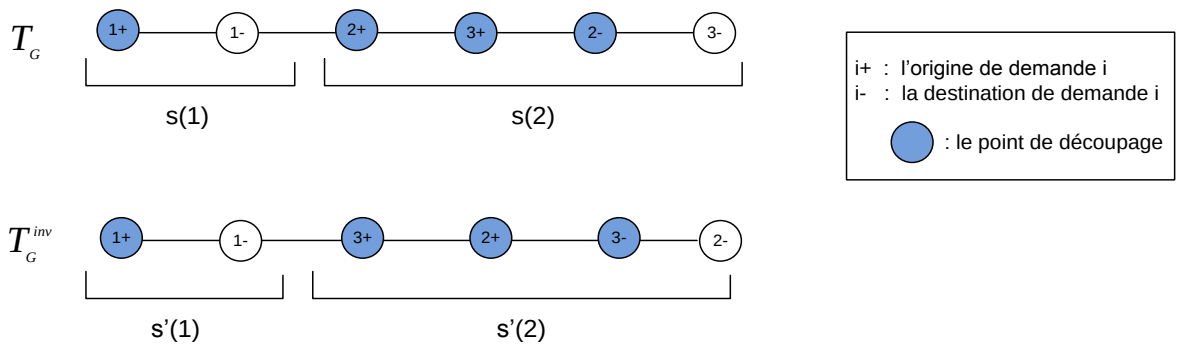

Figure 28 : Construction du tour géant inversé

La procédure SPLIT-SHIFT fonctionne alors ainsi, à partir du tour géant T_G :

- Construire le tour inversé T_G^{inv};
- Appliquer la procédure SPLIT à T_G^{inv};
- Récupérer un découpage de T_G^{inv} en K tournées, issu de cette application de SPLIT.

Dans le schéma général de transformation locale (voir la Figure 24), la procédure SPLIT et sa variante SPLIT-SHIFT peuvent être utilisées ensemble. Nous obtenons alors 2 découpages T et T^{inv}, et nous conservons celui qui fournit la meilleure performance.

3. Recherche Locale

3.1. Les opérateurs

La recherche locale nous permet d'améliorer la solution courante représentée par le couple (T, t). C'est-a-dire que l'on explore l'espace des solutions en passant d'une solution (T, t) vers une autre (T', t'). Le voisinage d'une solution est obtenu par l'application d'un mouvement de transformation locale. Les mouvements possibles concernent le déplacement de demandes et l'échange de demandes. Chaque mouvement peut impliquer une ou deux tournées. La stratégie de notre recherche locale se limite à une méthode de descente avec des opérateurs de transformation locale, à partir d'une solution initiale obtenue par la procédure SPLIT et sa variante. Nous proposons quatre mouvements :

3.1.1. Déplacement Interne

Le premier mouvement est défini par un opérateur $Deplace\_Interne(T(k))$ qui déplace un sommet dans la même tournée.

Déplacement Interne de sommet « origine »

On prend une tournée $T(k), k = 1, \dots, K$, un sommet $x = o_i$ de type « $Origine$ », présent dans $T(k)$ et un sommet y de $T(k)$ tel que : $x \ll_{T(k)} y \ll_{T(k)} Hom(x)$. On déplace alors x entre y et $succ(T(k), y)$. Cette procédure s'écrit $Deplace\_Interne\_Orig(k, x, y)$. On peut définir de la même façon $Deplace\_Interne\_Dest(k, x, y)$ où $x = d_i$ de type « $Destination$ » avec $Hom(x) \ll_{T(k)} y \ll_{T(k)} x$. La Figure 29 illustre une utilisation de cet opérateur.

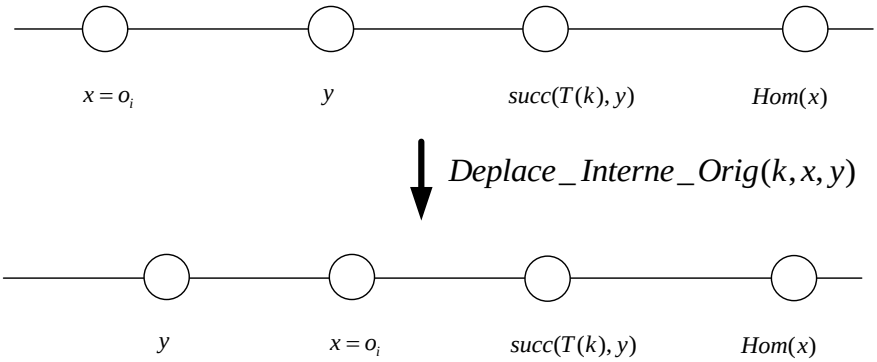

Figure 29 : Déplacement Interne

3.1.2. Déplacement Externe

Le second mouvement est défini par un opérateur $Deplace\_Externe$ qui déplace certaines demandes d'une tournée $T(k)$ vers une autre tournée $T(k')$, $k, k' \in \{1, \dots, K\}$ et $k \neq k'$. Cette procédure s'écrit $Deplace\_Externe(k, i, k', x, y)$:

- le couple (o_i, d_i) d'une demande D_i est extrait de la tournée $T(k)$;
- la demande D_i est insérée dans $T(k')$ selon les sommets d'insertion x et y : $T(k') \leftarrow Inserer(k', i, x, y)$.

La Figure 30 illustre une utilisation de cet opérateur.

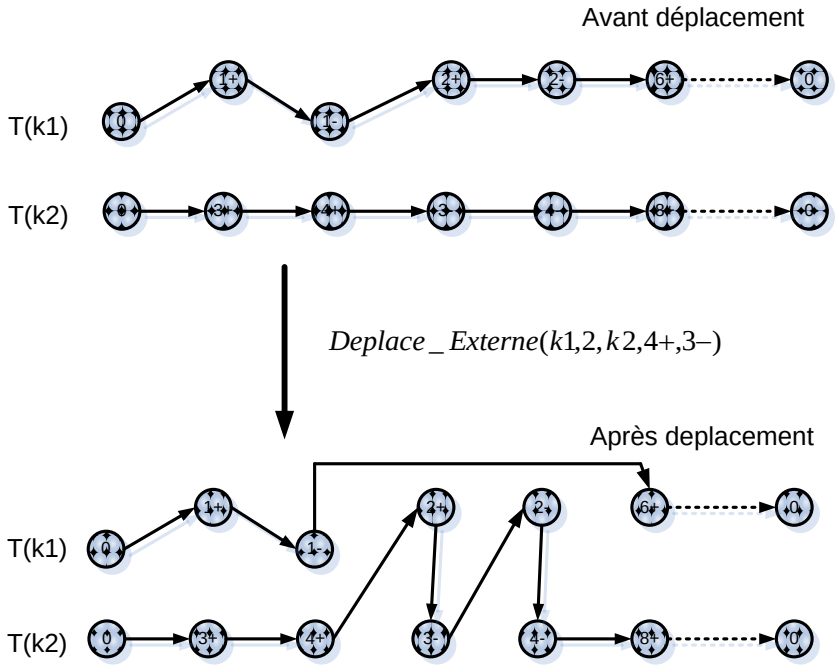

Figure 30 : Illustration de l'opérateur Déplacement Externe

3.1.3. Echange

Le troisième mouvement est défini par un opérateur $Echange(k, k', i, j, x, y, x', y')$ qui permet d'échanger une demande entre deux tournées différentes $T(k)$ et $T(k')$, $k, k' \in \{1, \dots, K\}, k \neq k'$ (voir la Figure 31).

- le couple (o_i, d_i) est extrait de la tournée $T(k)$;
- le couple (o_j, d_j) est extrait de la tournée $T(k')$;
- la demande i est insérée dans $T(k')$ selon les sommets d'insertion x et y ;
- la demande j est insérée dans $T(k)$ selon les sommets d'insertion x' et y'.

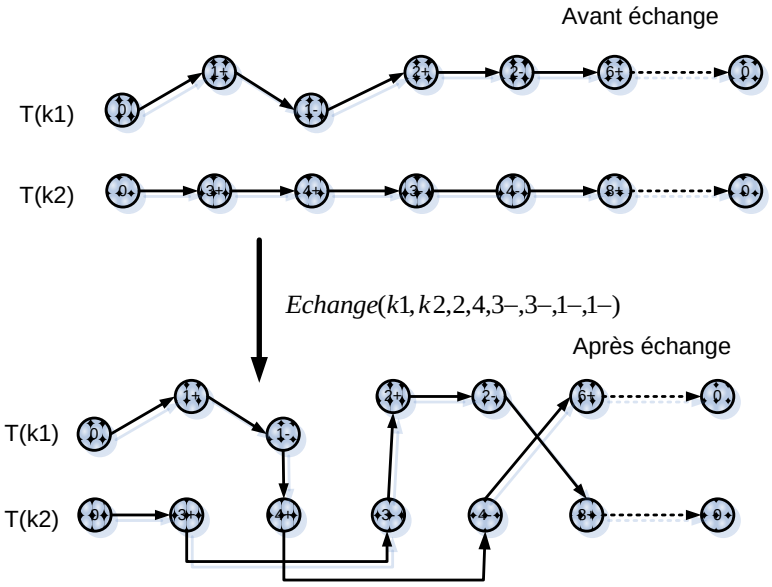

Figure 31 : Illustration de l'opérateur Echange

3.1.4. 2-Opt

Le dernier mouvement est défini par un opérateur $2 - Opt(k, k', x, y)$ qui permet d'échanger des sous-séquences entre deux tournées différentes $T(k)$ et $T(k')$, $k, k' \in \{1, \dots, K\}, k \neq k'$, on suppose que x, y sont des points de découpage pour $T(k)$ et $T(k')$. On pose :

- $s1 = \{u \in T(k)\ tq\ u \ll^{=}_{T(k)} x\}$;
- $s2 = \{u \in T(k)\ tq\ x \ll_{T(k)} u\}$;
- $s3 = \{u \in T(k')\ tq\ u \ll^{=}_{T(k)} y\}$;
- $s4 = \{u \in T(k')\ tq\ y \ll_{T(k)} u\}$.

Et on transforme alors $T(k) \leftarrow s1 \otimes s4$, $T(k') \leftarrow s3 \otimes s2$ où $\otimes$ est l'opérateur de concaténation, la Figure 32 montre un exemple d'utilisation de l'opérateur 2-Opt.

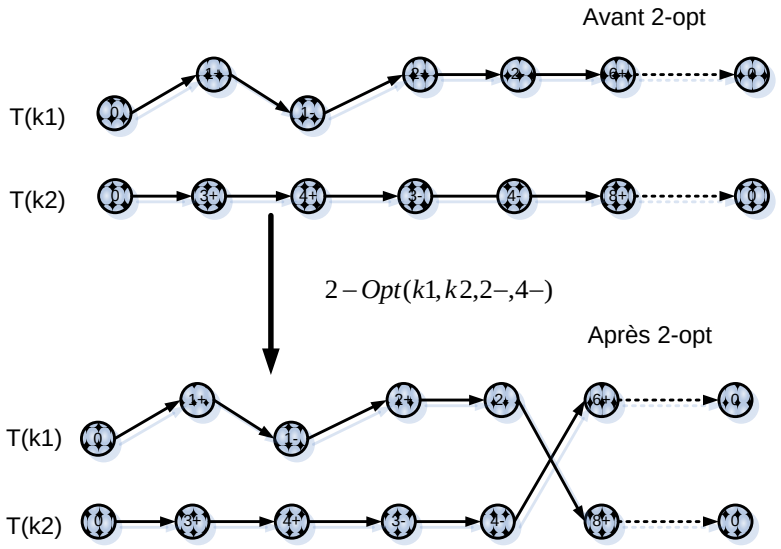

Figure 32 : Illustration de l'opérateur 2-Opt

3.2. Gestion des paramètres d'opérateur

Tous les opérateurs de recherche locale peuvent impliquer des paramètres qui concernent des tournées, des demandes ainsi que des sommets. En fait, selon les choix effectués pour quelques paramètres, les résultats obtenus sont radicalement différents. Nous présentons d'abord la sélection de paramètres :

3.2.1. Sélection de la tournée

Avant le démarrage de toutes les procédures de recherche locale, nous sélectionnons des tournées de manière aléatoire (et de sorte que la tournée choisie ne soit pas vide).

3.2.2. Sélection de la demande

Lorsque le choix des tournées est terminé, nous proposons aussi deux manières différentes de sélectionner les demandes pour les opérateurs « Déplacement Externe » et « Echange demandes » :

- La première est un choix aléatoire : les demandes sont choisies à l'aide d'un mécanisme stochastique ;

- La seconde est un choix déterministe : afin de pouvoir insérer la demande $D_i \in D$ dans une autre tournée, la demande D_i doit est compatible avec les demandes de la tournée considérée.

3.2.3. Sélection du sommet

Pour l'opérateur $Deplace\_Interne$, le sommet est choisi de manière aléatoire. Pour l'operateur *2-Opt*, nous cherchons un couple de points de découpage $(x, y), x \in \omega_1, y \in \omega_2$ pour les tournées $T(k)$ et $T(k')$ où $\omega_1 \in T(k)$ et $(\omega_2) \in T(k')$. Ce couple (x, y) est choisi tel que :

- $t(x) + Dist[x, succ(T(k'), y)] \leq \beta\big(succ(T(k'), y)\big)$;
- $t(y) + Dist[y, succ(T(k), x)] \leq \beta\big(succ(T(k), x)\big)$.

3.3. Stratégie

Chaque opérateur agit sur un couple (T, t). On cherche le meilleur voisin (T', t') (au sens des opérateurs définis précédemment) d'un couple (T, t) c'est-à-dire tel que $\sum_{k=1,..,K} PERF\big(T'(k)\big) < \sum_{k=1,..,K} PERF\big(T(k)\big), k \in \{1, \dots, K\}$. On remplace alors (T, t) par (T', t').

Les quatre opérateurs précédents sont utilisés par quatre procédures $RL\_DeplaceInterne$, $RL\_DeplaceExterne$, $RL\_Echange$ et RL_2-opt au sein de la recherche locale. Le principe de la recherche locale est présenté par l'Algorithme 22.

Algorithme 22 : Recherche locale

Entrée : T : l'ensemble de tournées ;
Ni : le nombre d'itération;
Sortie : T : l'ensemble de tournées ;

1. $cpt \leftarrow 0$;
2. **Répéter**
3. | $cpt \leftarrow cpt + 1$;
4. | $perf \leftarrow PERF(T)$;
5. | $T \leftarrow RL\_DeplaceInterne(T)$;
6. | $T \leftarrow RL\_DeplaceExterne(T)$;
7. | $T \leftarrow RL\_Echange(T)$;
8. | $T \leftarrow RL\_2-opt(T)$;
9. | **Si** $perf > Perf(T)$ **alors**
10. | | $T \leftarrow RL\_Split(T)$;
11. | **Fin**
12. **Jusqu'à** $cpt > Ni$;

Dans la procédure $RL\_DeplaceInterne$, nous choisissions d'abord aléatoirement une tournée $T(k)$. Ensuite, trois sommets sont sélectionnés au hasard dans la tournée $T(k)$. Nous déplaçons les sommets un par un, si un meilleur voisin est trouvé par un déplacement de sommet, alors la procédure $RL\_DeplaceInterne$ s'arrête.

Dans la procédure $RL\_DeplaceExterne$, deux tournées $T(k)$, $T(k')$ sont sélectionnées aléatoirement. L'opérateur « $Déplacement\ Externe$ » est appelé au maximum 3 fois. A chaque fois, une demande D_i de la tournée $T(k)$ est sélectionnée si elle est compatible avec toutes les demandes de la tournée $T(k')$ (c'est-à-dire, il n'existe aucun $non\_compatible(i,j)$, $\forall j, D_j \in T(k')$). On tente d'insérer la demande D_i est essayée d'insérer dans la tournée $T(k')$, si l'insertion réussie, alors la procédure $RL\_DeplaceExterne$ s'arrête.

Dans les procédures $RL\_Echange$ nous choisissions d'abord aléatoirement deux tournées $T(k)$, $T(k')$. L'opérateur « $Echange$ » est appelé trois fois, c'est-à-dire, nous essayons d'échanger 3 demandes entre deux tournées. Les demandes échangées sont choisies au hasard.

Pour la procédure RL_2-Opt, l'opérateur $2-Opt$ n'est appelé qu'une seul fois pour deux tournées en sélectionnant au hasard.

Une cinquième procédure RL_Split est utilisée lorsque les quatre procédures ci-dessus améliorent la solution courante. Nous limitons l'utilisation de l'opérateur RL_Split à cause de son coût très élevé en temps de calcul.

Nous utilisons la procédure « Evaluation_Cordeau_Laporte » (voir l'Algorithme 1 du chapitre 2) pour évaluer les tournées. Cette procédure peut déterminer si la tournée est temps admissible ou pas. Si la tournée est admissible, alors la procédure donne trois critères d'évaluation.

4. Evaluations numériques

4.1. Programmation et instances

Toutes les procédures ont été codées en Borland C++ 6.0 et exécutées sur un PC avec un processeur 2.4GHz sous Windows Xp avec 4 Go de mémoire. Les instances utilisées pour les tests sont celles proposées par Cordeau et Laporte [**77**]. Ils ont créé ces instances afin d'analyser le comportement de leur algorithme de résolution du DARP. Ils ont généré une série de 20 instances.

Ces instances, sont composées de 24 à 144 demandes. Dans chaque instance, chaque demande possède une fenêtre de temps associée, soit à son origine, soit à sa destination. Un temps de service (de durée 10) est imposé pour le chargement et le déchargement. Chaque demande est associée à une charge unitaire. La durée maximale d'une tournée est fixée à 480, la durée de transport maximale pour une demande est 90 et la capacité du véhicule est égale à 6. Les instances sont divisées en deux groupes : (Pr01…Pr10) et (Pr11…Pr20). Dans le premier

groupe, les fenêtres du temps sont plus serrées. Dans chaque groupe, le nombre de véhicules disponibles pour instance 1 à 6 est tel que les tournées ne sont pas trop chargées. Le nombre de véhicules disponibles pour les instances 7 à 10 est supposé être le nombre minimal de véhicules nécessaires.

La distance entre deux lieux x et y est définie comme étant la distance euclidienne entre les coordonnées des lieux x et y. On suppose que le temps de transport entre x et y est égal à la distance euclidienne entre x et y.

4.2. Résultats obtenus par la transformation locale

Nous comparons les résultats de transformation locale à la recherche taboue de Cordeau et Laporte [**77**]. Ils minimisent les distances parcourues alors que notre méthode minimise une combinaison des différentes durées. Nous mettrons les coefficients $\gamma = 10, \delta = 1, \varepsilon = 1$ afin de donner plus de poids à la durée totale des tournées. Le Tableau 16 montre les résultats obtenus par la transformation locale avec 10^5 itérations, les résultats obtenus par la recherche taboue [**77**] avec 10^5 itérations et les résultats obtenus par l'algorithme génétique avec 15000 itérations.

La comparaison entre les méthodes est difficile car [**77**] minimisent les distances parcourues, [**127**] minimisent une combinaison linéaire des critères suivantes :

- Distance totale;
- Temps de transport supplémentaire par rapport au temps de transport minimal pour chaque demande;
- Temps d'attente des clients;
- Durée totale des tournées;
- Pénalité sur la violation des fenêtres de temps;
- Pénalité sur la violation du temps de transport maximal des demandes;
- Pénalité sur la violation de la durée maximale des tournées.

Les coefficients ω_i, $i = 1, \dots, 7$ des critères sont donnés à $\omega_1 = 8$, $\omega_2 = 3$, $\omega_3 = 1$, $\omega_4 = 1$, $\omega_5 = n$, $\omega_6 = n$, $\omega_7 = n$. Le but de leur méthode étant de donner la priorité au confort des clients.

Les solutions initiales sont obtenues par l'heuristique « Insertion_Partielle_RF » qui a été présentée dans le chapitre 2. Les temps d'exécution sont comparables. Dans notre algorithme, nous utilisons quatre opérateurs, et les operateurs « $Déplacement\ Externe$ » et « $Echange$ » sont appelés maximum trois fois à chaque itération. Dans l'algorithme de Cordeau et Laporte, il n'a y qu'un seul opérateur « $Déplacement\ Externe$ ». Dans notre algorithme, une heuristique donne une solution initiale, nous nous intéressons uniquement à la tournée admissible pendant la recherche locale, donc la procédure d'évaluation s'arrête et donne un

signal négatif si une tournée viole les contraintes. Par contre, la tournée est évaluée complètement par la procédure d'évaluation dans la recherche taboue.

Pour les trois critères T_{Global}, T_{Wait}, T_{Ride}, les résultats en moyenne obtenus par notre algorithme sont meilleurs que ceux obtenus par la recherche taboue et l'algorithme génétique. Entre notre heuristique et la recherche taboue, notre heuristique fournit des durées totales 3% plus faibles, des temps de transport 31% plus faibles et des temps d'attente 93% plus faibles. Entre notre heuristique et l'algorithme génétique, notre heuristique fournit des durées totales 14% plus faibles, des temps de transport 25% plus faibles et des temps d'attente 86% plus faibles.

5. Conclusion

Dans ce chapitre, nous améliorons la solution du DARP obtenue par l'heuristique d'insertion proposée précédemment. Des opérateurs classiques de recherche locale sont utilisés pour l'amélioration. Nous proposons aussi la procédure SPLIT et une variante que nous avons adaptées pour le DARP. La procédure SPLIT consiste à découpage de manière optimale un tour géant pour obtenir des tournées admissibles. Dans le schéma de recherche locale, nous limitons l'utilisation à la procédure SPLIT car elle est très couteuse en temps de calcul, et son efficacité est amoindrie par les nombreuses contraintes de précédences et contraintes temporelles du DARP.

Tableau 16 : Comparaison des résultats entre transformation locale et la recherche locale

	Transformation locale				**Recherche Taboue**				**Algorithme génétique**			
Instance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}
pr01	2142,86	796,52	819,97	38,99	1140	881,2	1095,0	211,2	334,20	1041	477	252
pr02	3393,01	1696,00	1951,10	264,50	4836	1985,9	1976,7	723,9	685,80	1969	1367	470
pr03	3240,14	2375,01	2706,46	80,34	10308	2579,4	3586,7	607,3	3094,80	2779	3081	292
pr04	4048,43	2908,84	3466,21	3,28	17262	3583,2	5021,3	1090,4	-	-	-	-
pr05	5204,10	3508,49	4474,11	63,90	27744	3870,0	6156,5	833,0	3493,80	4250	5099	500
pr06	4310,07	4295,13	5815,29	4,68	32322	5056,8	7273,5	1375,4	-	-	-	-
pr07	2385,40	1205,35	1094,88	98,33	2634	1452,1	1508,9	440,3	-	-	-	-
pr08	5959,91	2250,26	2788,70	66,19	12264	2345,2	3691,5	410,3	-	-	-	-
pr09	9605,70	3365,41	3654,12	42,57	30306	3155,5	5621,8	323,1	2446,80	3597	6251	94
pr10	8343,199	4724,19	7244,48	94,11	52518	4480,1	7163,7	721,3	3958,80	5006	8413	315
pr11	2158,48	787,35	719,98	1,25	1158	965,1	1041,5	320,6	327,60	907	630	143
pr12	7340,63	1453,26	1712,98	4,37	4974	1564,7	2393,8	308,7	703,20	1719	1214	198
pr13	9294,34	2365,35	2630,70	1,15	11124	2263,7	3788,8	330,4	-	-	-	-
pr14	10613,86	2900,67	3783,02	2,15	18708	2882,2	4632,8	426,3	-	-	-	-
pr15	13346,79	3601,02	4162,67	1,24	32598	3595,6	6104,7	605,9	3535,80	4296	4615	552
pr16	18301,0	4324,84	5497,78	11.92	44220	4072,5	7347,4	448,9	4873,80	5309	6134	630
pr17	675,73	1200,76	629,3	1,95	2538	1097,3	1762,0	129,0	497,40	1299	990	102
pr18	1219,04	2275,76	1342,01	11,64	13716	2446,5	3659,0	523,4	-	-	-	-
pr19	15320,16	3386,41	2248,36	2,81	30768	3249,3	5581,0	487,3	2679,60	3679	5362	147
pr20	23879,29	4412,27	2812,13	0	55446	4041,0	7072,3	362,4	3984,60	4733	7969	113
Moyenne	7539,11	2691,64	2977,71	39,77	20329,20	2778,4	4323,9	534,0	2355.09	3121,85	3969,38	292,92

CHAPITRE 5
ELS POUR LE DARP MULTI-OBJECTIF

1. Introduction

Après avoir étudié des procédés de transformation locale pour le DARP, nous proposons une méthode de résolution basée sur une méta-heuristique évolutionnaire dans ce chapitre. Dans un premier temps, nous introduisons le principe de la méta-heuristique ELS (Evolutionnaire Local Search). La section suivante présente les composants de cette méta-heuristique pour le DARP monocritère. Nous proposons ensuite une adaptation au DARP multicritère (trois critères). L'objectif de la méthode est de trouver un front de Pareto dans une population de solutions. Les résultats obtenus ainsi que des comparaisons avec d'autres méthodes sont présentés à la fin du chapitre.

2. Présentation de la méta-heuristique ELS (Evolutionary Local Search)

La méta-heuristique ELS est une extension de l'ILS (*Iterated Local Search*). Le but de l'ILS est d'améliorer une solution initiale S de manière itérative à l'aide de procédés de mutation et une recherche locale (RL) : à chaque itération d'ILS, une copie S' de la solution courante S est modifiée par une procédure de mutation $Mut(S')$. La solution S' est ensuite améliorée par la recherche locale. La solution courante S est remplacée par la solution S' si et seulement si S' est meilleure que la solution S. Cette méthode est présentée par l'Algorithme 23.

La méthode ELS a récemment été introduite par Wolf et Merz [**134**] pour résoudre un problème de télécommunications. L'ELS est un algorithme évolutionnaire avec une méthode

de recherche locale. La méthode ELS se compose de deux parties principales : la recherche locale et l'algorithme évolutionnaire.

L'Algorithme 24 ci-dessous donne la structure générale de la méthode ELS. La méthode commence par l'initialisation d'une solution S et par son amélioration grâce à une recherche locale $(RL(S))$. Ensuite, l'ELS effectue itérativement ng fois le processus de génération : chaque génération construit nc solutions-filles qui sont obtenues en faisant une copie de la solution courant S. Chaque solution-fille $S_j, j = 1, \dots, nc$ est modifiée par la procédure de mutation Mut. La procédure mutation s'applique à la solution-fille S_j en utilisant un paramètre x, et la transforme. Après la mutation, la solution-fille courante S_j est améliorée par la recherche locale. La meilleure solution-fille S_j est conservée. Si S_j est meilleure que la solution courante S, alors S_j devient la solution courante.

Algorithme 23 : ILS général

Entrées : ng ;
Sorties S ;

1. Initialisation d'une solution S
2. $S \leftarrow RL(S)$;
3. **Pour** i **de** 1 **à** ng
4. | $S' \leftarrow S$;
5. | $S' \leftarrow Mut(S')$;
6. | $S' \leftarrow RL(S')$;
7. | **Si** $S' est\ meilleure\ que\ S$ **alors**
8. | | $S \leftarrow S'$;
9. | **Fin**
10. **Fin**

Algorithme 24 : ELS général

1. Initialiser S ;
2. $S \leftarrow RL(S)$;
3. **Pour** i **de** 1 **à** ng
4. | $(* \ processus\ de\ génération\ *)$
5. | Faire nc copies $S_1, \dots, S_{nc}$ de S ;
6. | **Pour** j **de** 1 **à** nc
7. | | Choisir au hasard une valeur de paramètre x $(* \ mutation\ aléatoire\ *)$;
8. | | $S_j \leftarrow Mut(S_j, x)$;
9. | | $S_j \leftarrow RL(S_j)$;
10. | **Fin**
11. | $S' \leftarrow la\ meilleure\ des\ solutions\ S_j, j = 1, \dots, nc$;
12. | **Si** S' est meilleure que S **alors**
13. | | $S \leftarrow S'$;
14. | **Fin**
15. **Fin**

La méthode ELS permet d'explorer une région de recherche autour d'un optimum locale, à travers les solution-filles. La méthode ELS peut être considérée comme une extension d'ILS car la ELS utilise nc fois la procédure de mutation afin de générer nc solutions-filles à chaque itération au lieu d'une seule dans l'ILS.

3. ELS pour le DARP monocritère

Dans cette section, nous concevons un algorithme « DARP_ELS_Mono » pour le DARP monocritère qui utilise le schéma ELS. Cet algorithme utilise des composants identiques à ceux utilisés dans l'algorithme de transformation locale proposé dans le chapitre précédent : le procédé SPLIT et les mouvements de recherche locale. Cette nouvelle méthode nécessite un autre composant important : une procédure de mutation. De plus, l'ELS est une méta-heuristique, elle a donc besoin d'une solution initiale. Les heuristiques introduites dans les chapitres précédents. Nous fournissent cette solution initiale.

Nous rappelons d'abord quelques notations nécessaires pour cette section :

- l'ensemble des véhicules homogènes $VH = \{1, \dots, K\}$;
- l'ensemble des sommets $X = \{o_i, d_i | i \in I\} \cup \{DepotD(k), DepotA(k) |\ k \in VH\}$;
- $Debut(\Gamma)$: le premier élément de la séquence Γ;
- $Fin(\Gamma)$: le dernier élément de la séquence Γ;
- pour chaque sommet $x \in X$:
  - si $x = o_i(\ respectivement\ d_i)$ alors
    on pose $Hom(x) = d_i(respectivement\ o_i)$;
  - si $x = DepotD(k)$ $(respectivement\ DepotA(k))$ alors
- pour tout $x, y \in X$, $Dist[x, y]$ présente le temps minimal pour aller de x vers y

Une solution S du DARP est un couple (T, t) où :

- $T = (T(k), k \in VH)$ est une famille de tournées admissibles, telle que les ensembles $X\big(T(k)\big), k \in VH$, de sommets couverts par les tournées $T(k)$, $k \in VH$, constituent une partition de l'ensemble des sommets X ;
- t est un ensemble de dates, qui fait de chaque tournée $T(k)$, $k \in VH$, une tournée « datée » : $\forall x \in X$, $t(x)$ représente la date de passage au sommet x.

Pour mesurer la performance d'une tournée $T(k), k \in VH$, on pose trois critères :

- $T_{Global}\big(T(k)\big) = t\left(Fin\big(T(k)\big)\right) - t\left(Debut\big(T(k)\big)\right)$
- $T_{Ride}\big(T(k)\big) = \sum_{x \in T(k), Type(x) \in \{Depot1, Origine\}} \left(t\big(Hom(x)\big) - t(x)\right)$
- $T_{Wait}\big(T(k)\big) = T_{Global}\big(T(K)\big) - \sum_{x \in T(k), x \neq Fin(T(k))} Dist\big(x, succ(T(k), x)\big)$

On suppose toujours que la performance $PERF$ d'un ensemble de tournées, est telle que : $PERF = \sum_{k=1,..,K} Cout(T(k))$ où

$Cout(T(k)) = \gamma \times T_{Global}(T(k)) + \delta \times T_{Ride}(T(k)) + \varepsilon \times T_{Wait}(T(k))$ Où $\gamma, \delta, \varepsilon$ sont coefficients.

3.1. Opérateurs de recherche locale

Dans la méthode proposée dans ce chapitre, les mouvements de recherche locale possibles pour le DARP monocritère sont :

- Déplacement interne : déplacement d'un sommet (origine ou destination) d'une demande dans une tournée. On considère, pour cet opérateur, un sommet $x = o_i$ de type $Origine$ présent dans $T(k)$ et un sommet y de $T(k)$ tels que : $x \ll_{T(k)} y \ll_{T(k)} Hom(x)$. On déplace alors x entre y et $succ(T(k), y)$. Cette procédure s'écrit $Deplace\_Interne\_Orig(k, x, y)$. On peut définir de la même façon $Deplace\_Interne\_Dest(k, x, y)$ où $x = d_i$ de type $Destination$ avec $Hom(x) \ll_{T(k)} y \ll_{T(k)} x$. La Figure 33 illustre une utilisation de cet opérateur.

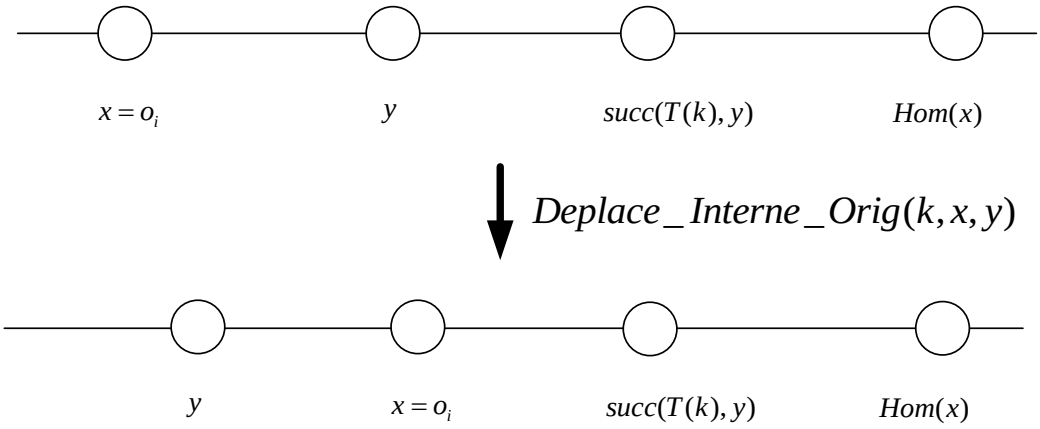

Figure 33 : Illustration de l'opérateur de déplacement interne sur la tournée *k*

- Déplacement externe (voir Figure 34) : déplacement d'une demande d'une tournée vers une autre. Ce mouvement est défini par un opérateur $Deplace\_Externe(k, i, k', x, y)$ qui déplace une demande i d'une tournée $T(k)$ vers une autre tournée $T(k')$, $k, k' \in VH, k \neq k'$. L'origine de i est placée après x dans $T(k')$ et la destination de i est placé après y dans $T(k')$;

- Echange (voir la Figure 35) : permutation de deux demandes appartenant à deux tournées différentes; Ce mouvement est défini par un opérateur $Echange(k, k', i, j, x, y, x', y')$ qui agit de la manière suivante :
  - le couple (o_i, d_i) est extrait de la tournée $T(k)$;
  - le couple (o_j, d_j) est extrait de la tournée $T(k')$;
  - la demande i est insérée dans $T(k')$ selon les sommets d'insertion x et y (o_i est placé après x et d_i est placé après y si $x \neq y$, après o_i sinon);
  - la demande j est insérée dans $T(k)$ selon les sommets d'insertion x' et y'.

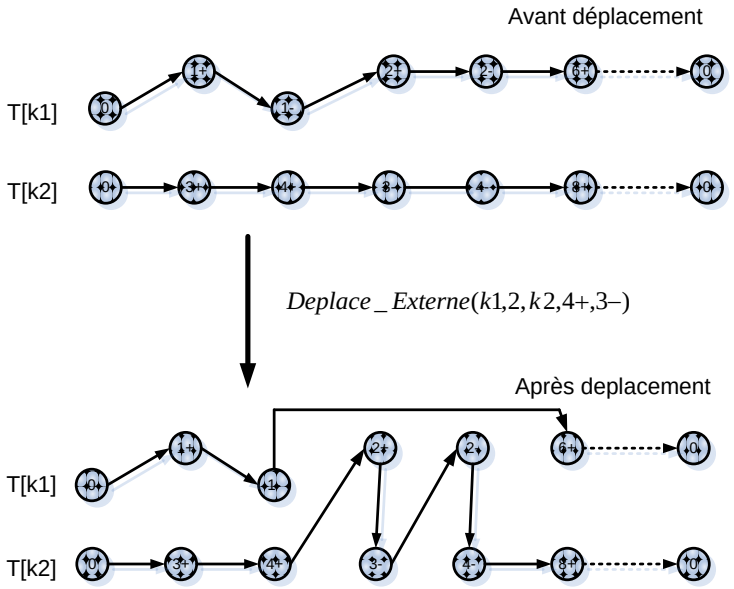

Figure 34 : Illustration de l'opérateur de déplacement externe

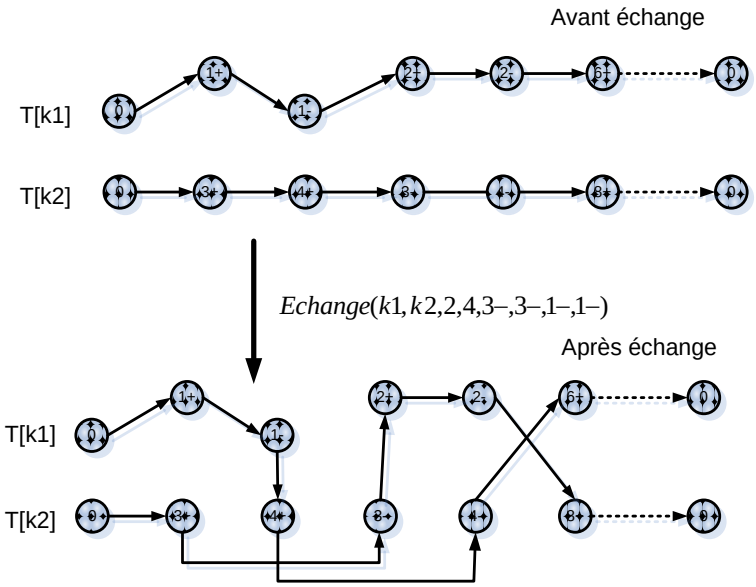

Figure 35 : Illustration de l'opérateur d'échange

- $2-Opt$ (voir la Figure 36) : échange de deux séquences de deux tournées ; Ce mouvement est défini par un opérateur $2-Opt(k,k',x,y)$, on suppose que x,y sont des points de découpage (voir le chapitre 4) pour $T(k)$ et $T(k')$. On pose :
  - $s1=\{u\in T(k)\ tq\ u \ll^{=}_{T(k)} x\}$;
  - $s2=\{u\in T(k)\ tq\ x \ll_{T(k)} u\}$;
  - $s3=\{u\in T(k')\ tq\ u \ll^{=}_{T(k)} y\}$;
  - $s4=\{u\in T(k')\ tq\ y \ll_{T(k)} u\}$.

On transforme alors les tournées de la manière suivante : $T(k)\leftarrow s1\otimes s4$ et $T(k')\leftarrow s3\otimes s2$ où $\otimes$ est l'opérateur de concaténation et $\leftarrow$ est l'opérateur d'affectation. La Figure 36 illustre une utilisation de l'opérateur $2-Opt$.

Lors de l'insertion d'une demande toutes les positions possibles d'insertion sont testées pour chacun des opérateurs présentés ci-dessus.

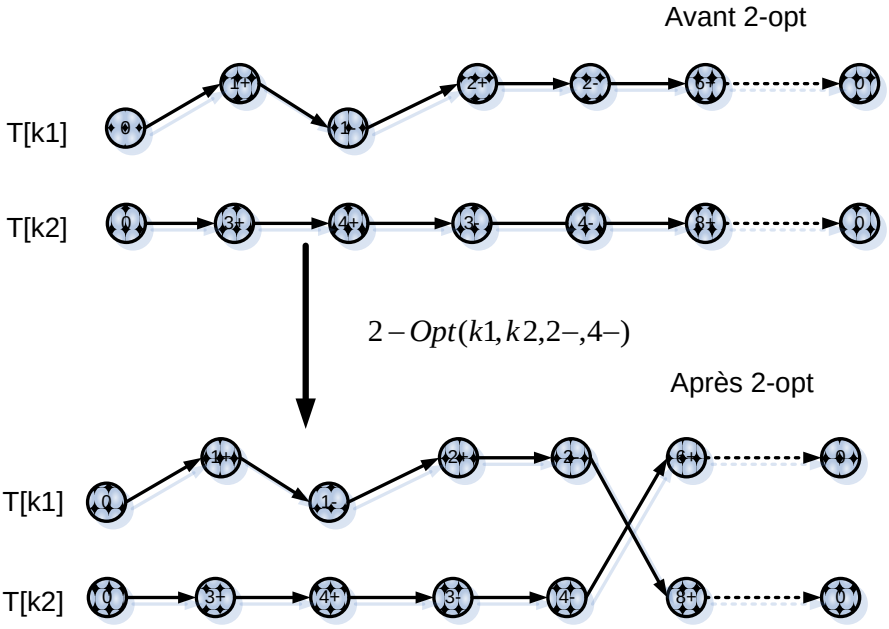

Figure 36 : Illustration de l'opérateur 2-Opt

3.2. Opérateur de mutation

L'opérateur de mutation est très important dans la méta-heuristique ELS. En effet, il permet d'éviter de converger vers des extrema locaux et assure la création d'éléments originaux. Si la mutation est trop faible, la solution transformée reste alors dans le même bassin d'attraction que l'optimum courant. Au contraire, si la mutation est trop forte, la recherche locale prendra beaucoup de temps pour trouver un nouvel optimum local.

La procédure SPLIT est choisie comme opérateur de mutation pour l'ELS : l'ordre des tournées dans le tour géant est modifié, on obtient ainsi un nouveau tour géant qui est redécoupé par SPLIT. Si on obtient de nouveaux tournées initiales alors on exécute K échanges aléatoires de demandes entre les tournées. Ces mouvements sont acceptés s'ils n'induisent aucune violation. Lorsque K échanges ont été effectués, on applique de nouveau la procédure SPLIT qui crée de nouvelle tournées.

3.3. Structure générale de l'algorithme

Le schéma général de l'ELS représentée par la Figure 37 : tout d'abord, on construit une solution initiale S du DARP en utilisant une heuristique. La solution S est ensuite améliorée par la recherche locale. La boucle principale de l'ELS comporte un nombre fixe ng d'itérations. A chaque itération, nc solution-filles sont créées grâce à l'opérateur de mutation, puis améliorées par la recherche locale. A la fin de chaque itération, la solution S est remplacée par la solution-fille qui a la meilleure performance $PERF$.

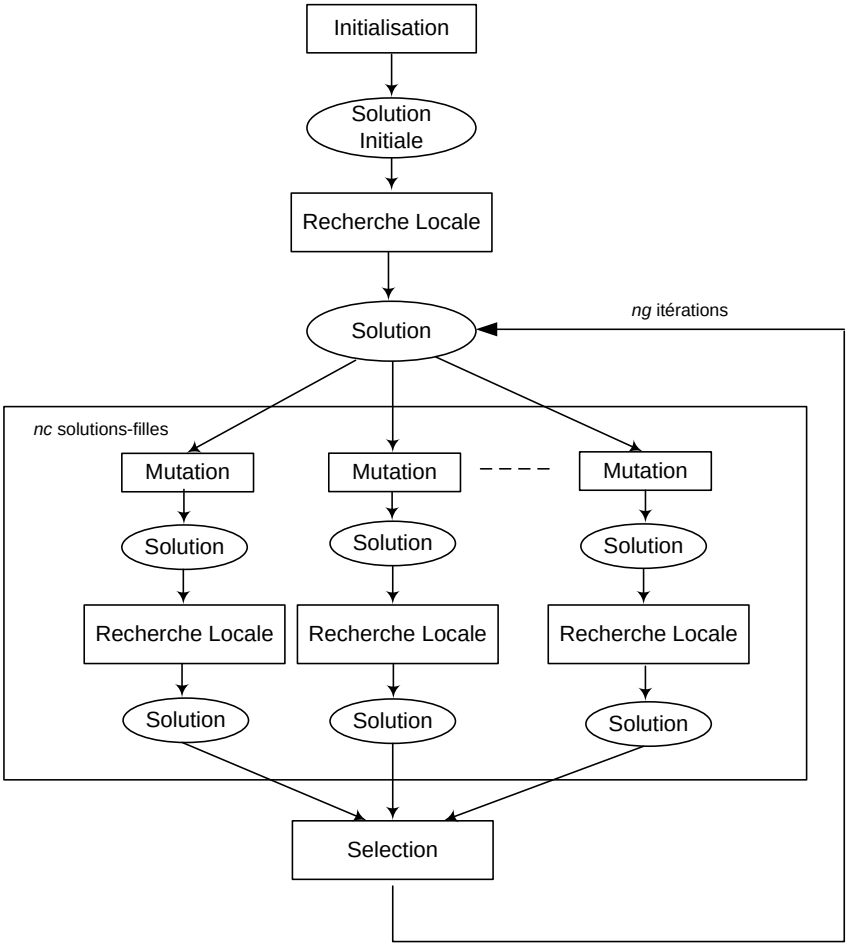

Figure 37 : Le schéma général de l'ELS

4. ELS pour le DARP multicritère

Dans les chapitres précédents, le DARP est abordé comme un problème de minimisation dans lequel on ne minimise qu'un critère qui est une combinaison linéaire de différentes valeurs. Dans cette section, le DARP est considéré comme un problème multi-objectifs (ou multicritère). On recherche alors un ensemble de solutions non-dominées (le « front de Pareto », voir la section 4.2.2) parmi lesquelles on ne peut décider si une solution est meilleure qu'une autre, aucune n'étant inférieure aux autres sur tous les critères. Dans cette section, nous présentons une adaptation de l'algorithme précédent (section 3). La principale différence réside dans le fait qu'on va manipuler, non plus une unique solution, mais une population de solutions.

4.1. Notations

On rappelle d'abord les trois critères utilisés pour mesurer la performance d'une tournée $T(k), k \in VH$:

- $T_{Global}\big(T(k)\big) = t\Big(Fin\big(T(k)\big)\Big) - t\Big(Debut\big(T(k)\big)\Big)$;
- $T_{Ride}\big(T(k)\big) = \sum_{x \in T(k), Type(x)\{Depot1, Origine\}} t\big(Hom(x)\big) - t(x)$;
- $T_{Wait}\big(T(k)\big) = T_{Global}\big(T(K)\big) - \sum_{x \in T(k), x \neq Fin(T(k))} Dist\big(x, succ(T(k), x)\big)$.

La performance multicritère d'une solution S du DARP est présentée par un triplet (h_1, h_2, h_3), où

- $h_1 = \sum_{k=1,\dots,K} T_{Global}(T(k))$;
- $h_2 = \sum_{k=1,\dots,K} T_{Ride}(T(k))$;
- $h_3 = \sum_{k=1,\dots,K} T_{Wait}(T(k))$.

4.2. La dominance et le front de Pareto

Dans les problèmes d'optimisation multicritère, la performance d'une solution dépend de plusieurs critères, il est donc parfois difficile de déterminer si une solution est meilleure qu'une autre (en effet elle peut être meilleure sur un critère mais plus mauvaise sur un autre). Nous allons voir dans cette section comment utiliser le front de Pareto pour résoudre ce problème.

4.2.1. La relation de la dominance

La relation de dominance (notée $\succ$) est une relation binaire permettant une comparaison indiscutable entre deux solutions. Une solution S domine une solution S'(on note $S \succ S'$) si et

seulement si la solution S est meilleure (ou égale) que la solution S' sur tous les critères : $\forall i = 1,2,3, h_i(S) \leq h_i(S')$.

4.2.2. Le front de Pareto

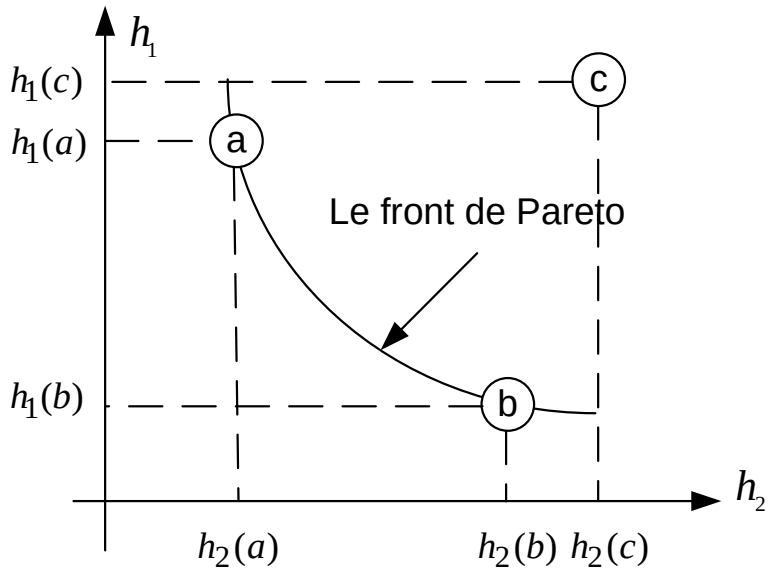

Figure 38 : L'exemple du front de Pareto

Les solutions qui dominent les autres solutions sont appelées solutions non-dominées (ou solutions optimales au sens du front de Pareto). La Figure 38 présente un exemple du front de Pareto dans le cas de la minimisation sur deux critères h_1 et h_2. Dans la Figure 38, on voit que a domine c, b domine c, mais a ne domine pas b et b ne domine pas a, On dit alors que a et b sont dans le même front de Pareto. Le front de Pareto est donc l'ensemble des meilleurs compromis entre les différents critères d'optimisation : il n'existe pas de solution (parmi l'ensemble de solutions considérées) qui soit meilleure sur tous les critères à la fois.

4.3. Structure générale de l'algorithme

La structure générale de l'algorithme que nous proposons est présentée par l'Algorithme 25. L'objectif est de chercher des solutions optimales au sens du front de Pareto. L'algorithme commence par la construction d'une population initiale $\mathcal{F}$ de np solutions différentes. Pour chaque solution, nous choisissons d'abord un critère qui doit être amélioré strictement, les autres critères sont maintenus à leur niveau initial.

Algorithme 25 : Schéma algorithmique DARP_ELS_Multi pour le DARP multicritère

1. Initialiser un ensemble $\mathcal{F}$ de solutison
2. **Pour** chaque solution S dans $\mathcal{F}$ **faire**
3. | Choisir un critère $h_i(S), i = 1, \dots, 3$
4. | Essayer d'améliorer au moins $h_i(S)$ sans dégrader les autres critères à l'aide de l'algorithme ELS
5. **Fin**
6. Extraire des solutions de Pareto

4.4. Initialisation de la population

Chaque solution initiale de « DARP_ELS_Multi » est générée par les heuristiques décrites dans les chapitres précédents en utilisant comme mesure de performance pour une solution (T, t) : $PERF\big(T(k)\big) = \gamma \times T_{Global}\big(T(k)\big) + \delta \times T_{Ride}\big(T(k)\big) + \varepsilon \times T_{Wait}\big(T(k)\big)$, les coefficients γ, δ , ε étant choisis au hasard dans l'intervalle $[0,1]$.

4.5. Recherche locale

Dans le DARP multicritère, les opérateurs de recherche locale gèrent simultanément les trois critères. Nous utilisons les mêmes opérateurs de recherche locale :

- Déplacement interne : déplacement d'un sommet d'une demande dans une tournée ;
- Déplacement externe : déplacement d'une demande d'une tournée vers une autre ;
- Echange : permutation de deux demandes appartenant à deux tournées différentes ;
- $2-\text{Opt}$: échange de deux séquences de deux tournées différentes.

Les mouvements sont acceptés si le critère choisi est amélioré strictement et que les autres critères ne sont pas dégradés.

4.6. L'opérateur de mutation

De même que pour le DARP monocritère, la procédure SPLIT est choisie comme un opérateur de mutation pour l'ELS : un nouveau tour géant T'_G est obtenu en changeant l'ordre de concaténation des tournées d'une solution initiale.

Le second opérateur de mutation exécute *K* échanges aléatoires de demandes parmi l'ensemble des tournées. Les mouvements sont acceptés s'ils n'induisent aucune violation. Lorsque les *K* échanges sont terminés, le tour géant obtenu est découpé grâce à la procédure SPLIT.

5. Adaptation de l'algorithme au DARP multicritère

Nous conservons tous les composants précédemment proposés pour le DARP monocritère. Pour le DARP multicritère, les composants doivent être adaptés à la gestion simultanée des trois critères. Dans cette section, nous présentons les modifications du SPLIT pour le cas multicritère.

5.1. La procédure SPLIT

Dans le chapitre précédent, nous avons utilisé la SPLIT pour le DARP monocritère. La SPLIT construit un découpage optimal pour un tour géant donné T_G obtenu en concaténant les tournées $T(k), k = 1, \dots, K$ d'une solution. Comme pour le DARP monocritère, la procédure SPLIT pour le DARP multicritère construit aussi un graphe auxiliaire $H = (\Psi, A, L)$ en utilisant l'ensemble de points de découpage dans T_G. Un sommet $x \in T_G$ est un point de découpage si $\sum_{y \ll_{T_G}^{=} x} c(y) = 0$, c'est-à-dire, ou le véhicule est vide. Nous notons que ω_G est l'ensemble de point de découpage. Le graphe auxiliaire $H = (\Psi, A, Z)$ pour le DARP multicritère est défini comme suit :

- L'ensemble Ψ des sommets du graphe H : $\{debut, x_k, fin \mid x \in \omega'_G, k \in \{1, \dots, K-1\}\}$ où les sommets $debut, fin$ sont des sommets fictifs et $x \in \omega'_G$ où ω'_G est un sous-ensemble de points de découpage ω_G pour le tour géant T_G : $\omega'_G = \omega_G - Fin(T_G)$.
- L'ensemble A des arcs du graphe H est défini en posant d'abord pour tout couple (x, y) dans ω'_G, $x \ll_{T_G} y$: $S_{x,y} = \{z \in T_G \ tq \ x \ll_{T_G} z \ll_{T_G}^{=} y\}$. Si $x = Null$, alors $S_{x,y}$ désigne l'ensemble des z tels que $z \ll_{T_G}^{=} y$; si $y = Null$, alors $S_{x,y}$ désigne l'ensemble des z tels que $x \ll_{T_G}^{=} z$; Nous notons que x_k : le sommet x est visité par le véhicule k. On définit alors A par :
  $A = \{(debut, x_1), x \in \omega_G$ tel que $\Gamma_{Null,x}$ est une tournée admissible$\}$
  $\cup \{(x_{K-1}, fin), x \in \omega_G$ tel que $\Gamma_{x,Null}$ est une tournée admissible$\}$
  $\cup \{(x_k, y_{k+1}), k = 1, \dots, k-2, x, y \in \omega_G$ tel que $\Gamma_{x,y}$ est une tournée admissible$\}$
- Cette construction est bien sûr la même que dans le cas monocritère. La spécificité du cas multicritère vient du fait que chaque arc (x, y) de A est accompagné, non pas d'une « longueur » (performance unique), mais d'un label constitué d'un triplet de longueurs. Ceci se formalise comme suit : on associe à chaque arc (x, y), $x, y \in \Psi$ de A un label $l_{x,y}$ de performance qui est défini par le triplet $(T_{Global}(\Gamma_{x,y}), T_{Ride}(\Gamma_{x,y}), T_{Wait}(\Gamma_{x,y}))$.

Pour le DARP multicritère, le procédé SPLIT consiste à chercher des chemins non dominés

depuis le sommet $debut$ vers le sommet fin. Le SPLIT (basé sur l'algorithme « Bellman-Ford ») pour le DARP multicritère est décrit par l'Algorithme 26.

Algorithme 26 : Le SPLIT pour le DARP multicritère

Entrée : H : le graphe auxiliaire
Sortie : $T(k)$: un ensemble de tournées $k = 1, \dots, K$
Paramètre : $Pred$: *tableau des prédécesseurs sur un plus court chemin depuis debut*
Procédures auxiliaires :
$add(l, x)$: Une fonction pour ajouter le label l à un sommet x
$dom(l, x)$: Une fonction qui renvoie vrai si le label l doit être ajouté au sommet x, faux sinon

1. **Pour** $k = 1, \dots, K$ **faire**
2. | Notons A_k l'ensemble des arcs de H de la forme (x_{k-1}, y_k)
3. | **Si** $k = 1$ **alors**
4. | | A_1 =ensemble des arcs $(debut, x_1)$;
5. | **Fin**
6. | **Si** $k = K$ **alors**
7. | | A_K =ensemble des arcs de la forme (x_{K-1}, fin) ;
8. | **Fin**
9. **Fin**
10. **Pour** $k = 1, \dots, K$ **faire**
11. | **Pour** chaque arc $(x, y) \in A_k$ **faire**
12. | | **Pour** chaque label l associé au sommet x **faire**
13. | | | $l' \leftarrow l + l_{x,y}$;
14. | | | **Si** $dom(l', y)$ est accepté **alors**
15. | | | $add(l', y)$;
16. | | | $Pred(l', y) \leftarrow (l, x)$;
17. | | | **Fin**
18. | | **Fin**
19. | **Fin**
20. **Fin**

La Figure 39 nous montre un exemple du SPLIT pour le DARP multicritère. Le tour géant T_G est obtenu en concaténant trois tournées $T(1), T(2), T(3)$ d'une solution de départ. L'ensemble de points de découpage est $\omega_G = \{1-, 2-, 3-, 4-, 5-\}$. Le graphe auxiliaire H est ensuite construit. On constate qu'il n'existe pas d'arc entre $début$ et $4-_1$ et entre $1-_2$ et fin dans le graphe H, parce que la tournée $\{0+, 1+, 1-, 2+, 2-, 3+, 3-, 4+, 4-, 0-\}$ et la tournée $\{0+, 2+, 2-, 3+, 3-, 4+, 4-, 5+, 5-, 0-\}$ ne sont pas admissibles. Nous parcourons ce graphe pour chercher des chemins non dominés. Au dessus de chaque sommet, sont indiqués des labels de performance. Par exemple, au sommet $3-_2$ (3- est la destination, 2 est le véhicule 2), deux labels (18, 30, 9) (20, 22, 5) sont dominés par un autre label (18, 22, 2). Au sommet final, nous obtenons trois labels non-dominés. C'est-à-dire que la procédure nous donne trois solutions possibles :

- La solution 1 : $T(1) = \{0+, 1+, 1-, 0-\}$, $T(2) = \{0+, 2+, 2-, 0-\}$, $T(3) = \{0+, 3+, 3-, 4+4-, 5+, 5-, 0-\}$;

- La solution 2 : $T(1) = \{0+, 1+, 1-, 2+, 2-, 0-\}$, $T(2) = \{0+, 0-\}$, $T(3) = \{0+, 3+, 3-, 4+\ 4-, 5+, 5-, 0-\}$;
- La solution 3 : $T(1) = \{0+, 1+, 1-, 2+, 2-, 3+, 3-, 0-\}$, $T(2) = \{0+, 0-\}$, $T(3) = \{0+, 4+ 4-, 5+, 5-, 0-\}$

Nous constatons que la solution de départ n'existe plus dans ces trois solutions.

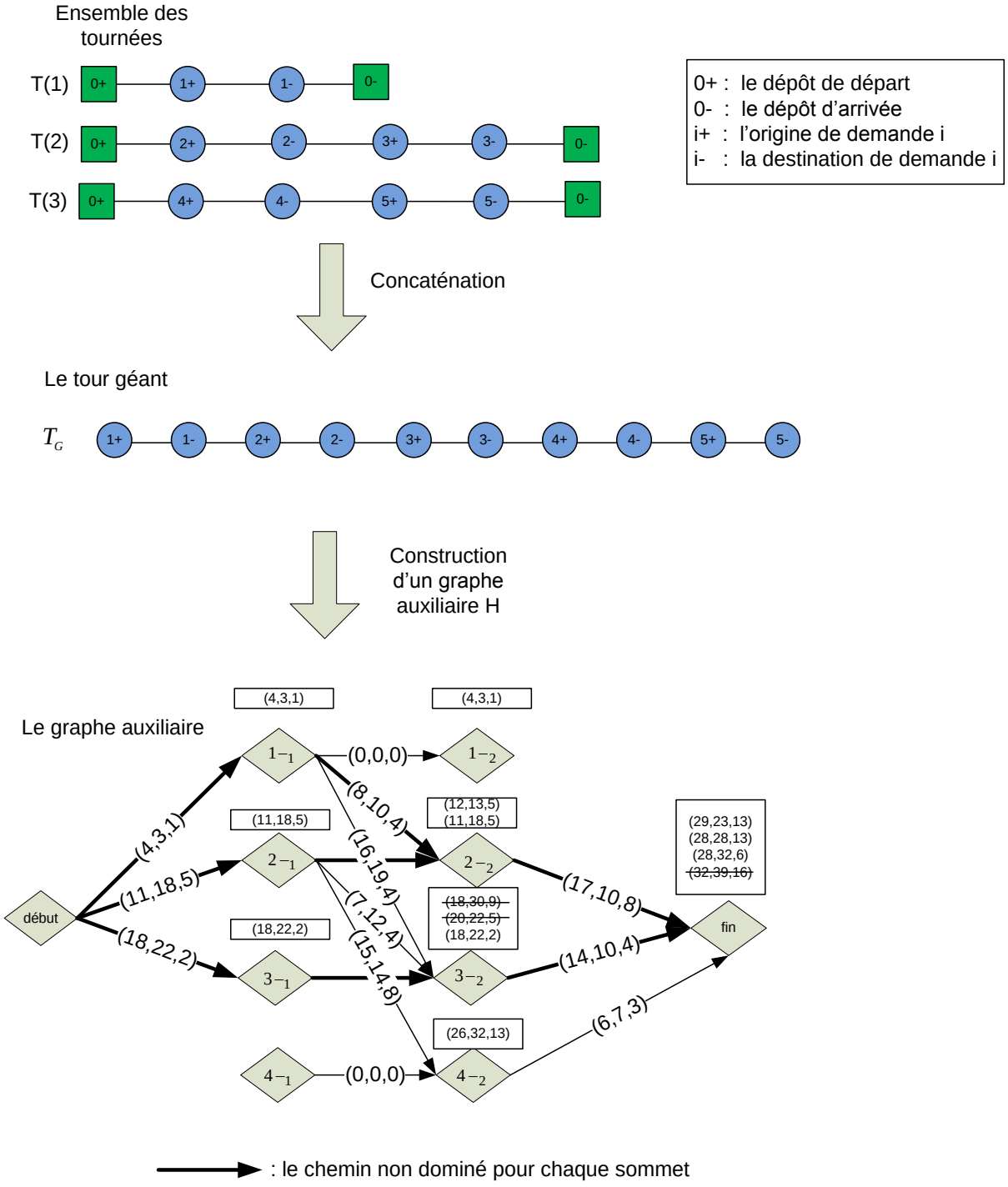

Figure 39 : Le SPLIT pour DARP multicritère

L'application du SPLIT à partir d'une solution du DARP, peut fournir plusieurs solutions dans le cas multicritère. Nous devons alors en choisir une, afin d'effectuer la boucle de recherche locale présente dans le schéma ELS. La section suivante présente la manière d'effectuer ce choix.

5.2. La direction de recherche locale

Nous travaillons sur une population de solutions. Lorsque la construction de la population $\mathcal{F}$ est terminée, la population est classée en nf groupes (voir Figure 40). Chaque groupe Θ_i, $i = 1, \dots, nf$ est un ensemble de solutions non-dominées. A chaque individu d'un groupe, on associe une direction pour la recherche locale.

Algorithme 27 : Classification de la population

$i := 1$;
Tant que $\mathcal{F}! = \phi$ **Faire**
| Chercher le front de Pareto Θ_i dans la population $\mathcal{F}$;
| $\mathcal{F} := \mathcal{F} - \Theta_i$;
| $i := i + 1$;
Fin

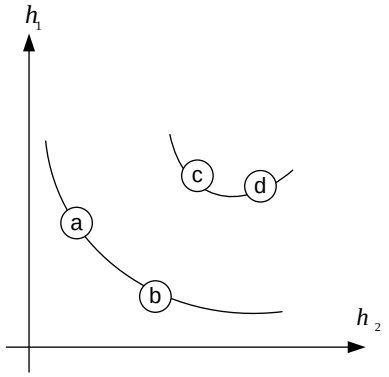

Figure 40 : Exemple de classification de la population

Nous représentons quatre directions pour la recherche locale (voir Figure 41).

- La première direction d_1 est utilisée lorsque l'on minimise le critère h_1 (T_{Gloabl});
- La seconde direction d_2 est utilisée lorsque l'on minimise le critère h_2 (T_{Ride});
- La troisième direction d_3 est utilisée lorsque l'on minimise le critère h_3 (T_{Wait});
- La quatrième direction d_4 est utilisée lorsque la minimisation concerne les trois critères en même temps.

Ces quatre directions sont définies de la manière suivante :

- Si $S \in \Theta_i, i = 1, \dots, nf$ a le plus mauvais critère en h_1 dans le groupe Θ_i, la direction d_1 est alors associée à cette solution;
- Si $S \in \Theta_i, i = 1, \dots, nf$ a le plus mauvais critère en h_2 dans le groupe Θ_i, la direction d_2 est alors associée à cette solution;
- Si $S \in \Theta_i, i = 1, \dots, nf$ a le plus mauvais critère en h_3 dans le groupe Θ_i, la direction d_3 est alors associée à cette solution;
- Sinon la direction d_4 est associée à cette solution.

La Figure 41 montre un exemple de directions de recherche locale pour un groupe Θ_i de solutions. Si le nombre d'individus du groupe Θ_i est égal à 3, on n'utilise que les trois directions d_1, d_2, d_3. Si le nombre d'individus du groupe Θ_i est égal à 2, on n'utilise que les directions d_1, d_2. Si le nombre d'individus du groupe Θ_i est égal à 1, on n'utilise alors que la direction d_4.

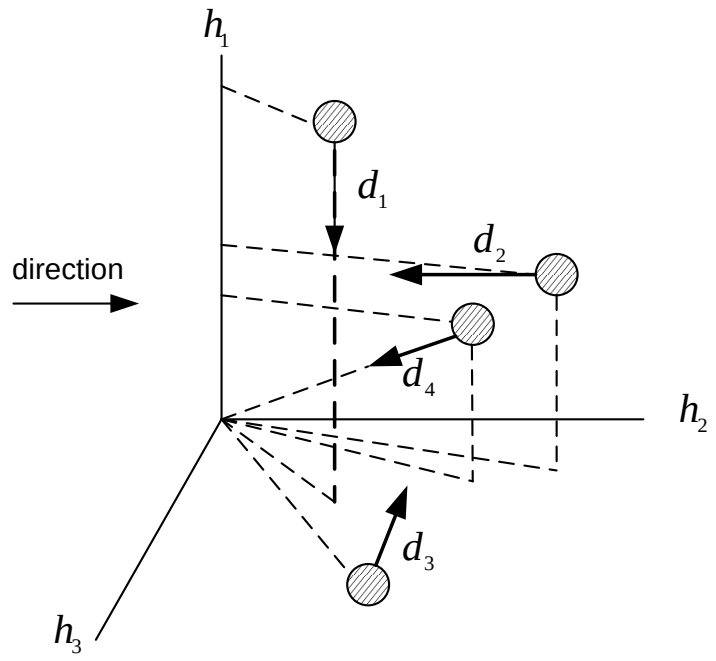

Figure 41 : Directions de recherche locale

En plus du choix du critère à améliorer, la direction d_i de recherche locale peut être utilisée pour :

- Sélectionner une solution dans un ensemble de solutions non-dominées obtenues par la procédure SPLIT :
  - Si $i = 1,2,3$, la solution qui a le meilleur critère h_i est sélectionnée;
  - Si $i = 4$, la solution est sélectionnée aléatoirement.

- Sélectionner une meilleure solution-fille : lorsqu'une génération est terminée, nous devons sélectionner une meilleure solution parmi les nc+1 solutions (nc solution-filles et la solution courante) pour remplacer la solution courante. Nous trouvons d'abord le front de Pareto dans cet ensemble de solutions. Si une solution domine les autres, cette solution est alors la meilleure. Sinon, nous utilisons la direction d_i de recherche locale pour sélectionner la meilleure solution dans le front de Pareto :
  - Si $i = 1,2,3$, la solution-fille qui possède le meilleur critère h_i est sélectionnée;
  - Si $i = 4$, la meilleure solution-fille est sélectionnée aléatoirement.

6. Evaluations numériques

6.1. Programmation et instances

Toutes les procédures ont été codées en Borland C++ 6.0 et exécutées sur un PC avec un processeur 2.4GHz sous Windows Xp avec 4 Go de mémoire. L'évaluation numérique a été réalisée sur 20 instances. Les instances utilisées pour les tests sont celles proposées par Cordeau et Laporte [**77**]. Ils ont créé ces instances afin d'analyser le comportement de leur algorithme de résolution du DARP. Les informations sur les fenêtres de temps, la capacité des véhicules, la durée maximale des tournées et des temps de transport sont fournis par la Commission de transport de Montréal (MTC). Toutes les instances sont disponibles sur Internet à l'adresse : http://www.hec.ca/chairedistributique/data/darp/.

Ces instances, sont composées de 24 et 144 demandes. Dans chaque instance, chaque demande de possède une fenêtre de temps associée, soit à son origine, soit à sa destination. Un temps de service (de durée 10) est imposé pour le chargement et le déchargement. Chaque demande est associée à une charge unitaire. La durée maximale d'une tournée est fixée à 480, la durée de transport maximale pour une demande est 90 et la capacité du véhicule est égale à 6. Les instances sont divisées en deux groupes : (pr01…pr10) et (pr11…pr20). Dans le premier groupe, les fenêtres du temps sont plus serrées. Dans chaque groupe, le nombre de véhicules disponibles pour instance 1 à 6 est tel que les tournées ne sont pas trop chargées. Le nombre de véhicules disponibles pour les instances pr7 à pr10 est supposé être le nombre minimal de véhicules nécessaires.

La distance entre deux lieux x et y est définie comme étant la distance euclidienne entre les coordonnées des lieux x et y. Le temps de transport entre x et y est égale à la distance euclidienne entre x et y.

Une description détaillée de ces instances et de la manière dont elles ont été générées est disponible dans [**77**].

6.2. Résultats Numériques

Les instances sont testées par l'heuristique « DARP_ELS_Multi » avec les paramètres suivants : une population de $np = 10$ individus, un nombre maximum d'itérations $ng = 150$ et un nombre de solution-filles $nc = 10$. Pour chaque instance, les meilleurs critères parmi les 10 solutions sont présentés dans le Tableau 17. Nous constatons que certaines solutions contiennent deux meilleurs critères à la fois. Pour l'instance pr09, une solution contient trois meilleurs critères.

Tableau 17 : Les meilleurs critères obtenus par l'heuristique « DARP_ELS_Multi »

Instance	T_{Global}	T_{Ride}	T_{Wait}	Instance	T_{Global}	T_{Ride}	T_{Wait}
pr01	**793,43**	366,48	**9,32**	pr11	**718,09**	398,11	1,18
	1009,79	**177,86**	211,32		767,48	**151,52**	**0**
pr02	**1589,66**	1767,45	159,4	pr12	**1388,91**	646,1	**0**
	1859,09	**1645,76**	415,52		1484,95	**268,68**	**0**
	1592,08	1819,35	**132,96**				
pr03	**2339,17**	2236,37	67,07	pr13	**2175,58**	2410,9	**0**
	2590,63	**606,21**	78,22		2403,42	**574,93**	**0**
	2480,84	833,63	**0**				
pr04	**2835,88**	3057,31	0,06	pr14	**2771,45**	2828,04	**0**
	3304,47	**999,85**	207,96		3004,33	**652,1**	**0**
	2932,81	2727	**0**				
pr05	**3445,19**	2505,5	20,6	pr15	**3356,83**	3824,71	**0**
	3708,17	**1104,19**	46,96		3662,22	**741,57**	**0**
	3520,04	2262,69	**1,3**				
pr06	**4454,36**	1989,8	20,45	pPr16	**4104,62**	3163,33	**0**
	4928,31	**1277,21**	338,66		4424,05	**1003,91**	**0**
	4454,36	1989,8	**20,45**				
pr07	**1215,82**	605,65	**14,87**	pr17	**1135,36**	560,32	**0**
	1289,9	**301,06**	76,25		1173,16	**250,53**	**0**
pr08	**2235,51**	1745,28	1,66	pr18	**2362,3**	705,39	**9,89**
	2401,22	**1077,19**	18,76		2442,51	**566,3**	70,97
	2280,09	1601,49	**0**				
pr09	**3332,7**	**2326,69**	**0**	pr19	**3175,76**	3415,05	**1,42**
					3518,12	**1141,48**	24,63
pr10	**4600,43**	2967,99	77,92	pr20	**4535,71**	2245,4	4,54
	4653,24	**2162,27**	135,85		4544,65	**2043,64**	**0**
	4653,12	2535,32	**38,93**				

Les résultats complètes obtenus par l'heuristique « DARP_ELS_Multi » sont représentés le Tableau 20.

6.3. Comparaison avec les résultats de Jorgensen et al. [127]

L'algorithme génétique pour le DARP proposé par Jorgensen et al. [**127**] utilise une combinaison linéaire des critères comme la fonction objectif telle que :

$$f(s) = \omega_1 c_1(s) + \omega_2 c_2(s) + \omega_3 c_3(s) + \omega_4 c_4(s) + \omega_5 c_5(s) + \omega_6 c_6(s) + \omega_7 c_7(s)$$

où $c_1(s)$ est la durée totale des tournées pour les clients (sans le temps d'attente du véhicule), $c_2(s)$ est le temps total de transport des clients, $c_3(s)$ est le temps d'attente total pour les clients, $c_4(s)$ est le temps total de tournées pour les véhicules (avec le temps d'attente), $c_5(s)$, $c_6(s)$ et $c_7(s)$ sont les violations sur les fenêtres de temps, le temps de transport des clients et la durée totale de tournée. Les poids sont définis :
$\omega_1 = 8, \omega_2 = 3, \omega_3 = 1, \omega_4 = 1, \omega_5 = n, \omega_6 = n, \omega_7 = n$ où n est le nombre de clients.

Le Tableau 18 montre la comparaison entre deux algorithmes, Les colonnes ont la signification suivante :

- Instance : nom de l'instance ;
- CPU : le temps de calcul est en minute ;
- $T_{Globale}$: la durée totale de tournées ;
- T_{Ride} : la durée totale de transport de demande ;
- T_{Wait} : le temps total d'attente pour les véhicules ;

Chaque instance est exécutée 5 fois avec 15000 itérations, on donne la moyenne des résultats obtenus. En moyenne sur les 13 instances, l'algorithme « DARP_ELS_Multi » donne la durée totale de tournées 11% plus faible, la durée de transport T_{Ride} 2 fois plus faible et le temps d'attente T_{Wait} 3 fois plus faible. Le temps de calcul est comparable entre deux heuristiques.

Tableau 18 : Comparaison des résultats entre notre heuristique et Jorgensen et al. [127]

	l'algorithme génétique [127]				**l'algorithme DARP_ELS_Multi**			
Instance	CPU(min)	T_{Global}	T_{Ride}	T_{Wait}	CPU(min)	T_{Global}	T_{Ride}	T_{Wait}
pr01	5,57	1041,00	477,00	252,00	8,12	930,24	271,98	140,06
pr02	11,43	1969,00	1367,00	470,00	11,96	1697,98	1740,19	246,09
pr03	51,58	2779,00	3081,00	292,00	24,11	2528,39	1046,98	86,33
pr05	58,23	4250,00	5099,00	500,00	47,83	3733,19	1688,26	146,85
pr09	40,78	3597,00	6251,00	94,00	29,68	3386,84	2725,43	7,82
pr10	65,98	5006,00	8413,00	315,00	64,55	4650,06	2508,10	82,10
pr11	5,46	907,00	630,00	143,00	8,49	754,20	350,26	3,35
pr12	11,72	1719,00	1214,00	198,00	23,49	1436,24	575,69	2,16
pr15	58,93	4296,00	4615,00	552,00	56,55	3566,53	1646,79	4,47
pr16	81,23	5309,00	6134,00	630,00	56,43	4235,37	2064,97	0,48
pr17	8,29	1299,00	990,00	102,00	13,76	1164,30	494,90	6,00
pr19	44,66	3679,00	5362,00	147,00	48,64	3449,40	1913,23	52,55
pr20	66,41	4733,00	7969,00	113,00	63,94	4564,68	2402,61	5,22
Moyenne	39,25	3121,85	3969,38	292,92	35,20	2776,72	1494,57	60,27

6.4. Comparaison avec les résultats de Cordeau et Laporte [77]

Les résultats obtenus par Cordeau et Laporte [**77**] sont représentés dans le Tableau 19 en utilisant la recherche taboue. La fonction objectif utilisée par [**77**] est $f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s)$ où $\alpha, \beta, \gamma, \tau$ sont des paramètres positifs auto-réglables. c(s) est le cout des tournées, $q(s), d(s), w(s), t(s)$ sont les violations sur la capacité, la durée de tournée, les fenêtres de temps et le temps de transport. Ces quatre paramètres sont réglables dynamiquement pendant la recherche.

Les résultats donnés par [**77**] sont obtenues avec 10^5 itérations, mais les temps de calcul sont donnés seulement pour 10^4 itérations. Pour notre algorithme « DARP_ELS_Multi », les expérimentations montrent que les résultats sont améliorés très peu après 15000 itérations. Parce que notre algorithme gère simultanément les trois critères, et des critères sont contradictoires lorsque la diminution d'un des critères entraine une augmentation d'un ou plusieurs autres critères. Les résultats donnés par notre algorithme sont obtenues avec 15000 itérations. En moyenne sur les 20 instances, L'algorithme « DARP_ELS_Multi » donne la durée de transport 68% plus faible et le temps d'attente 87% plus faible.

Tableau 19 : Comparaison des résultats entre notre heuristique et Cordeau et Laporte [77]

	La recherche taboue [77]				**L'algorithme DARP_ELS_Multi**			
Instance	CPU(min)	T_{Global}	T_{Ride}	T_{Wait}	CPU(min)	T_{Global}	T_{Ride}	T_{Wait}
pr01	1,90	881,20	1095,00	211,20	8,12	930,24	271,98	140,06
pr02	8,06	1985,90	1976,70	723,90	11,96	1697,98	1740,19	246,09
pr03	17,18	2579,40	3586,70	607,30	24,11	2528,39	1046,98	86,33
pr04	28,77	3583,20	5021,30	1090,40	33,18	3190,31	1785,99	178,73
pr05	46,24	3870,00	6156,50	833,00	47,83	3733,19	1688,26	146,85
pr06	53,87	5056,80	7273,50	1375,40	82,30	4757,45	1675,38	253,11
pr07	4,39	1452,10	1508,90	440,30	9,81	1260,23	399,58	67,70
pr08	20,44	2345,20	3691,50	410,30	24,33	2322,47	1311,37	7,74
pr09	50,51	3155,50	5621,80	323,10	29,68	3386,84	2725,43	7,82
pr10	87,53	4480,10	7163,70	721,30	64,55	4650,06	2508,10	82,10
pr11	1,93	965,10	1041,50	320,60	8,49	754,20	350,26	3,35
pr12	8,29	1564,70	2393,80	308,70	23,49	1436,24	575,69	2,16
pr13	18,54	2263,70	3788,80	330,40	29,04	2316,14	1127,78	0,62
pr14	31,18	2882,20	4632,80	426,30	40,20	2923,62	1183,91	0,09
pr15	54,33	3595,60	6104,70	605,90	56,55	3566,53	1646,79	4,47
pr16	73,70	4072,50	7347,40	448,90	56,43	4235,37	2064,97	0,48
pr17	4,23	1097,30	1762,00	129,00	13,76	1164,30	494,90	6,00
pr18	22,86	2446,50	3659,00	523,40	38,30	2424,98	702,65	65,47
pr19	51,28	3249,30	5581,00	487,30	48,64	3449,40	1913,23	52,55
pr20	92,41	4041,00	7072,30	362,40	63,94	4564,68	2402,61	5,22
Moyenne	33,88	2778,37	4323,95	533,96	35,74	2764,63	1380,80	67,85

La fonction objectif détermine les caractéristiques des tournées. Les résultats obtenus par notre heuristique sont plus favorables aux clients (meilleure qualité de service). L'heuristique [**77**] est plus favorable à l'opérateur.

Nous utilisons la procédure SPLIT pour traiter les solutions finales, certaines solutions (environ 5%) prouvent donner plusieurs solutions non dominées.

7. Conclusion

Ce chapitre présente une méthode de résolution du DARP dans le cas multi-objectif et propose une méthode pour déterminer un ensemble de solutions non-dominées. L'approche est basée sur la méta-heuristique ELS et utilise le front de Pareto. Les éléments initiaux sont générés par une heuristique. Les résultats obtenus montrent que la méthode peut rivaliser avec la méthode TABU [**77**] et la méthode GA [**127**].

Les travaux présentés dans ce chapitre ont fait l'objet d'une publication, présentée à la conférence internationale CIE39 2009.

Tableau 20 : Solutions obtenues par l'heuristique « DARP_ELS_Multi »

Instance	CPU [min]	T_{Global}	T_{Ride}	T_{Wait}
pr01	8,12	1009,79	177,86	211,32
		836,21	343,27	48,96
		1099,2	151,52	297,53
		793,43	366,48	9,32
		943,47	193,34	140,18
		792,95	389,39	23,15
		1088,29	188,76	289,67
		828,76	435,9	53,62
		809,38	293,94	31,5
		1100,91	179,29	295,33
pr02	11,96	1704,64	1834,08	215,79
		1859,09	1645,76	415,52
		1592,08	1819,35	132,96
		1846,9	1709,28	405,82
		1681,41	1789,24	261,53
		1747,66	1509,75	253,4
		1610,89	1689,94	164,1
		1632,19	1939,96	193,89
		1589,66	1767,45	159,4
		1715,29	1697,1	258,44
pr03	24,11	2618,81	765,24	143,7
		2483,44	832,03	0
		2611,2	826,67	134,3
		2605,71	663,54	71,43
		2480,84	833,63	0
		2590,63	606,21	78,22
		2360,53	1932,65	51,52
		2588,44	986,35	178,16
		2605,1	787,07	138,88
		2339,17	2236,37	67,07
pr04	33,18	3589,67	1145,26	510,32
		3304,47	999,85	207,96
		3232,96	1072,64	134,54
		2866,13	2417,16	0,52
		3564,28	1043,28	424,37
		2932,81	2727	0
		3571,33	1026,34	413,84
		2853,39	3155,61	0,06
		2835,88	3057,31	0,06
		3152,22	1215,47	95,59
pr05	47,83	3563,91	2255,99	71,31
		3999,04	1004,47	289,34
		3520,04	2262,69	1,3
		3911,62	1176,78	255,84
		3916,81	1134,42	257,45
		4064,58	1153,19	393,47
		3443,33	2475,52	20,6
		3708,17	1104,19	46,96
		3524,43	2279,8	0,28
		3679,95	2035,53	131,95

pr06	82,3	4454,36	1989,8	20,45
		4805,56	1684,7	275,99
		5172,69	1311,54	597,21
		4567,15	1929,98	134,52
		5021,27	1376,23	414,24
		4928,31	1277,21	338,66
		5043,72	1522,61	502,96
		4500,61	2131,62	33,29
		4489,73	2023,17	79,99
		4591,05	1506,9	133,75
pr07	9,81	1246,58	389,85	66,78
		1259,96	420,35	73,91
		1296,04	415,63	139,35
		1259,96	420,35	73,91
		1223,56	430,19	29,75
		1215,82	605,65	14,87
		1281,13	301,06	69,72
		1283,64	330,46	89,68
		1289,9	301,06	76,25
		1245,75	381,23	42,78
pr08	24,33	2312,8	866,74	3,19
		2331,37	942,37	0,84
		2235,51	1745,28	1,66
		2273,43	1641,22	8,84
		2280,55	1605,71	0
		2401,22	1077,19	18,76
		2380,21	1171,17	20,53
		2389,16	1356,25	8,63
		2280,09	1601,49	0
		2340,37	1106,24	14,96
pr09	29,68	3343,08	2595,11	0
		3424,38	2936,84	0
		3399,42	2637,04	21,15
		3338,09	2772,58	31,11
		3483,17	3255,27	0
		3332,7	2326,69	0
		3378,45	2502,88	0
		3389,97	2906,81	2,11
		3370,35	2644,16	0
		3408,75	2676,91	23,82
Pr10	64,55	4656,99	2461,18	104,34
		4681,12	2200,62	49,04
		4678,95	2238,08	69,36
		4679,58	2277	61,22
		4600,43	2967,99	77,92
		4625,22	2929,1	77,94
		4624,57	2754,05	110,06
		4647,41	2555,36	96,33
		4653,12	2535,32	38,93
		4653,24	2162,27	135,85
pr11	8,49	767,48	151,52	0
		760,89	284,3	0
		761,95	561,11	0
		749,92	372,93	5,93
		757,55	420,34	0
		782,96	371,96	2,89
		776,42	263,71	23,49
		747,34	305,47	0

		718,09	398,11	1,18
		719,39	373,14	0
pr12	23,49	1484,95	268,68	0
		1411,42	601,39	0
		1414,1	590,02	0
		1388,91	646,1	0
		1461,46	574,29	0
		1437,53	576,58	0
		1459,9	610,09	3,36
		1445,45	661,71	0
		1428,93	580,26	0
		1429,79	647,74	0
pr13	29,04	2203,39	2140,08	0
		2403,42	574,93	0
		2175,58	2410,9	0
		2254,37	1073,58	0
		2401,16	652,31	0
		2388,29	600,15	0
		2387,87	617,34	0,62
		2422,36	595,81	0
		2313,35	1079,8	0
		2211,65	1532,91	0
pr14	40,2	3004,33	652,1	0
		3020,92	673,17	0
		2960,62	1011,2	0
		2930,54	955,7	0
		2925,71	1084,28	0
		2935,79	1010,86	0
		2939,63	955,69	0,9
		2814,84	1650,66	0
		2771,45	2828,04	0
		2932,35	1017,39	0
pr15	56,55	3663,86	719,59	0
		3668,95	740,52	0
		3381,29	2859,79	0
		3664,51	743,1	4,69
		3577,05	1022,84	0
		3594,3	960,74	0
		3356,83	3824,71	0
		3395,27	3843,68	0
		3662,22	741,57	0
		3701	1011,31	39,97
pr16	56,43	4424,05	1003,91	0
		4187,46	2151,4	4,62
		4182,45	2286,26	0
		4160,2	2291,46	0
		4178,8	2303,93	0
		4185,01	2419,59	0,22
		4383,8	1150,38	0
		4143,94	2833,42	0
		4403,33	1046,02	0
		4104,62	3163,33	0
pr17	13,76	1188,17	660,38	0
		1173,16	250,53	0
		1221,43	345,33	41,42
		1166,55	457,16	0
		1161,27	533,6	0
		1135,42	550,71	0

		1135,36	560,32	0
		1142,18	749,3	1,6
		1164,71	383,05	9,41
		1154,77	458,62	7,52
pr18		2381,66	664,49	14,43
		2423,46	820,32	104,74
		2395,78	644,79	38,18
		2362,3	705,39	9,89
		2456,12	749,03	104,04
		2505,6	629,14	122,45
		2408,62	845,17	66,17
		2460,39	772,55	82,14
		2442,51	566,3	70,97
		2413,39	629,32	41,65
pr19	48,64	3518,12	1141,48	24,63
		3527,9	1320,33	57,93
		3515,29	1183,22	37,28
		3548,27	1227,6	63,13
		3548,17	1312,23	43,3
		3264,97	2901,19	3,23
		3607,96	1792,3	170,27
		3175,76	3415,05	1,42
		3582,25	1241,03	118,05
		3205,26	3597,91	6,24
Pr20	63,94	4583,19	2549,4	11,79
		4564,61	2531,72	0
		4547,13	2507,37	0
		4544,52	2567,92	0
		4579,63	2765,7	11,26
		4612,94	2439,33	17,11
		4552,07	2064,25	7,47
		4582,36	2311,41	0
		4535,71	2245,4	4,54
		4544,65	2043,64	0

CHAPITRE 6
DARP AVEC CONTRAINTE FINANCIERE

Dans ce chapitre, une extension du DARP est présentée prenant en compte des contraintes financières. Nous traitons ce problème en utilisant des heuristiques d'insertion et des techniques de propagation par contraintes.

1. Introduction

Les chapitres précédents présentent des heuristiques pour résoudre le problème du DARP. Dans ce chapitre, nous proposons une nouvelle extension à ce problème intégrant des contraintes financières : le Dial-a-ride avec contraintes financières (noté DARPF). Dans un premier temps, nous construisons un modèle qui étend le modèle conçu pour le DARP. Ce nouveau modèle intègre un coût financier pour chaque trajet en supposant que le client assure le paiement de sa course une fois arrivée à destination. On suppose disposer d'un avoir initial permettant d'assurer le paiement des frais pendant la course en attendant le paiement du client. A chaque instant, la quantité totale d'argent disponible doit être positive ou nulle ce qui signifie qu'on interdit les découverts bancaires. Nous proposons, pour résoudre ce problème, une approche basée sur l'heuristique d'insertion présentée dans le chapitre 3.

1.1. Définition du DARPF

La définition du DARP est présentée dans le chapitre 2. Le DARPF intègre de plus les hypothèses suivantes :

- Initialement, le gestionnaire de la flotte de véhicules fournit un avoir initial $\mathcal{R}$;
- Tout arc (x, y) , $x \neq y$ et $x, y \in X$ où $X = \{o_i, d_i | i \in I\} \cup \{DepotD(k), DepotA(k) | \ k \in \{1, \dots, K\}\}$ (I est l'ensemble d'indice des demandes et

K est le nombre de véhicules) est associé à un coût financier $Cost(x, y) \geq 0$, payable en x;

- Lorsqu'on arrive à une destination $d_i, i \in I$, la livraison rapporte un revenu R_i.

Nous supposons que les revenus et les dépenses ne concernent ni les temps de communication ni les coûts de communication. Nous supposons de plus que toutes les opérations financières sont directement effectuées par l'entité qui gère la flotte de véhicules.

D'une manière naturelle, la résolution de l'instance du DARP avec contraintes financières (DARPF) consiste à calculer le couple (T, t) de telle manière que:

- elle définit une solution faisable du DARP standard;
- à tout instant τ, l'équilibre financier $Global\_Cash(T, t, \tau)$ qui est la différence entre les revenus qui ont été reçus par le système pendant l'intervalle $[0, \tau]$ et les dépenses qui ont été payées au cours de la même période, est toujours **positive**; (**Contrainte financière**)
- la valeur $Perf_{\gamma,\delta,\varepsilon}(T, t)$ définie dans les chapitres précédents, est la plus petite possible.

Remarque

Nous considérons ici une ressource non renouvelable que nous interprétons comme une ressource financière. La démarche proposée par la suite pourrait tout à fait s'adapter à d'autres types de ressources non renouvelables comme l'énergie. La méthode peut aussi être adaptée à des ressources renouvelables (ressources humaines, dispositifs techniques).

2. La formulation linéaire du DARPF

Dans cette section, nous proposons un programme linéaire en nombres entiers pour le DARPF. Ce modèle généralise le modèle de Cordeau et al. [**62**,**77**] pour le DARP. Il est composé de deux parties: une partie concerne la formulation de DARP, l'autre concerne la formulation des contraintes financières. La fonction objectif est de minimiser : $\sum_{k=1,\dots,K}(\gamma \times T_{Globale}(k) + \delta \times T_{Ride} + \varepsilon \times T_{Wait})$.

2.1. Les contraintes du DARP

Pour chaque arc$(x, y) \in A$ (A est l'ensemble des arcs du graphe) et chaque véhicule $k \in \{1, \dots, K\}$, nous notons $Z_{x,y}^k = 1$ si et seulement si le véhicule k va du sommet x au sommet y.

Pour chaque sommet $x \in X$, $t(x)$ est la date de début de service au sommet x. $CH(x)$ est la charge du véhicule après avoir visité le sommet x après la visite d'un véhicule. $\Delta(x)$ est le temps maximal de transport.

Les contraintes du DARP se décomposent comme suit :

- Le vecteur T définit des « tournées » $T(k), k \in \{1, \dots, K\}$, de telles sorte que chaque sommet $x \in X$, apparait exactement une fois dans une tournée $T(k), k \in \{1, \dots, K\}$:
  - $DepotD(k)$ et $DepotA(k)$ sont dans la même tournée $T(k)$, en début et fin (voir la formule 6.1), et ne sont dans aucune autre tournée $T(k')$ $k \neq k'$(voir la formule 6.2):

$$\forall k \in \{1, \dots, K\} \qquad \sum_{x \in X} Z^k_{DepotD(k),x} = \sum_{x \in X} Z^k_{x,DepotA(k)} = 1 \qquad \textbf{6.1}$$

$$\sum_{\substack{k \in \{1,\dots,K\}, \\ x \in X}} Z^k_{DepotD(k),x} = \sum_{\substack{k \in \{1,\dots,K\}, \\ y \in X}} Z^k_{y,DepotA(k)} = 1 \qquad \textbf{6.2}$$

  - Si un sommet x (hormis le dépôt) est origine dans une tournée k, alors il y apparaît aussi comme destination. Nous notons un ensemble J de sommets $J = X - \left\{DepotD\big(T(k)\big), DepotA\big(T(k)\big) \,\middle|\, k \in \{1, \dots, K\}\right\}$:

$$\forall x \in J \qquad \sum_{\substack{y \in X, \\ k \in \{1,\dots,K\}}} Z^k_{x,y} = \sum_{\substack{y \in X, \\ k \in \{1,\dots,K\}}} Z^k_{y,x} = 1 \qquad \textbf{6.3}$$

$$\forall x \in J, \forall k \in \{1, \dots, K\} \qquad \sum\nolimits_{y \in X} Z^k_{x,y} = \sum\nolimits_{y \in X} Z^k_{y,x} \qquad \textbf{6.4}$$

  - Les sommets o_i et d_i sont dans la même tournée (6.6), et d_i est après o_i (6.5). ici, nous notons $Hom(x) = d_i$ si $x = o_i$

$$\forall x \in \{o_i | i \in I\} \qquad t(Hom(x)) \geq t(x) \qquad \textbf{6.5}$$

$$\forall x \in \{o_i | i \in I\}, \ \forall k \in \{1, \dots, K\} \qquad \sum\nolimits_{y \in X} Z^k_{x,y} = \sum\nolimits_{y \in X} Z^k_{y,Hom(x)} \qquad \textbf{6.6}$$

- Les tournées sont « Charge-Admissibles », on pose $D^* = \sum_{i \in I} c_i$ et on obtient alors :

$$\forall x \in X \qquad CH(x) \leq CAP \qquad \textbf{6.7}$$

$\forall k \in \{1, \dots, K\}, \forall x, y \in X$ $\quad Z_{x,y}^k$ implique $CH(y) \geq CH(x) + c(y)$

qui se linéarise :

$$Z_{x,y}^k + \frac{CH(x) + c(y) - CH(y)}{D*} \leq 1 \qquad \textbf{6.8}$$

- Les tournées sont « Temps-Admissibles », on obtient alors :

$$\forall x \in \{o_i | i \in I\} \qquad t(Hom(x)) - t(x) \leq \Delta(x) \qquad \textbf{6.9}$$

$$\forall x \in X \qquad \alpha(x) \leq t(x) \leq \beta(x) \qquad \textbf{6.10}$$

$\forall x, y \in X, \forall k \in \{1, \dots, K\}$ $\quad Z_{x,y}^k = 1$ implique $t(y) - t(x) \leq Dist[x, y]$

qui se linéarise :

$$Z_{x,y}^k + \frac{Dist[x, y] + t(x) - t(y)}{D*} \leq 1 \qquad \textbf{6.11}$$

2.2. Les contraintes financières

Pour représenter linéairement les contraintes financières, il est possible d'adapter la propostion faite par Artigues et al. sur le RCPSP en 2003 [**135**]. Les auteurs en effet, modélisent les consommations de ressources dans un RCPSP comme un problème de flot. Afin d'exprimer les contraintes financières, nous utilisons des flots (ou flux). Nous supposons posséder un avoir initial $\mathcal{R}$, une matrice de coûts financiers $Cost$, et un revenu $r(x)$ pour tout sommet $x \in X$. Pour chaque arc$(x, y) \in A$, nous notons $u_{x,y} = 1$ s'il existe un flux depuis le sommet x vers le sommet y. $F_{x,y}$ est la quantité d'argent envoyée depuis le sommet x vers le sommet y. $N = \{X\} \cup \{Source, Puits\}$ est un ensemble de sommets comprenant deux sommets fictifs $Source$ et $Puits$. Soit H une constante suffisamment grande.

Les contraintes de flux financiers se décomposent comme suit :

- Pour tout sommet sauf le sommet $Source$, il peut exister plusieurs flux entrants (6.12):

$$\forall y \in N - \{Source\} \qquad \sum_{x \in N - \{Puits\}} u_{x,y} \geq 1 \qquad \textbf{6.12}$$

- Pour tout sommet sauf $Puits$, il peut exister plusieurs flux sortants (6.13) :

$$\forall x \in N - \{Puits\} \qquad \sum_{y \in N - \{Source\}} u_{x,y} \geq 1 \qquad \textbf{6.13}$$

- S'il existe un flux financier envoyé depuis le sommet x vers le sommet y tel que $u_{x,y} = 1$, alors la quantité de flux doit être non nulle (6.14) :

$$\forall x, y \in N \qquad F_{x,y} \leq H * u_{x,y} \qquad \textbf{6.14}$$

- S'il existe un flux financier du sommet x vers le sommet y, c'est-à-dire $u_{x,y} = 1$ alors la date de début de service au sommet y (noté $t(y)$) doit être supérieure ou égale à la date $t(x)$ (6.15) :

$$\forall x, y \in X \qquad \big(t(y) - t(x)\big)u_{x,y} \geq 0 \qquad \textbf{6.15}$$

qui se linéarise :

$$\forall x, y \in X \qquad t(y) - t(x) + H\big(1 - u_{x,y}\big) \geq 0 \qquad \textbf{6.16}$$

- A chaque sommet x, la quantité totale d'argent disponible ne doit pas être négative (6.17) :

$$\forall x \in X \qquad \sum_{y \in N - \{Puits\}} F_{y,x} + r(x) - \sum_{k \in \{1,\dots,K\}} \sum_{y \in X} Z_{x,y}^{k} Cost[x,y] = \sum_{y \in N - \{Source\}} F_{x,y} \qquad \textbf{6.17}$$

- l'ensemble des flux financiers ayant pour origine $Source$ ne doit pas dépasser l'avoir initial (6.18) :

$$\sum_{x \in N} F_{Source,x} \leq \mathcal{R} \qquad \textbf{6.18}$$

- Lorsque toutes les demandes sont traitées (fin de service), la quantité totale d'argent disponible doit être positive (6.19) :

$$\sum_{x \in N} F_{x,Puits} \geq 0 \qquad \textbf{6.19}$$

3. Résolution heuristique

Dans cette section, nous proposons une heuristique pour le DARPF. Nous notons d'abord :

- $r(x) = R_i$ si x est la destination de la demande D_i, $i \in I$;
- $r(x) = 0$ sinon.

Pour toute tournée valide (ou admissible) Γ, et pour tout sommet x dans la tournée Γ, tel que $Type(x) \neq DepotA$ (dépôt d'arrive), on note :

- $Cash(\Gamma, x) = \sum_{y \in Segment(\Gamma, Debut(\Gamma), x)} \Big(r(y) - Cost\big(y, succ(\Gamma, y)\big)\Big)$;
- Si $t = (t(x), x \in \Gamma)$ est l'ensemble des dates liées à la tournée Γ, pour $\tau \in [0, +\infty[$:
  - $Node(\Gamma, t, \tau) =$ le sommet qui a la plus grande valeur $t(x) \leq \tau$;
  - $Tour\_Cash(\Gamma, t, \tau) = Cash(\Gamma, Node(\Gamma, t, \tau))$.

La Figure 42 présente un exemple de calcule de $Cash(\Gamma, x)$ pour la tournée Γ. Chaque arc est associé à une dépense (coût financier). Chaque sommet est associé une quantité de revenu $r(x)$. Nous essayons de calculer $Tour\_Cash(\Gamma, t, \tau)$ pour les dates $t1$, $t2$ et $t5$:

- $Node(\Gamma, t, t1) = 0$, $Tour\_Cash(\Gamma, t, t1) = Cash(\Gamma, 0) = -5$;
- $Node(\Gamma, t, t2) = 1+$, $Tour\_Cash(\Gamma, t, t2) = Cash(\Gamma, 1+) = -15$.
- $Node(\Gamma, t, t5) = 1-$, $Tour\_Cash(\Gamma, t, t5) = Cash(\Gamma, 1-) = 7$.

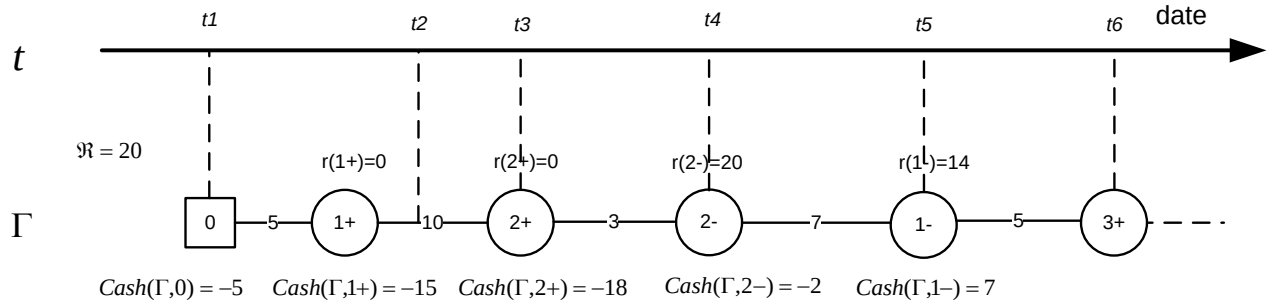

Figure 42 : Exemple des $Cash(\Gamma, x)$

3.1. La solution du DARPF

La solution du DARPF doit être une solution faisable (T, t) pour le DARP standard qui est définie par un ensemble des demandes $D = (D_i = (o_i, d_i, c_i, F(o_i), F(d_i), \Delta_i), i \in I)$, par un 4-uple $(X, Dist, K, CAP)$ où X est l'ensemble de sommet, $Dist$ est la matrice des distances entre deux sommets, K est le nombre de véhicules, CAP est la capacité commune de véhicules. De plus, la solution du DARPF doit satisfaire à la contrainte financière additionnelle définie par $Cost$, $\mathcal{R}$ et $R_i, i \in I$:

- à tout instant $\tau \in [0, +\infty[$:

$$Global\_Cash(T, t, \tau) = \mathcal{R} + \sum_{k \in \{1,\dots,K\}} Tour\_Cash(T(k), t, \tau) \geq 0$$

Donc, la résolution d'une instance du DARPF définie par un ensemble des demandes $D = (D_i = (o_i, d_i, c_i, F(o_i), F(d_i), \Delta_i), i \in I)$, par un 6-uple $(X, Dist, K, CAP, Cost, \mathcal{R})$, consiste à calculer un couple (T, t) telle que :

- les contraintes standards sont vérifiées;
- la contrainte financière est vérifiée;
- la valeur $Perf_{\gamma,\delta,\varepsilon}$ est la plus petite possible.

3.2. La stratégie de résolution

Nous rappelons le schéma général de résolution du DARP qui a été présenté dans le chapitre 3 :

1. Initialement, les tournées $T(k)$ sont initialisées à $\{DepotD(T(k)), DepotA(T(k))\}$ pour $k = \{1, \dots, K\}$. La liste $FREE(i)$ des insertions possibles est construite pour chaque demande D_i , $i \in I$ non encore insérée. $FREE(i)$ contient des 4-uples (k, x, y, v), signifiant que la demande D_i peut être insérée dans la tournée $T(k)$ de sorte que le somme o_i soit placé après $x \in T(k)$ et d_i soit placé après $y \in T(k)$. La valeur v est le coût (kilométrique) de cette insertion. La liste $N\_FREE(i)$ est l'ensemble des véhicules pouvant traiter la demande D_i ;

2. A chaque itération, on choisit une demande D_{i_0} non insérée parmi celles qui sont les plus contraintes de la manière suivante :
   - S'il existe des demandes D_i non insérées et telles que la liste $FREE(i)$ associée contient un seul élément, alors elles sont prioritaires (D_{i_0} est choisit aléatoirement parmi celles-ci) ;
   - Sinon on choisit aléatoirement D_{i_0} la demande non insérée parmi les demandes D_i de plus petites valeurs $N\_FREE(i)$.

3. Pour chaque demande D_{i_0} choisie, nous choisissons la liste $L\_Candidate$ de 4-uple (k_0, x_0, y_0, v_0) dans $FREE(i_0)$ qui possède la meilleure (plus petite) valeur v.

4. La demande D_{i_0} est insérée dans la tournée $T(k_0)$ selon les sommets x_0 et y_0 ;

5. On met à jour les listes $FREE(i)$ et $N\_FREE(i)$ pour les demandes D_i, $i \in I$ non encore insérées.

Cette stratégie peut être adaptée au DARPF en modifiant simplement les heuristiques basées sur les techniques d'insertion et de propagation de contraintes. Les contraintes financières sont prises en compte au moment de l'insertion d'une nouvelle demande D_{i_0}.

3.3. Contrainte temporelle supplémentaire

Nous considérons le couple (T, t), où $T = (T(k), k \in \{1, \dots, K\})$ est un ensemble des tournées qui assure le transport d'un sous-ensemble de demandes $\{D_i | i \in I_1\}$ et une demande D_{i_0}, $i_0 \notin I_1$. L'insertion de la demande D_{i_0} dépend des sommets d'insertion x_0 et y_0. Les flux financiers imposent des contraintes temporelles supplémentaires : s'il existe un flux financier envoyé depuis le sommet x au sommet x' alors on doit avoir : $t(x) \leq t(x')$.

Une contrainte a été imposée à un moment pendant le processus d'insertion afin d'assurer l'insertion de la demande D_{i_0} dans une tournée $T(k_0)$. A chaque insertion, les fenêtres de temps tiennent compte des contraintes supplémentaires liées à la ressource financière. Ces fenêtres de temps vont donc varier d'une manière non monotone, et elles deviennent interdépendantes.

Nous décidons de traiter la contrainte financière en effectuant l'insertion de la demande D_{i_0} dans une tournée $T(k_0)$ selon les sommets d'insertion x_0 et y_0 de la manière suivante : toutes les valeurs $t(x), x \in T(k), k \neq k_0$ sont considérées fixées. Cette hypothèse permet de considérer le problème lié à la faisabilité de l'insertion de la demande D_{i_0} dans une tournée $T(k_0)$ comme le calcul des valeurs $t(x), x \in INSERT(T(k_0), x_0, y_0, i_0)$.

Afin de formaliser ce problème, nous construisons d'abord une liste L_T des sommets, qui est la fusion de toutes les tournées $T(k), k \neq k_0$, triée suivant les valeurs croissantes de $t(x), x \in L\_T$. A chaque sommet x de la séquence L_T, nous associons une quantité $B\_Cash(L\_T, x)$ définie par :

$$B\_Cash(L\_T, x) = \mathcal{R} + \sum_{k \neq k_0} Cash(T(k), Node(T(k), t, t(x)))$$

Pour toute valeur de date τ, tous les sommets x dans L_T tels que $t(x) = \tau$ sont fusionnés en un unique sommet $x(\tau)$. Ce sommet $x(\tau)$ est associé à une valeur $B\_Cash(L\_T, x(\tau))$ qui est la somme des quantités $B\_Cash(L\_T, x)$, où x est tel que $t(x) = \tau$. Pour une tournée Γ, on définit une quantité $Global\_Cash(L\_T, t, \tau) = B\_Cash(L\_T, B\_Node(L\_T, t, \tau))$ pour toute date τ où $B\_Node(L\_T, t, \tau)$ est le plus grand sommet x dans L_T (vis à vis de la relation $\ll_{L\_T}$) tel que $t(x) \leq \tau$.

Un exemple de génération de la suite est présenté par la Figure 43. Dans cette figure, la suite L_T est générée par deux tournées $T(k_1)$ et $T(k_2)$. Chaque sommet $x \in L\_T$ est associée à

une valeur $B\_Cash(L\_T, x)$. Nous constatons que les débuts de service $t(2+)$ et $t(3+)$ sont identiques. Nous utilisons le sommet « $3+$ » pour présenter ces deux sommets dans la figure. Apres la génération de la séquence, on calcule la valeur B_Cash et $Global\_Cash$ pour tous les sommets.

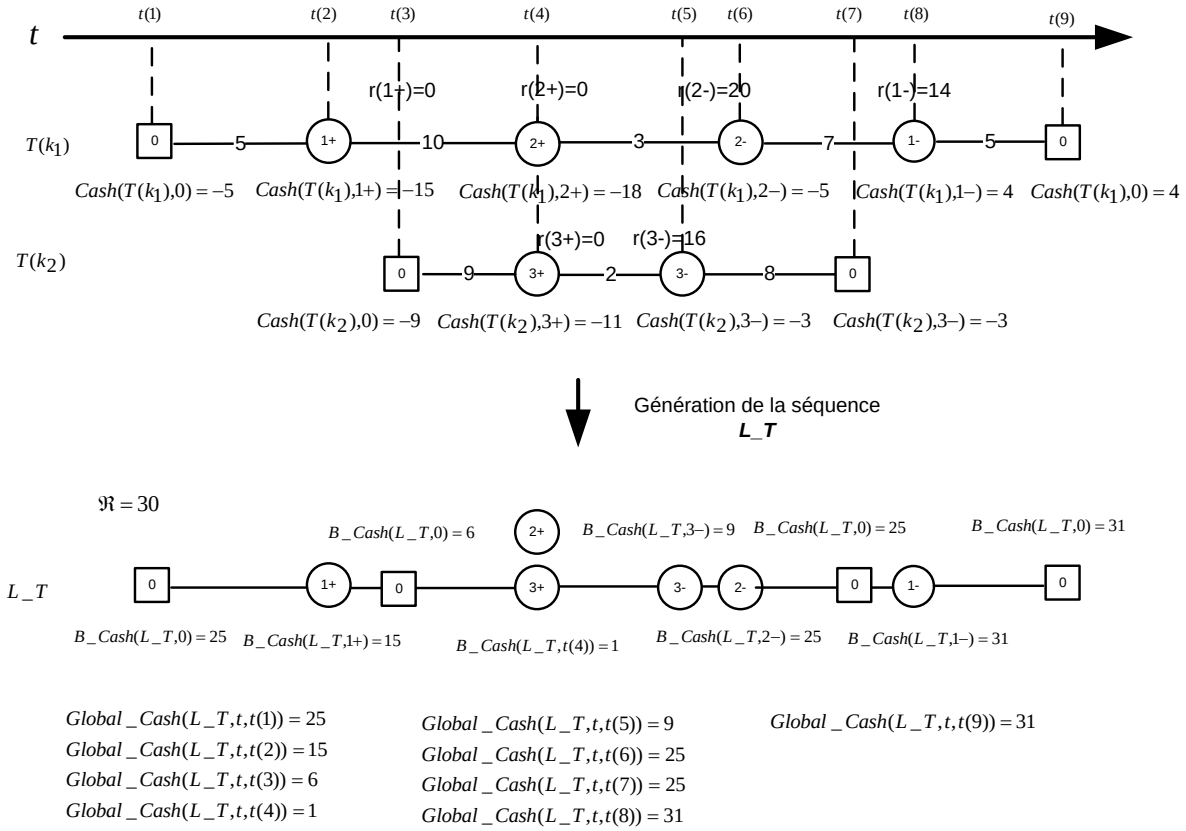

Figure 43 : Exemple de suite *L_T*, *B_Cash* et *Global_Cash*

3.4. Le problème « Cash_Insertion »

On considère une nouvelle tournée admissible $\Gamma = INSERT(T(k_0), x_0, y_0, i_0)$, ainsi que son ensemble de fenêtres de temps $FP(\Gamma)$ et les valeurs $Cash(\Gamma, x), x \in \Gamma$. Nous voyons que les tests sur la compatibilité financière de l'insertion de la demande D_{i_0} dans la tournée $T(k_0)$ consiste à vérifier l'existence de valeurs $t'(x), x \in \Gamma$ qui satisfont :

- $\forall x \in \Gamma, t'(x) \in FP(\Gamma)$;
- $\forall \tau, Global\_Cash(L\_T, t, \tau) + Tour\_Cash(\Gamma, t', \tau) \geq 0$.

Cela nous amène à définir un problème auxiliaire « $Cash\_Insertion$ » comme suit:

Le problème « $Cash\_Insertion$ » :

- Entrée :

- la suite L_T, l'ensemble de dates t et les valeurs $Cash(L\_T, x), x \in L\_T$;
- la tournée Γ , les valeurs $Cash(\Gamma, x), x \in \Gamma$, les fenêtres de temps $FP(\Gamma)(x), x \in \Gamma$;
- les trois coefficients $\gamma, \delta, \varepsilon$.

- Sortie : ensemble des dates $t' = (t(x), x \in \Gamma)$ telles que :
  - $\forall x \in \Gamma, t'(x) \in FP(\Gamma)$;
  - $\forall \tau, Global\_Cash(L\_T, t, \tau) + Tour\_Cash(\Gamma, t', \tau) \geq 0$;
  - la quantité : $Cout_{\gamma,\delta,\varepsilon}(\Gamma) = \gamma \times T_{Global}(\Gamma) + \delta \times T_{Ride}(\Gamma) + \varepsilon \times T_{Wait}(\Gamma)$ est la plus petite possible.

3.5. Le problème « Cash_Insertion_Flot »

Le problème $Cash\_Insertion$ peut être reformulé comme un problème de flot : la résolution de l'organisation de la circulation de la ressource financière entre les sommets de la tournée Γ et les sommets de la suite L_T. C'est-a-dire que tous les sommets de $\Gamma \cup L\_T$ doivent avoir un montant d'argent positif et la tournée Γ est une tournée « Temps-admissible » et « Charge-admissible ». Rappelons que tout échange d'argent entre le sommet x de la suite L_T et le sommet y de la tournée Γ introduit une contrainte temporelle supplémentaire : $t(y) \geq t(x)$ si le sommet x envoie de l'argent au sommet y (voire Figure 44) ou $t(y) \leq t(x)$ si x reçoit de l'argent de y.

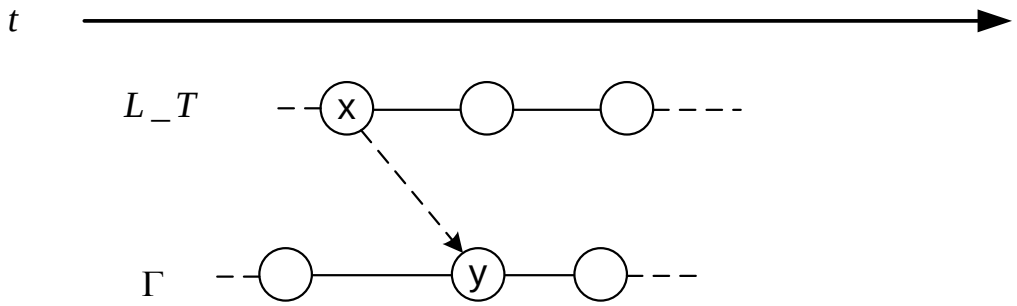

Figure 44 : Un flux financier entre L_T et Γ

Donc, le problème « $Cash\_Insertion\_Flot$ » peut être écrit comme suit :

Le problème « $Cash\_Insertion\_Flot$ » :

- Entrée :
  - la suite L_T, l'ensemble des dates t, les valeurs $B\_Cash(L\_T, x), x \in L\_T$;
  - la tournée Γ, les valeurs $Cash(\Gamma, x), x \in \Gamma$, l'ensemble des fenêtres de temps $FP(\Gamma)(x), x \in \Gamma$;
  - les trois coefficients $\gamma, \delta, \varepsilon$.

- Sortie : une famille $\mathcal{U}$ des triplets $(x, y, \theta_1), x \in L\_T, y \in \Gamma, \theta_1 \in \mathbb{Q}^+$. Elle présente un ensemble des flux financiers envoyés de la séquence L_T vers la tournée Γ. Une famille $\mathcal{V}$ des triplets $(y, x, \theta_2), x \in L\_T, y \in \Gamma, \theta_2 \in \mathbb{Q}^+$, elle présente un ensemble des flux financiers envoyés de Γ vers la suite L_T. L'ensemble de dates $t' = (t(x), x \in \Gamma)$ tel que :
  - $\forall x \in \Gamma, t'(x) \in FP(\Gamma)$;
  - la quantité : $Cout_{\gamma,\delta,\varepsilon}(\Gamma) = \gamma \times T_{Global}(\Gamma) + \delta \times T_{Ride}(\Gamma) + \varepsilon \times T_{Wait}(\Gamma)$ est la plus petite possible ;
  - pour $\forall x \in L\_T$ (voir la Figure 45) :

$$B\_Cash(L\_T, x) + \sum_{\substack{(y,z,\theta_2)\in\mathcal{V},\\ z\in segment(L\_T,NULL,x)}} \theta_2 - \sum_{\substack{(z,y,\theta_1)\in\mathcal{U},\\ z\in segment(L\_T,NULL,x)}} \theta_1 \geq 0$$

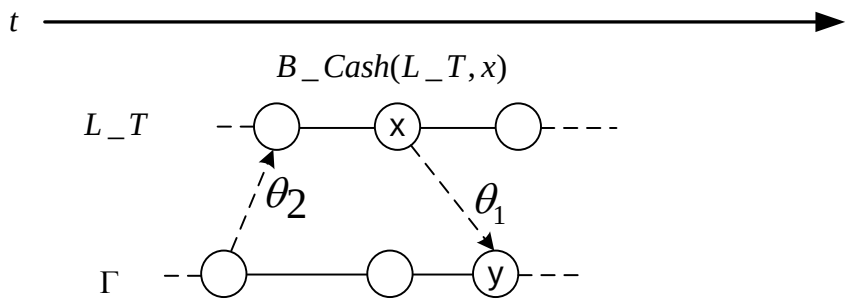

Figure 45 : L'équilibre financier pour la séquence *L_T*

  - pour $\forall y \in \Gamma$ (voir la Figure 46), on a :

$$Cash(\Gamma, y) - \sum_{\substack{(z,x,\theta_2)\in\mathcal{V},\\ z\in segment(\Gamma,NULL,y)}} \theta_2 + \sum_{\substack{(x,z,\theta_1)\in\mathcal{U},\\ z\in segment(\Gamma,NULL,y)}} \theta_1 \geq 0$$

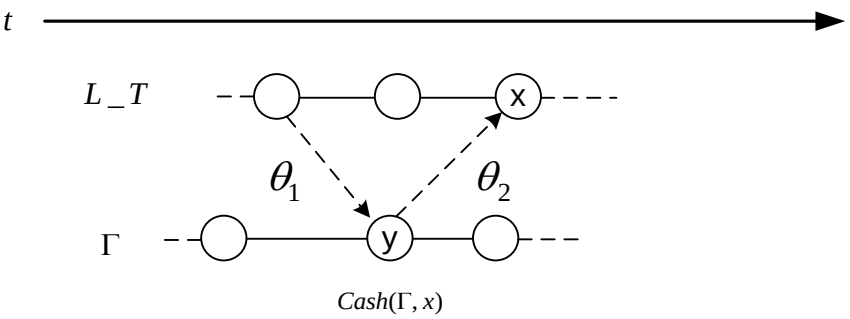

Figure 46 : L'équilibre financier pour la tournée *Γ*

Proposition 3 : une solution optimale $(\mathcal{U}, \mathcal{V}, t)$ du problème « $Cash\_Insertion\_Flot$ » peut être choisie de telle façon qu'il n'y a pas de flux croisés, c'est-à-dire qu'il n'existe pas de (x, y, θ_1) dans $\mathcal{U}$ et (y', x', θ_2) dans $\mathcal{V}$ tels que: $x \ll^{=}_{L\_T} x'$ et $y' \ll^{=}_{\Gamma} y$ (voir la Figure 47).

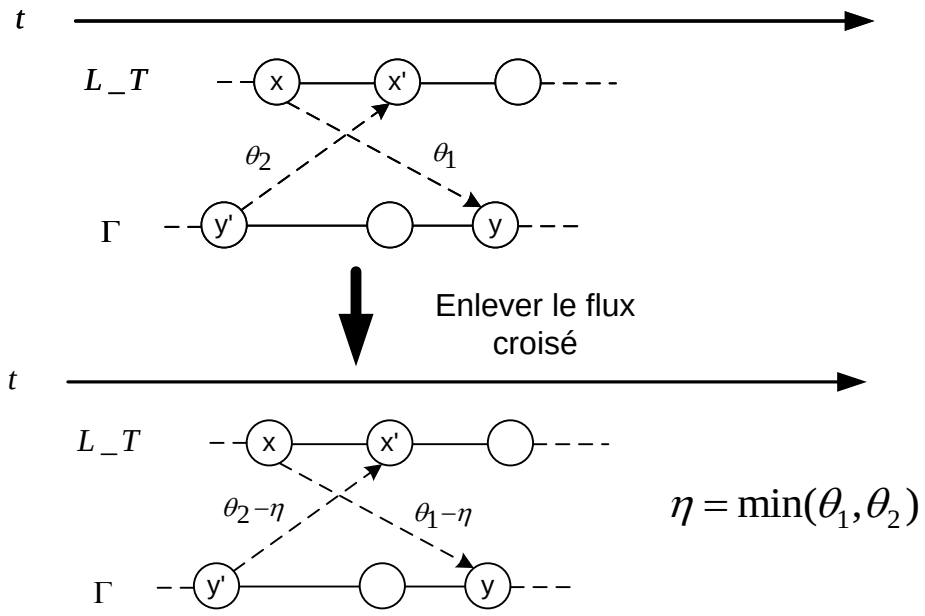

Figure 47 : L'enlèvement de flux croisé

Si deux tels triplets (x, y, θ_1) dans $\mathcal{U}$ et (y', x', θ_2) dans $\mathcal{V}$ existent, il suffit de poser : $\eta = min(\theta_1, \theta_2)$, et de remplacer :

- θ_1 par $\theta_1 - \eta$;
- θ_2 par $\theta_2 - \eta$.

Prenons maintenant quelques triplets (x_1, y_1, θ_1') dans $\mathcal{U}$, $x_1' = prev(L\_T, x_1)$ et supposons que : $B_{Cash(L_T, x_1')} + \sum_{(y,z,\theta_2)\in\mathcal{V}, z\in segment(L_T, NULL, x_1')} \theta_2 - \sum_{(z,y,\theta_1)\in\mathcal{U}, z\in segment(L_T, NULL, x_1')} \theta_1 \geq \theta_1'$. Nous voyons qu'il est possible de fusionner le triplet (x_1, y_1, θ_1) et le triplet (x_1', y_1, θ_1') dans $\mathcal{U}$, et que cette fusion nous permet d'obtenir une nouvelle solution faisable au problème « $Cash\_Insertion\_Flot$ », dont la valeur $Cout_{\gamma,\delta,\varepsilon}$ ne peut pas être pire que ce qu'elle était avant. La Figure 48 montre un exemple de la fusion de deux triplets dans l'ensemble $\mathcal{U}$. Un raisonnement similaire s'applique si l'on considère certains (y_1', x_1', θ_2').

Proposition 4 : Les deux problèmes « $Cash\_Insertion$ » et « $Cash\_Insertion\_Flot$ » sont équivalents. Toute solution faisable $(\mathcal{U}, \mathcal{V}, t)$ du problème « $Cash\_Insertion\_Flot$ » est telle que t est un composant d'une solution faisable de « $Cash\_Insertion$ » avec la même valeur de performance $Cout_{\gamma,\delta,\varepsilon}$.

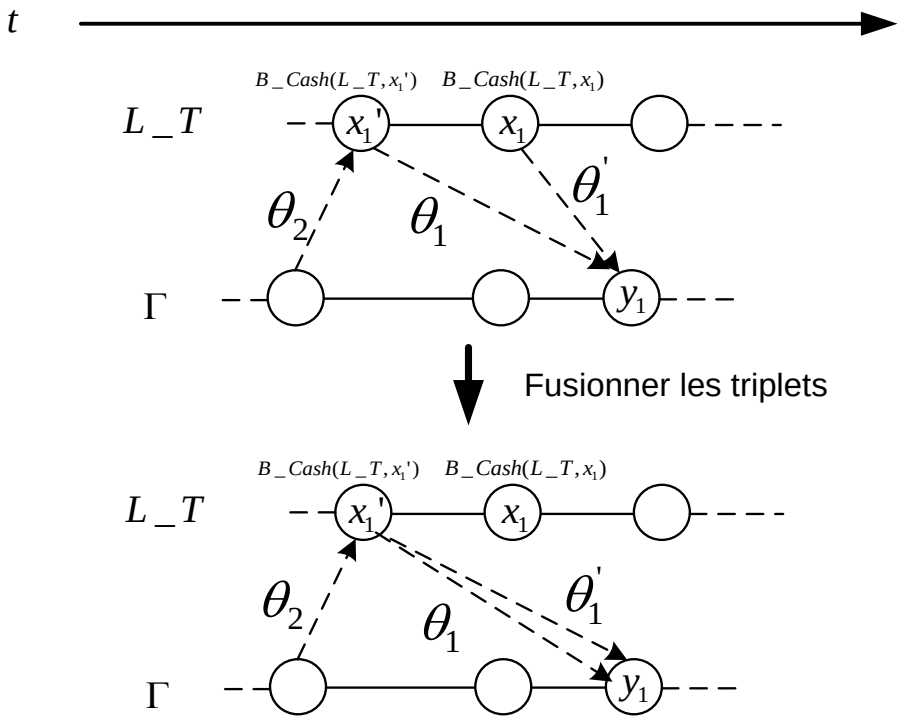

Figure 48 : La fusion des triplets dans l'ensemble $\mathcal{U}$

3.6. Le traitement du problème Cash_Insertion

Soit la tournée $T(k_0)$ dans laquelle on souhaite insérer la nouvelle demande D_{i_0}. Les contraintes temporelles, la contrainte de capacité et la contrainte financière doivent être respectées après l'insertion de la demande D_{i_0} dans la tournée $T(k_0)$. Avant d'effectuer l'insertion, nous effectuons des tests simples et rapides. Pour les contraintes temporelles, nous utilisons trois procédures qui sont présentées dans la section 5 du chapitre 3. Si les fenêtres de temps réduites sont valides, on utilise alors une autre procédure pour tester la faisabilité des flux financiers.

Pour la contrainte financière, on nomme une procédure « **Test_Finance** ». Cette procédure fonctionne de la manière suivante : On considère d'abord les dates de services des sommets dans la tournée $T(k), k \neq k_0$ fixées. On cherche les dates de service des sommets de la tournée $T(k_0)$, de sorte que les sommets $x \in T(k_0)$ qui génèrent plus de revenus que de dépenses ($r(x) - Cost(x, x') > 0$) soient le plus tôt possible $t(x) \leftarrow \alpha(x)$ et les autres le plus tard possible ($t(x) \leftarrow \beta(x)$). En suite, on trie les sommets $x \in \bigcup_{k\in\{1,\dots,K\}} T(k)$ dans l'ordre des date croissantes $t(x)$. A chaque instant λ, on calcule la différence entre les revenus totaux (inclure le capital initial) et les dépenses totales engendrées jusqu'en λ. Si la différence est négative, alors les contraintes financières ne sont pas respectées et la procédure s'arrête.

Algorithme 28 : Test_Finance

Entrée : $T(k), k \in \{1, \dots, K\}$: l'ensemble de tournées ;
k_0 : le numéro de la tournée pour laquelle on test l'insertion de la demande ;
FP : l'ensemble de fenêtres de temps réduites ; $\mathcal{R}$: la capital initial
Sortie : res : booléen ;

1. **Pour** chaque sommet x dans $T(k_0)$ et $\neq DepotA(T(k_0))$ **faire**
2. | $x \leftarrow succ(T(k_0), x)$;
3. | **Si** $r(x) - Cost(x, x') > 0$ **alors**
4. | | $t(x) \leftarrow FP.\alpha(x)$;
5. | **Sinon** $t(x) \leftarrow FP.\beta(x)$;
6. | **Fin**
7. **Fin**
8. $L\_T \leftarrow x \in \bigcup_{k\in\{1,\dots,K\}} T(k)$ dans l'ordre des $t(x)$ croissants ;
9. $res \leftarrow false$; $Total \leftarrow \mathcal{R}$; $i = 1$;
10. **Tant que** $i < |L\_T|$ et $res = false$ **faire**
11. | $x \leftarrow L\_T(i)$; // $i^{ème}$ élément dans la liste L_T
12. | $x' \leftarrow succ(L\_T, x)$; $Total \leftarrow Total + r(x) - Cost(x, x')$;
13. | **Si** $Total < 0$ **alors** $res \leftarrow true$;
14. | **Fin**
15. **Fin**

Si la procédure « Test_Finance » retourne un résultat positif, nous devons alors déterminer les flux financiers entre la séquence L_T et la tournée $T(k_0)$ en utilisant une procédure récursive « Propagation_Finance ». Pour chaque sommet $x \in L\_T \cup T(k_0)$, la différence entre revenus et dépenses doit être toujours positives. Nous supposons que les dates de début de service des sommets dans L_T sont fixées, nous devons mettre à jour les fenêtres de temps de $T(k_0)$ pour satisfaire la contrainte financière. Les fenêtres de temps sont mise à jour grâce à la procédure « Propage » qui est présentée dans le chapitre 3. Si la procédure « Propage » échoue, alors il n'est pas possible de trouver des flux financiers pour satisfaire la contrainte financière.
La nouvelle procédure Test_Insertion_Finance fonctionne de la manière suivante :

- Insérer la demande D_{i_0} dans la tournée ciblée $T(k_0)$;
- Mettre à jour les fenêtres de temps dans la tournée $T(k_0)$ avec la procédure **Propage** ;
- Déterminer les flux financiers

Algorithme 29 : Test_Insertion_Finance

Entées : Γ : la tournée; p1, p2 : les positions.
Sorties : res

1. Insérer le sommet o_{i_0} après p1 et le sommet d_{i_0} après p2 ;
2. $Propage(\Gamma)$;
3. **Si** $Test\_Finance() = vrai$ **alors**
4. | $res \leftarrow Propagation\_Finance()$;
5. **Fin**

Algorithme 30 : Propagation_Finance

Entrée : L_T : la séquence des sommets ; $T(k_0)$: la tournée
Sortie : res : booléen

1. Cherche le premier sommet x_0 dans L_T tel que $Cash(L\_T, x_0) < 0$;
2. Cherche le premier sommet y_0 dans Γ tel que $Cash(\Gamma, y_0) < 0$;
3. $\Gamma' \leftarrow \Gamma$;
4. **Si** x_0 et y_0 n'existent pas **alors**
5. | $res \leftarrow true$;
6. **Sinon**
7. | **// CAS1**
8. | **Si** x_0 existe et (y_0 n'existe pas ou $t(x_0) < \alpha(y_0)$) **alors**
9. | | Cherche le premier sommet y_1 dans Γ qui peut financer x_0 : $Cash(\Gamma, y_1) > -Cash(L\_T, x_0)$ et $\alpha(y_0) \leq t(x_0)$;
10. | | **Si** y_1 n'existe pas **alors**
11. | | | $res \leftarrow false$;
12. | | **Sinon**
13. | | | **Si** $\beta(y_1) > t(x_0)$ **alors**
14. | | | | $\beta(y_1) = t(x_0)$; $res1 \leftarrow Propage(\Gamma)$;
15. | | | | **Si** $res1 = true$ **alors**
16. | | | | | Supprimer x_0 dans L_T ainsi que tous les sommets qui le précédent ;
17. | | | | | Mettre à jour $Cash(L\_T, x_0)$ et $Cash(\Gamma, y_0)$ pour prendre en compte le flux ;
18. | | | | | financier de y_1 vers x_0 ;
19. | | | | | $res \leftarrow Propagation\_Finance(L\_T, \Gamma')$;
20. | | | | **Fin**
21. | | | **Fin**
22. | | **Fin**
23. | **Fin**
24. | **// CAS2**
25. | **Si** y_0 existe et (x_0 n'existe pas ou $t(x_0) > \beta(y_0)$) **alors**
26. | | Cherche le premier sommet x_1 dans L_T qui finance y_0 : $Cash(L\_T, x_1) > -Cash(\Gamma, y_0)$ et $\beta(y_0) \geq t(x_1)$;
27. | | **Si** y_1 n'existe pas **alors**
28. | | | $res \leftarrow false$;
29. | | **Sinon**
30. | | | Si $\alpha(y_0) < t(x_1)$ alors
31. | | | | $\alpha(y_1) = t(x_0)$; $res1 \leftarrow Propage(\Gamma)$;
32. | | | | **Si** $res1 = true$ **alors**
33. | | | | | Supprimer y_0 dans Γ' ainsi que tous les sommets qui le précédent ;
34. | | | | | Mettre à jour $Cash(L\_T, x_0)$ et $Cash(\Gamma, y_0)$ pour prendre en compte le flux ;
35. | | | | | financier de x_1 vers y_0 ;
36. | | | | | $res \leftarrow Propagation\_Finance(L\_T, \Gamma')$;
37. | | | | **Fin**
38. | | **Fin**
39. | **Fin**
40. | **Si** (x_0 et y_0 existent) et $\alpha(y_0) \leq t(x_0) \leq \beta(y_0)$ alors
41. | | Appliquer le CAS1
42. | | **Si** on rencontre l'échec **alors** Appliquer le CAS2
43. | **Fin**
44. **Fin**

4. Expérimentations numériques

4.1. Implémentation et jeux de tests

Les heuristiques ont été programmées en C++ et exécutées sur un PC sous Windows Xp. Pour tester l'heuristique avec flux financiers, nous modifions les instances de Cordeau et Laporte de la manière suivante :

- Chaque destination est associée à un revenu ;
- Chaque arc (x, y) est associé à un coût financier qui représente la dépense pour aller de x à y ;
- La durée maximale de transport d'une demande est fixée à 100.
- Les instances sont disponibles à l'adresse suivante : http://fc.isima.fr/~zhao/DARPF/instances.zip

4.2. Résultats des évaluations numériques

Nous utilisons les heuristiques présentées dans le chapitre 3 adaptées au DARPF.

- Heuristique basique ;
- Heuristique mémoire ;
- Heuristique avec recherche arborescente.

L'heuristique avec recherche arborescente présente quelques différences entre le DARP et le DARPF. Pour le DARPF : un revenu généré par l'insertion d'une nouvelle demande peut rendre insérables des demandes qui ne l'étaient pas. L'ajout de contrainte financière augmente la complexité du problème, si une demande peut être insérée dans plusieurs tournées, du point de vue des contraintes temporelles, il se peut qu'elle soit trop coûteuse pour être réellement insérée.

Nous modifions la création et l'utilisation des points de sauvegarde pour adapter ces différences comme suit :

- On ne crée qu'une point de sauvegarde à l'insertion de chaque demande ;
- A la fin du processus, si plus aucune demande ne peut pas être insérée on utilise les points de sauvegarde ;
- Une exécution est limitée à 600 secondes.

Le Tableau 21 montre une comparaison des trois heuristiques. Il donne le taux de réussite des trois heuristiques sur 100 exécutions. Les colonnes ont la signification suivante :

- Instance : nom de l'instance ;
- Basique : résultats pour l'heuristique basique ;

- Mémoire : résultats pour l'heuristique avec mémoire ;
- Arbre : résultats pour l'heuristique avec recherche arborescent ;
- Réussite : nombre d'exécution qui obtient une solution ;
- cpu : temps cpu en secondes pour 100 exécutions.

Comme pour le DARP classique, les fenêtres de temps sont réduites en utilisant des règles avant l'exécution des heuristiques (Voir le chapitre 2, la section 4.1). Parmi ces heuristiques, l'heuristique mémoire donne les meilleurs résultats.

Tableau 21 : Comparaison des trois heuristiques pour DARPF

	Basique		**Mémoire**		**Arbre**	
Instance	réussite	CPU(s)	réussite	CPU(s)	réussite	CPU(s)
F1	49	2,3	100	4,8	95	14,7
F2	79	9,2	100	11,9	100	15,3
F3	6	25,5	79	160,6	79	504,5
F4	16	65,1	88	321,9	97	1431,3
F5	54	118,5	100	187,7	100	889,8
F6	80	99,2	100	120,9	100	665,4
F7	3	6,1	99	38,8	44	99,5
F8	72	19,5	100	27,0	100	62,0
F9	69	49,3	100	73,9	100	394,0
F10	40	113,1	99	311,4	100	2357,8
F11	98	3,6	100	3,7	100	4,2
F12	83	25,6	100	33,8	100	46,3
F13	29	43,7	100	147,3	100	409,4
F14	25	112,2	100	415,9	100	1848,4
F15	17	276,0	79	1503,7	93	4514,5
F16	9	332,5	77	2135,6	92	9995,2
F17	48	11,7	100	24,9	100	28,8
F18	34	53,7	100	130,4	100	311,5
F19	31	178,4	99	550,7	100	1383,4
F20	50	237,4	99	507,4	100	2706,1
Moyenne	44,6	89,1	95,95	335,6	95	1384,1

Tableau 22 : Les solutions du DARPF obtenues par l'heuristique mémoire

Instance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Dist
F1	4,7	392,7	283,7	115,3	277,4
F2	11,9	592,8	970,6	98,1	494,7
F3	160,6	1209	1130	181,9	1027,1
F4	321,9	1236,8	1231,5	135,4	1101,4
F5	187,7	1305,1	1568,7	63,6	1241,5
F6	120,9	1567,6	2386,4	97,4	1470,2
F7	38,8	562,1	666,8	102,4	459,7
F8	27,0	1104,1	1128,2	142,2	961,9
F9	73,9	1190,9	2192,2	30,8	1160,1
F10	311,4	1564,8	2762,9	26	1538,8
F11	3,7	377,2	369,1	112,7	264,5
F12	33,8	457,9	711,9	1,6	456,3
F13	147,3	867,4	1181,1	9,6	857,8
F14	415,9	1029,6	1254,4	57,6	972
F15	1503,7	1122	2074,6	47,4	1074,6
F16	2135,6	1385	2272,8	6,6	1378,4
F17	24,9	459,6	850,3	46,8	412,8
F18	130,4	876,6	1129,1	63,9	812,7
F19	550,7	1101,7	2011,8	11,7	1090,0
F20	507,4	1485,6	2551,7	39,9	1445,7
Moyenne	335,6	994,4	1436,4	69,5	924,9

Tableau 23 : Les solutions du DARP obtenues par l'heuristique mémoire

Instance	CPU(s)	T_{Global}	T_{Ride}	T_{Wait}	Dist
F1	0,1	318,1	204,2	44,2	273,9
F2	0,8	469,7	680,9	24,7	445,1
F3	1,1	687,7	1303,5	3,5	684,2
F4	3,2	770,4	1562,5	1,1	769,3
F5	12,0	831,1	1593,3	13,2	817,9
F6	16,0	1006,4	2298,2	3,3	1003,1
F7	0,3	446,4	418,5	55,4	391,0
F8	0,1	672,4	744,1	0,7	671,7
F9	3,9	772,7	1891,3	1,1	771,6
F10	9,0	1178,8	1532,5	7,5	1171,3
F11	0,2	261,6	258,3	5,1	256,5
F12	1,5	358,4	657,4	0,0	358,4
F13	9,6	639,1	973,0	1,0	638,1
F14	18,6	683,6	1097,3	0,0	683,6
F15	25,9	771,0	1748,9	0,0	771,0
F16	94,5	952,8	1402,4	0,9	951,9
F17	1,1	361,0	323,3	0,0	361,0
F18	6,7	587,6	783,9	0,0	587,6
F19	54,0	771,5	1263,1	0,0	771,5
F20	72,5	1042,9	1547,2	6,0	1037,0
Moyenne	22,9	679,2	1114,2	8,4	670,8

Le Tableau 22 donne la durée totale des tournées, le temps total de transport, le temps total d'attente et la distance totale des tournées obtenus par l'heuristique mémoire.

Nous pouvons traiter les instances (F1-F20) du DARPF comme les instances du DARP classique. Le Tableau 23 présente les résultats obtenus par l'heuristique mémoire. Nous constatons la comparaison entre deux tableaux (le Tableau 22 et le Tableau 23). L'ajout de la contrainte financière augmente la complexité du problème, il augmente le temps de calcul (335.6 secondes en moyenne pour l'heuristique mémoire contre 22.90 dans le DARP classique). Il augmente la durée totale des tournées (994.4 secondes en moyenne pour l'heuristique mémoire contre 679.12 dans le DARP classique). Il augmente la durée totale du transport (1436.4 secondes en moyenne pour l'heuristique mémoire contre 1114.2 dans le DARP classique). Il augmente le temps d'attente (69.5 secondes en moyenne pour l'heuristique mémoire contre 8.4 dans le DARP classique). Il augmente aussi la distance totale (924.9 secondes en moyenne pour l'heuristique mémoire contre 670.8 dans le DARP classique).

5. Conclusion

Dans ce chapitre, nous avons considéré un problème de transport à demande (DARP) avec contrainte financière. A notre connaissance, ce problème n'avait jamais été étudié. Dans ce

problème, nous devons déterminer les tournées par une flotte de véhicules pour satisfaire un ensemble des demandes, et à chaque instant la différence entre le capital initial, les revenus et les dépenses doit être positive.

Nous proposons pour ce problème des méthodes d'insertion basée sur des techniques de propagation de contraintes. Ces méthodes ont été testées pour le DARP classique (voir le chapitre 3). Ces trois heuristiques ont été adaptées pour prendre en compte des contraintes financières. Les expérimentations ont montré que nos heuristiques sont capables de donner des résultats dans des temps de calcul courts.

Ce problème peut s'appliquer à différentes situations, par exemple pour une société de taxis : chaque course à un coût (dépense) et le conducteur reçoit le paiement de sa course arrivé à destination.

CONCLUSION GENERALE

Dans le cadre de cette thèse, nous nous sommes intéressés aux problèmes de tournées à la demande (DARP). Ce problème consiste à déterminer des tournées pour satisfaire les demandes de clients souhaitant être transportés depuis des lieux d'origine vers des lieux de destination. Une des difficultés de ce problème est de prendre en compte les fenêtres de temps imposées sur les origines ou sur les destinations.

Dans un premier temps, nous avons étudié et conçu un modèle simple pour le DARP multi-véhicules. Dans ce modèle, aucune préemption n'est autorisée, et aucune contrainte supplémentaire ne doit être pas prise en compte. Pour la résolution du DARP, nous avons développé un algorithme basé sur la technique d'insertion. Le but de ces méthodes est de fournir rapidement des solutions du DARP pour satisfaire toutes les demandes. Les expérimentations des algorithmes font apparaître de façon fréquente des demandes rejetées, donc nous proposons deux procédures de réparation pour diminuer le taux d'échec. Ensuite, nous proposons et testons une autre méthode heuristique d'insertion pour le DARP standard, impliquant de la propagation de contraintes.

Nous choisissons les heuristiques d'insertion pour résoudre le problème DARP, car elles sont rapides, elles peuvent produire des solutions de bonne qualité, elles sont faciles à mettre en œuvre et enfin, elles peuvent facilement être étendues. De plus, ces méthodes sont faciles à randomisées afin d'être intégrées à des schémas de type Monte-Carlo.

Dans un deuxième temps, nous avons traité le DARP en utilisant la recherche locale et la procédure SPLIT. Nous nous sommes ensuite intéressés à un cas particulier du DARP dans lequel le DARP est considéré comme un problème multi-objectif. Pour ce problème, nous avons utilisé trois critères : T_{Global} est la durée totale des tournées ; T_{Ride} est le temps total de transport, T_{Wait} est le temps total d'attente.

Enfin, nous avons traité une extension du DARP dans laquelle nous avons ajouté une contrainte financière. Ce problème a été résolu grâce à des heuristiques d'insertion et des

techniques de propagation de contraintes. Nous avons fournis des benchmark pour évaluer nos heuristiques proposées à ce nouveau problème.

La fonction objectif détermine les caractéristiques des tournées. Les expérimentations montrent que nos (méta-) heuristiques donnent des résultats plus favorables aux clients (meilleure qualité de service) par rapport aux autres heuristiques de la littérature.

Les perspectives que nous souhaitons donner à ce travail concernent :

- La proposition de méthodes pour le DARP avec correspondance;
- La proposition de méthodes pour le DARP multi-dépôts ;
- L'application de la propagation de contraintes au DARP dynamique.

BIBLIOGRAPHIES

1 Lilas. Lilas lille autopartege. [Internet]. Available from: http://www.lilas-autopartage.com/savoir.html.

2 Mobility. Mobility carsharing. [Internet]. Available from: http://www.mobility.ch/pages/index.cfm?srv=cms&pg=& dom=6&prub=526&rub=527.

3 CityCarClub. City car club – your monitoring solution. [Internet]. Available from: http://www.citycarclub.info.

4 PRAXITELE. [Internet]. Available from: http://www.rocq.inria.fr/praxitele.

5 LISELEC. [Internet]. Available from: http://www.agglo-larochelle.fr/services/info_liselec.htm.

6 Autolib. Mondial 2008 - des nouvelles d'autolib. [Internet]. Available from: http://www.avem.fr/video-245-mondial2008-des-nouvelles-d-autolib.html.

7 ZipCar. Zipcar offers a new model for automobile transportation. [Internet]. Available from: http://www.zipcar.com/about.

8 Communauto. La lettre de l'autpartage.. [Internet]. Available from: http://www.communauto.com/images/03. coupures_de_presse/CERTU_AutoPartage6.pdf.

9 Benoît r. Histore de l'autopartage. [Internet]. Available from: http://www.communauto.com/historique 01.html.

10 Psaraftis HN. Dynamic vehicle routing : Status and prospects. Annals of Operations Research. 1995;61(1):143-164.

11 Ropke S, Pisinger D. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research. 2006;171:750-775.

12 Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

13 Anily S, Hassin R. The swapping problem. Networks. 1992;22:419-433.

14 Chalasani P, Motwani R. Approximated capacitated routing and delivery problems. SIAM Journal on Computing. 1999;28:2133-2149.

15 Anily S, Gendreau M, Laporte G. The swapping problem on a line. SIAM Journal on Computing. 1999;29(327-335).

16 Erdoğan G, Cordeau JF, Laporte G. A branch-and-cut algorithm for the non-preemptive capacitated swapping problem. Technical Report. Montreal, Canada: CIRRELT-2008-44, CIRRELT; 2008.

17 Bordenave C, Gendreau M, Laporte G. Heuristics for the mixed swapping problem. Computers & Operations Research. 2010;37:108–114.

18 Hernández-Pérez H, Salazar-González JJ. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discret Appl Math. 2004a;145:126-139.

19 Balas E, Padberg M. Set partitioning: a survey. SIAM Rev. 1976;18:710-760.

20 Hernández-Pérez H, Salazar-González JJ. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. Transport Sci. 2004b;38:245–255.

21 Lin S. Computer solutions of the traveling salesman problem. Bell Syst Tech.

1965;44(J):2245-2269.

22 Johnson D, McGeoch L. The traveling salesman problem: a case study in local optimization. In: Aarts E, Lenstra J. Local search in combinatorial optimization. Wiley, Chilester 1997. p. 215-310.

23 Hernández-Pérez H, Rodriguez-Martin I, Salazar-González JJ. A Hybrid GRASP/VND Heuristic for the One-Commodity Pickup and Delivery Traveling Salesman Problem. Computers and Operations Research. 2009;36(5):1639–1645.

24 ZHAO F, Li S, Sun J, Mei D. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. Computers and Industrial Engineering archive. 2009;56(4):1642-1648.

25 Anily S, Bramel J. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. Nav Res Logist. 1999;46:654-670.

26 Gribkovskaia I, Halskau Ø, Laporte G, Vlcek M. General solutions to the single vehicle routing problem with pickups and deliveries. Eur J Oper Res. 2007;180:568-584.

27 Cordeau JF, Laporte G, Mercier A. A unified tabu search heuristic for vehicle routing problems with time windows. J Oper Res Soc. 2001;52:928-936.

28 Anily S. The vehicle routing problem with delivery and back-haul options. Nav Res Logist. 1996;43:415-434.

29 Min H. The multiple the vehicle routing problem with simultaneous delivery and pick-up points. Transport Res Part. 1989;A(5):377-386.

30 Anderberg M. Cluster analysis for applications. New York: Academic; 1973.

31 Nagy G, Salhi S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. Eur J Oper Res. 2005;162:126-141.

32 Chen JF, Wu TH. Vehicle routing problem with simultaneous deliveries and pickups. J Oper Res Soc. 2006 579-587.

33 Dueck G. New optimization heuristics: the great deluge algorithm and the record-to-record travel. J Comput Phys. 1993 86-92.

34 Bianchessi N, Righini G. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. Comput Oper Res. 2007 578-594.

35 Montané F, Galvão R. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. Comput Oper Res. 2006 595-619.

36 Deif I, Bodin L. Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In: Kidder A. Proceedings of the Babson conference on software uses in transportation and logistic management. Babson Park, FL 1984.

37 Clarke G, Wright J. Scheduling of vehicles from a central depot to a number of delivery points. Oper Res. 1964 568-581.

38 Toth P, Vigo D. A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls. Eur J Oper Res. 1999;113:528–543.

39 Toth P, Vigo D. VRP with backhauls. In: Toth P, Vigo D. The vehicle routing problem. Philadelphia: SIAM; 2002. p. 195–224.

40 Osman I, Wassan N. A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. Journal of Scheduling. 2002;5:263–285.

41 Brandão J. A new tabu search algorithm for the vehicle routing problem with backhauls. European Journal of Operational Research. 2006;173:540-555.

42 Gélinas S, Desrochers M, Desrosiers J. A new branching strategy for time constrained

routing problems with application to backhauling. Annals of Operations Research. 1995;61(1):91-109.

43 Potvin JY, Duhamel C, Guertin F. A genetic algorithm for vehicle routing with backhauling. Applied Intelligence. 1996;6(4):341-351.

44 Duhamel C, Potvin JY, Rousseau JM. A tabu search heuristic for the vehicle routing problem with backhauls and time windows. Transportation science. 1997;31(1):49-59.

45 Thangiah SR, Potvin JY, Sun T. Heuristic approaches to vehicle routing with backhauls and time windows. Computers & operations research. 1996;23(11):1043–1057.

46 Chisman JA. The clustered traveling salesman problem. Computers & Operations Research. 1975;2(2):115-119.

47 Gendreau M, Hertz A, Laporte G. The traveling salesman problem with backhauls. Computers & Operations Research. 1996;23:501-508.

48 Gendreau M, Hertz A, Laporte G. New insertion and postoptimization procedures for the traveling salesman problem. Operations Research. 1992;40:1086–1094.

49 Mladenović N, Hansen P. Variable neighborhood search. Computers & operations research. 1997;24:1097–1100.

50 Ghaziri H, Osman IH. A neural network algorithm for the traveling salesman problem with backhauls. Computers & industrial engineering. 2003;44(2):267-281.

51 Kohonen T. Self-organized formation of topologically correct feature maps. Biological Cybernetics. 1995;43:59-69.

52 Gendreau M, Laporte G, Vigo D. Heuristics for the traveling salesman problem with pickup and delivery. Computers & operations research. 1999;26(7):699-714.

53 Christofides N. Worst-case analysis of a new heuristic for the travelling salesman problem. Research report 388. Pittsburgh, PA: GSIA, Carnegie Mellon University; 1976.

54 Baldacci R, Hadjiconstantinou EA, Mingozzi A. An exact algorithm for the traveling salesman problem with deliveries and collections. Networks. 2003;42:26-41.

55 Frederickson G, Hecht M, Kim C. Approximation algorithms for some routing problems. SIAM. 1978;7:178–193.

56 Frederickson GGD. Nonpreemptive ensemble motion planning on a tree. Journal of algorithms. 1993;15(1):29-60.

57 Savelsbergh MWP. Local search in routing problems with time windows. Annals of Operations Research. 1985;4:285–305.

58 Dumas Y, Desrosiers J, Soumis F. The Pickup and Delivery Problem with Time Windows. European Journal of Operational Research. 1991;54:7-22.

59 Savelsbergh MWP, Sol M. Dynamic routing of independent vehicles. Operations research A. 1998;46(4):474-490.

60 Xu H, Chen ZL, Rajagopal S, Arunapuram S. Solving a practical pickup and delivery problem. Transportation science. 2003;37(3):347-364.

61 Ropke S, Cordeau JF. Branch-and-cut-and-price for the pickup and delivery problem with time windows. Transportation science. 2009;43(3):267-286.

62 Cordeau JF. A branch-and-cut algorithm for the dial-a-ride problem. Operations research. 2006;54(3):573-586.

63 Battiti R, Tecchiolli G. The reactive tabu search. ORSA Journal on Computing. 1994;6:126-140.

64 Nanry WP, Barnes JW. Solving the pickup and delivery problem with time windows

using reactive tabu search. Transportation research. 2000;34(2):107-121.

65 Stein D. An asymptotic, probabilistic analysis of a routing problem. Mathematics of Operations Research. 1978;3:89-101.

66 Psaraftis HN. Analysis of an O(N2) heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. Transportation Research. 1983;17B:133-145.

67 Psaraftis HN. An exact algorithm for the single-vehicle, many-to-many dial-a-ride problem with time windows. Transportation science. 1983;17(3):351-357.

68 Desrosiers J, Dumas Y, Soumis F. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences. 1986;6(3-4):301-325.

69 Kalantari B, Hill A, Arora SR. An algorithm for the traveling salesman problem with pickup and delivery customers. European Journal of Operational Research. 1985;22(3):377-386.

70 Little JDC, Murty KG, Sweeney DW, Karel C. An algorithm for the traveling salesman problem. Operations Research. 1963;11:972-989.

71 Ruland KS, Rodin EY. The pickup and delivery problem: faces and branch-and-cut algorithm. Computers & Mathematics with Applications. 1997;33:1-33.

72 Van Der Bruggen LJJ, Lenstra JK, Schuur PC. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. Transportation Science. 1993;27(3):298–311.

73 Landrieu A, Mati Y, Binder Z. A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. Journal of Intelligent Manufacturing. 2001;12:497-508.

74 Pacheco JA. Heurístico para los problemas de rutas con carga y descarga en sistemas LIFO. Qüestiió. 1997;21:153-175.

75 Carrabs F, Cordeau JF, Laporte G. Variable neighbourhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing. 2006;19(4):618-632.

76 Cordeau JF, Iori M, Laporte G, Salazar-González JJ. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks. 2010;55:46-59.

77 Cordeau JF, Laporte G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B. 2003;37:579-594.

78 Sexton T, Bodin L. Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: I. Scheduling. Transportation Science. 1985a;19:378-410.

79 Sexton T, Bodin L. Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: II. Routing. Transportation Science. 1985b;19:411-435.

80 Jaw J, Odoni A, Psaraftis H, Wilson N. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. Transportation Research. 1986;B 20:243–257.

81 Cullen FH, Jarvis J, Ratliff HD. Set partitioning based heuristic for interactive routing. Networks. 1981;11:125-143.

82 Roy S, Chapleau L, Ferland JA, Lapalme G, Rousseau JM. The construction of routes and schedules for the transportation of the handicapped. Working paper, Technical Report CRT-308. Université de Montréal; 1983.

83 Bodin L, Sexton T. The Multi-Vehicle Subscriber Dial-a-Ride Problem. TIMS Studies in

the Management Sciences. 1986;22:73-86.

84 Ropke S, Cordeau JF, Laporte G. Models and a branch-and-cut algorithm for pickup and delivery problems with time windows. Networks. 2007;49:258-272.

85 Cortés CE, Matamala M, Contardo C. The pickup-and-delivery problem with transfers: formulation and a branch-and-cut solution method. European Journal of Operational Research. 2010;200(3):711-724.

86 Mitrovic-Minic S, Laporte G. The pickup and delivery problem with time windows and transshipment. INFOR. 2006;44(3):217-227.

87 Powell WB, Bouzaiene-Ayari B, Simao HP. Dynamic models for freight transportation. In: Barnhart C, Laporte G. Handbooks in Operations Research and Management Science: Transportation. North Holland 2007. p. 285-365.

88 Powell WB. An operational planning model for the dynamic vehicle allocation problem with uncertain demands. Transportation Research B. 1987;21:217-232.

89 Powell WB, Sheffi Y, Nickerson KS, Butterbaugh K, Atherton S. Maximizing profits for North American Van Lines' truckload division: A new framework for pricing and operations. Interfaces. 1988;18(1):21-41.

90 Yang J, Jaillet P, Mahmassani.H.S. On-line algorithms for truck fleet assignment and scheduling under real-time information. Transportation Research Record. 1998;1667:107-113.

91 Tjokroamidjojo D, Kutanoglu E, Taylor GD. Quantifying the value of advance load information in truckload trucking. Transportation Research Part E. 2006;42:340-357.

92 Mes M, van der Heijden M, van Harten A. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. European Journal of Operational Research. 2007;181:59-75.

93 Gutenschwager K, Niklaus C, Vob S. Dispatching of an electric monorail system:applying metaheuristics to an online pickup and delivery problem. Transportation Science. 2004;38:434-446.

94 Swihart MR, Papastavrou JD. A stochastic and dynamic model for the single vehicle pick-up and delivery problem. European Journal of Operational Research. 1999;114:447-464.

95 Waisanen HA, Shah D, Dahleh MA. A dynamic pickup and delivery problem in mobile networks under information constraints. IEEE Transactions on Automatic Control. 2008;53:1419-1433.

96 Waisanen HA, Shah D, Dahleh MA. Fundamental performance limits for multi stage vehicle routing problems. Submitted to Operations Research. 2009.

97 Mitrovic-Minic S, Laporte G. Waiting strategies for the dynamic pickup and delivery problem with time windows. Transportation Research Part B. 2004;38:635-655.

98 Gendreau M, Guertin F, Potvin JY, Seguin R. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Transportation research. Part C. 2006;14(3):157-174.

99 Branke J, Middendorf M, Noeth G, Dessouky M. Waiting strategies for dynamic vehicle routing. Transportation Science. 2005;39:298-312.

100 Saez C, Cortés C, Nunez A. Hybrid adaptive predictive control for the multivehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. Computers & Operations Research. 2008;35:3412-3438.

101 Ghiani G, Manni E, Quaranta A, Triki C. Anticipatory algorithms for same-day courier

dispatching. Transportation Research Part E. 2009;45:96-106.

102 Hentenryck, P.Van, Bent RW. Online stochastic combinatorial optimization. Operations Research. 2004;52:977-987.

103 Cordeau JF, Laporte G. The dial-a-ride problem : Modeles and algorithms. Annals of Operations Research. 2007;153:29-46.

104 Psaraftis HN. A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. Transportation Science. 1980;14:130-154.

105 Madsen OBG, Ravn HF, Rygaard JM. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Annals of Operations Research. 1995;60:193–208.

106 Horn MET. Fleet scheduling and dispatching for demand-responsive passenger services. Transportation Research Part C. 2002;10:35-63.

107 Attanasio A, Cordeau JF, Ghiani G, Laporte G. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. Parallel Computing. 2004;30:377–387.

108 Coslovich L, Pesenti R, Ukovich W. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. European Journal of Operational Research. 2006;175:1605-1615.

109 Beaudry A, Laporte G, Melo T, Nickel S. Dynamic transportation of patients in hospitals. O.R. Spectrum. 2008;32:1-31.

110 Xiang Z, Chu C, Chen H. The study of a dynamic dial-a-ride problem under time dependent and stochastic environments. uropean Journal of Operational Research. 2008;185:534-551.

111 Cordeau JF, Laporte G, Potvin JY, Savelsbergh MWP. Transportation on demand. In: Transportation, editor. Handbooks in Operations Research and Management Science. Vol 14. North-Holland,Amsterdam 2007. p. 429-466.

112 Desrosiers J, Dumas Y, Soumis F. The multiple vehicle dial-a-ride problem. Computer-Aided Transit Scheduling, Lecture Notes in Economics and Mathematical System. 1988;308:15–27.

113 Baugh JW, Kakivaya GKR, Stone JR. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Engineering Optimization. 1998;30:91–123.

114 Toth P, Vigo D. Heuristic algorithms for the handicapped persons transportation problem. Transportation Science. 1997;31:60–71.

115 Hart SM. The modeling and solution of a class of dial-a-ride problems using simulated annealing. Control and Cybernetics. 1996;25:131–157.

116 Toth P, Vigo D. Fast local search algorithms for the handicapped persons transportation problem. In: Osman IH, Kelly JP, editors. Meta-Heuristics: Theory and Applications. 1996. p. 677–690.

117 Kikuchi S, Rhee JH. Scheduling method for demand-responsive transportation system. Journal of Transportation Engineering. 1989;115:630–645.

118 Potvin JY, Rousseau JM. Constraint-directed search for the advanced request dial-a-ride problem with service quality constraint. Computer Science and Operations Research: New Developments in Their Interfaces. 1992 457–474.

119 Diana M, Dessouky MM. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. Transportation Research Part B. 2004;38:539–557.

120 Dessouky M, Rahimi M, Weidner M. Jointly optimizing cost, service, and environmental

performance in demand-responsive transit scheduling. Transportation Research Part D. 2003;8:433–465.

121 Fu L. Scheduling dial-a-ride paratransit under time-varying, stochastic congestion. Transportation Research Part B. 2002;36:485–506.

122 FU L. Analytical model for paratransit capacity and quality-of-service analysis. Transportation Research Record. 2003;1841:81-89.

123 Campbell AM, Savelsbergh M. Efficient insertion heuristics for vehicle routing and scheduling problems. Transportation Science. 2004;38(3):369–378.

124 Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research. 1987;35(2):254–265.

125 Dessouky MM, Diana J. A new regret insertion heuristic for solving largescale dial-a-ride problems with time windows. Transportation research Part B. 2004;38:539-557.

126 Savelsbergh MWP. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. ORSA Journal on Computing. 1992;4:146–154.

127 Jorgensen RM, Larsen J, Bergvinsdottir KB. Solving the dial-a-ride problem using genetic algorithms. Journal of the Operational Research Society. 2007;58(10):1321-1331.

128 Beasley J. Route first-cluster second methods for vehicle routing. Omega. 1983;11(403-8).

129 Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research. 2004;31:1985-2002.

130 Lacomme P, Prins C, Ramdane-Chérif W. A Genetic Algorithm for the Capacitated Arc Routing Problem and Its Extensions. In: Proceedings of the EvoWorkshops on Applications of Evolutionary Computing. 2001. p. 473-483.

131 Lacomme P, Prins C, Ramdane-Chérif W. Competitive memetic algorithms for arc routing problems. Annals of Operations Research. 2004 159-85.

132 Lacomme P, Prins C, Sevaux M. Multi-objective capacitated arc routing problem. In: Fonseca C, editor. Evolutionary multi-criterion optimization. Lecture Notes in Computer Science ed. 2003. p. 550-64.

133 Duhamel C, Lacomme P, Prins C, Prodhon C. A memetic approach for the capacitated location routing problem. EU/meeting 2008. 2008 October 23-24.

134 Evolutionary local search for the minimum energy broadcast problem. In: Wolf S, Merz P. Proceedings of the Eighth European Conference on Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science 4972 (2008): Springer; 2008. p. 61-72.

135 Artigues C, Michelon P, Reusser S. Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research. 2003;149(2):Pages 249-267.