Optimisation de tournées de véhicules par programmation par contraintes : conception et développement d'un solveur industriel

Sylvain Ducomman

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Génie Industriel**

Arrêté ministériel : 25 mai 2016

Présentée par

Sylvain Ducomman

Thèse dirigée par **Bernard Penz**
et co-encadrée par **Hadrien Cambazard**

préparée au sein du laboratoire **G-SCOP**
et de l'école doctorale **IMEP2**

Optimisation de tournées de véhicules par programmation par contraintes : conception et développement d'un solveur industriel

Thèse soutenue publiquement le **09 mai 2017**,
devant le jury composé de :

M. Gilles Pesant
Professeur, École Polytechnique de Montréal, Rapporteur

M. Christian Prins
Professeur, Université de Technologie Troyes, Rapporteur

M. Dominique Feillet
Professeur, École des Mines de Saint-Etienne, Examinateur

M. Philippe Laborie
Maître de recherche IBM France Lab, Examinateur

M. Pierre Schaus
Professeur assistant, Université Catholique de Louvain, Examinateur

M. Marc Bannelier
Directeur technique GEOCONCEPT, Invité

M. Hadrien Cambazard
Maître de conférences, Grenoble INP, Co-Encadrant de thèse

M. Bernard Penz
Professeur, Grenoble INP, Directeur de thèse

OPTIMISATION DE TOURNÉES DE VÉHICULES PAR PROGRAMMATION PAR CONTRAINTES : CONCEPTION ET DÉVELOPPEMENT D'UN SOLVEUR INDUSTRIEL

SYLVAIN DUCOMMAN

Mai 2017

RÉSUMÉ

Les problèmes de tournées de véhicules sont des problèmes d'optimisation combinatoire épineux avec des enjeux économiques et environnementaux importants au sein de la chaîne logistique. Le problème fondamental est de desservir des clients avec un ensemble de véhicules de façon à minimiser la distance totale parcourue. En pratique, il y a une grande variété d'objectifs et de contraintes additionnelles, liées à la législation et à la diversité des domaines d'applications. Ces problèmes de tournées sont très fréquents pour de nombreuses industries et la conception d'approches de résolution génériques est devenue une question de recherche importante.

Cette thèse porte sur la conception et le développement d'un nouveau moteur de résolution pour les logiciels de tournées de véhicules proposés par l'entreprise GEOCONCEPT. Le solveur mis au point s'appuie sur la programmation par contraintes (PPC) pour améliorer la flexibilité (prise en compte de contraintes additionnelles), la déclarativité et la maintenance qui sont les limites des solveurs actuels de GEOCONCEPT fondés sur la recherche locale.

Dans un premier temps, un modèle de graphe est établi pour la représentation unifiée des données et de nombreuses contraintes métiers. La résolution s'effectue par des approches à base de voisinage large disponibles dans les solveurs de PPC modernes. On peut ainsi traiter des instances de très grandes tailles efficacement tout en conservant une approche déclarative pour exprimer une classe très large de problèmes de tournées de véhicules. Dans un second temps, des modèles PPC s'appuyant sur des représentations redondantes du problème sont proposés afin de renforcer le filtrage. Nous nous intéressons en détails aux mécanismes de filtrage c'est-à-dire aux processus d'élimination des valeurs infaisables ou sous-optimales dans les domaines des variables. Ces algorithmes permettent de simplifier rapidement le problème et de fournir des bornes inférieures afin d'évaluer la qualité des solutions obtenues. Les bornes inférieures sont obtenues en résolvant des relaxations du plus célèbre des problèmes de la Recherche Opérationnelle : le problème du voyageur de commerce (TSP). Ce problème est le cœur de la contrainte globale WEIGHTEDCIRCUIT permettant de modéliser les problèmes de tournées en PPC. Nous proposons de nouveaux mécanismes de filtrage pour cette contrainte s'appuyant sur trois relaxations du TSP. Ces relaxations sont comparées sur les plans théorique et expérimental. L'originalité de ce travail est de proposer un nouvel algorithme de filtrage permettant de raisonner à la fois sur les successeurs di-

rects d'un client et sur sa position dans la tournée. Ces raisonnements sont particulièrement utiles en présence de contraintes de fenêtres de temps, très communes dans les problèmes industriels.

Le nouveau moteur de résolution offre d'excellentes performances sur des problèmes académiques et industriels tout en proposant des bornes inférieures informatives à des problèmes industriels réels.

ABSTRACT

Vehicle routing problems are very hard combinatorial optimization problems with significant economic and environmental challenges. The fundamental problem is to visit a set of customers with a given fleet of vehicles in order to minimize the total distance travelled. Moreover, these problems arise with a wide variety of objectives and additional constraints, related to the legislation and the diversity of industrial sectors. They are very common for many industries and the design of generic solvers has become an important research issue.

This thesis focuses on the design and implementation of a new solver for the vehicle routing services offered by the company GEOCONCEPT. The proposed solver is based on constraint programming (CP) to improve flexibility (ability to take additional constraints into account), declarative modelling and maintenance, which are the limits of current GEOCONCEPT solvers based on local search.

Firstly, a graph model is established to provide a common representation of the input-data and the numerous business constraints. The resolution is performed using large neighbourhood search methods available in modern CP solvers. It is thus possible to deal with large instances efficiently with a declarative approach where a broad class of vehicle routing problems can be modelled. Secondly, several CP models based on redundant views of the problem are proposed to strengthen the filtering. We focus on the filtering mechanisms for removing infeasible or suboptimal values in the domains of the variables. These algorithms can quickly simplify the problem and derive lower bounds to assert the quality of the solutions found. The lower bounds are obtained by solving relaxations of the most famous problem in Operations Research: the Traveling Salesman Problem (TSP). This problem is the core of the global constraint WEIGHTEDCIRCUIT for modelling routing problems in CP. We propose new filtering algorithms for this constraint based on three relaxations of the TSP. These relaxations are compared theoretically and experimentally. The originality of this work is to propose a new filtering algorithm for reasoning on the direct successors of a customer as well as his position in the tour. It is particularly useful in the presence of time window constraints, which are very common in industrial problems.

The new solver shows excellent performance on academic and industrial problems and can compute informative lower bounds for real-life problems.

Point n'est besoin d'espérer pour entreprendre,
ni de réussir pour persévérer.

— Guillaume I^er^ d'Orange-Nassau

REMERCIEMENTS

Je remercie tout d'abord GEOCONCEPT qui m'a offert l'opportunité de construire cette thèse et qui m'a accordé sa confiance pour continuer dans leur équipe ce projet professionnel. Merci aux différentes équipes de GEOCONCEPT, l'équipe *Cloud*, avec Samuel Fenollosa, l'équipe *GSS*, avec Gilles Legoff et Eric Dupérier, l'équipe *Moteur* avec Laurent Pichon et Vandana Lemanchec qui me suivent dans le développement du nouveau moteur. Je remercie également les collègues de Fontaine, Elise, Marie, Sylvain, Hervé et Edo qui rendent l'environnement de travail très agréable et convivial. Je leur propose de la lecture en attendant les parties de pétanques le midi.

Ensuite, je tiens à remercier mes rapporteurs, Christan Prins et Gilles Pesant, ainsi que mes examinateurs Dominique Feillet, Philippe Laborie et Pierre schaus qui me font l'honneur d'être dans mon jury. Sans oublier mes deux tuteurs académiques, Bernard Penz et Hadrien Cambazard, qui ont su m'apporter connaissances, suivi et engagement. Sans eux, la réalisation de cette thèse n'aurait pas pu être conçue de cette sorte.

Je souhaite remercier tout le laboratoire *G-SCOP* pour l'accueil et les ressources mises en place pour le bon déroulement de la thèse, tout comme les collègues du laboratoire, Olivier, Matthieu, Gricha, Alex, Tom et bien d'autres. Merci aussi aux amis, Cyril, Lucile, Julien, Hugo et Caro pour les bons moments passés ensemble.

Je tiens à remercier maintenant mes parents Florence et Jacques, mes frères, Kevin et Clément, sans qui je n'aurai pas pu persévérer dans cette voie sans leur précieux soutien et leurs encouragements. Je remercie Françoise et Alain qui ont su s'intéresser et m'encourager dans mes efforts, ainsi que Bertand pour ses conseils de docteur et Annabelle pour ses corrections linguistiques, sans oublier Harry et Hella.

Enfin, je remercie Marie-Eve, ma fiancée, sans qui cette thèse n'aurait pu aboutir. Je la remercie de son amour, de sa confiance, de son soutien dans les moments difficiles et des merveilleux projets qu'ils nous restent à accomplir.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

La logistique et le transport sont devenus des éléments essentiels de l'économie moderne. L'optimisation de tournées de véhicules est donc un enjeu important pour une meilleure maîtrise des coûts et de la satisfaction client. Les problèmes de tournées de véhicules consistent à desservir des clients avec un ensemble de véhicules tout en minimisant la distance totale parcourue. Ce problème de base est plus complexe en présence de contraintes métiers, comme l'utilisation de véhicules différents ou encore la livraison de multiples produits, qui permettent d'être au plus proche de la problématique de l'utilisateur. Depuis 25 ans, la société GEOCONCEPT propose des solutions pour l'optimisation de tournées de véhicules. L'expérience acquise par l'entreprise justifie aujourd'hui la mise au point d'un nouveau moteur de résolution de tournées de véhicules. Par ailleurs, la déclarativité des contraintes métiers et la maintenance sont devenues pour GEOCONCEPT des enjeux de plus en plus importants. L'objectif de cette thèse est donc la conception et le développement d'un nouveau solveur répondant à ces enjeux actuels.

De nombreuses méthodes, s'appuyant sur la programmation mathématique ou la théorie des graphes, ont été proposées pour répondre à ces problèmes combinatoires. Ces dernières sont très efficaces pour des problèmes spécifiques dont les caractéristiques sont clairement définies. Cependant, elles manquent de flexibilité quant à la prise en compte de la diversité des contraintes métiers. De plus, la taille des problèmes peut être très grande (plusieurs centaines de clients à visiter) et l'utilisation de méthodes exactes est souvent inappropriée en termes de temps de résolution. Des méthodes heuristiques ont donc été proposées pour obtenir de bonnes solutions dans des temps de résolutions acceptables.

La *programmation par contraintes* (*PPC*) est un paradigme de programmation permettant de modéliser et résoudre des problèmes d'optimisation combinatoire. Elle offre des possibilités de résolution très variées dans un langage déclaratif très riche, ce qui permet une meilleure intégration des diverses contraintes métiers. La *PPC* se présente comme un cadre adapté à l'implémentation et la coopération de techniques de résolution très différentes comme la recherche locale, la théorie des graphes ou la relaxation lagrangienne.

Pour toutes ces raisons, GEOCONCEPT a choisi la *programmation par contraintes* comme outil de résolution pour le nouveau solveur industriel

de tournées de véhicules.

Le problème cœur des tournées de véhicules est le *problème du voyageur de commerce* (*TSP*). C'est pourquoi une partie de nos travaux ont porté sur ce dernier. Dans un premier temps, nous proposons une étude des relaxations pour le *TSP*. La résolution de ces relaxations permet d'obtenir des bornes inférieures du problème afin d'évaluer la qualité des solutions obtenues par le solveur.

Dans un second temps, nous proposons différentes modélisations en *PPC* pour le *problème du voyageur de commerce*. Nous améliorons les techniques de résolution grâce à de nouveaux mécanismes de filtrage pour la contrainte globale WEIGHTEDCIRCUIT. Ces algorithmes s'appuient sur les trois relaxations du *TSP* étudiées précédemment. Les différents algorithmes de filtrage s'avèrent complémentaires dès lors que des contraintes additionnelles viennent s'ajouter au problème, comme par exemple la contrainte de fenêtres de temps, très commune dans les problèmes industriels.

Nous étendons ces développements au *problème de tournées de véhicules avec fenêtres de temps et capacité* (*CVRPTW*) . Nous proposons de considérer le *CVRPTW* comme un *problème du voyageur de commerce avec fenêtres de temps* à travers une modélisation qui permet de réutiliser les résultats obtenus sur la WEIGHTEDCIRCUIT.

Les modèles proposés pour les problèmes fondamentaux, le *TSP* et le *CVRPTW*, servent de briques de base aux modèles intégrant les contraintes additionnelles dans le moteur de résolution. De plus, les algorithmes conçus pour la WEIGHTEDCIRCUIT peuvent fournir des bornes inférieures informatives pour des problèmes industriels réels. Le moteur de résolution montre de très bons résultats sur des données réelles avec différentes contraintes métiers.

Les différentes contributions de la thèse sont les suivantes :

1. un nouvel algorithme de filtrage pour la WEIGHTEDCIRCUIT permettant d'exploiter et de raisonner sur les successeurs directs d'un client et sa position dans la tournée. Les bornes inférieures ainsi obtenues améliorent significativement les bornes d'Held et Karp et du problème d'affectation pour des problèmes avec fenêtres de temps,
2. l'intégration des différents résultats dans le nouveau solveur de problèmes industriels. Ce dernier montre de bonnes performances sur divers problèmes clients et permet de fournir des bornes inférieures informatives sur ces problèmes.

Le chapitre 1 se focalise sur le contexte industriel et les objectifs de la thèse. Le Chapitre 2 présente un état de l'art et donne les notions nécessaires à la lecture de cette thèse. Le Chapitre 3 est dédié à l'étude de bornes inférieures pour le *problème du voyageur de commerce avec et sans fenêtres de temps*. Ce dernier est traité du point de vue de la *programmation par contraintes* en analysant différents modèles (Chapitre 4). Nous retrouvons, dans ce chapitre, la description des différents algorithmes de filtrage pour la contrainte WEIGHTEDCIRCUIT. Le *problème de tournées de véhicules avec fenêtres de temps et capacité* est étudié dans le Chapitre 5. Nous terminons par la comparaison expérimental du nouveau moteur de résolution, *OtSolver*, avec le moteur actuel, *TourSolver*, pour des problèmes réels de tournées de véhicules (Chapitre 6).

1

CONTEXTE ET APPLICATIONS INDUSTRIELLES

Dans ce chapitre, nous allons nous intéresser au contexte de cette thèse afin de mieux comprendre les enjeux industriels. Les choix de recherche effectués tant au niveau industriel qu'au niveau académique seront explicités.

1.1 PRÉSENTATION DE GEOCONCEPT

Cette section est dédiée à la présentation de l'entreprise GEOCONCEPT et du contexte de la thèse CIFRE.

1.1.1 *Généralités*

Sous l'effet de l'industrialisation et de la mondialisation, la logistique et le transport sont devenus des éléments essentiels de l'économie moderne. En effet, en 2013, en France, la branche transport et entreposage représentait 5% de la production nationale et les dépenses de transport représentaient 18% du produit intérieur brut [Repb]. C'est pourquoi ces secteurs sont devenus des enjeux stratégiques de développement et de compétitivité. De plus, en France, 80% des transports de marchandises s'effectuent par la route [Repa]. Il est donc important de maitriser et d'optimiser ces échanges de marchandises.

GEOCONCEPT propose entre autres des solutions pour l'optimisation de tournées de véhicules. C'est une société française d'édition de logiciels d'optimisation et d'applications cartographiques destinés à l'industrie. Elle a été fondée en 1990 et se place déjà parmi les sociétés les plus performantes d'Europe en la matière. Son siège social se situe à Bagneux et emploie environ 60 personnes. À la fin de l'année 2012, GEOCONCEPT a acheté Opti-Time, une société spécialisée dans l'optimisation de tournées de véhicules.

1.1.2 *Contexte de l'entreprise*

Le rachat d'Opti-Time a permis à GEOCONCEPT de faire l'acquisition de leurs produits, notamment le moteur d'optimisation *TourSolver*. GEOCONCEPT possédait également son propre moteur d'optimisation nommé *Dispatcher* s'appuyant sur le solveur d'IBM du même nom. Ces deux moteurs ont chacun leurs avantages et leurs inconvénients. A titre d'exemple, *TourSolver* est reconnu pour posséder une plus grande adaptabilité que son ancien concurrent *Dispatcher* qui lui, se caractérise par une plus grande robustesse, dans la mesure où la taille des problèmes qu'il peut traiter est plus élevée. Les deux moteurs d'optimisation sont redondants, puisqu'ils cherchent tous les deux à optimiser des tournées de véhicules, mais ils sont aussi complémentaires, ayant chacun des points forts. Par sa volonté d'améliorer et d'uniformiser la performance de ses produits, GEOCONCEPT souhaite fusionner les deux produits en développant un nouveau moteur d'optimisation plus robuste et plus adapté aux problèmes typiques rencontrés par les clients. Ma thèse s'inscrit dans ce contexte, la construction du nouveau moteur d'optimisation fondé sur la *programmation par contraintes*.

1.1.3 *Contraintes des utilisateurs*

Le principe des deux moteurs d'optimisation est de gérer une flotte de véhicules ainsi qu'un ensemble de clients. Le but est d'affecter les différents clients aux différents véhicules, d'ordonnancer les visites pour chaque véhicule, tout en minimisant un coût global. Dans la littérature, nous retrouvons ce problème sous le nom de *problème de tournées de véhicules* (*VRP*). Ce dernier consiste généralement à minimiser le nombre de véhicules utilisés pour ensuite minimiser la distance totale parcourue. Or, en pratique, la minimisation ne concerne pas seulement le nombre de véhicules. En effet, elle peut concerner aussi le coût de transport des véhicules, le coût de non prise en compte d'un client par un véhicule, ou encore, le coût du temps de travail des chauffeurs.

De plus, les deux moteurs d'optimisation doivent répondre à une multitude de contraintes données par les utilisateurs. Nous retrouvons des contraintes largement étudiées dans la littérature telles que l'ajout de fenêtres de temps pour les visites, l'hétérogénéité des véhicules de la flotte ou encore la présence de plusieurs dépôts (point de départ des camions). Cependant, il y a aussi des demandes très particulières, comme par exemple la possibilité pour un chauffeur de s'arrêter dans un hôtel la nuit ou encore de pouvoir effectuer des visites entrecoupées de pauses. Ces différentes contraintes des utilisateurs permettent au mieux de s'approcher des con-

traintes métiers, comme par exemple la prise en compte de la législation du temps de conduite ou encore la gestion des équipages.

Beaucoup de contraintes sont venues s'ajouter au fil des années, ce qui rend aujourd'hui plus complexe l'ajout de nouvelles demandes des utilisateurs. En effet, la modélisation utilisée au départ rend très difficile l'intégration de certaines contraintes, qui parfois peuvent compliquer la résolution. De plus, les méthodes de résolution ne s'adaptent pas obligatoirement aux nouvelles demandes. C'est pourquoi, il devient nécessaire de construire un nouvel outil permettant d'une part, d'appréhender ces demandes déjà existantes et d'autre part, d'offrir une plus grande adaptabilité du produit aux problèmes futurs.

1.1.4 *Le choix de la programmation par contraintes*

De part les différences entre les deux moteurs d'optimisation et de l'expérience acquise au fil du temps par ses concepteurs, le nouveau moteur se doit avant tout d'être modulaire et évolutif. En effet, il devra répondre à des contraintes diverses et variées tout en étant capable de pouvoir en intégrer de nouvelles. Cette modularité devra se retrouver à chaque étape du processus de résolution c'est-à-dire aussi bien au niveau de l'interface utilisateur qu'au niveau des méthodes de résolution. La modélisation des problèmes à traiter sera une étape importante dans la conception de l'outil, tout comme son architecture.

Pour garantir cette modularité et cette évolutivité de l'outil, le choix à été fait d'utiliser la *programmation par contraintes* (*PPC*). En effet, l'un des avantages de la *programmation par contraintes* est sa grande flexibilité. Il est aisé d'exprimer un problème et ses contraintes dans un programme en *PPC*, car la modélisation est intuitive. L'ajout de contraintes ne provoque pas d'incompatibilité vis-à-vis des autres contraintes. C'est-à-dire qu'il est possible d'ajouter des contraintes sans redéfinir un nouveau modèle. Certes les performances peuvent être impactées mais le problème respectera toutes les contraintes ajoutées.

Nous pouvons utiliser la *programmation par contraintes* de deux façons différentes. La première consiste à l'utiliser comme une boîte noire contenant toutes les contraintes d'un problème et permettant de vérifier très rapidement, grâce au processus propre de la *PPC*, si une solution respecte toutes les contraintes. La seconde consiste à exprimer clairement un modèle en *programmation par contraintes* et de pouvoir utiliser tous les outils offerts par la *PPC* (propagation de contraintes, recherche locale...) pour trou-

ver une solution satisfaisante. Ces deux utilisations de la *PPC* permettent d'augmenter la modularité du moteur d'optimisation. La première utilisation permet de vérifier facilement si une solution, trouvée grâce aux mécanismes propres de la *PPC* ou bien grâce à des algorithmes dédiés, satisfait toutes les contraintes des utilisateurs.

Le choix s'est porté sur la bibliothèque *ortools* [OPF14] comme solveur pour la *programmation par contraintes*. Ce solveur a l'avantage d'être open-source, sous licence Apache 2.0, et d'être reconnu dans la communauté *PPC*, grâce au challenge Minizinc [Net+07]. De plus, il possède une librairie dédiée aux problèmes de tournées de véhicules permettant de mettre en place beaucoup de techniques utiles.

1.2 MODÉLISATION

Pour modéliser les différents problèmes à résoudre, il est important d'identifier les contraintes essentielles auxquelles le solveur doit absolument répondre. Ensuite les contraintes plus spécifiques doivent pouvoir être intégrées facilement.

1.2.1 *Identification des contraintes essentielles*

De façon générale, le solveur doit gérer une flotte de véhicules ainsi qu'une base de clients à visiter. L'objectif est de visiter tous les clients, ou le maximum d'entre eux, par la flotte de véhicules. Les visites sont de deux types.

Le premier type de visite consiste à livrer chez le client une certaine quantité d'un produit. Le plus souvent les véhicules ont une capacité fixée, qui peut varier d'un véhicule à l'autre. Dans la pratique, les véhicules ne transportent pas obligatoirement un seul type de produit. Un véhicule peut posséder une capacité maximale globale (charge, poids et/ou volume, à ne pas dépasser) mais aussi une capacité maximale pour chaque type de produits. De plus, chaque véhicule peut avoir des caractéristiques différentes, aussi bien pour les capacités que pour les coûts d'utilisation. Ce type de problème est alors appelé *problème de tournées de véhicules avec flotte hétérogène*.

Nous pouvons aussi retrouver des contraintes de type collecte et livraison (*PDP*), c'est-à-dire que chaque véhicule peut, chez le client, livrer ou collecter les produits. Dans la littérature, ce type de problème se nomme *problème de tournées de véhicules avec collecte et livraison* (*PDPVRP*). Il existe plusieurs types de problèmes de *PDPVRP*. Il est possible pour un véhicule

de collecter en premier tous les produits chez des clients pour ensuite livrer les produits chez d'autres clients (*problème de tournées de véhicules avec rechargement* (*VRPB*)). Le problème symétrique existe avec d'abord les livraisons, puis des collectes. Nous pouvons citer à titre d'exemple le cas des sociétés de ramassage de linge sale pour l'hôtellerie puisque le linge sale ne doit pas être en contact avec le linge propre. Ainsi, le véhicule livre du linge propre dans différents hôtels pour ensuite récupérer le linge sale.

Un second type de visites classiques représente des visites de maintenance. Il ne s'agit plus ici de livrer ou récupérer un produit, mais de déplacer un technicien de maintenance entre ses sites d'interventions. Pour ce type de visite, il est fréquent d'avoir des temps de visite longs et devant être parfois découpés en plusieurs visites. Ces visites peuvent en effet se dérouler pendant toute une journée voire plusieurs journées et il faut donc gérer les différentes pauses des techniciens. Quand elles se déroulent sur plusieurs jours, il faut alors gérer les nuits du technicien et les prendre en compte au moment de la résolution du problème. Des plannings sur plusieurs jours doivent être alors pris en compte. Cette gestion est l'une des grandes difficultés des solveurs de tournées. La plupart sont destinés à résoudre des *problème de tournées de véhicules* sur une journée et il est alors difficile de modéliser un horizon temporel plus grand. Les nuitées peuvent être de différents types car le technicien peut retourner au dépôt poser son véhicule ou bien passer la nuit dans un hôtel sur le chemin entre deux visites.

Pour les deux problèmes décrits, nous retrouvons bien évidemment les contraintes de législation sur le temps de travail et le temps de conduite lorsque les temps de trajet sont longs.

Types contraintes	Problèmes académiques
Demandes de différents produits	Multi-produits
Véhicules de capacités différentes	Flotte hétérogène
Collectes et livraisons	Collecte et livraison
Livrer avant, collecter ensuite	Rechargement
Tournées de maintenance	Multi-périodes
Durée de visites longues	Multi-jours
Legislation du temps de travail	Planning des conducteurs

Tableau 1 – Récapitulatif des contraintes et des problèmes académiques associés

Le Tableau 1 donne un récapitulatif des différents types de contraintes rencontrées avec leurs dénominations académiques.

1.2.2 *Modélisation des contraintes utilisateurs et des objectifs*

La première étape du travail a consisté à recenser les différentes contraintes que l'outil doit gérer. Ensuite, les contraintes importantes ont été formalisées pour être incorporées à un unique modèle. Pour ne pas alourdir ce chapitre, le modèle détaillé est proposé dans l'Annexe A. Il est à noter que les contraintes présentées correspondent aux contraintes récurrentes que 80% des utilisateurs demandent.

La fonction objectif est une partie importante dans l'optimisation de tournées de véhicules, et celle-ci diffère en fonction des utilisateurs et de leurs attentes. Nous pouvons voir dans l'Annexe A que les coûts sont aussi très divers. Chaque utilisateur a sa vision du problème et son objectif propre. Certains utilisateurs préfèrent minimiser le nombre de véhicules utilisés et ensuite minimiser la distance totale parcourue (comme on peut le retrouver classiquement dans la littérature). Cependant, d'autres utilisateurs veulent utiliser la totalité de leur flotte de véhicules mais minimiser le temps d'attente ou équilibrer le temps de travail des différents chauffeurs. La fonction objectif du problème sera alors décidée par l'utilisateur de l'outil avec le catalogue des coûts pouvant être utilisés.

1.3 MODÉLISATION DES PROBLÈMES PAR UN GRAPHE UNIFIÉ

Dans cette section, nous allons étudier les choix effectués en terme de modélisation des problèmes utilisateurs au sein du moteur d'optimisation. Cette modélisation sera utilisée par les différentes techniques de résolution dans le moteur d'optimisation.

1.3.1 *Modèle commun fondé sur les graphes*

Afin de modéliser au mieux les problèmes utilisateurs, le moteur d'optimisation doit utiliser un modèle commun afin de pouvoir retranscrire les données et les résultats. Ce modèle peut s'apparenter à un graphe avec des caractéristiques spécifiques et des règles spécifiques. Ce graphe est construit à partir des données épurées (déjà pré-traitées, voir Section 1.4.1) et un module de vérification est intégré directement afin de vérifier rapidement, grâce à la *PPC*, si une solution est valide. Un diagramme UML est fourni Annexe B.

On modélise un arrêt d'un véhicule par un sommet dans le graphe. Chaque sommet possède des caractéristiques, comme par exemple la demande, le temps de service qui sont propres aux visites. Un sommet peut être de différents types. Le premier est un sommet représentant directement une visite. Le deuxième correspond à un sommet de début d'un tour, par exemple un dépôt, un hôtel. Enfin, le troisième est le sommet de fin d'un tour. Ces deux derniers types sont reliés par une ressource. Cette dernière représente un tour (ou un véhicule), elle possède des données de restriction, comme par exemple, le temps total maximum du tour, la distance totale maximum ou encore la capacité du tour.

Les sommets sont aussi connectés par des arcs qui sont caractérisés par un sommet source et un sommet puits. Ces arcs contiennent des consommables, c'est-à-dire des quantités propres consommées entre les deux visites, ce peut être par exemple la distance les séparant. Si deux visites ne sont pas connectées par un arc alors on peut aussi en déduire qu'elles ne peuvent pas se faire l'une après l'autre.

Un autre objet est une spécialisation d'un arc possédant une ressource, arcResource. Cet objet possède aussi des consommables comme un arc, à la différence que ce consommable est propre à une ressource et que par conséquent, il peut être restreint. On peut prendre comme exemple le temps de trajet entre deux sommets, qui peut varier selon la ressource utilisée ce qui va se traduire par des arcResources différents pour chaque ressource.

Le dernier objet du modèle sont des arêtes possédant plusieurs sommets. Ces arêtes permettent de modéliser des propriétés communes que doivent posséder les sommets. Par exemple, un utilisateur va vouloir qu'un ensemble de sommets soit traité par une même ressource (sans forcément connaître la ressource à l'avance), ou encore que les visites soient effectuées avec un espacement de temps prédéfini. Une solution contient donc

une succession de sommets commençant tous par un sommet de début et finissant par un sommet de fin.

Ce modèle permettra à la fois de communiquer sur les différentes contraintes associées aux problèmes utilisateurs mais aussi d'offrir un processus de vérification.

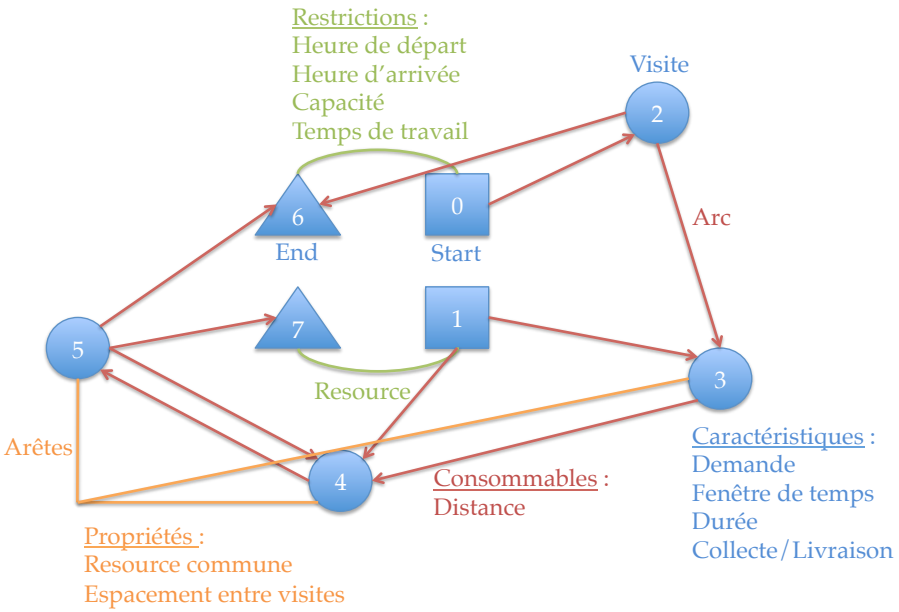


FIGURE 1 – Exemple d'un graphe commun

Un exemple de graphe commun est donné Figure 1. Ce dernier possède 8 sommets, dont deux sommets de début de tournées (sommets 0 et 1), deux sommets de fin de tournées (sommets 6 et 7) et 4 sommets représentant des visites (sommets 2, 3, 4 et 5). Chaque sommet de début et de fin de tournée est lié par une ressource. De plus, les sommets peuvent être connectés entre eux par des arcs. Plusieurs sommets peuvent être regroupés dans des arêtes (sommets 3, 4 et 5) afin d'expliciter leur propriétés communes.

1.3.2 *Modélisation des contraintes de nuitées*

Parmi les contraintes les plus difficiles à traiter, nous pouvons retenir les nuitées extérieures ainsi que les pauses déjeuners. Lorsque ce type de problème apparait, plusieurs possibilités de modélisation peuvent être mises en œuvre. Dans ce cas, une modélisation particulière est utilisée per-

mettant de représenter les contraintes des pauses des véhicules. En effet, pour chaque sommet de visite, une structure particulière est utilisée représentant les différentes transitions possibles. Ces dernières peuvent être les pauses déjeuners et les nuitées. Cette structure est représentée Figure 2.

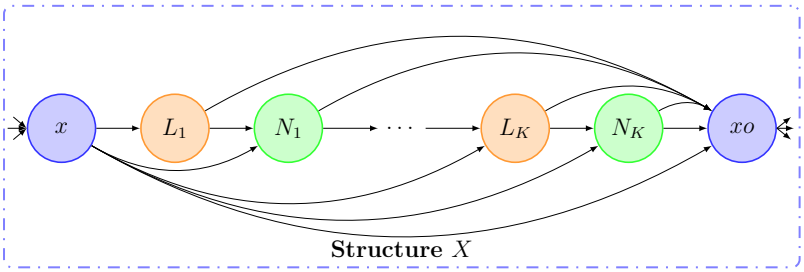


FIGURE 2 – Structure d'un nœud

Lorsque le véhicule traverse un nœud X, il entre dans la structure par le sommet x et en ressort par le sommet xo. Les sommets L représentent les pauses déjeuners et les sommets N les nuitées. Un chemin de x à xo représente les différentes transitions effectuées par le véhicule. À la sortie au sommet xo les variables de temps et de distance sont mises à jour.

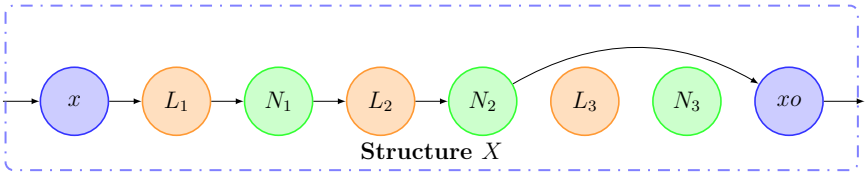


FIGURE 3 – Instanciation d'une structure d'un noeud

Un exemple d'instanciation d'une structure est donné Figure 3. Le véhicule effectue deux pauses déjeuners et deux nuitées. Le sommet xo est alors connecté à la visite suivante avec la mise à jour des temps et des distances correspondantes au sein de la structure.

1.4 ARCHITECTURE GLOBALE DE L'OUTIL

Le nouveau moteur d'optimisation doit pouvoir remplacer les anciens moteurs et fonctionner avec les applications externes utilisées par les outils actuels. Pour cela, la conception doit prendre en compte les différents modules.

1.4.1 *Communication des différents modules*

Le point d'entrée du moteur d'optimisation est constitué des données utilisateurs actuelles des deux outils d'optimisation de tournées, *TourSolver* et *Dispatcher*. Pour gérer cette diversité, nous avons décidé d'établir des formats de données en XML, en définissant le schéma dans un modèle xsd. Ce premier ensemble correspond au module d'entrée du solveur (*Parseur XML*).

Ces données sont les données brutes des utilisateurs, il est important alors d'épurer ces données pour faire une première modélisation qui sera envoyée au moteur de résolution. Ce modèle appelé modèle de transition est le module intermédiaire entre l'interface utilisateur et le moteur de résolution (*Transition Model*). Il sert notamment à vérifier les données, à récupérer les matrices de distance et de temps mais aussi à simplifier les données pour le moteur de résolution.

Ce dernier correspond au dernier module (*OtSolver*) qui a pour but de résoudre/trouver une solution au problème envoyé par le modèle de transition. Le modèle commun basé sur les graphes, vu Section 1.3.1 est alors instancié dans ce module.

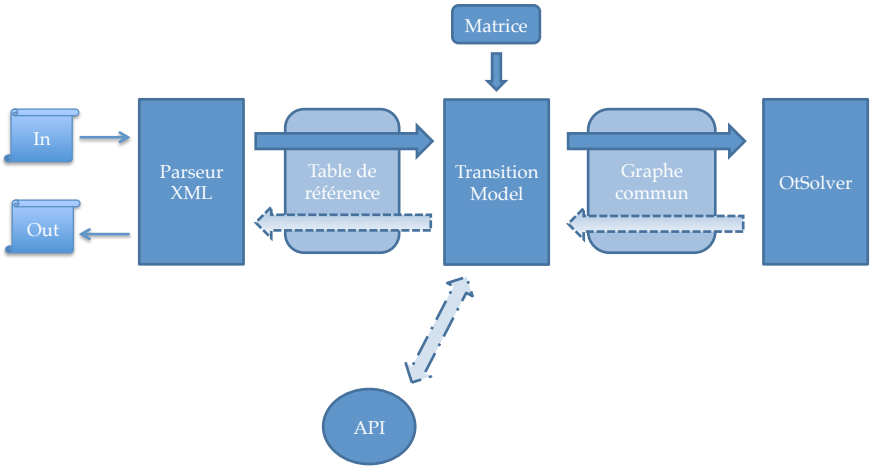


FIGURE 4 – Architecture globale

Ce principe est résumé sur la Figure 4. On y retrouve les trois modules présentés précédemment avec leurs communications en interne. Le module de modèle intermédiaire est connecté avec les différentes matrices de distances et de temps calculées avec un logiciel externe.

1.4.2 *Architecture et procédure de résolution du module de résolution*

Le moteur d'optimisation doit être modulaire et doit pouvoir répondre à des contraintes variées. En optimisation, il existe beaucoup de techniques de résolution et chacune peut être plus adaptée qu'une autre pour résoudre un type de problème. C'est pourquoi, nous proposons une architecture du module de résolution composée de plusieurs sous-modules. Ces derniers pourront utiliser des techniques d'optimisation diverses et répondre à des problèmes spécifiques.

De plus, tous ces modules doivent utiliser la représentation du modèle unifié présenté Section 1.3.1 permettant la vérification rapide d'une solution. Cette vérification utilise la *programmation par contraintes* pour indiquer si une solution est valide ou non. Le module de résolution proposera une identification du cas client utilisé afin d'activer ou non certaines briques pouvant résoudre ce type de problème. Un exemple de procédure de résolution est montré sur la Figure 5

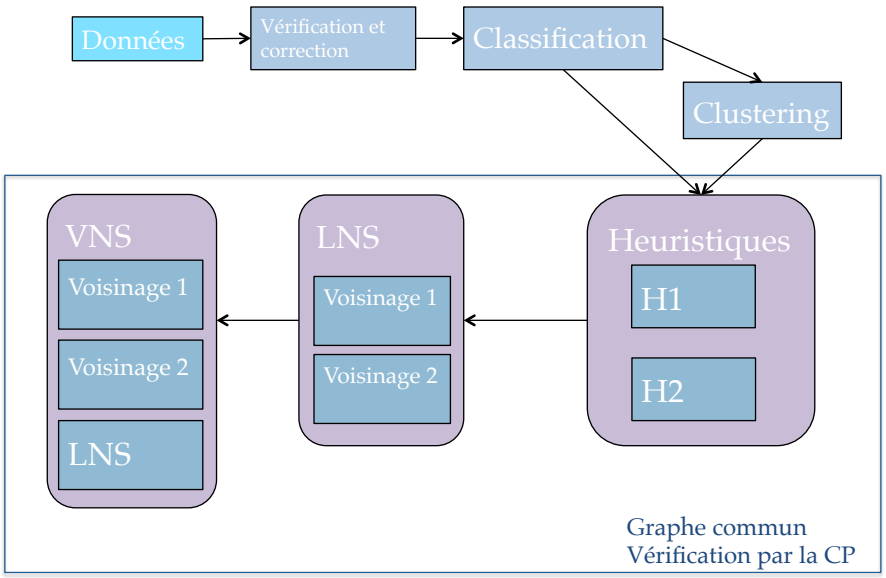


FIGURE 5 – Procédure de résolution

Le module de classification permet d'activer ou non certains modules de résolution afin de résoudre au mieux le problème d'entrée. Il est souvent utile de décomposer spatialement le problème afin d'être plus efficace sur les sous-problèmes générés, c'est le rôle du module de clustering.

CONCLUSION

Ce chapitre a permis de mieux appréhender le contexte dans lequel la thèse a été élaborée. Après avoir présenté l'entreprise GEOCONCEPT et cerné davantage ses besoins ainsi que ceux de ses utilisateurs, nous avons pu exposer différentes modélisations des contraintes clients. La modélisation choisie pour le moteur d'optimisation a par la suite été explicitée.

2

ÉTAT DE L'ART

Dans ce chapitre, nous introduirons, dans un premier temps, des notions de *programmation linéaire*. Dans un deuxième temps, nous étudierons le *problème du voyageur de commerce*. Dans un troisième temps, nous aborderons le *problème de tournées de véhicules*. Enfin, nous finirons par une introduction à la *programmation par contraintes*.

2.1 NOTIONS DE *programmation linéaire*

2.1.1 *Modélisation*

La *programmation linéaire* est une technique qui permet de modéliser et de résoudre un problème d'optimisation combinatoire [Chv83]. Un problème modélisé en *programmation linéaire* minimise ou maximise une fonction objectif sous certaines contraintes. Chaque équation (fonction objectif et contraintes) est linéaire, c'est-à-dire qu'elle peut être interprétée comme une fonction linéaire affine. Un programme linéaire est donc de la forme suivante :

$$(\mathbf{PL}_1) \quad \textit{Min} \qquad \sum_{j=1}^{n} c_j x_j \tag{1a}$$

$$\textit{s.t.} \qquad \sum_{j=1}^{n} a_{i,j} x_j \left\{ \begin{array}{l} = \\ \leqslant \\ \geqslant \end{array} \right\} b_i \qquad \forall i = 1, \ldots, m \tag{1b}$$

$$x \in X \tag{1c}$$

Le vecteur x de dimension n correspond aux variables. Le domaine de définition des variables est noté X. Le vecteur c correspond au vecteur des coûts. Les m contraintes 1b sont des fonctions affines. Dans le cas où X représente l'ensemble des réels ($X = \mathbb{R}^n$), nous parlons de *programmation linéaire* (*PL*). Si X représente l'ensemble des entiers ($X = \mathbb{N}^n$) alors c'est

un programme linéaire en nombres entiers (*PLNE*). Il existe aussi le cas où certaines variables peuvent être entières et d'autres réelles, nous parlerons alors de *programmation linéaire mixte* [Dan98].

Un programme linéaire peut être représenté sous forme matricielle :

$$
\begin{aligned}
(\mathbf{PL}_2) \quad & \textit{Min} && cx && (2a)\\
& \textit{s.t.} && Ax \leqslant b && (2b)\\
& && x \in X && (2c)
\end{aligned}
$$

Sous cette forme, nous pouvons exprimer l'ensemble des solutions réalisables par un polyèdre :

$$P = \{x \in \mathbb{R}^n \ : \ Ax \leqslant b, x \in X\}$$

Une solution $x^* \in P$ est optimale si et seulement si $cx^* \leqslant cx$, $\forall x \in P$.

2.1.2 *Résolution*

L'ensemble des solutions réalisables d'un programme linéaire est un polyèdre et l'ensemble des solutions optimales sont sur ces faces. L'algorithme du simplexe [Dan90] permet alors d'explorer les sommets du polyèdre, plus précisément les points extrêmes, jusqu'à atteindre une solution optimale si elle existe.

Un concept fondamental en *programmation linéaire* est la notion de dualité. Reprenons le programme linéaire sous forme matricielle ($\mathbf{PL}_2$) appelé primal, avec x le vecteur des n variables, c le vecteur des coûts, A la matrice $m \times n$ des contraintes et b le vecteur colonne des seconds membres.

$$
\begin{aligned}
(\mathbf{PL}_3) \quad & \textit{Min} && cx && (3a)\\
& \textit{s.t.} && Ax \leqslant b && (3b)\\
& && x \geqslant 0 && (3c)
\end{aligned}
$$

On associe pour chaque contrainte i ($i = 1, \ldots, m$) une variable duale y_i, le dual du programme linéaire ($\mathbf{PL}_3$) est alors :

$$
\begin{aligned}
(\mathbf{PL}_4) \quad & \textit{Max} && yb && (4a)\\
& \textit{s.t} && yA \leqslant c && (4b)\\
& && y \leqslant 0 && (4c)
\end{aligned}
$$

Nous pouvons remarquer que la matrice des contraintes du dual est la transposée de la matrice des contraintes du primal. De plus, le vecteur des coûts du primal (c) devient le vecteur des seconds membres du dual et inversement.

Un des théorèmes fondamentaux de la dualité est le théorème fort : si ($\mathbf{PL}_3$) admet une solution optimale x^*, alors ($\mathbf{PL}_4$) admet une solution optimale y^* et $cx^* = by^*$. Il existe alors des algorithmes primal-dual afin de résoudre un programme linéaire.

La dualité permet aussi d'effectuer de l'analyse de sensibilité sur un problème. En effet, le dual nous donne accès à ce que l'on nomme les coûts marginaux. La valeur optimale de la variable duale y_i représente le coût d'une petite variation du second membre b_i (coût marginal).

2.1.3 *Relaxation lagrangienne*

L'analyse et la compréhension d'un problème est important en *programmation linéaire* et plus particulièrement en *programmation linéaire* en nombre entiers. En effet, certains problèmes peuvent contenir des sous-problèmes connus avec des algorithmes efficaces pour les résoudre. Une des techniques utiles dans ces situations est de recourir à la relaxation lagrangienne [Ree93], [HKW72]. Soit le programme linéaire en nombre entiers suivant, avec $A = A_1 \cup A_2$ et $|A_1| = m'$:

$$
\begin{aligned}
(\mathbf{PL}_5) \quad & \textit{Min} && cx && (5a)\\
& \textit{s.t.} && A_1 x \leqslant b_1 && (5b)\\
& && A_2 x \leqslant b_2 && (5c)\\
& && x \geqslant 0 && (5d)\\
& && x \in \mathbb{N}^n && (5e)
\end{aligned}
$$

Supposons que le problème soit beaucoup plus facile sans les contraintes 5b qui compliquent significativement la résolution. L'idée de la relaxation lagrangienne est de relaxer ces contraintes en les incluant dans la fonction objectif avec une pénalité. Associons à chaque contrainte de A_1 un multiplicateur $\lambda_i \in \mathbb{R}_+$, $\forall i = 1, \ldots, m'$, appelé multiplicateur lagrangien. Pour des vecteurs données de λ, le sous-problème lagrangien, noté $L(\lambda)$ est le suivant :

$$
\begin{aligned}
(\mathbf{RL}_6) \quad L(\lambda) = \; & \mathit{Min} && cx + \lambda(A_1 x - b_1) && \text{(6a)}\\
& \mathit{s.t.} && A_2 x \leqslant b_2 && \text{(6b)}\\
& && x \geqslant 0 && \text{(6c)}\\
& && x \in \mathbb{N}^n && \text{(6d)}
\end{aligned}
$$

Nous définissons le dual lagrangien de $(\mathbf{PL}_5)$ comme :

$$
(\mathbf{DL}_7) \quad \max_{\lambda \in \mathbb{R}_+^{m'}} L(\lambda) \qquad \text{(7a)}
$$

Remarquons que pour tout $\lambda \in \mathbb{R}_+^{m'}$ et pour toute solution x réalisable de $(\mathbf{RL}_6)$, $L(\lambda) \leqslant cx$. De plus, $(\mathbf{DL}_7)$ est une fonction linéaire par morceau. Le dual lagrangien peut donc se résoudre avec des techniques de *programmation linéaire* mais aussi en utilisant un algorithme de sous-gradient afin d'obtenir une valeur proche de l'optimum du dual lagrangien.

Un théorème sur l'étude générale de la relaxation lagrangienne permet de faire un lien entre la valeur de la relaxation linéaire du problème général $(\mathbf{PL}_5)$ et la valeur optimale du dual lagrangien $(\mathbf{DL}_7)$. Le théorème de Geoffrion [Geo10] explique que si le sous-problème lagrangien $(\mathbf{RL}_6)$ respecte la propriété d'intégralité, alors la valeur optimale de $(\mathbf{PL}_5)$ est égale à celle de $(\mathbf{DL}_7)$. Autrement dit, si le sous-problème vérifie la propriété d'intégralité, il est plus facile d'utiliser la relaxation linéaire pour obtenir la même borne.

2.2 PROBLÈME DU VOYAGEUR DE COMMERCE

Le *problème du voyageur de commerce* (*Traveling Salesman Problem - TSP*) est un problème combinatoire largement étudié. Le problème rencontré par ce voyageur est de visiter chacun de ses clients une et une seule fois avant de revenir chez lui tout en parcourant une distance minimale.

2.2.1 *Problèmes généraux et notations*

Dans cette section, nous allons étudier différentes formulations du *TSP*. Soit $\mathcal{N}$ l'ensemble des nœuds représentant l'ensemble des clients devant être visités ($\mathcal{N} = \{0, \dots, n-1\}$). Généralement, le nœud 0 représente le dépôt où le voyageur doit commencer et terminer sa tournée. Les nœuds de $\{1, \dots, n-1\}$ représentent les $n-1$ clients devant être visités une et une seule fois.

2.2.1.1 *Graphe non-orienté et modélisation PLNE*

Historiquement, le *TSP* était défini dans un graphe non-orienté. Soit un graphe $\mathcal{G}_{no} = (\mathcal{N}, \mathcal{E})$, $\mathcal{E}$ est l'ensemble des arêtes du graphe $\mathcal{G}_{no}$. Pour chacune des arêtes, un poids $c_{i,j}$ représente la distance entre les nœuds i et j. Le *problème du voyageur de commerce* consiste à trouver dans $\mathcal{G}_{no}$ un cycle hamiltonien de poids minimum i.e. un cycle passant par chaque nœud exactement une seule fois.

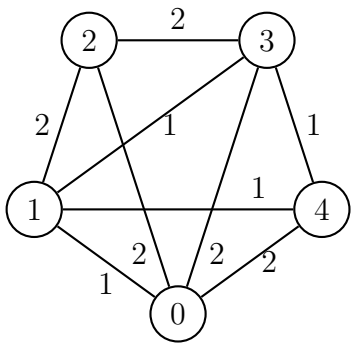
La Figure 6 montre un exemple de graphe non orienté pondéré (Figure 6a). La Figure 6b donne une solution réalisable pour le *TSP* (cycle hamiltonien) et la Figure 6c expose un cycle hamiltonien de longueur minimum qui constitue une des solutions optimales du *TSP*. Le coût de la solution associée est de 8.

Nous supposons que l'inégalité triangulaire est respectée, c'est-à-dire :

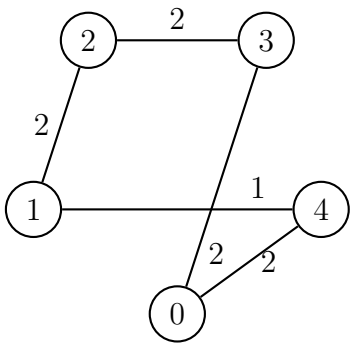
$$c_{i,j} \leqslant c_{i,k} + c_{k,j} \qquad \forall (i,j,k) \in \mathcal{N}^3$$

Le problème peut être formulé comme un programme linéaire en nombres entiers (*PLNE*). La formulation suivante est la plus souvent utilisée dans le cadre du *TSP*, elle a été proposée par Dantzig et al. dans [DFJ54]. Les variables de décision sont les suivantes, pour tout $(i,j) \in \mathcal{E}$:

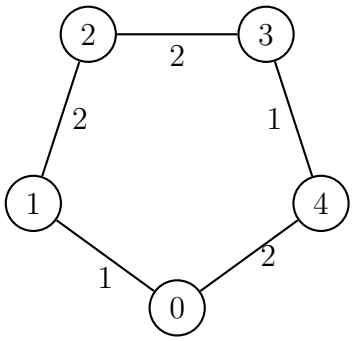
$$x_{i,j} = \begin{cases} 1 & \text{si l'arête } (i,j) \text{ est utilisée dans le tour} \\ 0 & \text{sinon} \end{cases}$$



(a) Exemple de graphe non orienté pondéré



(b) Une solution réalisable du *TSP* du graphe Figure 6a



(c) Une solution optimale du *TSP* du graphe Figure 6a

FIGURE 6 – Exemple de graphe non orienté

Nous notons $\Gamma(i)$, $\forall i \in \mathcal{N}$ l'ensemble des nœuds des arêtes incidentes au nœud i. Le programme linéaire est le suivant :

$$(\mathbf{PL}_8) \quad \textit{Min} \quad \sum_{(i,j)\in\mathcal{E}} c_{i,j}x_{i,j} \tag{8a}$$

$$\textit{s.t.} \quad \sum_{j\in\Gamma(i)|j>i} x_{i,j} + \sum_{j\in\Gamma(i)|j<i} x_{j,i} = 2 \qquad \forall i \in \mathcal{N} \tag{8b}$$

$$\sum_{i\in S, j\in S|(i,j)\in\mathcal{E}} x_{i,j} \leqslant |S| - 1 \qquad \forall S \subset \mathcal{N}, |S| \geqslant 2 \tag{8c}$$

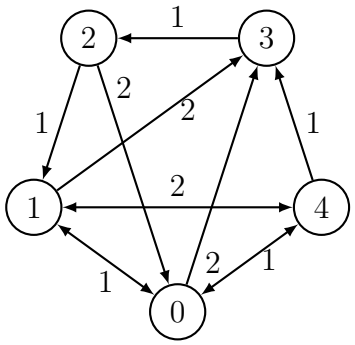
$$x_{i,j} = \{0,1\} \qquad \forall (i,j) \in \mathcal{E} \tag{8d}$$

L'objectif 8a représente la somme des poids des arêtes utilisées dans le tour. La contrainte 8b impose que le degré de chaque nœud soit de deux. La contrainte 8c, souvent appelée contrainte d'élimination de sous-

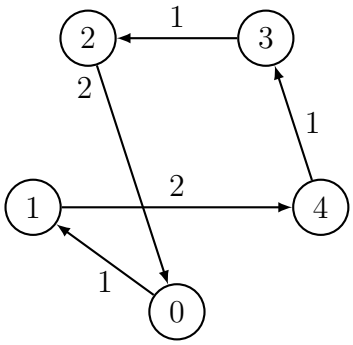
tours, implique qu'il n'existe aucun tour de taille inférieure à $|\mathcal{N}|$ dans la solution. Cette dernière contrainte impose la connexité du tour. Un graphe est dit connexe s'il existe pour chaque paire de nœuds une chaîne entre ces nœuds. La dernière contrainte 8d impose le domaine de définition des variables. Nous appelons relaxation linéaire le fait de relâcher la dernière contrainte ($x_{i,j} \in [0, 1], \forall (i, j) \in \mathcal{E}$).

2.2.1.2 *Graphe orienté et modélisation PLNE*

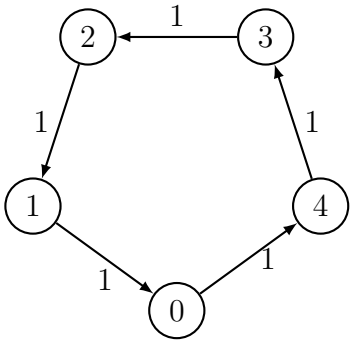
Le problème du *TSP* se pose souvent dans un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$, où $\mathcal{A}$ représente l'ensemble des arcs du graphe. En effet, lorsque les coûts sont asymétriques, il est nécessaire de définir une orientation. Nous supposons toujours l'inégalité triangulaire valide. Comme précédemment, un exemple est donné Figure 7.



(a) Exemple de graphe orienté pondéré



(b) Solution réalisable du *TSP* du graphe Figure 7a



(c) Solution optimale du *TSP* du graphe Figure 7a

FIGURE 7 – Exemple de graphe orienté

Nous pouvons étendre la formulation de Dantzig et al. (section 2.2.1.1) dans le graphe orienté $\mathcal{G}_o$. Les variables de décisions sont les suivantes, pour tout $(i,j) \in \mathcal{A}$:

$$x_{i,j} = \begin{cases} 1 & \text{si l'arc } (i,j) \text{ est utilisé pour le tour} \\ 0 & \text{sinon} \end{cases}$$

Le programme linéaire est le suivant :

$$(\mathbf{PL}_9) \quad \textit{Min} \quad \sum_{(i,j)\in\mathcal{A}} c_{i,j}x_{i,j} \tag{9a}$$

$$\textit{s. t.} \quad \sum_{j\in\Gamma^+(i)} x_{i,j} = 1 \qquad \forall i \in \mathcal{N} \tag{9b}$$

$$\sum_{i\in\Gamma^-(j)} x_{i,j} = 1 \qquad \forall j \in \mathcal{N} \tag{9c}$$

$$\sum_{i\in S, j\in S|(i,j)\in\mathcal{A}} x_{i,j} \leqslant |S| - 1 \qquad \forall S \subset \mathcal{N}, |S| \geqslant 2 \tag{9d}$$

$$x_{i,j} = \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{9e}$$

Nous notons $\forall i \in \mathcal{N}$, $\Gamma^+(i)$ (resp. $\Gamma^-(i)$) l'ensemble des nœuds des arcs sortants (resp. entrants) du nœud i. La contrainte 9b indique que chaque nœud du graphe doit avoir un degré d'arcs sortant égale à 1. Il en est de même pour les arcs entrants avec la contrainte 9c. Nous retrouvons aussi la contrainte d'élimination des sous-tours 9d et le domaine de définition des variables 9e. Une relaxation linéaire consiste donc à relâcher cette dernière contrainte.

Une formulation particulière a été proposée par Vadja [S.61] pour le *TSP* et qui nous sera utile par la suite. Cette formulation utilise une représentation par couche où chaque couche représente la position d'un nœud dans la séquence de la tournée. Les variables de décisions suivantes sont ajoutées, pour tout $(i,j) \in \mathcal{A}$ et $\forall k \in \{0,\dots,n\}$:

$$y^k_{i,j} = \begin{cases} 1 & \text{si l'arc } (i,j) \text{ est le } k^{\text{ième}} \text{ arc du tour} \\ 0 & \text{sinon} \end{cases}$$

Voici le programme linéaire utilisant ce modèle dit modèle de flot.

$$
\begin{aligned}
(\mathbf{PL}_{10}) \quad \textit{Min} \quad & \sum_{(i,j)\in\mathcal{A}} c_{i,j}x_{i,j} && && \text{(10a)}\\
\textit{s.t.} \quad & x_{i,j} - \sum_{k\in\{0,\dots,n\}} y^k_{i,j} = 0 && \forall (i,j)\in\mathcal{A} && \text{(10b)}\\
& \sum_{j\in\Gamma^+(i)} x_{i,j} = 1 && \forall i\in\mathcal{N} && \text{(10c)}\\
& \sum_{i\in\Gamma^-(j)} x_{i,j} = 1 && \forall j\in\mathcal{N} && \text{(10d)}\\
& \sum_{j\in\Gamma^+(0)} y^0_{0,j} = 1 && && \text{(10e)}\\
& \sum_{i\in\Gamma^-(0)} y^n_{i,0} = 1 && && \text{(10f)}\\
& \sum_{j\in\Gamma^+(i)} y^k_{i,j} - \sum_{j\in\Gamma^-(i)} y^{k-1}_{j,i} = 0 && \forall i\in\mathcal{N},\ \forall k\in\{1,\dots,n\} && \text{(10g)}\\
& x_{i,j}\in\mathbb{N} && \forall (i,j)\in\mathcal{A} &&\\
& y^k_{i,j} = \{0,1\} && \forall (i,j)\in\mathcal{A},\ \forall k\in\{0,\dots,n\} &&
\end{aligned}
$$

Les contraintes 10e et 10f imposent un degré 1 au nœud de la première et dernière couche. La contrainte 10g représente une contrainte de conservation du flot. Cela signifie que ce qui rentre dans un nœud doit en ressortir avec la même quantité.

Une étude des différentes formulations pour le *TSP* dans un graphe orienté est proposée dans [OW06].

2.2.1.3 *Problème du voyageur de commerce avec fenêtre de temps*

Souvent, des fenêtres de temps pour chaque client sont ajoutées au problème de base afin de modéliser la disponibilité des clients. Ainsi, le voyageur doit visiter chacun de ses clients une seule fois tout en respectant leurs fenêtres de temps avant de revenir chez lui en parcourant une distance minimale. Cette extension du *TSP* est appelée *problème du voyageur de commerce avec fenêtres de temps* (*Traveling Salesman Problem with Time Windows - TSPTW*). Le temps de trajet entre un nœud i et un nœud j est noté $t_{i,j}$ et la fenêtre de temps pour chaque nœud i est donnée par $[a_i, b_i]$. Le temps de service d'un nœud i est inclus, sans perte de généralité, dans

chaque temps de trajet $t_{i,j}$ vers les nœuds j.

Afin d'inclure cette contrainte dans les différentes modélisations, nous devons ajouter des variables $start_i \in [a_i, b_i],\ \forall i \in \mathcal{N}$ représentant le temps de début de service chez le client i. Les contraintes additionnelles sont les suivantes et présentées dans [SD88] :

$$start_i + t_{i,j} - start_j \leqslant M(1 - x_{i,j}) \qquad \forall (i,j) \in \mathcal{A} \qquad (11a)$$

$$a_i \leqslant start_i \leqslant b_i \qquad \forall i \in \mathcal{N} \qquad (11b)$$

M représente l'horizon de temps du problème. La contrainte d'élimination des sous-tours 9d devient alors redondante en présence de la contrainte 11a.

Une formulation en flot a été proposé dans [Lan+93] en modélisant le *TSPTW* comme un problème d'ordonnancement. Un modèle de *programmation par contraintes* a aussi été proposé par Pesant et al. dans [Pes+98].

2.2.2 *Relaxations du TSP et bornes inférieures*

En optimisation combinatoire, il est important de connaître des bornes inférieures ou supérieures aux problèmes. Pour cela, une des techniques utilisées consiste à relâcher une contrainte afin de résoudre un problème, dans la plupart des cas, plus facile. Une solution d'un problème relâché représente une borne inférieure au problème de base dans le cas d'un problème de minimisation. Dans la suite de la section, nous présentons différentes relaxations du *TSP*. Celles-ci constituent donc des bornes inférieures.

2.2.2.1 *Affectation*

L'une des premières relaxations étudiées du *TSP* est le problème d'affectation. De façon générale, les problèmes d'affectation consistent à répartir ou connecter un premier ensemble $\mathcal{B}^1$ vers un second ensemble $\mathcal{B}^2$. Dans le cas de problèmes d'optimisation, il est intéressant d'effectuer des affectations tout en minimisant ou maximisant une certaine fonction. Nous nous intéressons au problème d'affectation de poids minimum qui consiste à affecter des éléments de l'ensemble $\mathcal{B}^1$ à l'ensemble $\mathcal{B}^2$ tout en minimisant le coût d'affectation. En l'occurence dans notre cas ($|\mathcal{B}^1| = |\mathcal{B}^2|$), nous cherchons à affecter exactement un élément de $\mathcal{B}^1$ vers un élément de $\mathcal{B}^2$. Dans cette restriction, le problème revient à trouver un couplage parfait de poids minimum dans un graphe biparti. Soit un graphe biparti $\mathcal{G}_B = (\mathcal{B}^1 \cup \mathcal{B}^2, \mathcal{A}_B)$ tel que :

$$(i,j) \in \mathcal{A}_B \Leftrightarrow i \in \mathcal{B}^1 \ \ \& \ \ j \in \mathcal{B}^2$$

Chaque arc $(i,j) \in \mathcal{A}_B$ possède un poids $c_{i,j}$. Soit les variables de décisions suivantes :

$$x_{i,j} = \begin{cases} 1 & \text{si l'arc } (i,j) \in \mathcal{A}_B \text{ appartient au couplage} \\ 0 & \text{sinon} \end{cases}$$

Le programme linéaire du problème d'affectation de poids minimum est le suivant :

$$(\mathbf{PL}_{12}) \quad \textit{Min} \quad \sum_{(i,j)\in\mathcal{A}_B} c_{i,j}x_{i,j} \tag{12a}$$

$$\textit{s.t.} \quad \sum_{j\in\mathcal{B}^2} x_{i,j} = 1 \qquad \forall i \in \mathcal{B}^1 \tag{12b}$$

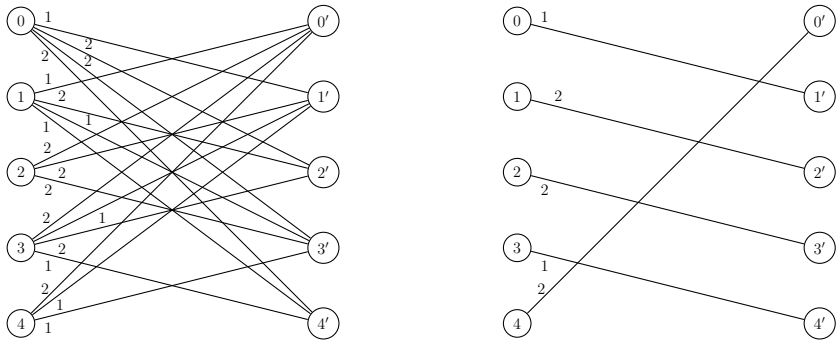
$$\sum_{i\in\mathcal{B}^1} x_{i,j} = 1 \qquad \forall j \in \mathcal{B}^2 \tag{12c}$$

$$x_{i,j} = \{0,1\} \qquad \forall (i,j) \in \mathcal{A}_B$$

Nous pouvons souligner que le programme linéaire $(\mathbf{PL}_9)$ sans la contrainte 9d correspond au programme linéaire $(\mathbf{PL}_{12})$. En effet, un problème d'affectation est une relaxation du *TSP* lorsque la contrainte d'élimination des sous-tours (de connexité) n'est plus vérifiée. Ainsi, toute solution du *TSP* est une solution réalisable pour le problème d'affectation, puique chaque nœud d'une solution du *TSP* est affecté au précédent nœud de la tournée. Cependant toute solution du problème d'affectation ne constitue pas une solution au *TSP* et ces solutions procurent des bornes inférieures sur l'objectif du problème de base.

Pour obtenir une relaxation du *TSP*, il suffit de dupliquer tous les nœuds du graphe considéré. Ainsi, les nœuds de base constituent le premier ensemble de nœuds du graphe biparti et les nœuds dupliqués le deuxième ensemble de nœuds. Cette relaxation peut être définie aussi bien pour les graphes orientés que pour les graphes non orientés.

La Figure 8 montre la transformation du graphe Figure 6 (Figure 8a) ainsi que la solution du *TSP* et du problème d'affectation Figure 8b. L'algorithme hongrois dédié à ce problème permet de résoudre ce dernier en $\mathcal{O}(n^3)$ [Kuh10].



(a) Graphe biparti associé au graphe Figure 6a

(b) Solution du *TSP* et du problème d'affectation du graphe Figure 6a

FIGURE 8 – Graphe biparti et problème d'affectation

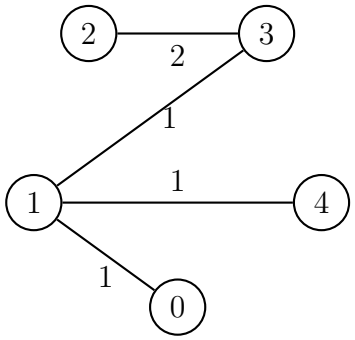
2.2.2.2 *Held-and-Karp*

La seconde relaxation du *TSP* étudiée et présentée dans [HK70 ; HK71] est fondée sur la recherche d'arbres dans un graphe, couplée à des techniques de relaxation lagrangienne. En théorie des graphes, un arbre est un graphe qui ne comporte pas de cycles (acyclique) et qui est connexe. Soit un graphe $\mathcal{G}_{no} = (\mathcal{N}, \mathcal{E})$, un arbre couvrant de $\mathcal{G}_{no}$ est un ensemble d'arêtes (sous-graphe) de $\mathcal{E}$ acyclique tel que chaque sommet dans $\mathcal{N}$ est atteint par au moins une arête de ce sous-graphe et qu'il existe un chemin entre toutes paires de nœuds. Lorsque les arêtes sont pondérées, nous pouvons rechercher un arbre couvrant de poids minimum. Ce dernier est un arbre couvrant dont la somme des arêtes sélectionnées est minimale (voir Figure 9a).

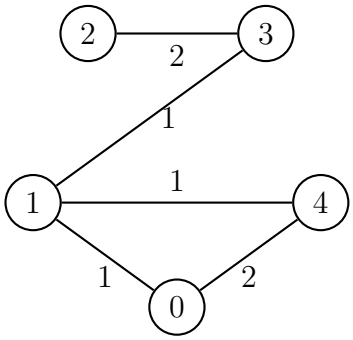
Un *1-arbre* (*One-Tree*) [HK70] dans un graphe $\mathcal{G}_{no} = (\mathcal{N}, \mathcal{E})$ est un arbre couvrant de poids minimum de $\mathcal{N}\backslash\{0\}$ plus deux arêtes de poids minimum incidentes au nœud 0 (voir Figure 9b).

Un *1-arbre* est une relaxation du *TSP*. En effet, toute solution du *TSP* vérifie les propriétés d'un tel arbre : la connexité de la solution ainsi que le coût minimum. Cependant, l'inverse est vérifié si et seulement si le degré des sommets du *1-arbre* de poids minimum est égal à deux.

Il existe deux algorithmes principaux pour la recherche d'arbres couvrants de poids minimum : l'algorithme de Kruskal [Kru56] et l'algorithme de Prim [Pri57]. Il suffit ensuite de rajouter les deux arêtes de poids minimum connectées au nœud 0 pour obtenir un *1-arbre* de poids minimum.



(a) Un arbre couvrant de poids minimum associé au graphe Figure 6a



(b) *1-arbre* du graphe Figure 6a

FIGURE 9 – Arbre couvrant et *1-arbre*

Nous pouvons étendre cette relaxation dans sa version lagrangienne (Section 2.1.3). En effet, une condition pour qu'un *1-arbre* soit un tour est que tous les degrés des nœuds soient égaux à deux. Nous pouvons donc appliquer une procédure de résolution s'appuyant sur la relaxation lagrangienne de cette contrainte ([HK70; HK71]). Pour cela, un potentiel est associé à chaque nœud. Ce dernier équivaut au coût de violation de la contrainte. Ainsi, si un nœud dans un *1-arbre* a un degré supérieur à deux, on aura tendance à augmenter le coût des arêtes incidentes afin de les exclure de l'arbre et inversement pour inclure des arêtes. Plusieurs algorithmes permettent de résoudre le dual lagrangien. Le plus utilisé est l'algorithme de sous-gradient. On peut aussi citer les méthodes de Bundle [Lem01].

2.2.2.3 *Plus court chemin*

La dernière relaxation étudiée est une relaxation s'appuyant sur les plus courts chemins. Il s'agit de la *n-path* présentée par Christofides et al. [CMT81b]. Soit un graphe orienté pondéré $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$, la relaxation du *TSP* consiste à rechercher un plus court chemin du nœud 0 jusqu'au nœud 0 dans $\mathcal{G}_o$ utilisant $|\mathcal{N}|$ arcs.

La Figure 10 montre un plus court chemin possible du graphe Figure 7a d'un poids total de 5. On s'aperçoit qu'il est possible de passer plusieurs fois par un même nœud (nœud 1) et il est aussi possible de ne pas visiter de nœud (nœud 2). De plus, il est possible de sélectionner plusieurs fois le même arc.

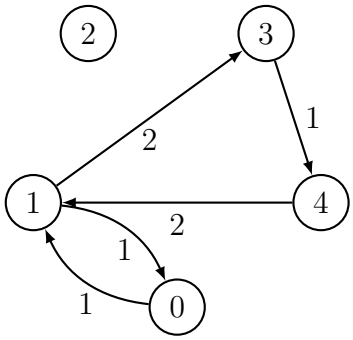


FIGURE 10 – Plus court chemin utilisant 5 arcs associé au graphe Figure 7a

Le problème du plus court chemin utilisant n arcs est résolu par un programme dynamique. Soit un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$ avec $n = |\mathcal{N}|$ et $c_{i,j}$, $\forall (i,j) \in \mathcal{A}$ le coût d'un arc. Le programme dynamique est défini comme suit :

$$
\begin{aligned}
(\mathbf{PD}_{13}) \quad & f^*(1,i) = c_{0i} && \forall i \in \mathcal{N}\backslash\{0\} && \text{(13a)}\\
& f^*(k,0) = +\infty && \forall k \in \{2,\dots,n-1\} && \text{(13b)}\\
& f^*(k,i) = \min_{j \in \Gamma^{-1}(i)} (f^*(k-1,j) + c_{j,i}) && \forall k \in \{2,\dots,n\}, \forall i \in \mathcal{N} && \text{(13c)}
\end{aligned}
$$

$\Gamma^{-1}(i)$ est l'ensemble des nœuds prédécesseurs du nœuds i, $\forall i \in \mathcal{N}$. La solution du problème est obtenue par la valeur $f^*(n,0)$. La complexité temporelle du programme dynamique est en $\mathcal{O}(n^3)$ et la complexité spatiale est en $\mathcal{O}(n^2)$.

Comme présenté précédemment Section 2.2.2.2, nous pouvons étendre l'étude par une relaxation lagrangienne. En effet, si le degré des nœuds d'une solution de la *n-path* est égale à deux, alors la solution est une solution valide pour le *TSP*. Ainsi, la contrainte de degré sur les nœuds est relâchée, le dual lagrangien peut être résolu par un processus de sous-gradient afin d'améliorer la borne inférieure du *TSP*.

2.2.2.4 *Classification des relaxations*

Nous pouvons définir les propriétés d'un tour de plusieurs façons. La Figure 11 résume les différentes propriétés qu'une solution du *TSP* doit respecter.

- La première propriété définit un tour comme étant un sous-graphe connexe et où chaque nœud possède un degré égal à deux.
  - Si le degré des nœuds n'est pas égal à deux, alors ce sous-graphe respecte les contraintes d'un *1-arbre* (Section 2.2.2.2).

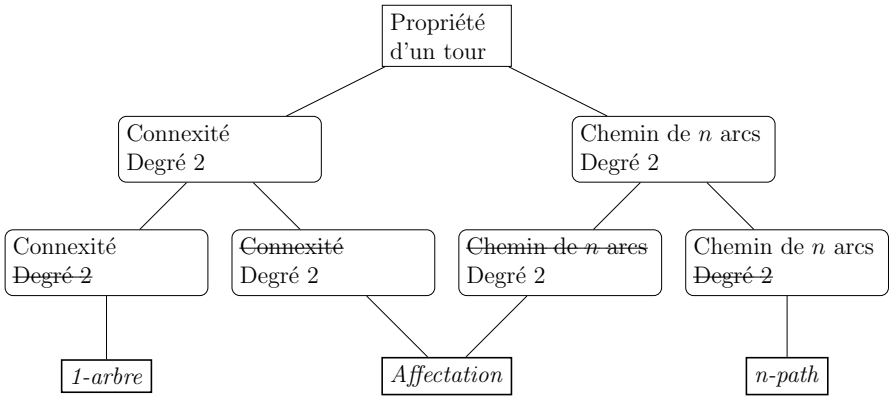


FIGURE 11 – Propriétés du sous-graphe induit du *TSP* et classification des différentes relaxations

- Si le sous-graphe n'est pas connexe, alors ce sous-graphe respecte les contraintes d'un problème d'affectation (Section 2.2.2.1).

— La seconde propriété définit un tour comme étant un sous-graphe constitué d'un chemin de n arcs et où chaque nœud possède un degré égal à deux.
  - Si le degré des nœuds n'est pas égal à deux, alors ce sous-graphe représente un chemin de n arcs (Section 2.2.2.3)
  - Si le sous-graphe ne constitue pas un chemin de n arcs, alors ce sous-graphe respecte les contraintes d'un problème d'affectation.

Une solution du *TSP* est un sous-graphe de poids minimum vérifiant les propriétés d'un tour.

2.2.3 *Algorithmes exacts*

Les approches exactes de résolution du *TSP* sont nombreuses. Nous pouvons citer tout d'abord Held et Karp avec l'utilisation de la relaxation lagrangienne et les *1-arbres* dans un environnement de branch and bound [HK70 ; HK71]. Ensuite, dans [BC81], les auteurs utilisent aussi la relaxation lagrangienne ainsi que la borne du problème d'affectation. Le meilleur algorithme connu pour les purs problèmes de *TSP* reste le moteur de calcul Concorde [App+11] utilisant des techniques de *programmation linéaire*. Cependant, lorsque des contraintes additionnelles viennent s'ajouter, comme les fenêtres de temps, il est alors difficile d'utiliser la *programmation linéaire*. Pour le *TSPTW*, nous pouvons citer les approches exactes de Dumas et al. [Dum+95], ainsi que Focacci et al. [FLM02] qui utilisent des approches de

programmation par contraintes. Le meilleur algorithme connu pour le *TSPTW* utilise la programmation dynamique combiné à la génération de colonne et est présenté dans [BMR12a]. De plus, l'idée de cette approche repose sur la relaxation proposée par Christofides et al. [CMT81b] (Section 2.2.2.3).

2.3 PROBLÈMES DE TOURNÉES DE VÉHICULES

Dans cette section, nous étudions les formulations associées aux *problèmes de tournées de véhicules avec capacité* (*CVRP*) ainsi que la version *avec fenêtres de temps* (*CVRPTW*). De plus, nous présentons des méthodes heuristiques pour résoudre des problèmes de tournées de véhicules.

2.3.1 *Problème de tournées de véhicules avec capacité*

2.3.1.1 *Modélisation*

Nous disposons d'une flotte de véhicules ainsi qu'un ensemble de clients ayant demandé chacun une certaine quantité de produit (demande q_i du client i) [DR59]. Nous devons alors satisfaire toutes les demandes des clients en affectant ces derniers à un des véhicules disponibles. L'affectation d'un certain nombre de clients à un véhicule forme une tournée. Par ailleurs, la somme des demandes d'une tournée ne doit pas excéder la capacité C du véhicule utilisé.

Le *CVRP* peut être vu comme une extension du *problème du voyageur de commerce* avec plusieurs véhicules. La formulation de Dantzig et al. [DFJ54] est alors étendue pour le *CVRP* [LND85]. Soit un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$, les variables de décisions sont les suivantes pour tout arc (i, j) de $\mathcal{A}$.

$$x_{i,j} = \begin{cases} 1 & \text{si l'arc } (i,j) \text{ est utilisé} \\ 0 & \text{sinon} \end{cases}$$

Le programme linéaire est le suivant avec K le nombre de véhicules utilisés :

$$
\begin{aligned}
(\mathbf{PL}_{14}) \quad \textit{Min} \quad & \sum_{(i,j)\in\mathcal{A}} c_{i,j}x_{i,j} && && (14a)\\
\textit{s. t.} \quad & \sum_{j\in\Gamma^+(i)} x_{i,j} = 1 && \forall i \in \mathcal{N}\backslash\{0\} && (14b)\\
& \sum_{i\in\Gamma^-(j)} x_{i,j} = 1 && \forall j \in \mathcal{N}\backslash\{0\} && (14c)\\
& \sum_{j\in\Gamma^+(0)} x_{0,j} = K && && (14d)\\
& \sum_{i\in\Gamma^-(0)} x_{i,0} = K && && (14e)\\
& \sum_{i\in S, j\in S|(i,j)\in\mathcal{A}} x_{i,j} \leqslant |S| - r(S) && \forall S \subset \mathcal{N}\backslash 0, |S| \geqslant 2 && (14f)\\
& x_{i,j} = \{0,1\} && \forall (i,j) \in \mathcal{A} && (14g)
\end{aligned}
$$

La fonction objectif 14a minimise la somme des distances parcourues. Nous retrouvons les contraintes de degré sur les arcs sortants 14b et sur les arcs entrants 14c. En revanche, ces dernières ne s'appliquent pas au nœud 0 puisque le nœud de dépôt doit posséder un degré d'arc sortant égale au nombre de véhicules utilisés (contrainte 14d), de même pour les arcs entrants (contrainte 14e). La contrainte d'élimination des sous-tours est aussi étendue dans le cas de plusieurs véhicules (contrainte 14f) avec $r(S)$ le nombre minimum de véhicules nécessaires à la demande totale de S. La prise en compte du respect des capacités se retrouve dans le calcul de $r(S)$. Ce dernier peut être pré-calculé en résolvant un problème de Bin-Packing [MT90] sur l'ensemble S. Les conditions d'intégralité sont données par les contraintes 14g.

Christofides et al. [CMT81a] propose une formulation différente qui permet de s'affranchir de la contrainte d'élimination des sous-tours en se rapprochant de la formulation en flot d'un *TSP*.

De la même façon que pour le *TSPTW*, nous ajoutons les contraintes de fenêtre de temps au modèle afin de modéliser le respect des disponibilités des clients, contraintes 11a et 11b (Section 2.2.1.3).

2.3.1.2 *Relaxation et résolution*

Divers relaxations ont été étudiées pour le *CVRP* et *CVRPTW*. Une étude est proposée dans [TV02].

Nous retrouvons l'utilisation du problème d'affectation pour calculer une borne inférieure du problème. En effet, si la contrainte d'élimination des sous-tours (eq. 14f) du ($\mathbf{PL}_{14}$) est relachée, alors le programme linéaire est un problème de transport. Ce dernier peut être transformé et résolu en problème d'affectation, ce qui a été proposé dans [CT80] et [LMN86].

L'emploi des arbres et arborescences a été utilisé afin de caractériser la borne inférieure du *CVRP*. L'extension d'un *1-arbre* pour le *CVRP* a été utilisée [CMT81a] et [Fis94]. Le problème relâché est alors appelé *k-arbre* et consiste à rechercher un arbre couvrant de poids minimum avec un degré au nœud 0 égal à k. On obtient ce problème en relâchant les contraintes de degré du programme linéaire du *CVRP*. Nous avons aussi la version orientée du *k-arbre*, la *k-arborescence* de poids minimum. La version lagrangienne a été utilisé dans [TV97].

Une relaxation s'appuyant sur les plus courts chemins a été utilisée dans [CMT81a] pouvant être étendue au *CVRPTW*. Elle a été reprise avec succès par Baldacci et al. dans [BMR11 ; BMR12b] permettant des résolutions efficaces en utilisant des algorithmes de Branch and Price.

Toutes ces relaxations ont été utilisées afin de résoudre le *CVRP* et le *CVRPTW* grâce à des algorithmes de Branch and Bound ou Branch and Cut.

2.3.2 *Heuristiques pour les problèmes de tournées de véhicules*

Les problèmes de tournées de véhicules sont en général très difficiles à résoudre à l'optimum. C'est pourquoi, il a été proposé des approches heuristiques afin de trouver de bonnes solutions dans des temps de résolutions acceptables. De plus, les approches heuristiques permettent de résoudre de nombreuses variantes pour les problèmes de tournées de véhicules. Gendreau et al. [Gen+08] propose une étude de six métaheuristiques pour les problèmes de tournées de véhicules comme par exemple les *tabu-search*, les colonies de fourmis ou encore les algorithmes génétiques. Des études se sont aussi focalisées sur des problèmes de tournées de véhicules larges [GT10 ; Lap+00]. Ces études décrivent les différents voisinages utilisés et les techniques de recherche du voisinage pour les problèmes de tournées

de véhicules avec fenêtres de temps.

Par ailleurs, des méthodes unifiées ont été proposées afin de résoudre un maximum de variantes avec un seul algorithme. Dans [Vid+13], Vidal et al. propose une étude et une classification des différentes contraintes pour les problèmes de tournées de véhicules. Cette étude propose de trouver quels éléments des différentes heuristiques sont les plus performants. Il identifie les éléments importants comme par exemple l'utilisation de différents voisinages, l'acceptation de solutions irréalisables (de préférence les fenêtres de temps non respectées) ou encore l'utilisation de mécanismes d'orientation de la recherche. Ces caractéristiques sont incluses dans une méthodes unifiées afin de trouver de bonnes solutions pour une grande variété de problèmes de tournées de véhicules.

2.4 LA PROGRAMMATION PAR CONTRAINTES

La *programmation par contraintes* (*PPC*) est un paradigme de programmation qui permet de résoudre des problèmes d'optimisation combinatoire [VHSD95], [RVBW06]. Trois ingrédients essentiels composent la *PPC* : la modélisation, le filtrage et la recherche. Nous allons voir en détails ces différentes composantes.

2.4.1 *La modélisation*

La *programmation par contraintes* est une technique pour résoudre des problèmes de satisfaction de contrainte (CSP).

Définition 2.4.1. *Un* CSP *est caractérisé par trois ensembles [Mon74] :*

- $X = (x_1, \ldots, x_n)$ *: l'ensemble des variables du problème,*
- $D = (D_{x_1}, \ldots, D_{x_n})$ *: l'ensemble des domaines de définitions des variables. Autrement dit,* $D_{x_i} \in D$ *est l'ensemble des valeurs que peut prendre* $x_i \in X$*. Nous notons* $\underline{x_i}$ *(resp.* $\overline{x_i}$*) la borne inférieure (resp. supérieure) de la variable* x_i*,*
- $C = (c_1, \ldots, c_m)$ *: l'ensemble des contraintes. Une contrainte* c *est une relation définie sur une séquence d'un sous ensemble* ($X' \subset X$) *de variables* $X(c) = (x_i, \forall x_i \in X')$*. Il s'agit d'un ensemble qui contient les combinaisons de valeurs (appelées tuples* τ*) qui satisfont* c*.*
- *Soit* $x_i \in X(c)$ *et* $v_i \in D_{x_i}$*, nous appelons* support *de* (x_i, v_i) *pour* c *un tuple* τ *vérifiant* $x_i = v_i$ *qui satisfait la contrainte* c*.*

L'arité d'une contrainte c est le nombre de variables concernées par c. Des contraintes unaires sont des relations posées sur une seule variable, par exemple $x_i \leqslant 9$. Des contraintes binaires lient deux variables entre

elles, comme par exemple $x_i + x_j \leqslant 5$. Il existe aussi des contraintes d'arité supérieure avec des ensembles de plusieurs variables.

Exemple 2.4.1. *Soit l'ensemble des variables* $X = (x_1, x_2, x_3)$ *et l'ensemble des domaines de définitions* $D_{x_i} = \{1, 2, 3\}$, $\forall i \in \{1, 2, 3\}$. *Soit la contrainte* c *suivante :*

$$x_1 + 2x_2 + x_3 \leqslant 4$$

Ainsi $X(c) = (x_1, x_2, x_3)$ *et les tuples permettant de satisfaire la contrainte* c *sont* $(1, 1, 1)$, $(2, 1, 1)$ *et* $(1, 1, 2)$. *Par ailleurs,* $(x_1, 1)$ *possède un support pour* c, *car il existe au moins un tuple* $(1, 1, 1)$ *qui satisfait la contrainte et qui utilise la valeur de* 1 *dans le domaine* D_{x_1}

Il existe aussi ce qu'on appelle des contraintes globales qui permettent de mettre en relation un ensemble de variables afin de satisfaire un même postulat ([Rég11], [Bel+07]). La contrainte globale ALLDIFFERENT$(x_1, \ldots, x_n)$, par exemple, impose que toutes les variables prennent des valeurs différentes. Il est possible de décomposer la contrainte ALLDIFFERENT avec des contraintes binaires, mais l'interêt d'utiliser une contrainte globale réside dans la modélisation simplifiée et dans les raisonnements spécifiques pour résoudre le postulat donné. De plus, certaines contraintes globales peuvent inclure plusieurs contraintes, même globales. Par exemple, la contrainte CIRCUIT$(next_1, \ldots, next_n)$, qui impose que les variables $next$ forment un circuit dans un graphe, implique une contrainte ALLDIFFERENT.

Définition 2.4.2. *Une solution d'un* CSP *est un tuple* $(s_1, \ldots, s_n)$ *de valeurs pour chaque variable qui satisfait toutes les contraintes de* C.

Exemple 2.4.2. *Soit le* CSP *comprenant trois variables* $X = \{x_1, x_2, e\}$ *de domaines* $D_{x_1} = \{0, \ldots, 5\}$, $D_{x_2} = \{1, \ldots, 10\}$, $D_e = \{1, \ldots, 10\}$ *et les contraintes suivantes :*

$$x_2 + e = 5 \tag{15a}$$

$$\text{ELEMENT}(x_1, t = [1, 3, 5, 7, 9], e) \tag{15b}$$

La contrainte ELEMENT *introduite dans [VHC88] permet d'indexer un tableau* t *à partir de la valeur d'une variable* x_1. *Par exemple, la contrainte 15b impose que la variable* e *soit égale au* $x_1^{ème}$ *élément du tableau* t. *Dans la suite nous préférons noter* $e = t_{x_1}$ *pour signifier la contrainte* ELEMENT.

Une solution de ce CSP *est par exemple* $x_1 = 0$, $x_2 = 4$ *et* $e = 1$.

2.4.2 *Filtrage*

Dans cette section, nous aborderons les raisonnements effectués par les contraintes. Nous finirons par une étude sur les contraintes globales.

2.4.2.1 *Raisonnements effectués par les contraintes*

Une des composantes importantes en *programmation par contraintes* est le filtrage des contraintes. Chaque contrainte en *PPC* possède un ou plusieurs algorithmes de filtrage. Ces derniers permettent d'éliminer certaines valeurs incohérentes d'une variable i.e. qui n'appartiennent à aucune solution de la contrainte. Les algorithmes peuvent utiliser des raisonnements logiques ou bien utiliser des outils de recherche opérationnelle, comme par exemple la relaxation lagrangienne ou la théorie des graphes [Rég11].

Il existe plusieurs niveaux de cohérence pour les contraintes.

Définition 2.4.3. *Une contrainte* c *est* arc-cohérente *(GAC) [VHSD95] si pour toute variable* $x_i \in X(c)$ *et pour toute valeur* $v \in D_{x_i}$*, il existe une affectation, telle que* $x_i = v$*, qui satisfait la contrainte* c*. Autrement dit, une contrainte* c *est* arc-cohérente *si toutes les valeurs des variables possèdent un support pour* c*.*

Ainsi, l'arc-cohérence garantit l'élimination de toutes les valeurs incohérentes d'une contrainte.

Définition 2.4.4. *Une contrainte* c *est* borne-cohérente *(BC) [RVBW06 ; Der15] si pour toute variable* $x_i \in X(c)$ *et chaque valeur des bornes de* D_{x_i} *(*$v_i \in \{\underline{D_{x_i}}, \overline{D_{x_i}}\}$*), il existe un tuple* $\tau = (v_1 \in [\underline{D_{x_1}}, \overline{D_{x_1}}], \ldots, v_i, \ldots, v_n \in [\underline{D_{x_n}}, \overline{D_{x_n}}])$ *qui satisfait la contrainte* c*.*

Les algorithmes garantissant la borne-cohérence ne prennent en compte et ne mettent à jour que les bornes des domaines. C'est un niveau moins élevé que l'arc-cohérence mais la complexité des algorithmes qui permet de maintenir ce niveau est, dans la plupart des cas, plus faible.

Exemple 2.4.3. *Reprenons l'exemple précédent 2.4.2. Nous séparons l'égalité 15a en deux inégalités et effectuons la borne-cohérence sur ces deux inégalités :*

$$\overline{x_2} \leqslant 5 - \underline{e}$$

$$\underline{x_2} \geqslant 5 - \overline{e}$$

Les bornes de la variables x_2 *peuvent être réduites à* $[1, 4]$*. De façon symétrique pour* e*, nous pouvons réduire sa borne supérieure à* 4*,* $D_e = [1, 4]$*.*

En ce qui concerne la contrainte ELEMENT *15b, le domaine de la variable* x_1 *peut être réduit. En effet,* $D_e = [1, 4]$*, il est alors impossible pour* e *de prendre les valeurs* 5*,* 7 *et* 9 *de* t *(*$D_e = \{1, 3\}$*). La variable* x_1 *ne peut alors pas prendre une valeur supérieure à* 1*,* $D_{x_1} = \{0, 1\}$*.*

Nous remarquons que l'ensemble de définition de e *ne contient pas toutes les valeurs de l'intervalle* $[1, 4]$*. Tentons alors de réaliser le filtrage arc-cohérence*

sur la contrainte 15a. Étant donné que les valeurs de e *sont impaires, il est alors nécessaire pour* x_2 *de prendre des valeurs paires afin de respecter la contrainte. En conséquence, nous pouvons déduire que* $D_{x_2} = \{2, 4\}$.

Lorsque des domaines sont mis à jour, les algorithmes de filtrage sont activés pour d'autres contraintes (ou parfois la même contrainte), nous appelons cela en *PPC* la propagation de contraintes.

2.4.2.2 *Contraintes globales et contraintes* NP-difficiles

Les contraintes globales possèdent le plus souvent des algorithmes de filtrage spécifiques afin d'éliminer les valeurs incohérentes du postulat proposé. Ces algorithmes exploitent la structure du problème sous-jacent et permettent efficacement de maintenir l'arc-cohérence. Par exemple, l'un des algorithmes de la contrainte ALLDIFFERENT utilise la théorie des graphes [Hoe01].

Il existe aussi des niveaux de cohérences plus faibles que la borne-cohérence ou arc-cohérence. En effet, lorsque le problème de trouver un support d'une contrainte globale est *NP-Complet*, il est alors impossible de maintenir la cohérence en temps polynomiale. C'est pourquoi, il est utile de dédier des algorithmes de filtrage spécifiques pour ce type de contrainte. Nous pouvons citer la contrainte CIRCUIT [Lau78] permettant de maintenir un circuit dans un graphe.

Face à un problème d'optimisation, i.e. un problème de minimisation ou de maximisation d'une fonction objectif, il est possible d'étendre la définition d'une contrainte globale en incluant le coût de cette fonction objectif. Nous pouvons prendre comme exemple la contrainte globale ALLDIFFERENT et sa version intégrant des coûts d'affectation MINIMUMWEIGHTALLDIFFERENT [FLM99]. Dans cette version, les valeurs des variables doivent être toutes différentes tout en minimisant le coût associé. L'algorithme de filtrage proposé dans [Sel02] modélise le problème comme un problème d'affectation de poids minimum. Il permet ainsi d'évaluer une borne inférieure sur le coût et de détecter les valeurs incohérentes afin de ne pas dépasser la borne supérieure du coût. Ce principe est appelé filtrage fondé sur les coûts [FLM99].

2.4.3 *La recherche arborescente*

Dans cette section, nous étudierons les principes de la recherche. Nous finirons par l'introduction de différentes stratégies de branchement.

2.4.3.1 *L'espace de recherche*

Une fois la propagation initiale effectuée, il est dans la plupart des cas nécessaire d'explorer l'espace de recherche afin de trouver une ou des solutions. L'exploration de l'espace de recherche est guidée par des stratégies de branchement afin de diviser l'espace de recherche.

Une stratégie de branchement consiste à ajouter une contrainte au modèle qui affecte une valeur dans le domaine de la variable sélectionnée. Un exemple de stratégie pour parcourir l'arbre de recherche consiste à prendre la première variable non affectée et lui affecter la première valeur de son domaine (recherche lexicographique).

Exemple 2.4.4. *Reprenons l'exemple précédent (15a et 15b). Après la propagation initiale, l'état des domaines est le suivant :* $x_1 \in \{0, 1\}$, $x_2 \in \{2, 4\}$ *et* $e \in \{1, 3\}$.

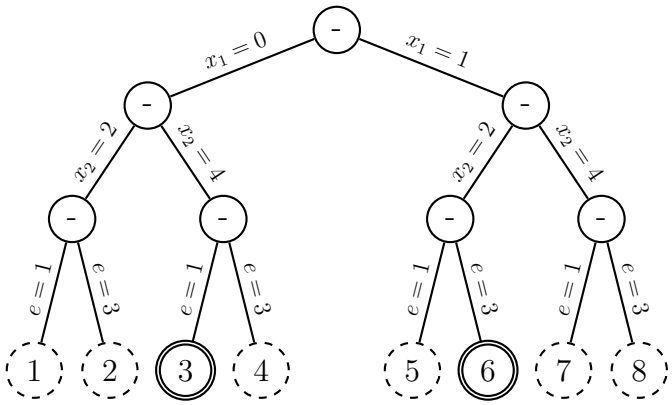


FIGURE 12 – Arbre de recherche complet

La Figure 12 montre l'espace de recherche complet pour le problème avec la stratégie de branchement lexicographique. Il existe seulement deux solutions pour ce problème (sommet 3 et sommet 6). Le reste des feuilles représente des incohérences et ne satisfait pas toutes les contraintes.

Dans la pratique, il n'est pas concevable de parcourir l'ensemble de l'espace de recherche, il faut alors réduire l'espace de recherche. Pour cela, en *PPC*, la propagation des contraintes est effectuée à chaque nœud, c'est-à-dire à chaque décision prise. Ainsi, l'espace de recherche est réduit et permet de détecter plus rapidement les incohérences ou de filtrer davantage de valeurs des domaines des variables.

— Si la contrainte ajoutée lors du branchement aboutit à une incohérence dans ce nœud, celui-ci correspond à un nœud d'échec. Par exemple, les nœuds en pointillés sur la Figure 12 sont des nœuds

d'échecs. Il faut alors retourner en arrière afin de continuer l'exploration. C'est ce que l'on appelle le *backtrack*.
— Si au moins une des variables n'est pas affectée à une valeur, alors il est nécessaire de continuer l'exploration.
— Si toutes les variables sont instanciées et sont compatibles avec les contraintes, nous sommes en présence d'une solution au problème. Par ailleurs, nous pouvons continuer l'exploration afin de trouver toutes les solutions du problème ou une solution de plus faible coût.

Exemple 2.4.5. *Effectuons la propagation des contraintes à chaque nœud pour le problème précédent. La Figure 13 montre l'ensemble de l'espace de recherche*

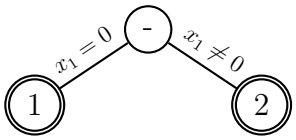


FIGURE 13 – Arbre de recherche avec la propagation des contraintes à chaque nœud

lorsque le filtrage des contraintes est effectué à chaque nœud. L'espace de recherche est alors réduit et les deux solutions du problème sont obtenues immédiatement.

2.4.3.2 *Stratégies de branchement*

Il est important de définir la stratégie de branchement afin de réduire le nombre de nœuds à parcourir. Un grand principe en programmation par contrainte est le *first-fail* [HE80] qui consiste à détecter les échecs le plus tôt possible. Nous retrouvons sous cette appellation beaucoup de stratégies de branchement. Ainsi en est-il de *mindom* qui sélectionne la variable ayant le plus petit domaine ou encore de *maxdeg* qui est une stratégie de branchement similaire sélectionnant la variable influant sur le plus grand nombre de contraintes.

Nous pouvons également citer comme stratégie de branchement l'*impact-based search* [Refo4] qui consite à détecter les variables ayant le plus d'impact sur le filtrage des contraintes. L'*impact-based search* évalue pour chaque variable la réduction de l'espace de recherche lorsque cette variable est affectée. Ainsi plus la valeur est grande, plus la variable influe sur l'espace de recherche.

Il existe aussi des techniques qui consistent à identifier et à retenir les affectations partielles, i.e. succession de conditions, provoquant des contradictions. De telles affectations se nomment des *nogoods* [FD94]. Le principe

est de stocker les *nogoods* car ces affectations aboutissent à un échec. Lorsqu'un *nogood* est identifié, nous ne parcourons pas la suite du sous-arbre de recherche. Ceci permet d'éviter d'explorer des sous-arbres identiques menant à des contradictions.

Lorsque nous sommes face à un problème d'optimisation, outre les contraintes globales qui peuvent se généraliser en prenant en compte des coûts (Section 2.4.2.2), la fonction objectif est prise en compte lors de la recherche. En effet, lorsqu'une solution est trouvée, nous pouvons repartir d'un autre nœud (*backtrak*) en ajoutant une contrainte sur la variable de coût à minimiser. Cela signifie que lors de la recherche de la prochaine solution, le coût devra être inférieur à celui de la meilleure solution connue. Ainsi, un problème d'optimisation peut être vu comme une séquence de problèmes de satisfaction de contraintes.

CONCLUSION

Ce chapitre a tout d'abord permis d'aborder des notions de *programmation linéaire* pour ensuite approfondir le *problème du voyageur de commerce* ainsi que différentes relaxations associées au problème. Nous avons également étudié le *problème de tournées de véhicules*, qui peut être considéré comme une extension du *TSP*. Enfin, une introduction à la *programmation par contraintes* a pu être présentée afin de mieux comprendre l'un des outils essentiels à la réalisation de cette thèse.

3

ÉTUDE THÉORIQUE DES BORNES INFÉRIEURES POUR LE PROBLÈME DU VOYAGEUR DE COMMERCE

Dans cette section, nous allons étudier d'un point de vue théorique les bornes inférieures du *TSP* données par les différentes relaxations vues dans la Section 2.2.2. Ces travaux ont fait l'objet d'un article présenté à *AAAI-2017* [DCP16].

3.1 RAPPEL DES BORNES ÉTUDIÉES

Nous allons commencer par rappeler et détailler les différentes bornes inférieures étudiées.

3.1.1 *1-arbre et Held et Karp*

La borne inférieure obtenue par la relaxation d'Held et Karp (Section 2.2.2.2) se base sur la recherche de *1-arbre*. Nous définissons *1-arbre*$(\{x_{i,j} | \forall (i,j) \in \mathcal{E}\})$ comme la contrainte imposant aux variables $x_{i,j} \in \{0,1\}$ associée aux arêtes $(i,j) \in \mathcal{E}$ de former un *1-arbre*. Nous appliquons alors une procédure de relaxation lagrangienne en relâchant et pénalisant la contrainte de degré d'un nœud, contrainte 8b de $(\mathbf{PL}_8)$:

$$\sum_{j \in \Gamma(i) | j > i} x_{i,j} + \sum_{j \in \Gamma(i) | j < i} x_{j,i} = 2 \qquad \forall i \in \mathcal{N}$$

Le sous-problème lagrangien sur le graphe $\mathcal{G}_{no} = (\mathcal{N}, \mathcal{E})$ avec les multiplicateurs lagrangiens $\lambda_i \in \mathbb{R}, \forall i \in \mathcal{N}$ est alors défini comme suit :

$$(\mathbf{RL}_{16}) \quad \textit{Min} \quad \sum_{(i,j) \in \mathcal{E}} (c_{i,j} - \lambda_i - \lambda_j) x_{i,j} + 2 \sum_{j \in \mathcal{N}} \lambda_j \tag{16a}$$

$$\textit{1-arbre}(\{x_{i,j} | \forall (i,j) \in \mathcal{E}\}) \tag{16b}$$

$$x_{i,j} \in \{0,1\}$$

Cette relaxation est proposée pour des graphes non orientés. Il est possible d'étendre la définition d'un *1-arbre* pour des graphes asymétriques, c'est-à-dire les graphes pour lesquels $\exists(i,j) \in \mathcal{N}^2 | c_{i,j} \neq c_{j,i}$. Pour cela, deux stratégies peuvent être mises en place.

La première stratégie consiste à transformer le graphe orienté en un graphe non orienté. Nous effectuons alors la transformation proposée par Jonker et Volgenant [JV83]. Les nœuds initiaux ($\{0,\ldots,n-1\}$) sont dupliqués afin d'obtenir des nœuds fictifs ($\{n,\ldots,2n-1\}$). Entre un nœud initial et son nœud fictif, la distance est considérée comme $-\infty$. Pour un nœud initial i, chaque arc sortant (i,j) est transformé en arête du nœud i au nœud fictif $n+j$. À l'inverse, chaque arc entrant (j,i) est transformé en arête du nœud fictif $(n+i)$ au nœud j. Pour le reste, une distance de $+\infty$ est attribuée. Ainsi pour un graphe asymétrique avec la matrice des distances suivantes :

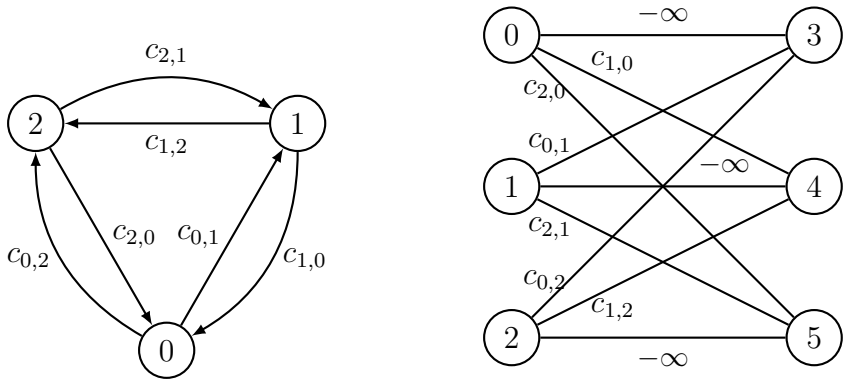
$$\begin{pmatrix} \infty & c_{0,1} & c_{0,2} \\ c_{1,0} & \infty & c_{1,2} \\ c_{2,0} & c_{2,1} & \infty \end{pmatrix}$$

La transformation de Jonker et Volgenant engendre la matrice symétrique suivante :

$$\begin{pmatrix} \infty & \infty & \infty & -\infty & c_{1,0} & c_{2,0} \\ \infty & \infty & \infty & c_{0,1} & -\infty & c_{2,1} \\ \infty & \infty & \infty & c_{0,2} & c_{1,2} & -\infty \\ -\infty & c_{0,1} & c_{0,2} & \infty & \infty & \infty \\ c_{1,0} & -\infty & c_{1,2} & \infty & \infty & \infty \\ c_{2,0} & c_{2,1} & -\infty & \infty & \infty & \infty \end{pmatrix}$$

La Figure 14 donne un exemple de la transformation de Jonker et Volgenant sur un graphe asymétrique composé de trois nœuds (Figure 14a). Le graphe ainsi obtenu est donné Figure 14b. Le lecteur pourra vérifier que la solution $(0-1-2-0)$ du graphe orienté (Figure 14a) produit alors la solution $(0-3-1-4-2-5-0)$ dans le graphe obtenue par la transformation de Jonker et Volgenant.

La seconde stratégie utilise la généralisation d'un *1-arbre* pour les graphes orientés : la *1-arborescence* présentée dans [HK70]. Une *1-arborescence* dans un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$ est une arborescence sortant du nœud 0 plus un arc supplémentaire entrant au nœud 0. Une arborescence sortant du nœud 0 signifie que n'importe quel nœud i dans $\mathcal{N}$ peut être atteint



(a) Graphe asymétrique avec 3 nœuds (b) Transformation de Jonker et Volgenant du graphe Figure 14a

FIGURE 14 – Transformation de Jonker et Volgenant sur un graphe avec 3 nœuds

depuis le nœud 0. Il est alors possible de rechercher une *1-arborescence* de poids minimum.

La Figure 15a montre un exemple de graphe asymétrique. Une *1-arborescence* est donnée Figure 15b et une *1-arborescence* de poids minimum est proposée sur la Figure 15c avec un poids de 7.

Dans le cas de graphes asymétriques (orientés), il est alors possible de définir la relaxation d'Held et Karp utilisant la *1-arborescence*. Nous définissons *1-arborescence*$(\{x_{i,j}|\forall(i,j) \in \mathcal{A}\})$ comme la contrainte imposant aux $x_{i,j}$ de former une *1-arborescence*. Une relaxation lagrangienne est effectuée sur le (**PL**$_9$) en relâchant et pénalisant la contrainte de degré des arcs entrants (eq. 9c) :

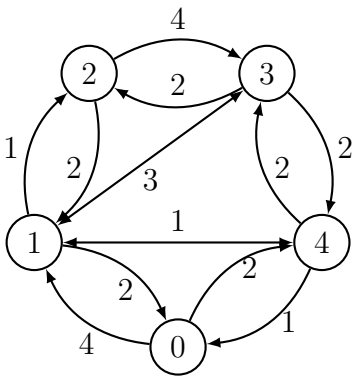
$$\sum_{i \in \Gamma^-(j)} x_{i,j} = 1 \qquad \forall j \in \mathcal{N}$$

Soit un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$ et les multiplicateurs lagrangiens $\lambda_i \in \mathbb{R}$, $\forall i \in \mathcal{N}$, le sous-problème de la relaxation d'Held et Karp utilisant la *1-arborescence* est alors définie comme suit :

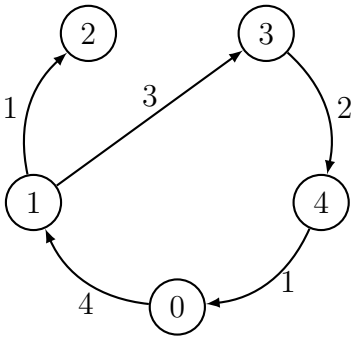
$$(\mathbf{RL}_{17}) \quad \textit{Min} \sum_{(i,j)\in\mathcal{A}} (c_{i,j} - \lambda_i)x_{i,j} + \sum_{i\in\mathcal{N}} \lambda_i \tag{17a}$$

$$\textit{1-arborescence}(\{x_{i,j}|\forall(i,j) \in \mathcal{A}\}) \tag{17b}$$

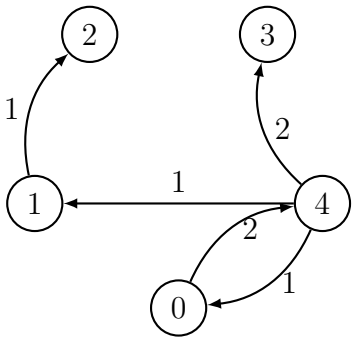
$$x_{i,j} \in \{0,1\}$$



(a) Graphe asymétrique



(b) Une *1-arborescence* de poids 11 du graphe Figure 15a



(c) Une *1-arborescence* minimale de poids 7 du graphe Figure 15a

FIGURE 15 – Graphe asymétrique et *1-arborescence*

La première méthode utilisant la transformation de Jonker et Volgenant n'augmente pas la complexité temporelle au pire cas ($\mathcal{O}(n)$) mais double la taille du problème. Suivant l'algorithme utilisé pour la recherche de *1-arbre*, la complexité de l'algorithme de Kruskal est en $\mathcal{O}(m \log(n))$ et celle de l'algorithme de Prim en $\mathcal{O}(m + n \log(n))$. n étant le nombre de nœuds dans le graphe et m le nombre d'arêtes. Pour la recherche de *1-arborescence*, la complexité temporelle est de $\mathcal{O}(m + n \log(n))$ [Gab+86]. Cependant la recherche des multiplicateurs lagrangiens optimums pour ($\mathbf{RL}_{17}$) est beaucoup plus instable [Ben+12] que celle pour ($\mathbf{RL}_{16}$) en terme de temps et de qualité de la borne inférieure.

3.1.2 *n-path*

La relaxation *n-path* peut être définie par le programme dynamique donné dans la Section 2.2.2.3. Nous définissons *n-path*$(\{x_{i,j} | \forall (i,j) \in \mathcal{A}\})$

comme la propriété imposant aux $x_{i,j}$ de former un chemin de n-arcs donné par le programme dynamique ($\mathbf{PD}_{13}$).

Comme pour la relaxation d'Held et Karp, la contrainte de degré sur les nœuds est relâchée. Une relaxation lagrangienne est donc appliquée sur la contrainte de degré afin de pénaliser la violation de cette dernière. Pour cela, un multiplicateur lagrangien $\lambda_i \in \mathbb{R}$ est associé à chaque nœud $i \in \mathcal{N}$. Le sous-problème lagrangien est défini comme suit :

$$(\mathbf{RL}_{18}) \quad Min \quad \sum_{(i,j)\in\mathcal{A}} (c_{i,j} - \lambda_i - \lambda_j)x_{i,j} + 2\sum_{j\in\mathcal{N}} \lambda_j \tag{18a}$$

$$n\text{-}path(\{x_{i,j} | \forall (i,j) \in \mathcal{A}\}) \tag{18b}$$

$$x_{i,j} \in \mathbb{N}$$

Le sous-problème implique que le domaine de définition des $x_{i,j}$ soit $\mathbb{N}$, $\forall (i,j) \in \mathcal{A}$. En effet, il est possible de passer plusieurs fois par le même arc.

3.1.3 *n-path sans 1-circuit*

Il est possible d'améliorer la borne inférieure de la relaxation *n-path* sans modifier la complexité temporelle de l'algorithme. En effet, la relaxation représentée par le programme dynamique ($\mathbf{PD}_{13}$) autorise les sous-circuits de taille 1. Soient les nœuds x,y, un sous-circuit de taille 1 est de la forme $x \to y \to x$ (Figure 16).

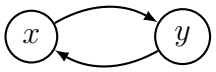


FIGURE 16 – Sous-circuit de taille 1

Comme mentionné dans [CMT81b], il est possible d'interdire ce type de sous-circuit sans modifier la complexité temporelle. Pour cela, nous devons stocker la valeur $\phi^*(k,i)$, $\forall i \in \mathcal{N}$, $\forall k \in \{2,\ldots,n\}$, correspondant à la valeur du meilleur chemin pour atteindre i en k arcs ayant un prédécesseur différent de celui obtenu avec $f^*(k,i)$. La fonction $\pi(k,i)$, $\forall i \in \mathcal{N}, \forall k \in \{2,...,n\}$ retourne le dernier nœud visité du meilleur chemin de valeur $f^*(k,i)$.

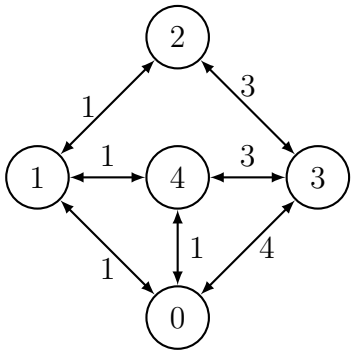
Le programme dynamique est le suivant, $\forall i \in \mathcal{N}$:

$$
\begin{aligned}
(\mathbf{PD}_{19}) \quad & f^*(1,i) = c_{0,i} \qquad \phi^*(1,i) = \infty && \text{(19a)}\\
& f^*(k,i) = \min\{ \min_{j \in \Gamma^-(i) | \pi(k-1,j) \neq i} (f^*(k-1,j) + c_{j,i}), \\
& \qquad \min_{j \in \Gamma^-(i) | \pi(k-1,j) = i} (\phi^*(k-1,j) + c_{j,i})\} \\
& \quad \forall k \in \{2,\dots,n\} && \text{(19b)}\\
& \phi^*(k,i) = \min\{ \min_{j \in \Gamma^-(i) | \pi(k,i) \neq j \wedge \pi(k-1,j) = i} (\phi^*(k-1,j) + c_{j,i}), \\
& \qquad \min_{j \in \Gamma^-(i) | \pi(k,i) \neq j \wedge \pi(k-1,j) \neq i} (f^*(k-1,j) + c_{j,i}), \infty\}, \\
& \quad \forall k \in \{2,\dots,n\} && \text{(19c)}
\end{aligned}
$$

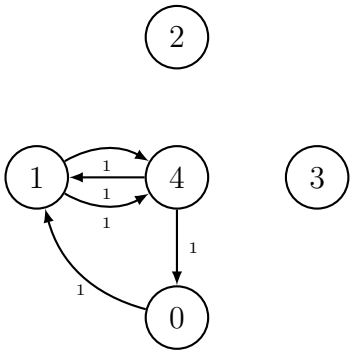
Les équations 19a définissent les valeurs initiales pour la fonction f^* et ϕ^*. La valeur $f^*(k,i)$ provient soit de la meilleure valeur de $f^*(k-1,i)$, soit de la meilleure valeur de $\phi^*(k-1,i)$, équation 19b. Si le nœud i n'est pas le prédécesseur de j dans le chemin utilisant $k-1$ arcs ($\pi(k-1,j) \neq i$), alors la meilleure valeur provient de $f^*(k-1,i)$. Sinon la condition suivante est vérifiée $\pi(k-1,j) = i$ et pour éviter l'apparition d'un sous-circuit, la valeur de $\phi^*(k-1,i)$ doit être sélectionnée. L'équation 19c définissant $\phi^*(k,i)$ est similaire à l'équation précédente, nous devons cependant nous assurer que le prédécesseur de $j \in \Gamma^-(i)$ ne soit pas i ($\pi(k,i) \neq j$).

La Figure 17 montre l'amélioration obtenue pour un graphe donné (Figure 17a) entre la *n-path* originale (Figure 17b) et la *n-path* sans les 1-circuits (Figure 17c). La borne inférieure obtenue par la *n-path* est de 5 (Figure 17b) avec des séquences $1 \to 4 \to 1 \to 4$. La borne inférieure obtenue en interdisant de telles séquences est de 9 (Figure 17c) et correspond ici à un tour (la solution optimale du *TSP*).

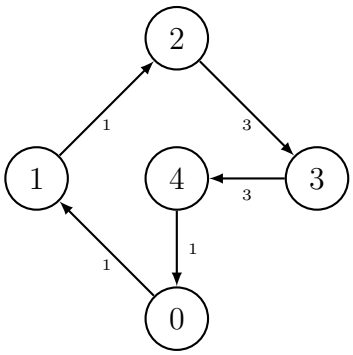
Dans la suite, nous définissons $\textit{n-path}^{\text{no1Circuit}}(\{x_{i,j} | \forall (i,j) \in \mathcal{A}\})$ comme un chemin de n-arcs ne possédant pas de 1-circuit. De la même façon, il est possible d'appliquer une relaxation lagrangienne pénalisant la contrainte de degré d'un nœud. Pour cela, la contrainte 18b de $(\mathbf{RL}_{18})$ est remplacée par $\textit{n-path}^{\text{no1Circuit}}(\{x_{i,j} | \forall (i,j) \in \mathcal{A}\})$.



(a) Graphe composé de 5 nœuds



(b) *n-path* autorisant les 1-circuits du graphe Figure 17a



(c) *n-path* sans 1-circuit du graphe Figure 17a

FIGURE 17 – *n-path* avec 1-circuit et *n-path* sans 1-circuit

3.2 CARACTÉRISATION PAR LA PROGRAMMATION LINÉAIRE

3.2.1 *Held et Karp*

La valeur optimale de la relaxation d'Held et Karp utilisant les *1-arbres* ($Max_\lambda(\mathbf{RL}_{16})$, Section 3.1.1) peut se caractériser par la valeur optimale de la relaxation linéaire de ($\mathbf{PL}_8$) (Section 2.2.1.1). En effet, le sous problème lagrangien respecte la propriété d'intégralité et d'après le théorème de Geoffrion, Held et Karp ont montré que l'optimisation sous λ ($Max_\lambda(\mathbf{RL}_{16})$) produisait la même valeur que la relaxation linéaire de ($\mathbf{PL}_8$) (Section 2.2.1.1).

De la même façon, il est démontré dans [Wil90] que la valeur optimale de la relaxation d'Held et Karp utilisant les *1-arborescences* ($Max_\lambda(\mathbf{RL}_{17})$) produit la même valeur que la relaxation linéaire de ($\mathbf{PL}_9$) pour les graphes orientés (Section 2.2.1.2).

Observation 3.2.1. *Les valeurs des relaxations lagrangiennes données par les sous-problèmes* ($\boldsymbol{RL}_{16}$) *et* ($\boldsymbol{RL}_{17}$) *ne dépendent pas du nœud de départ choisi.*

Démonstration. En effet, dans les définition des relaxations linéaires de ($\mathbf{PL}_8$) (équivalente à la relaxation lagrangienne constitué du sous-problème ($\mathbf{RL}_{16}$)) et ($\mathbf{PL}_9$) (équivalente à la relaxation lagrangienne constitué du sous-problème ($\mathbf{RL}_{17}$)), il n'y a pas de différence entre les nœuds et aucune contrainte n'est imposée sur le nœud 0.

Cela signifie que n'importe quel nœud du graphe peut être utilisé comme point de départ pour la recherche de *1-arbre* ou de *1-arborescence*.

Observation 3.2.2. *Soit un graphe orienté* $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$ *symétrique* ($c_{i,j} = c_{j,i}$, $\forall (i,j) \in \mathcal{A}$), *alors la résolution des relaxations lagragiennes* ($\boldsymbol{RL}_{16}$) *et* ($\boldsymbol{RL}_{17}$) *donnes des bornes inférieures identiques.*

Démonstration. Held et Karp ont démontré que la valeur optimale de la relaxation lagrangienne utilisant le *1-arbre* était équivalente à la valeur optimale de la relaxation linéaire de ($\mathbf{PL}_8$), nous rappelons sa définition :

$$(\mathbf{PL}_8) \quad \textit{Min} \qquad z = \sum_{(i,j)\in\mathcal{E}} c_{i,j}x_{i,j} \tag{20a}$$

$$\textit{s.t.} \qquad \sum_{j\in\Gamma(i)|j>i} x_{i,j} + \sum_{j\in\Gamma(i)|j<i} x_{j,i} = 2 \qquad \forall i \in \mathcal{N} \tag{20b}$$

$$\sum_{i\in S, j\in S} x_{i,j} \leqslant |S| - 1 \qquad \begin{matrix}\forall S \subset \mathcal{E},\\ |S| \geqslant 2\end{matrix} \tag{20c}$$

$$x_{i,j} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{E} \tag{20d}$$

Williamson dans [Wil90] a prouvé que la relaxation lagrangienne utilisant la *1-arborescence* était équivalente à la relaxation linéaire de (**PL**$_9$) :

$$
\begin{aligned}
(\mathbf{PL}_9) \quad \textit{Min} \quad & z' = \sum_{(i,j)\in\mathcal{A}} c_{i,j}x'_{i,j} && && (21a)\\
\textit{s.t.} \quad & \sum_{j\in\Gamma^+(i)} x'_{i,j} = 1 && \forall i \in \mathcal{N} && (21b)\\
& \sum_{i\in\Gamma^-(j)} x'_{i,j} = 1 && \forall j \in \mathcal{N} && (21c)\\
& \sum_{i\in S, j\in S} x'_{i,j} \leqslant |S| - 1 && \forall S \subset \mathcal{A}, |S| \geqslant 2 && (21d)\\
& x'_{i,j} \in \{0,1\} && \forall (i,j) \in \mathcal{A} && (21e)
\end{aligned}
$$

Il faut donc montrer que les relaxation linéaires de (**PL**$_8$) et (**PL**$_9$) sont équivalentes lorsque $c_{i,j} = c_{j,i}$, $\forall (i,j) \in \mathcal{N}^2$.

— Démontrons d'abord que $z^* \leqslant z'^*$. Soit x' une solution fractionnaire réalisable de (**PL**$_9$) :

$$x_{i,j} = x'_{i,j} + x'_{j,i} \qquad \forall (i,j) \in \mathcal{E}$$

Il faut vérifier que les contraintes de (**PL**$_8$) sont respectées. En sommant les contraintes 21b et 21c, nous obtenons alors la contrainte 20b. La contrainte 21d implique la contrainte 20c. De plus, l'ensemble de définition est bien vérifié. Nous avons donc :

$$z^* = \sum_{(i,j)\in\mathcal{E}} c_{i,j}(x'_{i,j} + x'_{j,i}) = \sum_{(i,j)\in\mathcal{E}} c_{i,j}x'_{i,j} + \sum_{(i,j)\in\mathcal{E}} c_{i,j}x'_{j,i}$$

Étant donné que $c_{i,j} = c_{j,i}$, nous obtenons alors :

$$z^* = \sum_{(i,j)\in\mathcal{A}} c_{i,j}x'_{i,j} = z'^*$$

— Démontrons maintenant que $z'^* \leqslant z^*$. Soit x une solution fractionnaire réalisable de (**PL**$_8$) :

$$
\begin{aligned}
x'_{i,j} &= \frac{x_{i,j}}{2} && \forall (i,j) \in \mathcal{N}^2 | i < j && (22a)\\
x'_{j,i} &= \frac{x_{i,j}}{2} && \forall (i,j) \in \mathcal{N}^2 | i > j && (22b)
\end{aligned}
$$

Les équations 22a et 22b assurent que $\forall i \in \mathcal{N}$:

$$\sum_{j\in\mathcal{N}|j>i} x'_{i,j} = \sum_{j\in\mathcal{N}|j<i} x'_{j,i}$$

De plus l'équation 20b nous donne l'égalité suivante $\forall i \in \mathcal{N}$:

$$\sum_{j \in \mathcal{N} | j > i} x'_{i,j} + \sum_{j \in \mathcal{N} | j < i} x'_{j,i} = 2$$

Ainsi, nous pouvons en déduire que 21c et 21b sont vérifiées. Les équations 20c et 20d impliquent respectivement les équations 21d et 21e. La fonction objectif peut donc s'exprimer de la façon suivante :

$$\begin{aligned} z'^{*} &= \sum_{(i,j) \in \mathcal{A}} c_{i,j} x'_{i,j} = \sum_{(i,j) \in \mathcal{A} | i < j} c_{i,j} \frac{x_{i,j}}{2} + \sum_{(i,j) \in \mathcal{A} | i > j} c_{i,j} \frac{x_{i,j}}{2} \\ &= \sum_{(i,j) \in \mathcal{E}} c_{i,j} x_{i,j} = z^{*} \end{aligned}$$

Il a été montré que $z^{*} \leqslant z'^{*}$ et $z'^{*} \leqslant z^{*}$, c'est à dire $z^{*} = z'^{*}$, alors les relaxations linéaires de $(\mathbf{PL}_8)$ et $(\mathbf{PL}_9)$ sont équivalentes pour les graphes orientés symétriques.

Autrement dit, pour les graphes symétriques, l'emploi de la relaxation d'Held et Karp utilisant les *1-arbres* ou celle utilisant les *1-arborescences* donnent les mêmes valeurs.

3.2.2 *n-path*

Nous utilisons une formulation par couches proche de celle de Vajda [S.61] pour modéliser le *TSP*. Soit un graphe orienté $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$, une formulation valide du *TSP* est donnée par :

$$
\begin{aligned}
(\mathbf{PL}_{23}) \quad \textit{Min} \quad & \sum_{(i,j)\in\mathcal{A}} c_{i,j}x_{i,j} && && (23a)\\
\textit{s.t.} \quad & \sum_{j\in\Gamma^+(i)} x_{i,j} + \sum_{j\in\Gamma^-(i)} x_{j,i} = 2 && \forall i \in \mathcal{N} && (23b)\\
& x_{i,j} - \sum_{k\in\{0,\dots,n\}} y^k_{i,j} = 0 && \forall (i,j) \in \mathcal{A} && (23c)\\
& \sum_{j\in\Gamma^+(0)} y^0_{0,j} = 1 && && (23d)\\
& \sum_{i\in\Gamma^-(0)} y^n_{i,0} = 1 && && (23e)\\
& \sum_{j\in\Gamma^+(i)} y^k_{i,j} - \sum_{j\in\Gamma^-(i)} y^{k-1}_{j,i} = 0 && \forall i \in \mathcal{N},\ \forall k \in \{1,\dots,n\} && (23f)\\
& x_{i,j} \in \mathbb{N} && \forall (i,j) \in \mathcal{A} &&\\
& y^k_{i,j} \in \mathbb{N} && \forall (i,j) \in \mathcal{A},\ \forall k \in \{0,\dots,n\} &&
\end{aligned}
$$

Il apparait que les contraintes 10c et 10d du $(\mathbf{PL}_{10})$ pour le *TSP* ont été additionnées pour donner la contrainte 23b. De plus, les variables x et y ne sont pas restreintes à l'ensemble $\{0, 1\}$ du fait des contraintes 23d et 23e.

Une relaxation lagrangienne, avec les multiplicateurs $\lambda_i \in \mathbb{R}, \forall i \in \mathcal{N}$, est appliquée sur la contrainte de degré 23b. Le sous-problème lagrangien est le suivant :

$$(\mathbf{PL}_{24}) \quad f_\lambda^{24}(x) = Min \sum_{(i,j)\in\mathcal{A}} (c_{i,j} - \lambda_i - \lambda_j)x_{i,j} + 2\sum_{i\in\mathcal{N}} \lambda_i \tag{24a}$$

$$s.t. \quad x_{i,j} - \sum_{k\in\{0,\dots,n\}} y_{i,j}^k = 0 \qquad \forall (i,j) \in \mathcal{A} \tag{24b}$$

$$\sum_{j\in\Gamma^+(0)} y_{0,j}^0 = 1 \tag{24c}$$

$$\sum_{i\in\Gamma^-(0)} y_{i,0}^n = 1 \tag{24d}$$

$$\sum_{j\in\Gamma^+(i)} y_{i,j}^k - \sum_{j\in\Gamma^-(i)} y_{j,i}^{k-1} = 0 \qquad \forall i \in \mathcal{N}, \ \forall k \in \{1,\dots,n\} \tag{24e}$$

$$x_{i,j} \in \mathbb{N} \qquad \forall (i,j) \in \mathcal{A}$$

$$y_{i,j}^k \in \mathbb{N} \qquad \forall (i,j) \in \mathcal{A}, \ \forall k \in \{0,\dots,n\}$$

Ainsi le $(\mathbf{PL}_{24})$ équivaut au sous-problème lagrangien $(\mathbf{RL}_{18})$ (Section 3.1.2).

Observation 3.2.3. *L'optimisation sous λ du $(\boldsymbol{PL}_{24})$, à savoir $Max_\lambda(f_\lambda^{24}(x))$ donne la même valeur optimale que la relaxation linéaire du $(\boldsymbol{PL}_{23})$.*

Démonstration. D'après le théorème de Geoffrion [Geo10] (Section 2.1.3), il faut démontrer que $(\mathbf{PL}_{24})$ respecte la propriété d'intégralité. Soit le graphe donné par la Figure 18.

Le $(\mathbf{PL}_{24})$ consiste à rechercher le plus court chemin du nœud 0 de la couche 0 au nœud 0 de la couche n dans le graphe $\mathcal{G}_0$. Les contraintes 24d, 24c et 24e consistent à rechercher un flot de valeur 1 de coût minimum. Or, la matrice d'un problème de flot à coût minimum est totalement unimodulaire. Les variables y sont donc entières. La contrainte 24b impose donc les variables x entières, puisque les variables y sont entières. Nous pouvons en déduire que les contraintes du $(\mathbf{PL}_{24})$ vérifient la propriété d'intégralité.

3.2.3 *n-path sans 1-circuit*

Le sous-problème formulé comme un problème de plus court chemin (Figure 18) peut se généraliser en interdisant les 1-circuits. En effet, il suffit

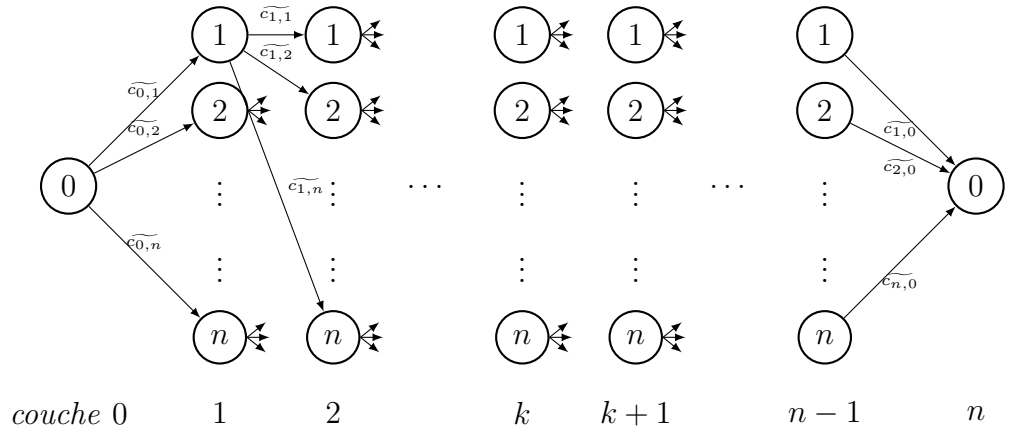


FIGURE 18 – Graphe par couches
$\widetilde{c_{i,j}} = c_{i,j} - \lambda_i - \lambda_j, \forall (i,j) \in \mathcal{A}$

de considérer les paires de nœuds (arcs) à chaque couche du graphe. La Figure 19 montre un exemple de la couche k et $k-1$ lorsque les nœuds correspondent à un arc.

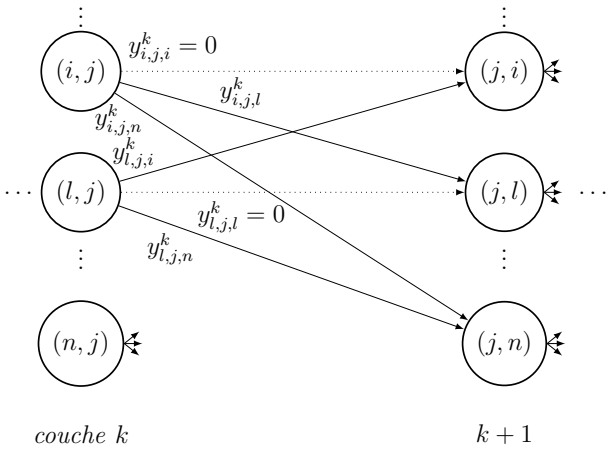


FIGURE 19 – Graphe par couches suivant les arcs

Nous introduisons donc une formulation sur 4 indices afin d'éliminer les 1-circuits. Les variables de décisions sont donc $x_{i,j}$ et $y^k_{i,j,l}$, $\forall (i,j,l) \in \mathcal{A}^2, \forall k \in \{0,\dots,n\}$.

Ainsi pour éliminer les 1-circuits, il faut imposer la condition suivante :

$$y^k_{i,j,i} = 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \{0,\dots,n\}$$

Voici une reformulation valide pour le *TSP*, s'appuyant sur le ($\mathbf{PL}_{23}$).

$$
\begin{aligned}
(\mathbf{PL}_{25}) \quad \textit{Min} \quad & \sum_{(i,j)\in\mathcal{A}} c_{i,j}x_{i,j} && && \text{(25a)}\\
\textit{s.t.} \quad & \sum_{j\in\Gamma^+(i)} x_{i,j} + \sum_{j\in\Gamma^-(i)} x_{j,i} = 2 && \forall i \in \mathcal{N} && \text{(25b)}\\
& \sum_{k\in\{0,\dots,n\}} \Big(\sum_{l\in\Gamma^+(j)} y^k_{i,j,l} + \sum_{l\in\Gamma^-(j)} y^k_{l,j,i} \Big) = x_{i,j} && \forall (i,j) \in \mathcal{A} && \text{(25c)}\\
& \sum_{l\in\Gamma^+(0)} y^0_{0,0,l} = 1 && && \text{(25d)}\\
& \sum_{j\in\Gamma^-(0)} \sum_{i\in\Gamma^-(j)} y^n_{i,j,0} = 1 && && \text{(25e)}\\
& \sum_{l\in\Gamma^+(j)} y^k_{i,j,l} - \sum_{l\in\Gamma^-(i)} y^{k-1}_{l,i,j} = 0 && \forall (i,j) \in \mathcal{A}, \; \forall k \in \{1,\dots,n\} && \text{(25f)}\\
& y^k_{i,j,i} = 0 && \forall (i,j) \in \mathcal{A}, \; \forall k \in \{0,\dots,n\} && \text{(25g)}\\
& x_{i,j} \in \mathbb{N} && \forall (i,j) \in \mathcal{A} &&\\
& y^k_{i,j,l} \in \mathbb{N} && \forall (i,j,l) \in \mathcal{A}^2, \; \forall k \in \{0,\dots,n\} &&
\end{aligned}
$$

Nous retrouvons la contrainte 25g permettant d'éliminer les 1-circuits. La complexité spatiale du programme linéaire est en $\mathcal{O}(n^4)$, tandis que celle du programme dynamique ($\mathbf{PD}_{19}$) est en $\mathcal{O}(n^2)$.

De la même façon que pour la *n-path*, une relaxation lagrangienne est appliquée sur la contrainte 25b. Ceci équivaut à une reformulation de ($\mathbf{RL}_{18}$) avec la contrainte *n-path*$^{\text{no1Circuit}}$ (Section 3.1.3).

Observation 3.2.4. *L'optimisation sous λ de ($\boldsymbol{RL}_{18}$) avec la n-path*$^{\text{no1Circuit}}$*, à savoir $\mathrm{Max}_\lambda(\boldsymbol{RL}_{18})$ donne la même valeur optimale que la relaxation linéaire du ($\boldsymbol{PL}_{25}$).*

Démonstration. La démonstration est la même que pour la *n-path* (observation 3.2.3), il nous suffit, d'après le théorème de Geoffrion, de montrer que le sous-problème respecte la propriété d'intégralité. Le sous-problème consiste à rechercher un plus court chemin dans le graphe représenté par la Figure 19.

Observation 3.2.5. *La valeur de la relaxation lagrangienne du sous problème qui utilise la n-path*$^{\text{no1Circuit}}$ $\text{Max}_\lambda(\boldsymbol{RL}_{18})$ *dépend du nœud de départ choisi.*

Démonstration. La preuve utilise un graphe particulier étudié Section 3.3.2. Le lecteur pourra se reporter sur l'Annexe C pour la preuve complète.

3.3 COMPARAISON THÉORIQUE DES BORNES

Dans cette section, nous allons étudier les différentes bornes du *TSP* présentées précédemment. En effet, cette étude permet, avec des exemples, de montrer les limites des différentes relaxations utilisées. La borne utilisant les arborescences couvrantes de poids minimum est notée z_{oa} (Section 3.1.1). Nous notons z_{ap} la borne obtenue grâce à la relaxation utilisant le problème d'affection (Section 2.2.2.1). z_{np} désigne la borne de la relaxation utilisant un chemin de n-arcs sans 1-circuit (Section 3.1.3). Les versions lagrangiennes des bornes *1-arborescence* et *n-path*$^{\text{no1Circuit}}$ sont notées respectivement z_{oa}^{rl} et z_{np}^{rl}.

3.3.1 *Version non-lagrangienne*

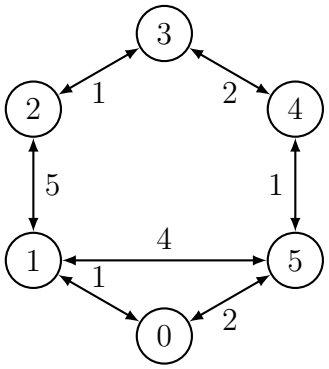
Dans cette section, nous nous intéressons aux bornes fournies par les relaxations *1-arborescence*, *affectation* et *n-path*$^{\text{no1Circuit}}$. Cette étude permet de mettre en lumière les cas extrêmes des différentes relaxations et de montrer qu'elles sont incomparables.

Proposition 3.3.1. *Les valeurs des bornes inférieures z_{oa}, z_{ap} et z_{np} sont incomparables.*

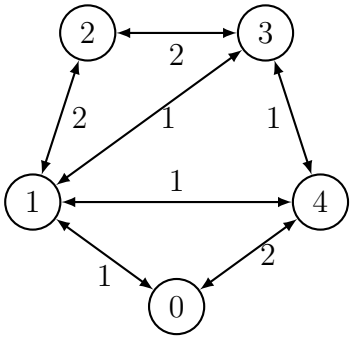
Démonstration. Notre démonstration s'effectue en deux étapes en s'appuyant sur des exemples. Soit les graphes G_1 et G_2 définis sur la Figure 20.

Nous allons tout d'abord prouver que $z_{ap} < z_{oa} < z_{np}$ pour le graphe G_1 pour ensuite démontrer l'inégalité inverse pour le graphe G_2.

— $z_{ap} < z_{oa} < z_{np}$: nous utilisons le graphe G_1 Figure 20a pour montrer l'inégalité.
- $z_{ap}(G_1) = 6$: l'affectation présentée Figure 21a est optimale puisqu'elle utilise uniquement des arcs de poids minimum.
- $z_{oa}(G_1) = 10$: une arborescence de poids minimum sur le graphe G_1 consiste à sélectionner les arcs de poids minimum si le nœud incident n'a pas encore été atteint. Ainsi, l'algorithme sélectionnera, dans l'ordre, l'arc $(0,1)$, $(5,4)$, $(3,2)$, $(4,2)$ et enfin $(1,5)$. Il faut compléter par l'arc entrant au nœud o de poids minimum $(1,0)$ (Figure 21b).

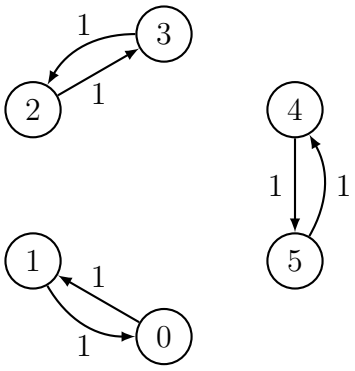


(a) Graphe G_1

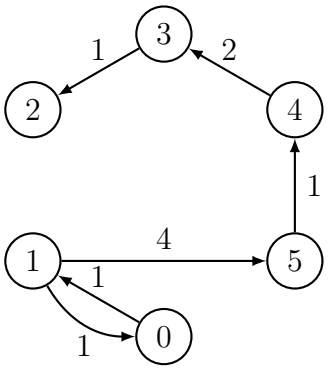


(b) Graphe G_2

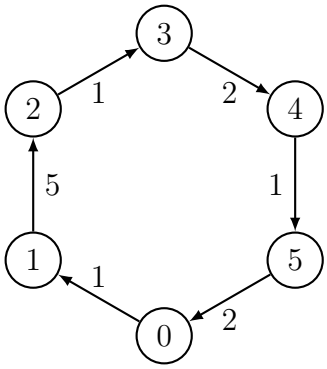
FIGURE 20 – Graphe G_1 et G_2



(a) Solution de *affectation*



(b) Solution de *1-arborescence*



(c) Solution de *n-path*$^{\text{no1Circuit}}$

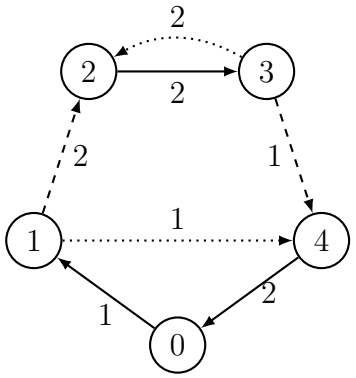
FIGURE 21 – Solutions des relaxations sur le graphe G_1 Figure 20a

- $z_{np}(G_1) = 12$: la relaxation *n-path*$^{\text{no1Circuit}}$ possède deux solutions symétriques avec un coût de 12, l'une d'entre elle est donnée Figure 21c. Supposons que nous démarrons notre che-

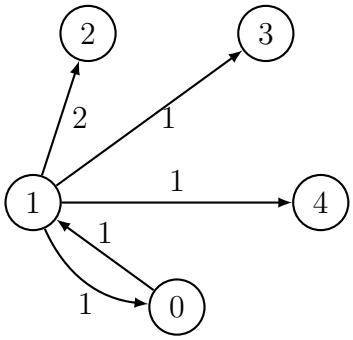
min par l'arc $(0,1)$, alors deux choix sont ensuite possibles : l'arc $(1,2)$ ou l'arc $(1,5)$. Si l'arc $(1,5)$ est sélectionné, alors il sera impossible d'atteindre le nœud 0 en utilisant 6 arcs sans créer de 1-circuit $(0-1-5-4-5-1-0)$. Ainsi, le seul choix possible pour ne pas créer de 1-circuit est l'arc $(1,2)$. Le même raisonnement s'applique en commençant avec l'arc $(0,5)$.

Nous avons donc $z_{ap}(=6) < z_{oa}(=10) < z_{np}(=12)$

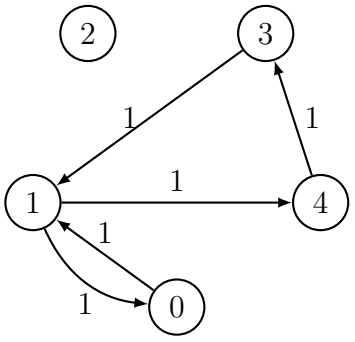
— $z_{ap} > z_{oa} > z_{np}$: nous utilisons le graphe G_2 Figure 20b pour montrer l'inégalité.



(a) Solution de *affectation*



(b) Solution de *1-arborescence*



(c) Solution de *n-path*$^{\text{no1Circuit}}$

FIGURE 22 – Solutions des relaxations sur le graphe G_2 Figure 20b

- $z_{ap}(G_2) = 8$: deux solutions optimales ayant en commun les arcs incidents au nœud 0 sont possibles pour le problème d'affectation sur le graphe G_2. Ces deux solutions sont représentées sur la Figure 22a avec un coût de 8. Supposons l'arc $(0,1)$ sélectionné dans le couplage. Alors l'arc $(2,1)$ est rejeté, ce qui implique l'utilisation de l'arc $(2,3)$ dans le couplage car il n'existe qu'un seul arc sortant du nœud 2. De la même façon, l'arc $(4,0)$

est sélectionné pour le couplage. Il y a donc deux couplages parfaits possibles. Le premier utilise les arcs $(1,2)$ et $(3,4)$, le second les arcs $(3,2)$ et $(1,4)$ avec comme coût 8. La démonstration est la même si nous avions sélectionné l'arc $(0,4)$ en premier.

- $z_{oa}(G_2) = 6$: une arborescence sortant du nœud 0 dans le graphe est obligée d'atteindre le nœud 2. Pour cela, un arc de poids 2 ($(1,2)$ ou $(3,2)$) est obligatoirement sélectionné. Pour atteindre les autres nœuds, il est possible de prendre les arcs de poids 1. L'arc entrant au nœud 0 est l'arc de poids minimum, à savoir l'arc $(1,0)$. Nous obtenons donc une *1-arborescence* de poids 5. Une des solutions de la relaxation est donnée Figure 22b.
- $z_{np}(G_2) = 5$: un plus court chemin utilisant 5 arcs consiste à passer par les arcs de poids 1 en passant deux fois par le nœud 1. La solution est représentée Figure 22c.

Ainsi $z_{ap}(=8) > z_{oa}(=7) > z_{np}(=5)$

Nous avons démontré les deux inégalités suivantes : $z_{ap} > z_{oa} > z_{np}$ et $z_{ap} < z_{oa} < z_{np}$ pour deux graphes différents. Ainsi les bornes des relaxations *affectation*, *1-arborescence* et *n-path*$^{\text{no1Circuit}}$ sont incomparables.

3.3.2 *Version lagrangienne*

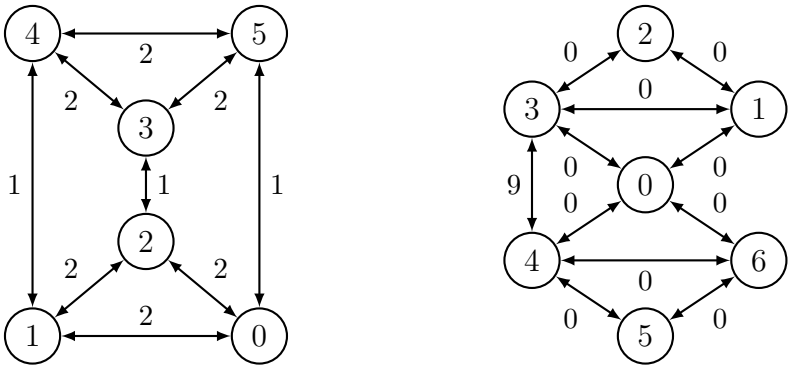
Seules les relaxations *n-path*$^{\text{no1Circuit}}$ et *1-arborescence* ont été étendues dans leur version lagrangienne en pénalisant une contrainte de degré ($(\mathbf{RL}_{18})$ et $(\mathbf{RL}_{17})$ Section 3.1).

Proposition 3.3.2. *Les valeurs des bornes inférieures z^{rl}_{oa} et z^{rl}_{no} sont incomparables.*

Démonstration. De la même façon que la proposition précédente, nous allons nous appuyer sur des exemples. Soit les graphes G_3 et G_4 définis sur la Figure 23.

— $z^{rl}_{oa} < z^{rl}_{np}$: nous utilisons le graphe G_3 Figure 23a pour montrer l'inégalité.

- $z^{rl}_{oa} = 9$: l'étude faite dans [BB08] exhibe des graphes où la relaxation d'Held et Karp ne donne pas la valeur optimale du *TSP*, mais possède un écart d'optimalité bien définie. Ainsi, sur le graphe G_3, il a été montré que la marge d'erreur était de $10/9$. L'étude est proposée pour la relaxation d'Held et Karp utilisant les *1-arbres*. Le graphe G_3 est symétrique, donc la valeur de la relaxation qui utilise la *1-arborescence* est la même que celle qui utilise l'*1-arbre* sur ce graphe (Observation 3.2.2), soit $z^{rl}_{oa} = 9$.



(a) Graphe G_3 (b) Graphe G_4

FIGURE 23 – Graphe G_3 et G_4

- $z^{rl}_{np} = 10$: tous les chemins provenant du nœud 0 sans 1-circuit de 6 arcs possèdent un coût de 10. Or $z^{rl}_{np} \geqslant z_{np} = 10$, donc $z^{rl}_{no} \geqslant 10$.

Nous avons donc $z^{rl}_{oa} < z^{rl}_{no}$ sur le graphe G_3.

— $z^{rl}_{oa} > z^{rl}_{no}$: nous utilisons le graphe G_4 Figure 23b pour démontrer l'inégalité.

- $z^{rl}_{oa} \geqslant 8$: pour un graphe composé de 7 nœuds, il a été montré dans [BB08] que l'écart d'optimalité pour la relaxation d'Held et Karp était de 8/9 par rapport au tour optimal. Sur le graphe G_4 la valeur du tour optimal est de 9 puisque au moins l'un des deux arcs $(3,4)$ et $(4,3)$ doit apparaitre dans un tour. D'après l'observation 3.2.2, le graphe G_4 étant symétrique nous pouvons en déduire que $z^{rl}_{oa} \geqslant 8$.
- $z^{rl}_{np} \leqslant 0$: le dual lagrangien du programme défini par les équations 25 possède, entres autres, les deux contraintes suivantes :

$$
\begin{aligned}
z^{rl}_{np} &= \max w \\
w &\leqslant -2\lambda_1 - 2\lambda_2 - 2\lambda_3 + 2\lambda_4 + 2\lambda_5 + 2\lambda_6 && \text{(26a)} \\
w &\leqslant +2\lambda_1 + 2\lambda_2 + 2\lambda_3 - 2\lambda_4 - 2\lambda_5 - 2\lambda_6 && \text{(26b)}
\end{aligned}
$$

La contrainte 26a correspond au chemin possible $(0-1-2-3-1-2-3-0)$. Les nœuds 1, 2 et 3 possèdent un degré de 4, il faut donc soustraire 2 fois leurs multiplicateurs lagrangiens dans la fonction objectif de $(\mathbf{PL}_{25})$, inversement pour les nœuds 4, 5 et 6 qui possèdent un degré nul. La contrainte 26b correspond au

chemin $(0-6-5-4-6-5-4-0)$. En faisant la somme de ces deux contraintes, nous montrons que $z^{rl}_{np} \leqslant 0$.

Ainsi $z^{rl}_{oa} > z^{rl}_{np}$ sur le graphe G_4.

Nous avons donc démontré les deux inégalités $z^{rl}_{oa} < z^{rl}_{np}$ et $z^{rl}_{oa} > z^{rl}_{np}$, ce qui implique que les relaxations lagrangiennes qui utilisent les *1-arborescences* et *n-path*^{no1Circuit} sont incomparables.

Nous pouvons observer que le graphe G_3 montre un exemple où la relaxation lagrangienne utilisant la *n-path*^{no1Circuit} (**RL**$_{18}$) donne le tour optimal tandis que le graphe G_4 montre un exemple où la relaxation reste complètement non informative et ce, malgré l'utilisation de multiplicateurs lagrangiens.

CONCLUSION

Nous avons pu expliciter les différentes bornes inférieures obtenues grâce aux relaxations utilisées pour le *problème du voyageur de commerce*. Cela a permis de caractériser les diverses relaxations en programmation linéaire. Une comparaison théorique des différentes bornes inférieures a permis de montrer qu'elles pouvaient être incomparables.

4

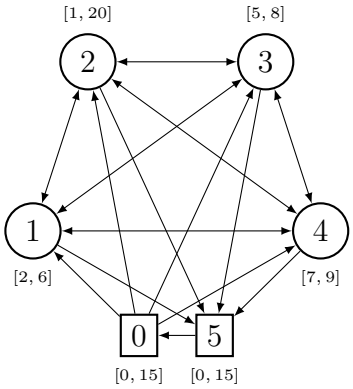
APPROCHES EN PROGRAMMATION PAR CONTRAINTES POUR LE TSP ET LE TSPTW

Dans ce chapitre, nous allons voir les différents modèles et algorithmes pour le *TSP* et la variante avec fenêtres de temps en *programmation par contraintes*. Deux articles ont été présentés pour ces travaux [DCP16 ; DCP15].

4.1 MODÈLES

Soit un graphe $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$ avec $\mathcal{N} = \{0, \ldots, n+1\}$. L'ensemble $\{1, \ldots, n\}$ correspond aux clients à visiter. Contrairement au modèle classique de *TSP* (Section 2.2), le nœud de dépôt est dupliqué. Le nœud du début du tour correspond au nœud 0 et le nœud de fin de tour correspond au nœud $n+1$. Ce modèle est explicité dans [BMR12a]. Ainsi pour une instance avec $n = 4$, nous disposons de 4 clients ($\{1, \ldots, 4\}$), du nœud de départ ($\{0\}$) et du nœud de fin ($\{5\}$) du tour. On note $d_{i,j}$ (resp. $t_{i,j}$) la distance (resp. le temps de trajet) entre le nœud i et le nœud j. Chaque nœud i possède une fenêtre de temps, notée $[a_i, b_i]$. Sans perte de généralité, le temps de service du nœud i est compris dans les temps de trajet partant de ce nœud. Le principe du *problème du voyageur de commerce avec fenêtres de temps* est de visiter tous les nœuds en commençant par le nœud 0 et en finissant par le nœud $n+1$ tout en minimisant la distance parcourue et en respectant les fenêtres de temps des clients. Nous notons D_x, le domaine de la variable x et $\overline{x}$ (resp. $\underline{x}$) la borne supérieure (resp. inférieure) de x.

Un exemple de graphe est donné Figure 24. La Figure 24a montre le graphe d'entrée du problème composé de quatre clients (nœuds 1, 2, 3 et 4) ainsi que le nœud de début de tournée (nœud 0) et le nœud de fin de tournée (nœud 5). La fenêtre de temps de chaque nœud est aussi indiquée. Le Tableau 24b donne la matrice des distances entre les nœuds. Nous considérons dans cet exemple que le temps de trajet entre chaque nœud est égal à la distance entre les nœuds. Cet exemple sera le fil conducteur pour illustrer les modèles présentés dans ce chapitre.



(a) Exemple de graphe pour le *TSPTW* avec 4 clients

	0	1	2	3	4	5
0	0	2	3	3	3	0
1	3	0	2	4	4	3
2	4	2	0	2	3	4
3	3	3	2	0	1	3
4	4	4	4	1	0	4
5	0	2	3	3	3	0

(b) Matrice des distances (égales aux temps de trajets dans l'exemple)

FIGURE 24 – Exemple de données pour le *TSPTW* avec $n = 4$

4.1.1 *Modèle de base*

Le modèle de base s'appuie sur les variables $next_i$ pour chaque nœud $i \in \mathcal{N}$. Ces variables représentent le successeur immédiat du nœud i dans le tour (par convention $next_{n+1} = 0$). Des variables représentant une accumulation de quantité sont aussi utilisées. Nous introduisons les variables $start_i \in [a_i, b_i], \forall i \in \mathcal{N}$ pour le temps accumulé le long du trajet. Ces dernières représentent le temps de début du service au nœud i. Le modèle de base est le suivant :

$$(\mathbf{MO}_{27}): \quad \textit{Min} \quad z \tag{27a}$$

$$z = \sum_{i=0}^{n} (d_{i,next_i}) \tag{27b}$$

$$\text{CIRCUIT}(next_0, ..., next_{n+1}) \tag{27c}$$

$$start_{next_i} \geqslant start_i + t_{i,next_i} \qquad \forall i \in \mathcal{N}\backslash\{n+1\} \tag{27d}$$

$$start_0 = 0 \tag{27e}$$

L'équation 27b définit l'objectif à minimiser. Elle correspond à la somme des distances utilisées pour le tour. $dist_{i,next_i}$ est la distance parcourue entre le nœud i et son successeur dans le tour. Cette variable utilise la contrainte globale ELEMENT présentée dans [VHC88]. Il en est de même pour les variables $start_{next_i}$ et $t_{i,next_i}$. La contrainte 27c, présentée dans [Lau78] et impliquant notamment la contrainte $\text{ALLDIFFERENT}(next_0, \ldots, next_{n+1})$, permet de maintenir un circuit dans le graphe en visitant une seule fois

chaque nœud.

Exemple 4.1.1. *Reprenons l'exemple donné dans la Figure 24 et examinons les domaines des différentes variables après la première propagation (Tableau 2) :*

i	0	1	2	3	4	5
$next_i$	$\{1,2,3,4\}$	$\{2,3,4,5\}$	$\{1,3,4,5\}$	$\{1,2,4,5\}$	$\{2,3,5\}$	$\{0\}$
$start_i$	0	$[2,6]$	$[\mathbf{3},\mathbf{13}]$	$[5,8]$	$[7,9]$	$[0,15]$
$d_{i,next_i}$	$[2,3]$	$[2,4]$	$[2,4]$	$[1,3]$	$[1,4]$	0
$start_{next_i}$	$[2,13]$	$[4,15]$	$[3,15]$	$[6,15]$	$[8,15]$	0

Tableau 2 – État des domaines des variables

Pour les variables successeurs $next$, *seule la valeur* 1 *de* $next_4$ *a été filtrée. En effet, du fait des fenêtres de temps des nœuds* 1 *et* 4, *le nœud* 1 *ne peut être visité après le nœud* 4. *Le domaine de la variable* $d_{i,next_i}$ *(dans cet exemple* $t_{i,next_i} = d_{i,next_i}$*) d'un nœud* i *correspond à l'intervalle entre la distance minimum et maximum partant du nœud* i. *Nous avons pu aussi diminuer le domaine de la variable* $start_2$ *qui initialement était de* $[1,20]$. *En effet, la valeur maximum de la variable* $start_5$, *correspondant au nœud de fin de tournée, est égale à* 15. *Le temps de trajet entre le nœud* 2 *et* 5 *est de deux. Il est alors impossible de partir du nœud* 2 *après le temps* 13, *d'où la réduction de la variable* $start$ *du nœud* 2. *La variable* $start_{next_0}$ *prend ces valeurs entre la borne minimum et la borne maximum des variables* $start$ *correspondant au domaine de* $next_0$, *d'où le domaine* $[2,13]$.

Il est très courant de rajouter des variables et contraintes supplémentaires afin d'augmenter le filtrage. Nous ajoutons donc des variables prédécesseurs $pred_i$, $\forall i \in \mathcal{N}$, afin de représenter le prédécesseur immédiat du nœud i dans le tour. Ces variables sont liées aux variables $next$ par la propriété suivante : $next_i = j \Leftrightarrow pred_j = i$ pour chaque paire $(i,j) \in \mathcal{A}$ ou $next_{pred_i} = i$, $\forall i \in \mathcal{N}$. Cette propriété se traduit en termes de *programmation par contraintes* par l'utilisation de la contrainte globale INVERSE :

$$\text{INVERSE}(\{next_0,\ldots,next_{n+1}\},\{pred_0,\ldots,pred_{n+1}\})$$

L'utilisation des prédécesseurs renforce la borne inférieure sur z. En effet, z peut être défini de la façon suivante :

$$z = \sum_{i=0}^{n}(d_{i,next_i}) = \sum_{i=1}^{n+1}(d_{pred_i,i})$$

Les quantités $\sum_{i=0}^{n}(\underline{d_{i,next_i}})$ et $\sum_{i=1}^{n+1}(\underline{d_{pred_i,i}})$ ne sont pas forcément égales (en particulier dans les problèmes asymétriques) ce qui explique la contrainte précédente.

Exemple 4.1.2. *Lorsque nous incluons les variables prédécesseurs* pred*, l'état des domaines est le suivant (Tableau 3) :*

i	0	1	2	3	4	5
$pred_i$	$\{5\}$	$\{0,2,3\}$	$\{0,1,3,4\}$	$\{0,1,2,4\}$	$\{0,1,2,3\}$	$\{1,2,3,4\}$
$d_{pred_i,i}$	0	$[2,3]$	$[2,4]$	$[1,4]$	$[1,4]$	$[3,4]$

Tableau 3 – État des domaines des variables des variables pred

Nous nous apercevons que $\sum_{i=0}^{n}(\underline{d_{i,next_i}}) = 8$*. L'utilisation des prédécesseurs permet d'obtenir une borne inférieure* $\sum_{i=1}^{n+1}(\underline{d_{pred_i,i}}) = 9$ *meilleure que celle proposée seulement par les variables* next*.*

Cependant, ces quantités sont égales lorsque les variables sont instanciées. En outre, ces variables offrent des possibilités supplémentaires lors de l'étape de recherche (Section 2.4.3).

4.1.2 *Modèles redondants*

Dans cette section, nous allons étudier différents modèles redondants. Pour le *problème du voyageur de commerce avec fenêtres de temps*, de nombreuses contraintes supplémentaires ont été étudiées afin d'améliorer la résolution. Ainsi, on ajoute au modèle de base la contrainte sur l'élimination des arcs proposée dans [Lan+93]. Elle permet l'élimination des valeurs dans le domaine des next en considérant les fenêtres de temps :

$$start_i + t_{i,j} > start_j \Rightarrow next_i \neq j \qquad \forall (i,j) \in \mathcal{A} \tag{28}$$

Cette contrainte logique signifie simplement qu'au moment de servir le client i, si le temps de trajet entre le client i et j est supérieur à la date au plus tard de visite de j, alors le client j ne peut pas être le successeur immédiat de i. Cette contrainte est vérifiée dès lors que nous avons la condition suivante :

$$\underline{start_i} + t_{i,j} > \overline{start_j}$$

La contrainte 28 permet un meilleur filtrage que la contrainte 27d du modèle ($\mathbf{MO}_{27}$).

Exemple 4.1.3. *Lorsque nous ajoutons cette contrainte au modèle précédent, l'état des domaines est le suivant (Tableau 4) :*

i	0	1	2	3	4	5
$next_i$	$\{1,2,3,4\}$	$\{2,3,4,5\}$	$\{1,3,4,5\}$	$\{\mathbf{2},\mathbf{4},\mathbf{5}\}$	$\{2,3,5\}$	$\{0\}$
$pred_i$	$\{5\}$	$\{\mathbf{0},\mathbf{2}\}$	$\{0,1,3,4\}$	$\{0,1,2,4\}$	$\{0,1,2,3\}$	$\{1,2,3,4\}$

Tableau 4 – État des domaines des variables modifiées avec l'ajout de la contrainte d'élimination d'arc

Les variables modifiées par rapport au modèle précédent sont en gras. L'ajout de la contrainte d'élimination d'arcs permet de supprimer la valeur 1 *du domaine de* $next_3$.

Cette contrainte est couplée avec la contrainte de réduction des variables $start$ présentée dans [DDS92].

$$start_i \geqslant \min_{k \in D(pred_i)} (start_k + t_{k,i}) \qquad \forall i \in \mathcal{N}\backslash\{0\}$$
$$start_i \leqslant \max_{k \in D(next_i)} (start_k - t_{i,k}) \qquad \forall i \in \mathcal{N}\backslash\{n+1\} \tag{29}$$

Les contraintes 29 permettent d'affiner les bornes des variables $start$ en parcourant le domaine des $next$ ou $pred$. Dans [Pes+98], les deux dernières contraintes peuvent se généraliser en utilisant le temps de plus court chemin entre chaque paire de clients (i, j). En conséquence, il est possible de considérer non plus les successeurs et prédécesseurs directs d'un nœud, mais l'ensemble des clients devant être visités avant ou après le nœud. Toutefois, dans [FLM02], les expérimentations ont montré que le calcul du plus court chemin était trop coûteux pour le gain obtenu.

Exemple 4.1.4. *L'ajout de ces deux contraintes 29 permet d'obtenir l'état des domaines des variables suivant (Tableau 5) :*

i	0	1	2	3	4	5
$start_i$	0	$[2,6]$	$[\mathbf{3},\mathbf{11}]$	$[5,8]$	$[7,9]$	$[\mathbf{5},\mathbf{15}]$
$start_{next_i}$	$[\mathbf{2},\mathbf{11}]$	$[4,15]$	$[\mathbf{5},\mathbf{15}]$	$[6,15]$	$[8,15]$	0

Tableau 5 – État des domaines des variables modifiées avec l'ajout des contraintes de mise à jour des fenêtres de temps

Nous nous apercevons que les bornes inférieures des variables $start$ *et les variables* $start_{next}$*, qui utilisent la contrainte* ELEMENT*, sont principalement mises à jour.*

Dans la suite, nous présentons deux modèles redondants pour le *problème du voyageur de commerce avec fenêtres de temps*.

4.1.2.1 *Modèle logique*

Les fenêtres de temps apportent une dimension asymétrique et ordonnée par rapport au problème de base (*TSP*). En effet, la temporalité du problème permet de définir et de raisonner sur les visites pouvant être faites au début du tour ou à la fin du tour. C'est pourquoi, des variables booléennes $b_{i,j} \in \{0,1\}, \forall (i,j) \in \mathcal{A}$ peuvent être ajoutées permettant d'indiquer si le nœud i est visité avant le nœud j : $b_{i,j} = 1$. Inversement, si le nœud i est visité après le nœud j : $b_{i,j} = 0$. Au lieu de raisonner uniquement sur les successeurs et prédécesseurs directs, nous nous intéressons aux clients situés avant et après dans le tour. Cela aura pour conséquence de mieux prendre en compte la temporalité du problème dans les raisonnements. On associe les contraintes suivantes au modèle $(\mathbf{MO}_{27})$, $\forall (i,j) \in \mathcal{N}^2$ avec $i \neq j$:

$(\mathbf{MO}_{30})$: $(\mathbf{MO}_{27})$

$$b_{i,j} + b_{j,i} = 1 \tag{30a}$$

$$(b_{i,j} = 1 \wedge b_{j,k} = 1) \Rightarrow b_{i,k} = 1 \qquad \forall k \in \mathcal{N} \backslash k \neq i \neq j \tag{30b}$$

$$(b_{i,j} = 1) \Rightarrow next_j \neq i \tag{30c}$$

$$(b_{i,j} = 1) \wedge (b_{j,k} = 1) \Rightarrow next_i \neq k \qquad \forall k \in \mathcal{N} \backslash k \neq i \neq j \tag{30d}$$

$$(b_{i,j} = 1) \Rightarrow start_j \geqslant start_i + t_{i,j} \tag{30e}$$

La contrainte 30a implique qu'un nœud i ne peut pas être visité avant et après un nœud j. L'équation 30b, appelée fermeture transitive, permet de définir les successeurs et prédécesseurs indirects, c'est-à-dire que s'il existe un nœud j devant être visité après i et avant k, alors i doit être visité avant k. Évidemment, si le nœud i est visité avant le nœud j, alors i ne peut pas être le successeur direct de j, équation 30c. L'équation 30d permet de lier cette fermeture transitive aux valeurs des variables $next$. De plus, cette contrainte est équivalente à la contrainte 28 d'élimination des arcs. Les valeurs des variables $start$ sont aussi filtrées grâce à la contrainte 30e. L'équivalence $start_j \geqslant start_i + t_{i,j} \Rightarrow (b_{i,j} = 1)$ est alors implicite car l'équation 30e est définie sur les paires $(i,j) \in \mathcal{A}$ et grâce à l'équation 30a.

Ce modèle apporte beaucoup en termes de raisonnement sur les relations de précédences entre les nœuds. Il ajoute malheureusement de l'ordre de $\mathcal{O}(n^3)$ contraintes supplémentaires, à cause des contraintes 30b et 30d.

Ce modèle permet d'encoder un graphe de précédence utilisé en *PPC* pour des problèmes d'ordonnancement, notamment avec la contrainte NO-OVERLAP [Car82 ; Vil04].

4.1.2.2 *Modèle positionnel*

Le modèle précédent ($\mathbf{MO}_{30}$) apporte une représentation précise des relations de précédence. Les équations 30b ajoutent malheureusement de l'ordre de $\mathcal{O}(n^3)$ contraintes supplémentaires. Cependant, il est possible de relâcher ce modèle tout en gardant un niveau de cohérence proche. Pour cela, nous introduisons pour chaque nœud $i \in \mathcal{N}$ une variable position $pos_i \in \{0, \dots, n+1\}$ permettant de représenter la position du nœud dans le tour. Il est évident que pour le nœud de début (resp. fin), $pos_0 = 0$ (resp. $pos_{n+1} = n+1$). Les contraintes de ce modèle sont les suivantes, $\forall (i,j) \in \mathcal{N}^2$ avec $i \neq j$:

($\mathbf{MO}_{31}$) : ($\mathbf{MO}_{27}$)

$$pos_i = \sum_{j \in N \setminus \{i\}} b_{j,i} \tag{31a}$$

$$b_{i,j} + b_{j,i} = 1 \tag{31b}$$

$$(b_{i,j} = 1) \Rightarrow next_j \neq i \tag{31c}$$

$$(b_{i,j} = 1) \Rightarrow start_j \geqslant start_i + t_{i,j} \tag{31d}$$

$$pos_j > pos_i \Leftrightarrow b_{i,j} = 1 \tag{31e}$$

$$pos_j > pos_i + 1 \Rightarrow next_i \neq j \tag{31f}$$

$$\text{ALLDIFFERENT}(pos_0, \dots, pos_{n+1}) \tag{31g}$$

La contrainte 31a permet de définir les variables pos. Nous retrouvons les contraintes 30a, 30c et 30e du modèle logique ($\mathbf{MO}_{30}$). La contrainte 31e permet de remplacer la contrainte 30b. Un filtrage des variables next grâce aux variables pos est effectué par la contrainte 31f. Cette contrainte est équivalente à la contrainte d'élimination des arcs 28. Évidemment, chaque position d'un nœud dans le tour est unique et les positions sont donc toutes différentes, d'où la contrainte 31g.

4.1.2.3 *Approche de type ordonnancement*

Le *problème du voyageur de commerce avec fenêtres de temps* peut être vu également comme un problème d'ordonnancement sur une seule machine avec un temps de *set-up* dépendant de la séquence et des dates de disponibilités et de fins de tâches. Le modèle précédent ($\mathbf{MO}_{31}$) est adapté à la mise en oeuvre de raisonnements énergétiques semblables à ceux de la

contrainte NOOVERLAP [Car82] utilisés en ordonnancement. L'application du filtrage de cette contrainte est trop coûteux dans ce contexte et pas toujours utile. Une des grandes difficultés est la prise en compte des temps de *set-up* entre les différentes activités dues au temps de trajet entre chaque visite. Dans ce sens, les bornes inférieures pour le *TSP* ont été utilisées pour la contrainte NOOVERLAP [DVCS15]. Nous souhaitons renforcer le filtrage sur les variables pos grâce à des raisonnements énergétiques simples.

Un filtrage supplémentaire est alors ajouté au modèle qui utilise les positions ($\mathbf{MO}_{31}$), noté $(\mathbf{MO}_{31})^+$ permettant de mettre à jour les variables positions pos par rapport à l'ensemble des autres clients et de leurs possibilités de placement avant ou après.
Soit A_i l'ensemble des clients qui doivent être placés après le client i et B_i l'ensemble des clients pouvant être visités avant ou après le client i :

$$A_i = \{j \in \mathcal{N} \mid b_{i,j} = 1\}$$
$$B_i = \{j \in \mathcal{N} \mid \overline{b_{i,j}} = 1 \wedge \underline{b_{i,j}} = 0\}$$

Le temps minimum requis après i est noté EA_i :

$$EA_i = \underline{t_{i,next_i}} + \sum_{j \in A_i} \underline{t_{j,next_j}}$$

Dans un premier temps, nous remarquons que la borne supérieure de $start_i$ ne peut pas dépasser le temps minimum requis par les visites de A_i :

$$\overline{start_i} \leqslant \overline{start_{n+1}} - EA_i$$

Dans un second temps, nous filtrons la borne inférieure de pos_i. Considérons les visites $x_1, \ldots, x_k$ de B_i rangées par ordre croissant de distance aux successeurs :

$$\underline{t_{x_1,next_{x_1}}} \leqslant \underline{t_{x_2,next_{x_2}}} \leqslant \ldots \leqslant \underline{t_{x_k,next_{x_k}}}$$

Soit c le plus grand entier tel que :

$$EA_i + \sum_{j=0}^{c} \underline{t_{x_j,next_{x_j}}} \leqslant (\overline{start_{n+1}} - \underline{start_i})$$

$c + |A_i|$ est donc le nombre de visites maximum pouvant avoir lieu après la visite i, ce qui nous donne une borne inférieure pour pos_i :

$$\underline{pos_i} \geqslant n - (c + |A_i| + 1)$$

Un raisonnement symétrique est effectué pour la borne inférieure de $start_i$ et la borne supérieure de pos_i en prenant en compte le nombre minimum de visites pouvant se placer avant le client i.

Ce raisonnement est un cas particulier de raisonnement énergétique que ferait la contrainte NOOVERLAP [Car82] pour l'algorithme d'*edge-finding*. On pourrait écrire un modèle *PPC* s'appuyant sur une NOOVERLAP avec des durées variables pour les tâches en ajoutant une variable end_i telle que $start_i + t_{i,next_i} = end_i$. Ce modèle n'a pas été testé car les développements sur les durées de transition variables sont encore faibles pour les raisonnements de la contrainte NOOVERLAP [DVCS15].

4.1.2.4 *Comparaison des différents modèles*

Nous pouvons classer les différents modèles proposés ($\mathbf{MO}_{27}$), ($\mathbf{MO}_{30}$) et ($\mathbf{MO}_{31}$) selon le niveau de propagation atteint. En effet, le modèle de base ($\mathbf{MO}_{27}$) possède un niveau de propagation plus faible comparé au modèle ($\mathbf{MO}_{30}$) qui possède la plus forte propagation. Le modèle ($\mathbf{MO}_{31}$) est un modèle intermédiaire en termes de propagation. Par ailleurs, nous verrons dans les résultats numériques que le modèle $(\mathbf{MO}_{31})^+$ peut avoir un niveau de propagation plus élevé que le ($\mathbf{MO}_{30}$).

Modèle	i	0	1	2	3	4	5
($\mathbf{MO}_{27}$)	$next_i$	$\{1,2,3,4\}$	$\{2,3,4,5\}$	$\{1,3,4,5\}$	$\{2,4,5\}$	$\{2,3,5\}$	$\{0\}$
	$start_i$	0	$[2,6]$	$[3,11]$	$[5,8]$	$[7,9]$	$[5,15]$
($\mathbf{MO}_{30}$)	$next_i$	1	$\{2,3,4\}$	$\{3,4,5\}$	$\{2,4,5\}$	$\{2,3,5\}$	$\{0\}$
	$start_i$	0	$[2,4]$	$[4,11]$	$[6,8]$	$[7,9]$	$[11,15]$
$(\mathbf{MO}_{31})^+$	$next_i$	1	$\{2,3,4\}$	$\{3,4,5\}$	$\{2,4,5\}$	$\{2,3,5\}$	$\{0\}$
	$start_i$	0	$[2,4]$	$[4,11]$	$[6,8]$	$[7,9]$	$[11,15]$
	pos_i	0	1	$\{2,3,4\}$	$\{2,3,4\}$	$\{2,3,4\}$	5

Tableau 6 – État des domaines des variables avec le modèle position

Exemple 4.1.5. *Le Tableau 6 montre l'état des domaines des variables* next *et* start *suite à la première propagation des trois modèles étudiés. Les modèles* ($\mathbf{MO}_{30}$) *et* $(\mathbf{MO}_{31})^+$ *filtrent davantage que le modèle de base pour l'exemple. Cependant le filtrage est identique entre les deux derniers modèles.*

4.2 FILTRAGE POUR LA CONTRAINTE WEIGHTEDCIRCUIT

Les modèles précédents souffrent de l'absence d'une borne inférieure globale de la fonction objectif. Nous allons donc étudier dans cette section,

le remplacement de la contrainte CIRCUIT (27c) par la contrainte WEIGHTEDCIRCUIT présentée dans [Ben+12] ainsi que les différents algorithmes de filtrage. La contrainte WEIGHTEDCIRCUIT permet de maintenir un circuit dans un graphe pondéré et fournit une borne inférieure sur z. Les différents filtrages proposés s'appuient sur les coûts [FLM99] ce qui signifie qu'ils utilisent des relaxations du problème afin de procéder au filtrage. Pour cela, les trois relaxations étudiées dans le Chapitre 3 sont utilisées.

Soit $\mathcal{G}_{no} = (\mathcal{N}, \mathcal{E})$, nous notons $\mathcal{G}_{relaxation} = (\mathcal{N}, \mathcal{E}_{relaxation})$ le sous-graphe induit de $\mathcal{G}_{no}$ respectant la propriété de la *relaxation* utilisée. On note z l'objectif à minimiser et $w(i,j)$ le coût de l'arête (i,j) et $w(\mathcal{G})$ la somme des coûts des arêtes du graphe $\mathcal{G}$. Une notation similaire est utilisée pour les graphes orientés $\mathcal{G}_o = (\mathcal{N}, \mathcal{A})$.

4.2.1 *Filtrage avec Held et Karp*

L'un des filtrages utilisés est celui s'appuyant sur la relaxation d'Held et Karp définie pour les graphes non orientés. L'algorithme de filtrage utilisant la relaxation d'Held et Karp est décrit dans [Ben+12]. Dans cette version, la contrainte WEIGHTEDCIRCUIT se note :

$$\text{WEIGHTEDCIRCUIT}(next, z)$$

Deux composantes importantes sont identifiées dans l'algorithme de filtrage présenté : le processus de résolution du dual lagrangien et le filtrage par le sous-problème de recherche de *1-arbre*.

4.2.1.1 *Résolution du dual lagrangien*

L'intérêt de la relaxation d'Held et Karp (Section 3.2.1) réside dans la résolution de la relaxation lagrangienne associée au sous-problème de *1-arbre*. La fonction associée au dual lagrangien est une fonction concave. C'est pourquoi, pour résoudre le dual lagrangien, une procédure de sous-gradient est appliquée. Chaque nœud se voit attribuer un multiplicateur lagrangien λ. Ces derniers sont inclus dans le coût d'une arête (i,j), $w(i,j) = c_{i,j} - \lambda_i - \lambda_j$. L'idée pour résoudre le dual lagrangien par une procédure de sous-gradient est de pénaliser les nœuds ne respectant pas la contrainte de degré. Pour cela, lorsqu'un nœud possède un degré strictement supérieur à 2, le multiplicateur lagrangien augmente afin de réduire son degré dans une nouvelle solution. De façon symétrique, lorsqu'un nœud possède un degré strictement inférieur à 2, le multiplicateur lagrangien diminue pour tenter de l'inclure dans la solution suivante.

À chaque étape du sous-gradient, les multiplicateurs lagrangiens sont mis à jour. Nous utilisons alors la formule proposée par Held et Karp [HK70 ; HK71] :

$$\lambda_i^{t+1} = \lambda_i^t + \tau^t \left(\frac{\overline{z} - w(\mathcal{G}_{\textit{1-arbre}})}{\|C^t\|} \right) c_i^t \qquad \forall i \in \mathcal{N}$$

$$\|C^t\| = \sum_{i \in \mathcal{N}} (c_i^t)^2$$

$$c_i^t = \deg(i) - 2 \qquad \forall i \in \mathcal{N}$$

$$0 < \tau^t \leqslant 2$$

La fonction deg renvoie le degré du nœud. Ainsi c_i^t permet de quantifier la violation de la contrainte de degré pour un nœud i à l'étape t. De plus, nous divisons, après un certain nombre d'itérations, la valeur de τ par deux lors du processus de sous-gradient.

Le filtrage du *1-arbre* peut être appliqué à chaque étape du sous-gradient, c'est à dire pour chaque valeur de λ utilisée par le sous-gradient.

4.2.1.2 *Filtrage pour le 1-arbre*

Le sous-problème associé à la relaxation d'Held et Karp est la recherche de *1-arbre*.

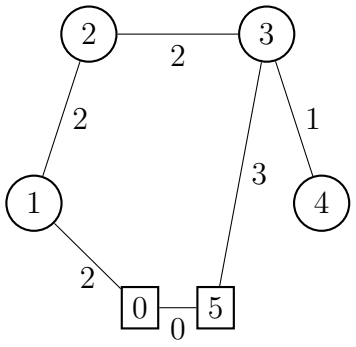


FIGURE 25 – Graphe $\mathcal{G}_{\textit{1-arbre}}$ pour un coût $w(\mathcal{G}_{\textit{1-arbre}}) = 10$ avec $n = 4$

Exemple 4.2.1. *La Figure 25 montre le 1-arbre de poids minimum pour le problème de TSP présenté précédemment après la première propagation donnée par le Tableau 6.*

L'algorithme de filtrage de la WEIGHTEDCIRCUIT est principalement fondé sur le filtrage d'un *1-arbre*. Dans ce filtrage, les arêtes interdites et les arêtes obligatoires sont identifiées.

ARÊTES INTERDITES L'identification des arêtes interdites s'inspire de l'analyse de sensibilité pour la recherche d'arbres couvrants de poids minimum [Tar82]. En effet, le coût marginal d'une arête est calculé. Le coût marginal d'une arête $\widetilde{w}(i,j)$, $(i,j) \in \mathcal{E}$ représente le coût supplémentaire lorsque l'on force l'arête (i,j) à être dans le graphe $\mathcal{G}_{\textit{1-arbre}}$. Nous avons alors la relation suivante :

$$w(\mathcal{G}_{\textit{1-arbre}}) + \widetilde{w}(i,j) > \bar{z} \Rightarrow next_i \neq j$$

Rappelons que la recherche d'*1-arbre* est définie sur un graphe non orienté et que nous utilisons la transformation de Jonker et Volgenant (Section 3.1.1) permettant de transformer un graphe orienté en graphe non orienté. Ainsi lorsque nous forçons l'arête (i,j) dans la solution du *1-arbre* et que cela entraine le dépassement de la borne supérieure de l'objectif, alors aucune solution ne peut contenir l'arête (i,j).

Tous les coûts marginaux sont alors calculés. Ce calcul, similaire à la contrainte SPANNINGTREE [Rég08], s'exécute en $\mathcal{O}(mn)$ en utilisant l'algorithme de Kruskal. Le filtrage des arêtes interdites est donc en $\mathcal{O}(m+n+n\log(n))$ [Rég+10].

ARÊTES OBLIGATOIRES Nous pouvons aussi identifier les arêtes obligatoires, c'est-à-dire celles qui se trouvent dans les solutions de coût inférieur à $\bar{z}$. Pour cela, le coût de remplacement $\widehat{w}(i,j)$ d'une arête $(i,j) \in \mathcal{E}_{\textit{1-arbre}}$ est calculé. Nous avons alors la relation suivante :

$$w(\mathcal{G}_{\textit{1-arbre}}) + \widehat{w}(i,j) > \bar{z} \Rightarrow next_i = j$$

Le coût de remplacement d'une arête correspond au coût supplémentaire minimum si cette arête ne devait pas être dans le *1-arbre*. Si ce dernier entraine une augmentation importante, alors l'arête se trouve dans toutes les solutions ayant un coût inférieur à la borne supérieure actuelle. Elle devient alors obligatoire.

Le calcul des coûts de remplacement découle du calcul des coûts marginaux et peut s'effectuer en $\mathcal{O}(m\alpha(m,n))$ [Tar82] avec α représentant la fonction inverse de la fonction d'Ackermann. Le filtrage des arêtes obligatoires est donc en $\mathcal{O}(m\alpha(m,n))$ [Rég+10].

4.2.2 *Filtrage avec n-path*

Nous étudions dans cette section un autre algorithme de filtrage pour la contrainte WEIGHTEDCIRCUIT. Il s'agit d'un filtrage fondé sur les coûts et

qui utilise la relaxation *n-path*$^{\text{no1Circuit}}$ (Section 3.2.3). Nous pouvons renforcer la borne obtenue grâce à la relaxation lagrangienne. Tout comme le filtrage avec la relaxation d'Held et Karp, le même processus de sous-gradient est utilisé pour résoudre le dual lagrangien.

Le sous-problème associé à la relaxation est défini par un programme dynamique ($\mathbf{PD}_{19}$) (Section 3.1.3). Dans le cadre du *TSP*, le programme dynamique est redéfini afin d'inclure les variables de position dans la recherche de plus court chemin. Ainsi, nous incluons dans la contrainte WEIGHTEDCIRCUIT les variables pred et pos :

$$\text{WEIGHTEDCIRCUIT}(\text{next}, \text{pred}, \text{pos}, z)$$

4.2.2.1 *Redéfinition du programme dynamique*

La *n-path* consiste, dans cette version, à trouver le plus court chemin du nœud 0 au nœud $n+1$ en utilisant exactement $n+1$ arcs. Le programme dynamique de la *n-path* est redéfini grâce aux variables dans la portée de la WEIGHTEDCIRCUIT, $\forall i \in \{1, \dots, n\}$:

$$(\mathbf{PD}_{32}) \quad f^*(1,i) = c_{0,i} \qquad \text{s.t } 1 \in D_{\text{pos}_i}$$

$$f^*(1,i) = \infty \qquad \text{s.t } 1 \notin D_{\text{pos}_i} \tag{32a}$$

$$f^*(k,i) = \min_{j \in D_{\text{pred}_i}} (f^*(k-1,j) + c_{j,i}) \qquad \forall k \in \{2, \dots, n\}, \; i \text{ s.t } k \in D_{\text{pos}_i} \tag{32b}$$

Rappelons que $f^*(k,i)$ représente la valeur optimale du plus court chemin entre le nœud 0 et le nœud i avec k arcs. Nous pouvons noter l'utilisation des domaines des variables pred et des variables pos afin de réduire l'espace de recherche. La borne inférieure sur z est alors donnée par $f^*(n+1, n+1)$. $f^*_{\text{rev}}(k,i)$ est aussi calculé et représente le plus court chemin entre le nœud i et le nœud $n+1$ utilisant k arcs. La complexité spatiale du programme dynamique est en $\mathcal{O}(n^2)$ et la complexité temporelle est de $\mathcal{O}(n^2 m)$ avec m la taille maximale des domaines des variables pred. En outre, nous utilisons la *n-path*$^{\text{no1Circuit}}$ en redéfinissant de la même façon le programme dynamique associé (Section 3.1.3).

Exemple 4.2.2. *La Figure 26 montre le plus court chemin de $n+1$ arcs qui ne contient pas de 1-circuit sur le graphe obtenu après la première propagation donnée Tableau 6.*

Grâce à ces modifications, les positions d'un nœud (Section 4.1.2.2) peuvent être prises en compte et ainsi réduire la complexité du programme dynamique, ce qui n'est pas le cas lors de la recherche de *1-arbre*.

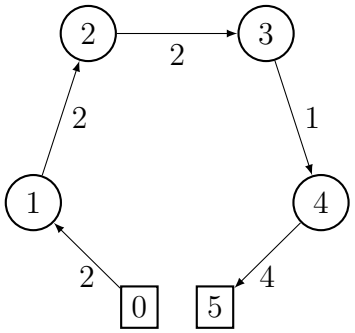


FIGURE 26 – Graphe $\mathcal{G}_{\textit{n-path}^{\text{noCircuit}}}$ pour un coût $w(\mathcal{G}_{\textit{n-path}^{\text{noCircuit}}}) = 11$ pour l'exemple avec $n = 4$

4.2.2.2 *Filtrage grâce au programme dynamique*

Grâce au programme dynamique, le domaine de plusieurs variables peut être filtré. Ainsi, nous filtrons les arcs interdits et les positions interdites d'un nœud. Rappelons que la relaxation lagrangienne est étudiée Section 3.1.3.

IDENTIFICATION DES ARCS INTERDITS Pour identifier les arcs interdits, le coût supplémentaire de l'ajout de cet arc à chaque position possible doit être estimé. Nous notons $w(i,j) = c_{i,j} - \lambda_i - \lambda_j$ et $C = 2\sum_{j\in\mathcal{N}} \lambda_j$. La relation suivante $\forall i \in \mathcal{N}$, $\forall j \in D_{next_i}$ identifie les arcs interdits :

$$(\forall k \in D_{pos_i},\ f^*(k,i) + w(i,j) + f^*_{rev}(n-k,j) - C > \bar{z}) \Rightarrow next_i \neq j$$

$f^*(k,i)$ représente le coût minimum pour atteindre le nœud i à partir du nœud 0 en utilisant k arcs. $f^*_{rev}(n-k,j)$ représente le coût minimum pour atteindre le nœud $n+1$ à partir de j en $n-k$ arcs. Il nous suffit alors de tester si l'ajout de l'arc (i,j) est acceptable pour n'importe quelle position, ce qui correspond bien à un plus court chemin composé de $n+1$ arcs. Or, si cette augmentation n'est pas valide pour toutes les positions possibles, alors cet arc ne fait pas partie d'une solution de coût inférieur à $\bar{z}$.

IDENTIFICATION DES POSITIONS INTERDITES Il est possible aussi d'identifier les positions interdites pour un nœud. En effet, nous avons redéfini le programme dynamique de façon à prendre en considération le domaine des variables pos. Ainsi, $\forall i \in \mathcal{N}$, $\forall k \in D_{pos_i}$:

$$f^*(k,i) + f^*_{rev}(n-k+1,i) - C > \bar{z} \Rightarrow pos_i \neq k \tag{33}$$

Le principe est de comparer le coût minimum pour atteindre i en k arcs et celui en partant de i utilisant $n-k+1$ arcs. Si ces coûts sont supérieurs

à la valeur de la meilleure solution courante, alors le nœud i ne peut pas se trouver à la position k.

FILTRAGE SANS RELAXATION LAGRANGIENNE Il est possible de filtrer les variables $start$ avec la relaxation *n-path*$^{\text{no1Circuit}}$ sans la présence des multiplicateurs lagrangiens. En effet, le programme dynamique peut être appliqué sur la matrice de temps puisque $c_{ij} = t_{i,j}$, $\forall (i,j) \in \mathcal{A}$. En conséquence, nous pouvons filtrer les bornes des variables $start$ grâce aux relations suivantes, $\forall i \in \mathcal{N}$:

$$start_i \geqslant f^*(\underline{pos_i}, i) \tag{34a}$$

$$start_i \leqslant \overline{start_{n+1}} - f^*_{rev}(n - \overline{pos_i} + 1, i) \tag{34b}$$

Lorsque $c_{ij} = t_{i,j}$, $f^*(k,i)$ correspond au temps minimum pour atteindre le nœud i en utilisant k arcs. C'est pourquoi, les variables $start$ peuvent être bornées à l'aide du programme dynamique. Le temps minimum pour atteindre le nœud i est représenté par $f^*(\underline{pos_i}, i)$, $start_i$ ne peut donc pas prendre des valeurs inférieures au temps minimum (contrainte 34a). De la même façon, la contrainte 34b réduit la borne supérieure de la variable $start$.

Il est aussi possible de réduire les bornes des variables positions pour un nœud i. Nous avons les relations suivantes, $\forall i \in \mathcal{N}$, $\forall k \in D_{pos_i}$:

$$f^*(k,i) > \overline{start_i} \Rightarrow pos_i < k \tag{35a}$$

$$\underline{start_i} + f^*_{rev}(n-k+1, i) > \overline{start_{n+1}} \Rightarrow pos_i > k \tag{35b}$$

Si en utilisant k arcs, le temps minimum pour atteindre le nœud i est supérieur à la borne supérieure de la variable $start$, alors le nœud i ne pourra jamais se trouver après la position k (contrainte 35a). Nous avons le raisonnement symétrique grâce à la contrainte 35b.

4.2.3 *Filtrage avec l'affectation*

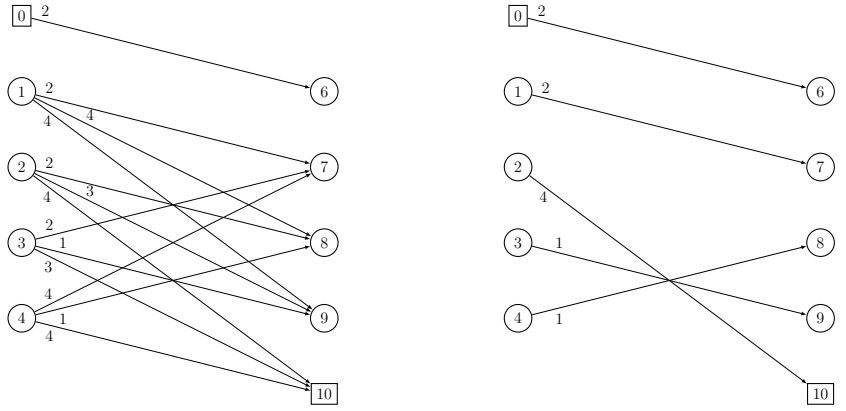
La troisième relaxation étudiée pour le *TSP* est le problème d'affectation (Section 2.2.2.1). Cette relaxation est utilisée pour augmenter la borne inférieure pour la WEIGHTEDCIRCUIT mais aussi pour apporter du filtrage sur les variables $next$.

FILTRAGE EXISTANT Le problème d'affection consiste à trouver un couplage parfait de poids minimum dans un graphe biparti. Dans le cas de la WEIGHTEDCIRCUIT, le graphe biparti correspondant est noté $\mathcal{G}_B = (\mathcal{N}\backslash\{n+1\} \cup \mathcal{V}, \mathcal{A}_B)$, avec $\mathcal{V} = \{n+i, \forall i \in \mathcal{N}\backslash\{0\}\}$ tel que :

$$(i,j) \in \mathcal{A}_B \Leftrightarrow i \in \mathcal{N}\backslash\{n+1\} \ \wedge \ (j-n) \in D_{next_i} \tag{36}$$

Afin d'obtenir une borne inférieure pour le *TSP*, nous résolvons le problème par l'algorithme de Kuhn [Kuh10]. Notons $\mathcal{G}_{affectation} = (\mathcal{N}\backslash\{n+1\} \cup \mathcal{V}, \mathcal{A}_{affectation})$, le graphe ainsi obtenu et $w(\mathcal{G}_{affectation})$ la valeur optimale du problème d'affectation.

Exemple 4.2.3. *Reprenons l'exemple de l'état des domaines des variables avec le modèle position, Tableau 6. Nous pouvons alors utiliser le problème d'affectation dans la contrainte* WEIGHTEDCIRCUIT *: La Figure 27 montre la modélisation en*



(a) Graphe biparti $\mathcal{G}_B$ défini par la relation 36

(b) Graphe $\mathcal{G}_{affectation}$ avec le coût associé $w(\mathcal{G}_{affectation}) = 10$.

FIGURE 27 – Graphe biparti et résolution du problème d'affectation

problème d'affectation. La Figure 27a représente le graphe biparti défini par l'équation36. La Figure 27b représente la solution du problème d'affectation, nous notons le graphe obtenu $\mathcal{G}_{affectation}$ et le coût associé $w(\mathcal{G}_{affectation})$.

Le filtrage utilisant la relaxation du problème d'affectation a été étudié dans [FLM02]. Les auteurs utilisent les coûts réduits $\widetilde{w}(i,j)$ d'un arc $(i,j) \in \mathcal{A}_B$. Ce dernier correspond à l'augmentation minimum du coût $w(\mathcal{G}_{affectation})$ si la valeur de $next_i$ est égale à j. Nous avons donc la relation suivante :

$$w(\mathcal{G}_{affectation}) + \widetilde{w}(i,j) > \bar{z} \Rightarrow next_i \neq j \tag{37}$$

Les coûts réduits peuvent être approximés grâce au programme linéaire associé et en utilisant la notion de dualité [FLM02]. Cependant, le calcul des coûts réduits exacts peut néanmoins se faire en temps polynomial.

CALCUL DES COÛTS RÉDUITS EXACTS Les coûts réduits exacts d'un arc $(i,j) \in \mathcal{A}_B$ sont obtenus à partir du graphe résiduel [Rég02]. Ce dernier est un graphe obtenu à partir du graphe $\mathcal{G}_B$ dont tous les arcs ayant une capacité résiduelle nulle ont été supprimés. La capacité résiduelle d'un arc (i,j) correspond à la quantité de flot pouvant encore traverser l'arc (i,j) dans la solution $\mathcal{G}_{affectation}$.

Il est alors nécéssaire d'identifier les chemins augmentants alternatifs dans le graphe résiduel. Pour cela, nous utilisons l'algorithme de Floyd-Warshall permettant de calculer les plus courts chemins entre toutes les paires de nœuds d'un graphe. Cet algorithme a une complexité en $\mathcal{O}(n^3)$. Notons sp_{ij}, $\forall (i,j) \in \mathcal{A}_B$ la valeur du plus court chemin de l'arc (i,j) obtenue à partir du graphe résiduel. Le coût réduit $\widetilde{w}(i,j)$ d'un arc (i,j) correspond à l'augmentation du coût global $w(\mathcal{G}_{affectation})$ lorsque l'arc (i,j) se trouve dans le graphe $\mathcal{G}_{affectation}$.

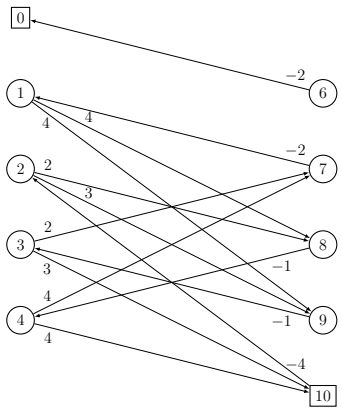


FIGURE 28 – Graphe résiduel de $\mathcal{G}_{affectation}$

Exemple 4.2.4. *La Figure 28 montre le graphe résiduel correspondant à l'exemple* $\mathcal{G}_{affectation}$.

Notons $j' \in \mathcal{V}$ le nœud puits de l'arc $(i,j) \in \mathcal{A}_{affectation}$ dans la solution du graphe $\mathcal{G}_{affectation}$. De la même façon, $i' \in \mathcal{N} \backslash \{n+1\}$ le nœud source de

l'arc $(i,j) \in \mathcal{A}_{affectation}$ dans le graphe $\mathcal{G}_{affectation}$. Le coût réduit exact s'obtient de la façon suivante, $\forall (i,j) \in \mathcal{A}_B$:

$$\widetilde{w}(i,j) = (w(i,j) + sp_{i'j'}) - (w(i,j') + w(i',j)) \tag{38}$$

Le premier terme représente le coût minimum qu'impliquerait l'ajout de l'arc (i,j) dans la solution, tandis que le second terme enlève les coûts relatifs aux arcs de la solution. Grâce à la contrainte 37 et au calcul des coûts réduits exacts (38), il est possible d'identifier les arcs interdits.

Il n'est pas nécessaire d'identifier les arcs obligatoires car un arc $(i,j) \in \mathcal{V}$ est un arc obligatoire si tous les autres arcs $(i,k) \in \mathcal{V}$ sont des arcs interdits.

4.3 STRATÉGIES DE BRANCHEMENT

Les stratégies de branchement sont la troisième composante importante en *programmation par contraintes* (Section 2.4.3). Elles visent à parcourir l'espace de recherche le plus efficacement possible. Dans cette section, nous présentons différentes stratégies de branchement étudiées dans le cadre du *problème du voyageur de commerce*. Nous commencerons par étudier trois stratégies de branchement classiques. Nous finirons par trois stratégies de branchement alternatives.

4.3.1 *Stratégies de branchement classiques*

La première stratégie étudiée est une stratégie de branchement dynamique courante en *programmation par contraintes* : *mindom*. Cette stratégie consiste à sélectionner la variable ayant le plus petit domaine. Dans le cadre du *problème du voyageur de commerce*, les variables comparées sont les variables $next$ ainsi que les variables $pred$. Lorsque la variable x ($next_i$ ou $pred_i$) est sélectionnée, nous devons lui affecter une valeur dans le domaine de D_x. Pour cela, nous décidons de sélectionner la valeur j dont le coût associé $w(i,j)$ est le plus faible. Ceci correspond au nœud le plus proche de i. Cette stratégie de branchement est notée *MinDom*.

La deuxième stratégie de branchement étudiée est une stratégie de branchement développée par Pesant et al. dans [Pes+98]. L'idée de cette heuristique est de sélectionner la variable qui, dans son domaine, possède les valeurs qui apparaissent le plus de fois dans le domaine des variables non affectées. Pour sélectionner la variable, les règles suivantes sont appliquées :

1. Soit s la taille du plus petit domaine des $next$ et $pred$. Soit $\mathcal{V} = \{next_i \mid |D_{next_i}| = s, \forall i \in \mathcal{N}\} \cup \{pred_i \mid |D_{pred_i}| = s, \forall i \in \mathcal{N}\}$,
2. Si $|\mathcal{V}| = 1$, nous choisissons la variable contenue dans $\mathcal{V}$,
3. Sinon
   a) Pour chaque valeur e dans $\cup_{v \in \mathcal{V}} D_v$, calculer $e^\sharp$ le nombre d'apparition de e dans le domaine des variables de $\mathcal{V}$,
   b) Choisir la variable qui maximise $f(v) = \sum_{e \in D_v} e^\sharp$.

Lorsque la variable est sélectionnée, nous décidons d'affecter la valeur dont le nœud correspond au coût le plus faible. Cette stratégie de branchement est notée *Pesant*.

La troisième stratégie de branchement étudiée consiste à choisir la variable correspondant à la valeur précédemment sélectionnée. Ce schéma de branchement étend un chemin partant du nœud 0. Après avoir affecté $next_i$ à j, la prochaine variable sélectionnée est $next_j$ afin de prolonger le chemin en cours. La valeur choisie est le nœud le plus proche. Cette stratégie de branchement est notée *Path*.

Le Tableau 7 récapitule les différentes stratégies de branchement.

	Sélection variables	Sélection valeurs
Heuristique 1	*MinDom*	Client le plus proche
Heuristique 2	*Pesant*	Client le plus proche
Heuristique 3	*Path*	Client le plus proche

Tableau 7 – Récapitulatif des stratégies de branchement

4.3.2 *Stratégies de branchement alternatives*

Nous présentons dans cette section trois stratégies de branchement alternatives. Elles s'appuient sur les stratégies de branchement étudiées précédemment.

4.3.2.1 *nogoods*

Nous proposons d'améliorer la stratégie *Path* en enregistrant des *nogoods* (Section2.4.3.2). Un *nogood* est une affectation partielle qui ne peut être étendue à une solution réalisable. L'enregistrement de *nogoods* peut permettre d'éviter l'exploration redondante de sous-arbres de recherche. La stratégie de branchement *Path* représente un chemin partiel depuis le nœud 0. Cette affectation est définie par :

— $S \subset \mathcal{N}$ l'ensemble de clients visités sur ce chemin,
— $k \in S$ le dernier nœud parcouru,
— w le coût atteint au dernier nœud,
— t le temps indiquant le service au plus tôt du nœud k.

On caractérise donc une affectation partielle de l'heuristique *Path* par le quadruplet (S, k, w, t). Si l'affectation (S_1, k_1, w_1, t_1) est un *nogood*, alors il est inutile d'essayer d'étendre toutes affectations (S_2, k_2, w_2, t_2) telles que :

$$S_2 = S_1,\ k_2 = k_1,\ w_2 \geqslant w_1 \ \wedge\ t_2 \geqslant t_1$$

Sachant que (S_1, k_1, w_1, t_1) est un *nogood*, il est sans intérêt d'explorer un chemin partiel contenant les mêmes nœuds S_1, terminant au même nœud k_1 et arrivant plus tard que t_1 tout en ayant un coût plus élevé.

Les *nogoods* sont enregistrés au backtrack de la stratégie de branchement *Path*, c'est-à-dire à chaque fois où le parcours de l'arbre conduit à un échec ou à l'obtention d'une solution. De plus, à partir du dernier nœud i sur le chemin partiel, le nœud $j \in D_{next_i}$ est éliminé si le chemin partiel est prouvé irréalisable par un *nogood* connu.

Exemple 4.3.1. *Reprenons l'exemple précédent et supposons que lors de la recherche nous avons stocké le* nogood *suivant, Figure 29a :*

— $S_1 = \{0, 1, 2, 3\}$,
— $k_1 = 3$,
— $w_1 = 6$,
— $t_1 = 6$

La recherche continue et nous arrivons à l'état suivant, Figure 29b :

— $S_2 = \{0, 2, 1, 3\}$,
— $k_2 = 3$,
— $w_2 = 9$,
— $t_2 = 9$

Il est alors inutile de continuer la recherche puisque nous avons déjà un nogood *$((S_1, k_1, w_1, t_1)$ de meilleure qualité.*

(a) *nogood* enregistré

(b) État du problème lors de la recherche

FIGURE 29 – Exemples de *nogoods*

4.3.2.2 *Stratégies de branchement guidées par la relaxation lagrangienne*

Nous étudions aussi des stratégies de branchement guidées par la relaxation lagrangienne utilisée dans certains filtrages de la WEIGHTEDCIRCUIT. L'idée de ces stratégies de branchement est d'utiliser le support de la dernière solution obtenue grâce aux relaxations lagrangiennes et de sélectionner le nœud ayant la somme de ses coûts réduits la plus élevée. Les stratégies de branchement *MinDom* et *Pesant* sont alors étendues.

Pour la stratégie de branchement *MinDom*, lorsque plusieurs variables possèdent le plus petit domaine, nous départageons les variables en calculant la somme de leurs coûts réduits dans le dernier support de la relaxation lagrangienne. La variable possédant la somme la plus élevée est alors sélectionnée. Nous affectons alors la valeur dont l'arc possède le plus faible coût réduit. Cette stratégie de branchement est notée *MinDomLag*.

Nous procédons de la même façon pour *Pesant*. Lors du choix de la variable dans l'heuristique *Pesant* (étape b), nous devons sélectionner la variable qui maximise la fonction $f(v)$ (Section 4.3.1). Si plusieurs variables existent, alors la variable ayant la somme des coûts réduits la plus élevée est sélectionnée. Nous lui affectons ensuite la valeur dont le coût réduit de l'arc correspondant est le plus faible. Cette stratégie de branchement se nomme *PesantLag*.

	Sélection variables	Sélection valeurs
NoGoods	*Path* + enregistrement de *nogoods*	Client le plus proche
MinDomLag	*MinDom* + maximum des sommes des coûts réduits	Client dont l'arc formé possède le plus faible coût réduit
PesantLag	*Pesant* + maximum des sommes des coûts réduits	Client dont l'arc formé possède le plus faible coût réduit

Tableau 8 – Récapitulatif des stratégies de branchement alternatives

Le Tableau 8 récapitule les différentes extensions des stratégies de branchement.

4.4 RÉSULTATS NUMÉRIQUES

Cette section présente les résultats numériques obtenus pour différents benchmarks du *TSP* et *TSPTW*. Nous commencerons par une présentation des outils et instances utilisés pour réaliser ces tests. Ensuite, nous aborderons les différents modèles proposés (Section 4.1) puis une comparaison des différentes WEIGHTEDCIRCUIT étudiées (Section 4.2). Enfin, les différentes stratégies de branchement (Section 4.3) seront comparées.

4.4.1 *Machine et benchmarks*

Les tests sont lancés sur un seul processeur sur un ordinateur Intel(R) Xeon(R) CPU, 2,27GHz avec 20 Go de mémoire vive avec Windows 7 comme système d'exploitation. Le langage utilisé pour l'implémentation des différentes études est le langage *c++ 11*. Nous utilisons, par ailleurs, la bibliothèque *ortools 4.3* [OPF14] pour la *programmation par contraintes*.
Les instances prises en compte sont des instances académiques pour le *TSP* et le *TSPTW*. Les instances pour le *TSP* proviennent de la TSPLIB. Nous considérons seulement les problèmes symétriques et asymétriques avec un nombre de nœuds inférieur à cent. Nous avons alors 28 instances pour le *TSP* symétrique et 13 instances pour le *TSP* asymétrique. Nous décidons de classifier les différentes instances suivant leurs nombres de nœuds. Ainsi, nous notons TSP< et ATSP< les instances possédant au plus 50 nœuds et TSP> et ATSP> les instances ayant un nombre de nœuds supérieur à 50.

Les instances utilisées pour le *TSPTW* proviennent de la base de données de Manuel López-Ibáñez et Christian Blum [1]. Nous avons alors 110 instances symétriques de [Dum+95] séparées suivant la taille des problèmes : 25 problèmes de taille 20 (noté D20), 25 problèmes de taille 40 (noté D40), 25 problèmes de taille 60 (noté D60), 20 problèmes de taille 80 (noté D80) et 15 problèmes de taille 100 (noté D100). Nous utilisons aussi 30 problèmes symétriques de [PB96] (noté SPO) ainsi que 27 problèmes symétriques de [Pes+98] (SPE). Les 70 instances de [Lan+93] ont été utilisées et séparées suivant leurs nombres de nœuds. On note L< pour 40 instances dont le nombre de nœuds est inférieur à 50, noté L> pour les 30 autres instances. Pour finir, nous avons testé sur les 39 instances asymétriques de [Asc95], dont 30 problèmes avec un nombre de nœuds inférieur à 50 (noté AFG<) et 9 problèmes restants (noté AFG>).

L'intérêt principal est d'évaluer l'impact des différents niveaux de filtrage proposés par les modèles ou les algorithmes de filtrage de la WEIGHTEDCIRCUIT. Par ailleurs, l'efficacité des algorithmes de filtrage qui s'appuient sur les coûts augmente lorsque la borne supérieure est proche de la borne inférieure. Ainsi, pour chaque instance, la borne supérieure de l'objectif correspond à la meilleure solution connue, cette méthode est proposée dans [Ben+12]. Nous imposons aussi un temps limite de calcul de 10 minutes.

4.4.2 *Récapitulatif des résultats numériques*

Notre étude se décompose en trois parties. Dans un premier temps, nous allons comparer les trois modèles étudiés dans la Section 4.1 : le modèle de base (Section 4.1.1) noté $(\mathbf{MO}_{27})$ ainsi que les deux modèles redondants. L'un des modèles redondants est fondé sur l'encodage du graphe de précédence par des variables booléennes (Section 4.1.2.1) noté $(\mathbf{MO}_{30})$. L'autre modèle redondant s'appuie sur des variables positions d'un nœud dans le tour (Section 4.1.2.2) noté $(\mathbf{MO}_{31})^{+}$.

Dans un deuxième temps, nous analysons les différentes relaxations utilisées pour la contrainte WEIGHTEDCIRCUIT étudiée Section 4.2. Notons AP la relaxation utilisant le problème d'affectation (Section 4.2.3), HK la relaxation utilisant les *1-arbres* (Section 4.2.1) et NP la relaxation utilisant les plus courts chemins (Section 4.2.2).

Dans un troisième temps, nous étudions les différentes stratégies de branchement vues Section 4.3.

1. http ://iridia.ulb.ac.be/ manuel/tsptw-instances

Afin de rendre le chapitre lisible, nous décidons de reporter l'ensemble des résultats dans les Annexes D, E et F. Ainsi, des extraits de résultats numériques seront utilisés afin d'appuyer nos propos dans les sections suivantes. Par ailleurs, les meilleures valeurs sont reportées en gras. De plus, l'analyse des résultats se décompose en deux parties. Nous étudions d'abord l'apport des différents modèles ou WEIGHTEDCIRCUIT au nœud racine, c'est-à-dire après la propagation initiale. Pour cela, nous étudions la valeur de la borne inférieure de l'objectif, colonne (**%Opti**), c'est-à-dire l'écart entre la borne inférieure et la borne supérieure. L'écart est calculé comme suit : $100\frac{\overline{z}-\underline{z}}{\overline{z}}$. Nous étudions aussi le gain apporté aux domaines des variables $next$ (notée **Next**), $position$ (notée **Pos**) et $start$ (notée **Start**) pour les instances de *TSPTW*. Ce gain est exprimé en pourcentage de la réduction du domaine des variables après la propagation.

La résolution sur les différents problèmes est ensuite comparée. Pour cela, nous devons retrouver la solution optimale (valeur optimale fournie par la borne supérieure) et prouver l'optimalité dans le temps limite imparti. Nous notons le nombre de problèmes dont l'optimum a été prouvé dans le temps limite imparti. La colonne notée **CPU** (resp. **Branche**) correspond au temps de calcul en seconde (resp. nombre de branches parcourues) du parcours de l'arbre de recherche.

Quand cela est nécessaire, nous notons Moy la moyenne des résultats, EcTy la valeur de l'écart type des résultats et Méd la valeur médiane des résultats.

	TSP sym	TSP asym	TSPTW sym	TSPTW asym
Modèles	$(\mathbf{MO}_{31})^+$	$(\mathbf{MO}_{31})^+$	$(\mathbf{MO}_{31})^+$	$(\mathbf{MO}_{31})^+$
Filtrage des WEIGHTEDCIRCUIT	HK	HK AP	NPath	NPath AP
Stratégies de branchement	MinDom Pesant	MinDom Pesant	MinDom Pesant	MinDom Pesant

Tableau 9 – Récapitulatif des comparaisons des résultats numériques

Le Tableau 9 récapitule les différentes comparaisons effectuées dans ce chapitre ainsi que les principales observations. Dans la section suivante, nous allons montrer que le choix du modèle utilisant les variables position est un bon compromis entre temps de résolution et filtrage. Ensuite, nous verrons, Section 4.4.4, que suivant les caractéristiques des problèmes, il sera parfois plus intéressant d'utiliser tel ou tel type de filtrage. Enfin, Section

4.4.5, nous montrons que les stratégies de branchement efficaces restent celles de la littérature.

4.4.3 *Comparaison des différents modèles*

Dans cette section, nous analysons les résultats numériques obtenus pour les différents modèles proposés Section 4.1.

4.4.3.1 *Propagation initiale*

Commençons par étudier l'apport des différents modèles après la propagation initiale. Le Tableau 10 propose des extraits des Tableaux 26, 27 et 28 en Annexe D.

Benchs	Modèle	%Opti			Next		
		Moy	EcTy	Méd	Moy	EcTy	Méd
TSP<	(all)	31.76	7.54	28	4.17	1.83	4.11
ATSP<	(all)	42.02	35.29	23.85	5.01	7.15	2.5
D60	$(\mathbf{MO}_{27})$	38.05	9.36	37.73	45.08	**7.16**	44.7
	$(\mathbf{MO}_{31})^+$	25.93	9	**25.31**	80.68	11.09	84.15
	$(\mathbf{MO}_{30})$	**25.61**	**8.77**	**25.31**	**81.73**	10.14	**84.85**
AFG<	$(\mathbf{MO}_{27})$	9.49	6.82	6.59	27.67	**14.65**	25.01
	$(\mathbf{MO}_{31})^+$	**4.87**	2.95	**3.74**	45.44	21.57	45.24
	$(\mathbf{MO}_{30})$	4.9	**2.92**	**3.74**	**45.81**	21.3	**45.72**

Tableau 10 – Extrait de résultats numériques pour la comparaison des modèles au noeud racine

Pour les problèmes de type *TSP* symétrique et asymétrique, nous nous apercevons que quel que soit le modèle, il n'y a pas de différence de filtrage. En effet, les raisonnements apportés par les modèles de précédence jouent un rôle important lorsque nous sommes en présence de fenêtres de temps.

Pour les instances de *TSPTW*, les modèles de précédences $(\mathbf{MO}_{30})$ et $(\mathbf{MO}_{31})^+$ sont plus efficaces en termes de propagation. En effet au nœud racine, la borne inférieure de l'objectif est nettement plus proche de sa borne supérieure pour les deux modèles pré-cités. De plus, les domaines des variables $next$ sont aussi davantage réduits grâce à ces modèles.

Cependant, il y a une légère différence entre les deux modèles puisque le modèle utilisant les variables booléennes $(\mathbf{MO}_{30})$ est en général légère-

ment plus efficace que le modèle utilisant les variables position $(\mathbf{MO}_{31})^+$. Ce dernier évite l'utilisation de l'ordre de $\mathcal{O}(n^3)$ contraintes par rapport au modèle $(\mathbf{MO}_{30})$, sans toutefois perdre beaucoup de raisonnement. Par ailleurs, ce modèle permet parfois d'être plus proche de l'optimum comme pour les instances AFG< grâce à l'ajout de filtrages supplémentaires donnés par l'approche de type ordonnancement (Section 4.1.2.3).

4.4.3.2 *Résolution*

Dans cette section, nous allons comparer les trois modèles présentés lors du parcours de l'arbre de recherche ainsi que les stratégies de branchement. Les Tableaux de l'Annexe D correspondants sont les Tableaux 29, 30 et 31. Nous présentons seulement les problèmes de *TSP* dont le nombre de nœuds est inférieur à 50 car aucun des modèles ne prouve l'optimalité pour les problèmes de taille supérieure. Par ailleurs, nous ne représentons pas les problèmes L< car pour les trois modèles, l'optimum des 40 problèmes est facilement prouvé.

Nous pouvons d'ores et déjà conclure, d'après les tableaux de l'Annexe D, que les stratégies de branchement efficaces sont les stratégies *MinDom* et *Pesant* car pour tous les modèles l'optimum a été trouvé et prouvé sur davantage d'instances.

Le Tableau 11 propose des extraits des tableaux précédents avec les moyennes des différentes colonnes associées.

En ce qui concerne la différence entre les modèles, le modèle de base $(\mathbf{MO}_{27})$ prouve l'optimum sur un plus grand nombre de problèmes pour les problèmes de *TSP* symétrique et asymétrique. Du fait de la complexité temporelle des raisonnements proposés par les modèles de précédences $(\mathbf{MO}_{30})$ et $(\mathbf{MO}_{31})^+$, nous ne pouvons pas parcourir suffisamment de branches dans l'arbre de recherche dans le temps imparti.

En moyenne le nombre de branches parcourues est bien inférieur à celui du modèle de base dans le temps limite. Il en est de même entre le modèle $(\mathbf{MO}_{30})$ et le modèle $(\mathbf{MO}_{31})^+$ puisque le modèle $(\mathbf{MO}_{30})$ qui utilise seulement les variables booléennes propose un raisonnement plus fort que le modèle mixte qui utilise les variables booléennes et positions.

Pour les problèmes de *TSPTW*, l'apport des raisonnements sur les précédences devient important. Les modèles $(\mathbf{MO}_{30})$ et $(\mathbf{MO}_{31})^+$ prouvent l'optimum dans beaucoup plus de problèmes que le modèle de base qui ne propose pas de raisonnements propres liés aux fenêtres de temps.
Pour la même raison que pour les problèmes de *TSP*, le modèle utilisant les

Benchs	Modèle	Preuve	CPU (s)	Branches
		Moy	Moy	Moy
TSP<	$(\mathbf{MO}_{27})$	**6.67**	**336.38**	1753575.00
	$(\mathbf{MO}_{31})^+$	6.00	377.92	591231.31
	$(\mathbf{MO}_{30})$	5.33	398.07	**288152.71**
ATSP<	$(\mathbf{MO}_{27})$	**1.33**	**523.23**	3067165.96
	$(\mathbf{MO}_{31})^+$	1.00	568.43	1058543.67
	$(\mathbf{MO}_{30})$	0.33	593.92	**545023**
D60	$(\mathbf{MO}_{27})$	4.00	533.28	1216652.40
	$(\mathbf{MO}_{31})^+$	**5.67**	**497.39**	337198.57
	$(\mathbf{MO}_{30})$	5.33	505.23	**233099.43**
AFG<	$(\mathbf{MO}_{27})$	11.67	390.61	3624961.77
	$(\mathbf{MO}_{31})^+$	**17**	**298.07**	1224442.52
	$(\mathbf{MO}_{30})$	16.33	302.27	**1085042.01**

Tableau 11 – Extrait de résultats numériques pour la comparaison des modèles lors de la recherche

variables pos parcourt plus de branches lors du temps imparti par rapport au modèle $(\mathbf{MO}_{30})$.

4.4.4 *Comparaison des relaxations de la* WEIGHTEDCIRCUIT

Dans cette section, nous comparons les différentes relaxations utilisées pour la contrainte WEIGHTEDCIRCUIT. Le modèle utilisé est le modèle $(\mathbf{MO}_{31})^+$ car il offre un bon compromis en termes de propagation et de temps de résolution. Les résultats numériques sont répertoriés dans l'Annexe E. Nous trouvons dans cette annexe le Tableau 33 pour les instances de *TSP* après la propagation initiale ainsi que le Tableau 34 pour les instances de [Dum+95] et le Tableau 35 pour le *TSPTW*.

Le Tableau 12 présente des extraits des résultats numériques pour les instances de *TSP* après la propagation initiale.

Pour tous les problèmes de *TSP*, la valeur obtenue par la relaxation d'Held et Karp (HK) est beaucoup plus proche de l'optimum que les autres relaxations, en particulier pour les problèmes symétriques.

Cependant, pour les problèmes asymétriques, nous nous apercevons d'un éloignement à la valeur optimum. De plus, pour les problèmes de plus de 50 nœuds, la valeur de la relaxation *n-path* est beaucoup plus proche de

Benchs	WC	%Opti			Next		
		Moy	EcTy	Méd	Moy	EcTy	Méd
TSP<	AP	17.18	4.77	17.68	22.01	14.24	19.91
	HK	**0.21**	**0.45**	**0.04**	**89.74**	**3.49**	**90.68**
	NP	3.47	2.75	3.47	68.01	18.8	71.65
ATSP<	AP	30.42	42.21	7.42	44.86	**25.81**	56.86
	HK	**13.47**	30.82	**0.58**	21.5	26.68	**4.59**
	NP	17.48	**30.25**	4.1	**58.8**	29	70.92

Tableau 12 – Extraits des résultats numériques pour le *TSP* au nœud racine

Benchs	WC	%Opti			Next		
		Moy	EcTy	Méd	Moy	EcTy	Méd
D60	AP	10.28	8.86	10.51	80.95	**11.35**	84.34
	HK	5.96	6.29	4.66	82.06	11.9	84.34
	NP	**3**	**3.45**	**2.24**	**85.6**	12.38	**86.54**
AFG<	AP	0.79	1.14	0.22	75.06	**12.26**	73.8
	HK	4.54	2.98	4.22	45.21	21.6	43.78
	NP	**0.45**	**0.73**	**0.2**	**76.75**	13.76	**77.48**

Tableau 13 – Extraits des résultats numériques pour le *TSPTW* au nœud racine

celle des autres relaxations.

Pour les problèmes symétriques, le filtrage utilisant Held and Karp permet une réduction significative des domaines des variables $next$. Cependant, la tendance s'inverse lorsque les problèmes sont asymétriques. Les deux autres relaxations permettent alors une plus forte réduction.

La relaxation qui utilise le problème d'affectation ne jouit pas de l'efficacité proposée par l'utilisation des relaxations lagrangiennes. Par conséquent, elle reste plus faible mais obtient de meilleures bornes inférieures pour les problèmes asymétriques.

Le Tableau 13 présente des extraits des résultats numériques pour les instances de *TSPTW* après la propagation initiale.

Pour les problèmes de *TSPTW*, le filtrage proposé avec la relaxation *n-path* est plus efficace que les autres filtrages étudiés. En effet, dans la plupart des instances, nous sommes en moyenne plus proche de l'optimum

avec une réduction des domaines des variables plus élevée.

Le filtrage proposé par la relaxation d'Held et Karp reste efficace par rapport au problème d'affectation.

Cependant, pour les problèmes asymétriques (AFG), ce dernier devient moins efficace et le filtrage qui utilise le problème d'affectation s'avère aussi performant que la relaxation utilisant les plus courts chemins.

4.4.5 *Comparaison des stratégies de branchement*

Dans cette section, nous analysons les différentes stratégies de branchements étudiées Section 4.3.1. Nous comparons aussi les différentes relaxations utilisées. L'annexe correspondant aux résultats est l'Annexe D avec les Tableaux 36, 37, 38, 39, 40 et 41.

Le Tableau 14 présente des extraits des résultats numériques pour les instances de *TSP* lors du parcours de l'arbre de recherche.

Quels que soient les problèmes et grâce à la contrainte WEIGHTEDCIRCUIT, la solution optimale est trouvée et prouvée pour un plus grand nombre de problèmes.

Comme pour la comparaison des modèles, les stratégies de branchement les plus efficaces sont les stratégies *MinDom* et *Pesant*.

Pour le *TSP*, la relaxation d'Held et Karp est beaucoup plus efficace en termes de résolution que la relaxation *n-path*.

Par ailleurs, pour chaque relaxation, les stratégies alternatives permettent de prouver un plus grand nombre de problèmes à l'optimum. En moyenne, un problème de plus est prouvé à l'optimum par rapport aux stratégies de branchement de base. En outre, grâce aux stratégies de branchement alternatives, nous arrivons à parcourir plus de branches en moyenne dans le temps imparti.

Le Tableau 15 présente des extraits des résultats numériques pour les instances de *TSPTW* lors du parcours de l'arbre de recherche.

La contrainte WEIGHTEDCIRCUIT qui utilise le filtrage *n-path* est beaucoup plus performante que celle qui utilise le filtrage d'Held et Karp pour les problèmes du voyageur de commerce avec fenêtres de temps. En effet, pour les problèmes de 60 nœuds (D60), nous arrivons à prouver plus de problèmes à l'optimum en utilisant *n-path* que Held et Karp.

Benchs	WC	Strat	Preuve	CPU (s)	Branches
				Moy	Moy
TSP<	AP	Path	9	270.72	96888.36
		MinDom	**13**	147.71	19884.21
		Pesant	12	**139.28**	**18506.14**
	HK	Path	13	44.07	34268.5
		NoGoods	13	43.87	34911.6
		MinDom	13	43.58	**8716.14**
		MinDomLag	13	**43.16**	14261.86
		Pesant	13	43.49	9346
		PesantLag	13	**43.16**	17945.64
	NP	Path	11	173.33	103336.93
		NoGoods	11	166.26	147521.29
		MinDom	11	112.47	15650.36
		MinDomLag	**12**	**99.51**	**14965.36**
		Pesant	**12**	111.30	21705.43
		PesantLag	**12**	99.87	29855.43
ATSP<	AP	Path	4	319.73	258308
		MinDom	**7**	126.05	33446
		Pesant	**7**	**112.61**	**29574**
	HK	Path	5	226.39	140672.63
		NoGoods	5	412.3	291458.86
		MinDom	**6**	206.54	**93170.63**
		MinDomLag	**6**	**189.37**	95398.38
		Pesant	5	245.26	112823.13
		PesantLag	**6**	199.93	93590.38
	NP	Path	5	326.52	146536.00
		NoGoods	6	**244.12**	**10681.38**
		MinDom	5	309.19	47265.25
		MinDomLag	**7**	310.67	43665.25
		Pesant	5	259.56	50670.00
		PesantLag	6	322.48	43960.00

Tableau 14 – Extrait des résolutions pour les instances de *TSP* symétrique et asymétrique

Benchs	WC	Strat	Preuve	CPU (s)	Branches
				Moy	Moy
D60	AP	Path	11	375.16	36475.12
		MinDom	**13**	**348.48**	31022.64
		Pesant	**13**	348.51	**30114.68**
	HK	Path	9	402.21	208548.72
		NoGoods	9	420.38	246485.33
		MinDom	**10**	385.13	162869.12
		MinDomLag	**10**	390.63	203685.36
		Pesant	**10**	**377.73**	**141614.12**
		PesantLag	**10**	401.15	150394.2
	NP	Path	15	300.88	59399.16
		NoGoods	15	313.22	61834.53
		MinDom	**17**	241.20	42854.36
		MinDomLag	**17**	255.91	45468.48
		Pesant	**17**	**239.27**	**37086.48**
		PesantLag	**17**	249.08	38607.03
AFG<	AP	Path	26	83.68	25466.47
		MinDom	**28**	58.28	11317.77
		Pesant	**28**	**49.28**	**9642.33**
	HK	Path	18	280.19	164616.47
		NoGoods	18	277.63	297670.21
		MinDom	18	245.96	**116509.93**
		MinDomLag	18	264.31	188414.07
		Pesant	**19**	**226.02**	184699.63
		PesantLag	**19**	264.87	194512.53
	NP	Path	27	93.11	43515.13
		NoGoods	27	75.02	40690.53
		MinDom	27	65.55	32145.67
		MinDomLag	27	**60.82**	**31087.27**
		Pesant	27	80.96	47314.63
		PesantLag	27	60.88	411084.07

Tableau 15 – Extrait des résolutions des instances de *TSPTW* symétrique et asymétrique

Par ailleurs, il est intéressant de noter que pour les problèmes avec un grand nombre de nœuds, les stratégies de branchement *Path* et *NoGoods* semblent efficaces avec la WEIGHTEDCIRCUIT *n-path*.

Une forte progression est constatée pour les problèmes *TSPTW* asymétriques (AFG) avec la contrainte WEIGHTEDCIRCUIT utilisant les plus courts chemins.

Par ailleurs, la contrainte WEIGHTEDCIRCUIT qui utilise le filtrage du problème d'affectation est aussi efficace par rapport au filtrage défini avec la relaxation d'Held et Karp.

CONCLUSION

Nous avons d'abord étudié trois modèles en *programmation par contraintes* pour le *problème du voyageur de commerce avec fenêtres de temps*. L'un des modèles est un modèle intermédiaire entre le modèle de base et le modèle booléen permettant un filtrage efficace tout en gardant une complexité acceptable. Ensuite, nous nous sommes concentrés sur la contrainte WEIGHTEDCIRCUIT en présentant différents filtrages s'appuyant sur les relaxations étudiées dans le chapitre précédent. Enfin, il était nécessaire de définir des stratégies de branchement afin d'explorer au mieux l'arbre de recherche.

Nous avons vu que les différents filtrages proposés pour la contrainte WEIGHTEDCIRCUIT influaient sur la résolution selon le type de problèmes. En effet, pour les problèmes avec peu de relations de précédence, il est préférable d'utiliser le filtrage qui utilise Held et Karp. A l'inverse, pour des problèmes avec de fortes relations temporelles, l'utilisation du filtrage avec la *n-path* est beaucoup plus efficace. Par ailleurs, dans les filtrages d'Held et Karp et de *n-path*, nous résolvons le dual lagrangien par un processus de sous-gradient. Nous pensons que l'utilisation de méthode de bundle [Lem01] permettrait de résoudre plus efficacement le dual lagrangien. En effet, cela diminuerait le nombre d'itérations nécessaires à la résolution tout en garantissant la valeur optimale du dual lagrangien.

5

APPROCHES CP POUR LE PROBLÈMES DE TOURNÉES DE VÉHICULES AVEC FENÊTRES DE TEMPS

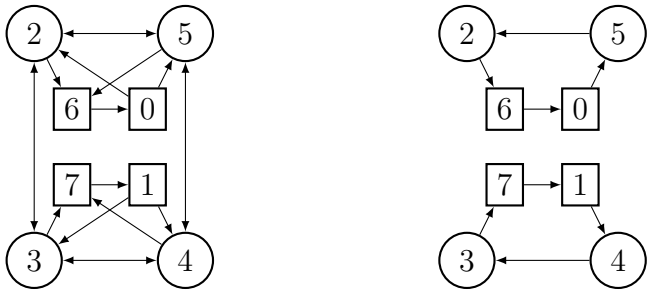
Dans ce chapitre, nous étudions le *problème de tournées de véhicules avec fenêtres de temps et capacité* (*CVRPTW*) (Section 2.3) en *programmation par contraintes*. Nous définissons d'abord le modèle utilisé ainsi que les contraintes redondantes. Nous abordons, ensuite, le *problème de tournées de véhicules* comme un problème de *TSP* afin d'utiliser les raisonnements décrits dans le chapitre précédent. Enfin, nous étudions différentes stratégies de branchement dans le cadre du *CVRPTW*.

5.1 MODÈLES

Soit un ensemble $\mathcal{M}$ de m véhicules et un ensemble $\mathcal{N}$ de n clients. Chaque client i a une demande q_i et chaque véhicule possède une capacité maximale C. Notons $[a_i, b_i]$ la fenêtre de temps d'un nœud i. Le principe du *problème de tournées de véhicules avec fenêtres de temps* est de satisfaire toutes les demandes des clients sans dépasser la capacité maximale d'un véhicule tout en respectant les fenêtres de temps des clients et en minimisant la distance totale parcourue.

Comme pour le *TSP*, les nœuds de début et de fin de tournée d'un véhicule sont dupliqués. Nous définissons ainsi $\mathcal{M} = \{o_1, \ldots, o_m\}$ l'ensemble des nœuds de départ des véhicules et $\mathcal{M}' = \{f_1, \ldots, f_m\}$ l'ensemble des nœuds de fin des véhicules. Soit $\mathcal{V} = \mathcal{M} \cup \mathcal{N} \cup \mathcal{M}'$ l'ensemble des nœuds du graphe. Soit $\mathcal{G}_o = (\mathcal{V}, \mathcal{A})$ le graphe sur lequel le *problème de tournées de véhicules avec fenêtres de temps* est résolu. $d_{i,j}$ (resp. $t_{i,j}$) représente la distance (resp. le temps) entre le nœud i et le nœud j.

Un exemple de graphe est donné Figure 30a. L'exemple montre un graphe composé de 4 clients ainsi que deux véhicules. Ces derniers sont chacun décomposés par leurs nœuds de départ (nœuds $o_1 = 0$ et $o_2 = 1$) et leurs nœuds d'arrivée (nœuds $f_1 = 6$ et $f_2 = 7$). La Figure 30b montre une solution possible. Le premier véhicule distribue le nœud 5 puis le nœud 2.



(a) Exemple de graphe pour le *CVRPTW* avec 4 clients et deux véhicules

(b) Exemple de solution

FIGURE 30 – Exemple de modélisation pour le *CVRPTW*

Le second véhicule visite le nœud 4 puis le nœud 3 avant de terminer sa tournée.

5.1.1 *Modèle de base*

Le modèle de base en *programmation par contraintes* s'appuie sur les variables $next_i$ pour chaque nœud $i \in \mathcal{V}$. Ces variables représentent le successeur immédiat du nœud i dans la tournée.

— $next_i \in \mathcal{N} \cup \mathcal{M}'$, $\forall i \in \mathcal{M} \cup \mathcal{N}$,

— $next_{f_r} = o_r$, $\forall r \in \{1, \ldots, m\}$.

Chaque nœud de fin d'une tournée est connecté à son nœud de début de tournée. Nous définissons aussi les variables prédécesseurs $pred_i$ pour chaque nœud i.

— $pred_i \in \mathcal{M} \cup \mathcal{N}$, $\forall i \in \mathcal{N} \cup \mathcal{M}'$,

— $pred_{o_r} = f_r$, $\forall r \in \{1, \ldots, m\}$.

Les variables $start_i$ pour chaque nœud i représentent le temps accumulé le long du trajet :

— $start_i \in [a_i, \ldots, b_i]$, $\forall i \in \mathcal{N} \cup \mathcal{M}'$,

— $start_i = 0$, $\forall i \in \mathcal{M}$.

En outre, des variables d'accumulation de quantité $load_i$ sont introduites pour chaque nœud i.

— $load_i \in [0, \ldots, C]$, $\forall i \in \mathcal{N} \cup \mathcal{M}'$,

— $load_i = 0$, $\forall i \in \mathcal{M}$.

Le modèle de base est le suivant :

$$
\begin{aligned}
(\mathbf{MO}_{39}): \quad \textit{Min} \quad & z && && (39a)\\
& z = \sum_{i \in \mathcal{M} \cup \mathcal{N}} (d_{i,next_i}) = \sum_{i \in \mathcal{N} \cup \mathcal{M}'} (d_{pred_i,i}) && && (39b)\\
& \textsc{alldifferent}(\{next_i | \forall i \in \mathcal{V}\}) && && (39c)\\
& \textsc{inverse}(next, pred) && && (39d)\\
& start_{next_i} \geqslant start_i + t_{i,next_i} && \forall i \in \mathcal{M} \cup \mathcal{N} && (39e)\\
& load_{next_i} = load_i + q_i && \forall i \in \mathcal{M} \cup \mathcal{N} && (39f)
\end{aligned}
$$

L'équation 39b définit l'objectif à minimiser. Elle correspond à la somme des distances utilisées pour les tournées. La contrainte 39e représente l'accumulation du temps de trajet le long des tournées et évite ainsi les sous-tours. La contrainte 39f permet de mettre à jour les variables de quantité le long des tournées.

5.1.2 *Modèle redondant*

Afin d'améliorer le modèle, des variables redondantes sont ajoutées :
- pos_i, variables positions pour chaque nœud $i \in \mathcal{V}$
  - $pos_i \in \{1, \ldots, n+1\}$, $\forall i \in \mathcal{N} \cup \mathcal{M}'$,
  - $pos_i = 0$, $\forall i \in \mathcal{M}$.
- $route_i$, variables d'appartenance d'un nœud $i \in \mathcal{V}$ à une tournée.
  - $route_i \in \{1, \ldots, m\}$, $\forall i \in \mathcal{N}$,
  - $route_{o_r} = route_{f_r} = r$, $\forall r \in \{1, \ldots, m\}$.
- $card_r$, variables du nombre de clients d'une tournée $r \in \mathcal{M}$.
  - $card_r \in \{0, \ldots, n\}$, $\forall r \in \mathcal{M}$.

Les contraintes suivantes sont alors ajoutées au modèle ($\mathbf{MO}_{39}$) :

$$(\mathbf{MO}_{40}): \quad (\mathbf{MO}_{39}) \tag{40a}$$

$$pos_{next_i} = pos_i + 1 \qquad \forall i \in \mathcal{N} \tag{40b}$$

$$pos_{pred_i} = pos_i - 1 \qquad \forall i \in \mathcal{N} \cup \mathcal{M}' \tag{40c}$$

$$pos_j > pos_i + 1 \Rightarrow next_i \neq j \qquad \forall i \in \mathcal{N} \tag{40d}$$

$$next_i = j \Rightarrow route_i = route_j \qquad \forall (i,j) \in \mathcal{A} \tag{40e}$$

$$\sum_{r \in \mathcal{M}} (card_r) = n \tag{40f}$$

$$\text{GCC}(\{route_i | \forall i \in \mathcal{N}\}, \{1, \ldots, m\}, \{card_r | \forall r \in \mathcal{M}\}) \tag{40g}$$

$$\text{BINPACKING}(\{load_r | \forall r \in \mathcal{M}'\}, \{route_i | \forall i \in \mathcal{V}\}, \{q_i | \forall i \in \mathcal{V}\}) \tag{40h}$$

Les contraintes 40b et 40c permettent de mettre à jour les variables positions pos. Nous pouvons éliminer des valeurs des variables next grâce aux positions des nœuds (contrainte 40d). Lorsqu'un nœud i possède un successeur j, alors ces nœuds font partie de la même tournée (contrainte 40e). Tous les nœuds doivent être visités, ainsi la somme du nombre de clients des tournées est égale à n (contrainte 40f).

La contrainte GCC (40g) ou GLOBALCARDINALITY permet à chaque valeur prise par les variables route d'apparaitre exactement le nombre de fois que la valeur de la variable card, représentant le nombre de clients dans la tournée. Par exemple, si $card_{o_1} = 3$, alors exactement trois variables route doivent prendre la valeur 0. Cette contrainte globale est présentée dans [OMT89].

La contrainte 40h permet de borner les variables load des nœuds de fin de tournée suivant les demandes des clients ainsi que leur affectation dans les tournées correspondantes. Cette contrainte globale dérive de la contrainte CUMULATIVE [AB93] utilisée pour des problèmes d'ordonnancement. Ainsi, les véhicules peuvent être vus comme des *bin* avec une capacité maximum (variable load) à ne pas dépasser. L'affectation des clients aux routes doit être compatibles avec les capacités.

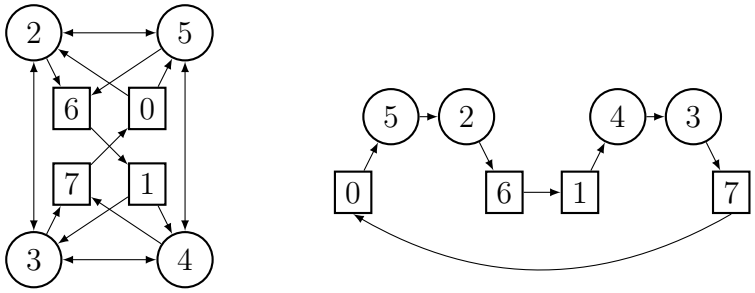
5.2 UTILISATION DU MODÈLE TSP

Dans cette section, nous allons étendre le modèle vu précédemment afin d'utiliser les résultats obtenus pour le *TSP* (Chapitre 4), le but étant d'obtenir, grâce à une modélisation différente du *CVRPTW*, une borne in-

férieure globale. Grâce à cette modélisation, les différents filtrages de la contrainte WEIGHTEDCIRCUIT étudiés précédemment peuvent être appliqués.

5.2.1 *Modélisation chromosomique*

Le *CVRPTW* peut être modélisé grâce à la modélisation dite chromosomique. Celle-ci a été très largement utilisée dans les algorithmes génétiques pour résoudre des problèmes de tournées de véhicules [Prio4 ; Vid+14]. Elle permet un encodage facile et une comparaison des coûts simplifiée.
Pour ce faire, les nœuds $f_r \in \mathcal{M}'$ de fin de tournée peuvent être connectés à la tournée suivante et donc au véhicule suivant. Dès lors, les variables $next_{f_r}$ pour les nœuds de fin de tournée sont instanciées au nœud de début de tournée suivante. Le principe est alors de trouver un circuit hamiltonien de poids minimum dans ce graphe.



(a) Exemple de modélisation chromosomique fondé sur la Figure 30a

(b) Exemple de solution pour la modélisation chromosomique

FIGURE 31 – Exemple de modélisation chromosomique pour le *CVRPTW*

La Figure 31a montre un exemple de modélisation chromosomique à partir de la Figure 30a. Les nœuds de fin de tournée (nœuds 6 et 7) sont connectés aux nœuds de début de la tournée suivante. Un exemple de solution est présenté Figure 31b.

L'intérêt de cette modélisation en *PPC* est de pouvoir poser une contrainte unique de WEIGHTEDCIRCUIT afin de calculer une borne inférieure globale pour le problème.

5.2.2 *Modèle étendu*

Le modèle ($\mathbf{MO}_{40}$) est donc étendu en utilisant le modèle chromosomique. Nous avons en conséquence les variables supplémentaires suivantes :

- Variables successeur :
  - $next_i^{tsp} \in \mathcal{N} \cup \mathcal{M}',\ \forall i \in \mathcal{M} \cup \mathcal{N}$
  - $next_{f_r}^{tsp} = o_{r+1},\ \forall r \in \{1, \ldots, m-1\}.$
  - $next_{f_m}^{tsp} = o_1$
- Variables prédécesseur :
  - $pred_i^{tsp} \in \mathcal{M} \cup \mathcal{N},\ \forall i \in \mathcal{N} \cup \mathcal{M}'$
  - $pred_{o_1}^{tsp} = f_m$
  - $pred_{o_r}^{tsp} = f_{r-1},\ \forall r \in \{2, \ldots, m\}$
- Variables position :
  - $pos_i^{tsp} \in \{1, \ldots, n + 2m - 1\},\ \forall i \in \mathcal{V} \backslash \{0\}$
  - $pos_{o_1}^{tsp} = 0$

Les contraintes suivantes sont ajoutées au modèle ($\mathbf{MO}_{40}$) :

$$(\mathbf{MO}_{41}) : \quad (\mathbf{MO}_{40}) \tag{41a}$$

$$\textsc{weightedcircuit}(\{next_i^{tsp} | \forall i \in \mathcal{V}\}, z) \tag{41b}$$

$$\textsc{inverse}(next^{tsp}, pred^{tsp}) \tag{41c}$$

$$\textsc{alldifferent}(\{pos_i^{tsp} | \forall i \in \mathcal{V}\}) \tag{41d}$$

Nous pouvons alors utiliser la contrainte WEIGHTEDCIRCUIT (41b) présentée précédemment (Section 4.2). Ceci permettra d'obtenir une borne inférieure sur z globale au problème. Nous retrouvons la relation entre les variables $next^{tsp}$ et les variables $pred^{tsp}$ (contrainte 41c). Dans ce modèle, toutes les valeurs des variables positions pos^{tsp} doivent être différentes (contrainte 41d).

Il faut de plus connecter les variables du modèle *TSPTW* aux variables du modèle de *CVRPTW*. Les contraintes suivantes sont ajoutées :

$$(\mathbf{MO}_{42}) : \quad (\mathbf{MO}_{41}) \tag{42a}$$
$$next_i^{tsp} = next_i \qquad \forall i \in \mathcal{M} \cup \mathcal{N} \tag{42b}$$
$$pred_i^{tsp} = pred_i \qquad \forall i \in \mathcal{N} \cup \mathcal{M}' \tag{42c}$$
$$pos_{f_r}^{tsp} = pos_{o_r}^{tsp} + card_{o_r} + 1 \qquad \forall r \in \{1, \dots, m\} \tag{42d}$$
$$pos_i = pos_i^{tsp} - pos_{route_i}^{tsp} \qquad \forall i \in \mathcal{N} \cup \mathcal{M}' \tag{42e}$$
$$route_i < route_j \Rightarrow pos_i^{tsp} < pos_j^{tsp} \qquad \forall (i,j) \in \mathcal{A} \tag{42f}$$

Les variables $next$ et $next^{tsp}$ sont équivalentes pour les nœuds de début de tournée et les nœuds de visite (contrainte 42b), seules les variables $next$ des nœuds de fin de tournée doivent être modifiées. Nous retrouvons l'équivalence symétrique pour les variables $pred$ et $pred^{tsp}$ (contrainte 42c).

La contrainte 42d permet de mettre à jour les variables positions des nœuds de fin de tournée et de début de tournée. En effet, dans la modélisation chromosomique, la position du nœud de fin de tournée est la somme de la position du nœud de début de tournée et du nombre de nœuds associés à la tournée (variables $card$).

Il est possible aussi de connecter les variables positions des deux modèles (contrainte 42e). Pour cela, la contrainte ELEMENT sur les variables pos^{tsp} et $route$ permet d'identifier la position du nœud de début de la route considérée dans le modèle chromosomique. Ainsi, la position locale (dans le modèle *CVRPTW*) d'un nœud dans le tour est égale à sa position dans le modèle chromosomique moins la position du nœud de début de tournée.

Puisque le modèle chromosomique ordonne les différents nœuds de départ et de fin, il est alors possible d'ordonner les positions dans le modèle chromosomique suivant les variables routes afin de casser les symétries (contrainte 42f).

5.3 STRATÉGIES DE BRANCHEMENT

Dans cette section, nous allons voir les stratégies de branchement mises en place pour le *CVRPTW*. Pour ces problèmes, nous devons classer les solutions trouvées. En effet, nous imposons que les premières routes possèdent au moins autant de clients que les suivantes. Nous obtenons donc la relation suivante :

$$card_{o_r} \geqslant card_{o_{r+1}} \qquad \forall r \in \{1, \dots, m-1\} \tag{43}$$

	C1	C2	RC1	RC2	R1	R2
25 nœuds	9	8	8	8	12	11
50 nœuds	9	8	8	7	12	9
100 nœuds	9	8	6	3	10	1

Tableau 16 – Nombre de problèmes testés pour chaque instance

Deux stratégies de branchement sont utilisées pour le *CVRPTW* :

— La première stratégie de branchement consiste à utiliser la stratégie de branchement *Pesant* (Section 4.3.1),

— La seconde stratégie de branchement affecte d'abord les variables `card` puis utilise la stratégie de branchement *Pesant*. Pour affecter les variables `card`, nous décidons de sélectionner la première variable non bornée et de lui affecter la plus petite valeur de son domaine.

5.4 RÉSULTATS NUMÉRIQUES

Nous présentons dans cette section les résultats numériques obtenus pour des instances de *CVRPTW*. Nous utilisons les mêmes outils que pour le chapitre précédent. Les instances testées sont les instances de *CVRPTW* présentées par Solomon dans [Sol87]. Les instances sont composées de 25 clients, 50 clients et 100 clients. Chaque groupe d'instances se décompose en trois sous-groupes : le sous-groupe C concerne les clients qui sont géographiquement regroupés. Le sous-groupe R indique que les clients sont placés géographiquement de façon aléatoire. Le sous-groupe RC est un mélange des deux groupes précédents. Pour chaque sous-groupe, il existe deux variétés de problèmes : les problèmes avec un horizon de temps court (les problèmes C1, R1, RC1) et les problèmes avec un grand horizon de temps (les problèmes C2, R2, RC2).

De la même façon que pour les résultats numériques du chapitre précédent, nous étudions d'abord les résultats après la propagation initiale et ensuite la résolution des différentes instances. Le Tableau 16 récapitule le nombre de problèmes testés pour les différents groupes de benchmarks de Solomon [Sol87].

5.4.1 *Propagation initiale*

Nous présentons dans cette section les résultats numériques pour les différents benchmarks après la propagation initiale.

Benchs	%Opti			Next			Position		
	Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
C1	13.42	11.92	18.08	54.85	35.45	33.16	45.70	41.65	8.22
C2	18.41	20.40	8.09	60.65	18.54	56.10	27.67	34.14	5.01
RC1	28.44	3.92	27.72	23.22	11.92	22.47	9.20	1.47	8.22
RC2	22.06	5.84	23.80	22.45	13.06	17.39	9.78	3.72	8.22
R1	19.34	5.57	20.69	22.09	13.98	14.56	16.07	4.59	14.10
R2	11.07	3.75	10.01	25.76	13.11	18.05	7.52	2.31	8.22

Tableau 17 – Propagation initiale *CVRPTW* 25 nœuds

Benchs	%Opti			Next			Position		
	Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
C1	29.20	19.93	33.55	52.44	37.47	30.17	38.58	33.93	8.04
C2	7.54	2.10	8.52	49.69	16.19	52.35	14.94	19.87	4.07
RC1	45.43	7.10	46.38	27.89	13.90	28.43	11.02	1.73	10.03
RC2	33.67	4.88	34.20	17.73	8.58	19.17	6.91	1.30	8.04
R1	25.22	9.26	30.43	24.58	15.21	17.00	13.23	3.33	12.01
R2	17.82	5.48	17.24	15.44	7.46	12.62	6.28	1.62	6.06

Tableau 18 – Propagation initiale *CVRPTW* 50 nœuds

Les Tableau 17, 18 et 19 présentent les résultats obtenus pour les instances de [Sol87] 25, 50 et 100 clients après la propagation initiale.

Pour les problèmes clusterisés (C1 et C2), l'écart à l'optimum est plus proche que pour les autres problèmes. Cependant, nous restons assez loin de la valeur optimum (en moyenne 10 %). De plus, pour ce type de problème, il est intéressant de voir que nous arrivons à filtrer en moyenne la moitié des arcs du problème. Étonnamment, nous sommes plus proche de l'optimum pour les problèmes où les clients sont placés aléatoirement (les problèmes R) par rapport aux problèmes mixtes (RC). En outre, il semblerait que les problèmes avec un horizon de temps court (les problèmes 1) soient plus difficiles à résoudre. Ceci peut s'expliquer par le fait qu'ils possèdent un plus grand nombre de véhicules par rapport aux problèmes 2. Nous ne présentons pas les instances R1 et RC1 pour les problèmes avec 100 nœuds car pour ces instances nous ne parvenons pas à terminer la propagation initiale de toutes les contraintes dans le temps imparti.

Benchs	%Opti			Next			Position		
	Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
C1	38.86	28.56	40.70	64.18	35.62	91.06	45.21	36.27	36.01
C2	6.41	1.60	7.30	53.20	16.87	54.87	14.12	21.18	2.02
RC2	49.29	25.36	37.81	45.30	27.35	30.52	30.26	34.87	7.51
R2	12.09	0.00	12.09	36.01	0.00	36.01	7.01	0.00	7.01

Tableau 19 – Propagation initiale *CVRPTW* 100 nœuds

Benchs	Strat	Preuve	CPU (s)			Branches		
			Moy	EcTy	Méd	Moy	EcTy	Méd
C1	Pesant	5	271.36	292.34	39.77	1249.33	1333.04	232.00
	Card	9	159.11	172.06	83.02	651.00	769.11	274.00
C2	Pesant	5	15.75	15.80	7.05	39.38	38.81	17.50
	Card	4	27.10	27.90	12.21	80.25	83.88	37.00
RC1	Pesant	1	600.21	0.11	600.23	6665.38	5366.81	3492.00
	Card	4	462.33	189.76	600.05	6795.13	5099.41	4855.50
RC2	Pesant	4	477.96	183.43	600.12	3092.88	2238.78	2234.00
	Card	5	248.02	176.11	174.60	5196.75	7329.31	857.00
R1	Pesant	2	551.09	90.18	600.16	2775.75	1177.58	2563.00
	Card	5	550.93	90.37	600.11	5732.00	2503.17	4965.50
R2	Pesant	5	482.12	172.22	600.24	3342.18	1834.74	2708.00
	Card	7	467.00	169.72	600.08	3379.45	1472.51	3859.00

Tableau 20 – Résolution des instances de *CVRPTW* 25 nœuds

5.4.2 *Résolution*

Nous présentons dans cette section les résultats numériques pour les différentes instances lors du parcours de l'arbre de recherche.

Les Tableaux 20, 21 et 22 présentent les résultats obtenus pour les instances de [Sol87] 25, 50 et 100 clients pour la preuve d'optimalité.

Nous pouvons conclure que la stratégie de branchement consistant à instancier en premier les variables `card` est beaucoup plus efficace que la stratégie de branchement *Pesant*. Les résultats restent faibles même pour les problèmes avec 25 nœuds. Il est étonnant de remarquer la différence de résolution entre les problèmes où les clients sont placés aléatoirement (les problèmes R) par rapport aux problèmes mixtes (RC). En effet, la méthode semble plus efficace pour les problèmes R.

Benchs	Strat	Preuve	CPU (s)			Branches		
			Moy	EcTy	Méd	Moy	EcTy	Méd
C1	Pesant	3	338.14	292.72	600.21	163.67	120.81	210.00
	Card	4	337.52	292.67	600.18	326.56	275.16	403.00
C2	Pesant	2	486.51	172.92	600.48	400.75	151.13	452.00
	Card	3	526.68	131.49	600.65	331.00	85.75	364.00
RC1	Pesant	0	600.66	0.36	600.52	519.25	168.50	496.00
	Card	0	600.73	0.63	600.32	1022.63	466.28	899.50
RC2	Pesant	0	601.08	0.83	600.74	456.57	132.08	440.00
	Card	0	600.37	0.25	600.25	3040.14	2219.31	1871.00
R1	Pesant	1	599.34	2.61	600.50	332.15	136.50	285.00
	Card	1	600.60	0.42	600.43	620.69	268.13	457.00
R2	Pesant	0	601.09	0.86	600.71	378.33	86.81	359.00
	Card	0	600.64	0.35	600.45	637.33	126.30	602.00

Tableau 21 – Résolution des instances de *CVRPTW* 50 nœuds

Benchs	Strat	Preuve	CPU (s)			Branches		
			Moy	EcTy	Méd	Moy	EcTy	Méd
C1	Pesant	1	546.23	120.69	607.64	53.22	77.46	4.00
	Card	1	561.96	124.18	623.62	70.33	106.30	2.00
C2	Pesant	1	531.97	132.21	603.43	33.75	16.25	35.50
	Card	1	534.31	132.79	607.88	47.88	14.41	51.00
RC2	Pesant	0	664.70	74.51	620.76	8.50	5.50	9.50
	Card	0	673.78	73.85	635.10	5.50	3.50	6.00
R2	Pesant	0	600.85	0.00	600.85	13.00	0.00	13.00
	Card	0	630.50	0.00	630.50	8.00	0.00	8.00

Tableau 22 – Résolution des instances de *CVRPTW* 100 nœuds

CONCLUSION

Nous avons proposé dans ce chapitre une modélisation en *programmation par contraintes* pour le *problème de tournées de véhicules avec fenêtres de temps*. Cette modélisation est renforcée par l'utilisation d'un modèle chromosomique de *CVRPTW* permettant d'appliquer la contrainte globale WEIGHTEDCIRCUIT étudiée dans le chapitre précédent et d'obtenir une borne inférieure. Les deux stratégies de branchement utilisées sont des stratégies de branchement classiques de la littérature.

Les résultats numériques sont satisfaisants et nous espérons pouvoir les améliorer en proposant une contrainte WEIGHTEDSOUSCIRCUIT appliquée à chaque route.

6

OTSOLVER : PREMIERS TESTS SUR DES DONNÉES INDUSTRIELLES

Dans ce chapitre, nous allons nous concentrer sur le nouvel outil en développement, *OtSolver*, à travers des exemples clients ainsi que la comparaison avec le moteur actuellement opérationnel *TourSolver*. La partie industrielle de cette thèse porte sur la conception d'un moteur de résolution pour des problèmes variés de tournées de véhicules (Chapitre 1). Un modèle unifié s'appuyant sur les graphes permet la représentation des données et intègre des contraintes métiers variées.

Nous présenterons d'abord plus en détail les contraintes prises en compte dans le module de résolution *OtSolver*. Nous effectuerons ensuite une première comparaison expérimentale entre ce nouveau solveur à base de *PPC* et le solveur actuel s'appuyant sur la recherche locale : *TourSolver*.

6.1 CONTRAINTES PRISES EN COMPTE DANS LE MOTEUR DE RÉSOLUTION OTSOLVER

Dans cette section, nous présentons brièvement le fonctionnement du moteur de résolution. Le point d'entrée d'*OtSolver* est le graphe unifié qui représente les données du problème de tournées de véhicules. Actuellement, ce moteur est composé d'un module principal de résolution fondé sur la librairie de *programmation par contraintes ortools* [OPF14]. Cette librairie permet de mettre en place facilement le modèle *PPC* des problèmes de tournées de véhicules. Elle offre aussi des possibilités de résolution efficace par des algorithmes de recherche locale avec ou sans voisinage large. Il est aussi aisé d'intégrer des développements spécifiques tout en garantissant l'intégrité des modèles déjà mis en place. Ainsi, les calculs des bornes inférieures des relaxations étudiées Chapitre 3 sont intégrés en utilisant la modélisation chromosomique (Section 5.2).

Ce moteur de résolution intègre plusieurs contraintes essentielles. Nous différencions les contraintes relatives aux ressources et celles appliquées

sur les visites. Une ressource peut être un véhicule ou une personne selon les applications. Pour les tournées de livraison de biens volumineux ou lourds, ce sera le véhicule et sa capacité qui seront importants. Pour de la distribution de journaux ou des interventions de maintenance, c'est la personne qui sera la ressource. Chaque contrainte est optionnelle. Si l'utilisateur ne définit pas la contrainte alors elle ne sera pas prise en compte. Nous présentons les contraintes liées aux ressources actuellement prises en compte :

- Lieu de départ et de fin d'une tournée,
- Fenêtres de temps : définition de l'heure de début et de fin d'utilisation d'une ressource,
- Coût horaire de travail : permet de définir le coût horaire de travail d'une ressource,
- Coût kilométrique : définit le coût kilométrique d'une ressource,
- Coût d'utilisation : définit le coût d'utilisation de la ressource,
- Liste de capacités : définit la capacité maximale de la ressource pour chaque produit du problème,
- Distance/temps de trajet/ temps de travail maximale : permet de définir des limitations maximales sur la distance, le temps de trajet et le temps de travail,
- Coût de compatibilité : définit une liste de compétences que la ressource possède ainsi qu'un coût associé pour chaque compétence.

Nous présentons les contraintes liées aux clients actuellement prises en compte :

- Liste de fenêtres de temps : définit les différentes fenêtres de temps possibles pour une visite. Un même client peut avoir plusieurs fenêtres de temps distinctes,
- Temps de service : définit le temps de service nécessaire pour effectuer la visite,
- Liste de demandes : définit la demande du client pour chaque produit du problème,
- Coût de non-visite : coût supplémentaire si la visite ne peut pas être effectuée ou qu'il est préférable qu'une entreprise externe fasse la visite,
- Service inclus dans la fenêtre de temps : contrairement à la littérature, cette contrainte impose que la ressource parte (service compris) avant la fin de la fenêtre de temps,
- Affectation/exclusion de ressources : définit une liste de ressources que la visite autorise ou exclut,
- Coût de compatibilité : définit une liste de compétences dont la visite a besoin ainsi qu'un coût associé.

Nom	Nb ressources	Nb clients	Caractéristiques
Italie	6	95	Affectation ressources 1produit
Mexico	5	172	Affectation ressources 1 produit
Belgique	5	65	Plusieurs produits
Espagne	5	123	Multi fenêtres de temps Service inclus dans la fenêtre
France	69	2191	Service time
France agglo	6	601	Service time Livraison de journaux

Tableau 23 – Caractéristiques des instances testées

6.2 INSTANCES CLIENTS ET COMPARAISON

Dans cette section, nous allons décrire différentes instances utilisateurs avec des contraintes différentes. Nous proposons également la comparaison des résolutions du nouveau moteur d'optimisation *OtSolver* et l'un des moteurs actuellement en place *TourSolver*. La version de *TourSolver* est la version 7.0 pour *MapInfo*.

Nous avons six instances utilisateurs avec des caractéristiques diverses. Le Tableau 23 montre les différentes instances testées en indiquant le nombre de ressources, le nombre de clients et leurs caractéristiques. Les instances Italie et Mexico sont assez classiques. Les clients demandent une certaine quantité d'un seul produit. De plus, chaque client est pré-affecté à une ressource du problème. L'instance Belgique est une instance avec plusieurs produits. Chaque ressource possède une capacité maximale différente pour les produits (cela peut être zéro pour certains produits) et les clients ont une demande sur les produits. Le client doit être visité une et une seule fois. Ainsi, si aucun véhicule ne peut satisfaire la demande d'un client, celui-ci ne pourra pas être visité. L'instance Espagne est une instance avec plusieurs fenêtres de temps pour les clients et la contrainte du service inclus dans la fenêtre. Il n'y a pas de capacité, seulement un temps de service pour chaque client. Les instances France sont des instances avec beaucoup de clients. Les clients ont seulement un temps de service, ils n'ont aucune demande ni de fenêtre de temps. Les ressources du problème possèdent une fenêtre de temps. France agglo à la particularité d'être un problème de livraison de journaux. Ce sont des problèmes assez denses et

Nom	Solveur	Nb ressources	Nb clients non visité	Temps de résolution (s)	Coût total	Coût de trajet	Coût de travail
Italie	TourSolver	6	0	144	897	301	596
	OtSolver	6	0	**106**	**887**	**295**	**591**
Mexico	TourSolver	5	0	600	1163	**212**	951
	OtSolver	5	0	**75**	**1089**	218	**871**
Belgique	TourSolver	5	0	600	1681	**1098**	583
	OtSolver	5	0	**81**	**1562**	1107	**457**
Espagne	TourSolver	5	0	600	833	201	632
	OtSolver	5	0	**416**	**819**	**189**	**630**
France	TourSolver	51	0	600	10977	4945	6032
	OtSolver	**47**	0	600	**10491**	**4781**	**5710**
France agglo	TourSolver	6	33	600	6231	52	180
	OtSolver	6	**0**	**472**	**6220**	**42**	**178**

Tableau 24 – Résultats numériques des différentes instances

les clients sont généralement très proches les uns des autres.

Nous avons testé les instances sur le même ordinateur que précédemment. Nous définissons un temps limite de résolution de 10 min mais chaque moteur de résolution peut s'arrêter automatiquement lorsqu'aucune amélioration n'est détectée après un certain nombre d'itérations.

Le Tableau 24 présente les résultats numériques des différentes instances entre les deux solveurs *TourSolver* et *OtSolver*. Sur toutes les instances, le nouveau solveur *OtSolver* donne de meilleurs résultats. Ces derniers sont meilleurs en terme de temps de résolution mais aussi en terme de réduction du coût total. De plus le solveur *OtSolver* réussit à utiliser moins de véhicules pour l'instance France que *TourSolver* tout en diminuant le coût global.

Un des développements d'*OtSolver* consiste à utiliser les bornes inférieures étudiées Chapitre 3. Pour cela, la modélisation chromosomique (Section 5.2) est utilisée afin de pouvoir appliquer le calcul des bornes inférieures.

Nom	Coût total	Bornes inférieures			% Opti
		AP	HK	NP	
Italie	887	707	751	**818**	7.8
Mexico	1089	**1011**	1005	941	7.2
Belgique	1562	1191	1271	**1494**	4.3
Espagne	819	666	675	**716**	12.5
France	10491	**10026**	9356*	-	4.4
France agglo	220	**192**	180*	-	12.6

Tableau 25 – Bornes inférieures des différentes instances obtenues par *OtSolver*

Le Tableau 25 donne les bornes inférieures obtenues au nœud racine pour les différentes instances testées. Notons AP la borne obtenue à partir de la relaxation qui utilise le problème d'affectation. HK représente la borne inférieure en appliquant la relaxation d'Held et Karp et NP celle de la *n-path*. La colonne % Opti donne le meilleur gap obtenu en pourcentage par rapport au coût total. Pour l'instance France agglo, le coût total est égal à la somme du coût de trajet et du coût de travail. En effet, dans l'instance les ressources possèdent un coût fixe d'utilisation.

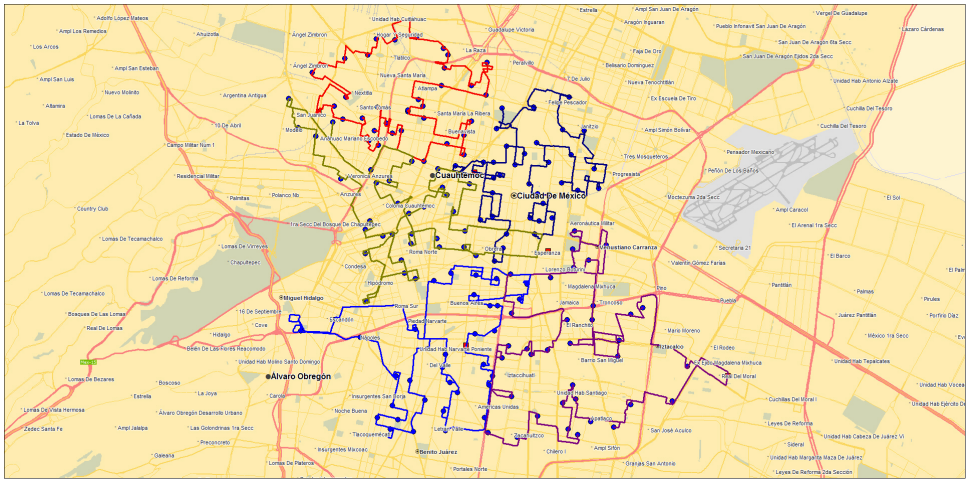
Il existe une grande différence entre les bornes inférieures obtenues. En effet, pour l'instance Belgique, la relaxation *n-path* est beaucoup plus proche de la borne supérieure alors que pour l'instance Mexico c'est la relaxation qui utilise le problème d'affectation qui est plus proche. Malheureusement, pour des problèmes de plus de 500 clients, nous n'arrivons pas à calculer la relaxation utilisant les plus courts chemins. De plus, pour ces problèmes, seule la recherche d'un *1-arbre* est effectué pour obtenir la borne inférieure sans résolution du dual lagrangien. Sur ces instances, nous garantissons un gap par rapport à la borne supérieure entre 4% et 13% .

Nous allons maintenant étudier en détail trois des instances. Ces trois instances représentent diverses problématiques rencontrées par les solveurs actuels et constituent un large éventail d'utilisateurs.

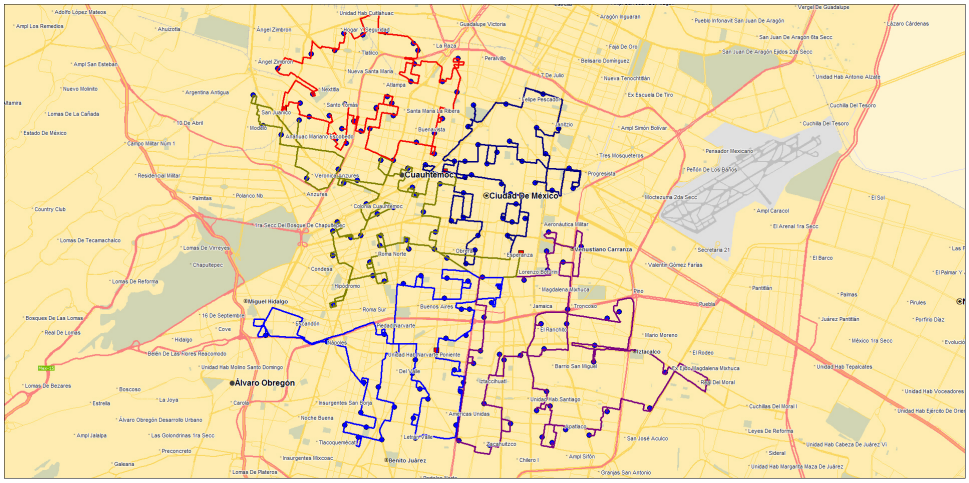
6.2.1 *Instance Mexico*

L'instance Mexico est une instance assez classique où l'utilisateur final dispose d'une flotte de véhicules avec des points de départs différents. Chaque véhicule possède une capacité maximale pour chaque type de produit et doit donc livrer les clients demandant ce produit.

Il est souvent conseillé aux utilisateurs d'utiliser au préalable le logiciel *Territory Manager* afin de répartir l'ensemble des clients en secteurs. Il est alors possible de définir les caractéristiques des secteurs voulus pour une équité entre les différents secteurs suivant plusieurs paramètres, par exemple le chiffre d'affaire réalisé. Cela permet donc d'avoir une affectation des clients aux différents véhicules disponibles. Ainsi lorsque la préaffectation est effectuée nous pouvons lancer l'optimisation de chaque tournée.



(a) Résultat graphique de *TourSolver* pour l'instance Mexico



(b) Résultat graphique de *OtSolver* pour l'instance Mexico

FIGURE 32 – Résultat graphique pour l'instance Mexico

La Figure 32 montre les résultats graphiques sur l'instance Mexico pour les deux solveurs testés *TourSolver* (Figure 32a) et *OtSolver* (Figure

32b). Sur ce type d'instance, comme pour Italie, il y a très peu de différences entre les deux solveurs. Il y a seulement un gain de temps de résolution et un changement d'ordre de passage des clients dans les tournées.

6.2.2 *Instance Belgique*

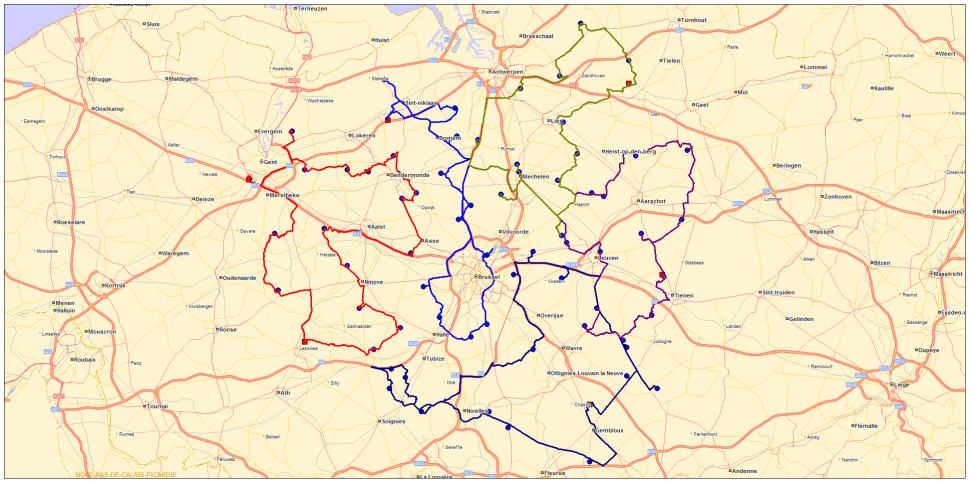
L'instance Belgique ne bénéficie pas de la pré-affectation. La principale caractéristique de cette instance est la possibilité d'avoir plusieurs produits. C'est-à-dire que chaque véhicule possède une quantité maximale possible pour chaque produit. De plus, chaque client peut demander une certaine quantité de chaque produit. Le but est toujours de visiter tous les clients en les livrant une seule fois. Autrement dit, le client ne peut pas être visité deux fois, la première fois pour le premier produit, la seconde fois pour le second produit.

Nous pouvons voir sur les Figures 33 les résultats graphiques de l'instance Belgique pour les deux solveurs *TourSolver* (Figure 33a) et *OtSolver* (Figure 33b). Nous pouvons noter une répartition différente des clients aux véhicules. Le coût total est un peu amélioré pour cette instance, mais nous voyons que le coût de trajet est plus élevé que la solution proposée par *TourSolver*. Cependant, le coût total est balancé avec le coût de travail qui est nettement diminué dans la solution proposée par *OtSolver*.

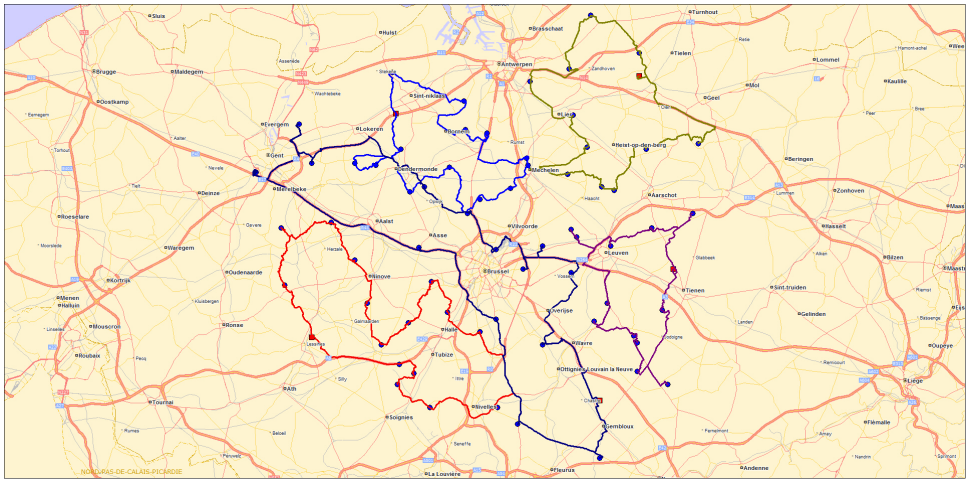
6.2.3 *Instance France agglo*

L'instance France agglo est une instance dite dense. Le problème correspond à de la livraison de journaux. Nous disposons donc de beaucoup de clients (601) avec un temps de service faible (30 secondes à 1 minute) dans un secteur géographique resseré. Les ressources, au nombre de six, doivent visiter un maximum de client entre 5h30 et 7h00.

La Figure 34 montre les résultats graphiques pour l'instance France agglo pour les deux solveurs *TourSolver* (Figure 34a) et *OtSolver* (Figure 34b). Nous résolvons le problème en deux fois moins de temps et en visitant tous les clients. A l'inverse, *TourSolver* ne parvient pas à livrer 33 des 601 clients. Graphiquement, les clients sont très proches les uns des autres et beaucoup de clients se situent dans la même rue.



(a) Résultat graphique de *TourSolver* pour l'instance Belgique

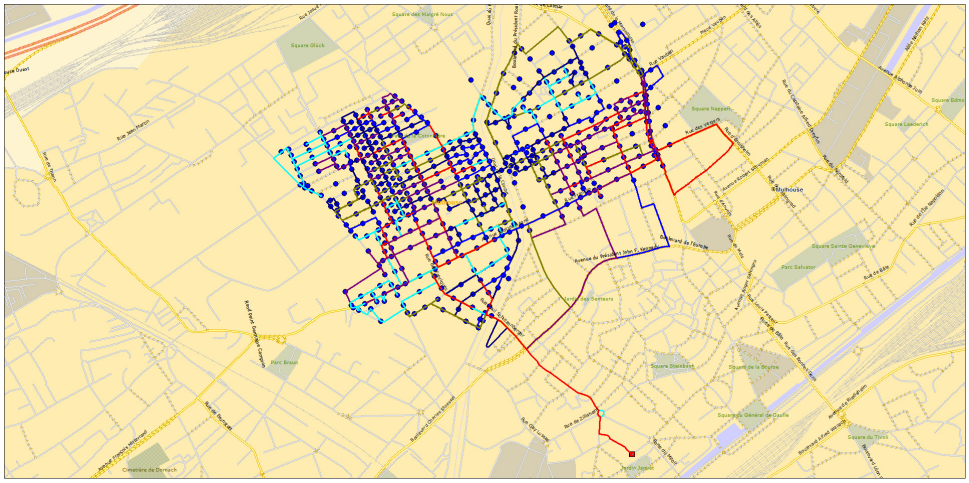


(b) Résultat graphique de *OtSolver* pour l'instance Belgique

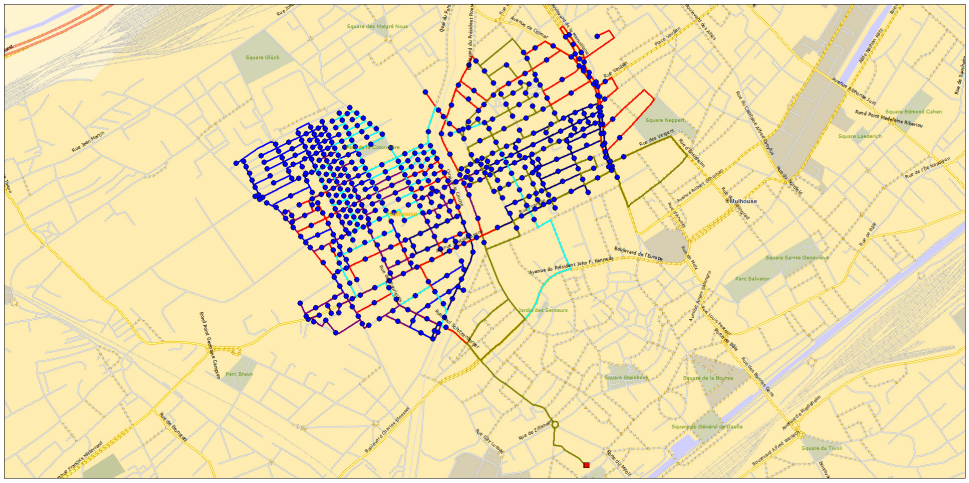
FIGURE 33 – Résultat graphique pour l'instance Belgique

CONCLUSION

Le nouveau moteur de résolution de tournées de véhicules est encore en cours de développement mais nous avons vu qu'il est aisé de pouvoir prendre en compte des contraintes utilisateurs variées grâce au modèle de graphe unifié (Chapitre 1). Les premiers résultats expérimentaux sont très prometteurs face au solveur actuel *TourSolver*. De plus nous avons vu que le nouveau moteur de résolution intègre des parties de développement étudiées dans les chapitres précédents. Il permet, grâce à la modélisation chromosomique, de fournir des bornes inférieures provenant des différentes relaxations étudiées.



(a) Résultat graphique de *TourSolver* pour l'instance France agglo



(b) Résultat graphique de *OtSolver* pour l'instance France agglo

FIGURE 34 – Résultat graphique pour l'instance Franceagglo

Dans les prochaines étapes, il est prévu d'inclure d'autres contraintes et d'intégrer le filtrage de la contrainte WEIGHTEDCIRCUIT pour les instances industrielles. L'intégration complète du calcul des bornes inférieures est perfectible pour les instances de grandes tailles. La relaxation qui utilise la *n-path* doit être approfondie pour une meilleure intégration. De plus, l'arrêt dans un temps limite de la relaxation lagrangienne nous permettrait d'affiner les bornes inférieures.

CONCLUSION

La conception et le développement d'un nouveau moteur de résolution pour des problèmes de tournées de véhicules ont été proposés dans cette thèse (Chapitre 1). Pour représenter un large éventail de problématiques utilisateurs, un modèle unifié s'appuyant sur des graphes est intégré dans le solveur. Ce nouveau moteur s'appuie sur des mécanismes de *programmation par contraintes* et l'utilisation de recherche locale avec ou sans voisinage large pour résoudre au mieux les problèmes industriels.

Nous avons réalisé une étude sur différentes relaxations du *problème du voyageur de commerce* (Chapitre 3). Ces relaxations permettent de fournir des bornes inférieures au *TSP*. De plus, nous avons démontré, à travers des exemples, que les relaxations sont incomparables.

Nous proposons trois modèles en *PPC* ainsi que l'utilisation de la contrainte WEIGHTEDCIRCUIT pour le *TSP* (Chapitre 4). L'étude se concentre sur l'efficacité des différents algorithmes de filtrage proposés pour cette contrainte qui permet de faciliter la résolution. Les résultats expérimentaux démontrent, suivant les caractéristiques des problèmes, la nécessité de sélectionner des filtrages appropriés. En particulier, avec l'utilisation des positions et de la relaxation *n-path*, nous améliorons l'algorithme de filtrage historique qui utilise la relaxation d'Held et Karp pour des problèmes avec des contraintes additionnelles, telles que des fenêtres de temps et des précédences.

Nous avons aussi étudié le *problème de tournées de véhicules avec fenêtres de temps et capacité* en proposant un modèle en *PPC* (Chapitre 5). Ce modèle est renforcé par la modélisation chromosomique permettant d'appliquer la contrainte WEIGHTEDCIRCUIT. Les premiers résultats sont encourageants et montrent l'intérêt d'utiliser la contrainte WEIGHTEDCIRCUIT sur l'ensemble du problème, d'une part pour obtenir une borne inférieure du problème et d'autre part pour éliminer des valeurs infaisables ou sous-optimales dans les domaines des variables.

Le Chapitre 6 a présenté les résultats du nouveau moteur de résolution sur des données réelles. Le solveur fournit par ailleurs les bornes inférieures des différentes relaxations étudiées grâce à la modélisation chromosomique pour le *problème du voyageur de commerce*. Les premiers résultats expérimentaux sont très prometteurs et dominent significativement les résultats obtenus avec le solveur actuel.

Certes, toutes les contraintes utilisateurs ne sont pas encore prises en compte mais le solveur est encore en cours de construction et il est nécessaire de prendre en compte les contraintes les plus communes aux utilisateurs.

Cette thèse s'est déroulée en deux étapes : la première étape consistait à étudier les problèmes fondamentaux de tournées de véhicules, le *TSP* et *CVRPTW*. La seconde étape met en application de façon plus pragmatique les résultats obtenus sur les problèmes fondamentaux pour des cas industriels réels. Ces deux étapes sont complémentaires et peuvent très bien interagir ensemble sans perte de performances, comme par exemple l'utilisation de bornes inférieures informatives dans le solveur. Pour une meilleure intégration, nous mettrons en place les différents filtrages proposés pour la WEIGHTEDCIRCUIT dans le solveur *OtSolver*.

La *programmation par contraintes* offre un cadre de résolution pour des problèmes variés et de grande taille. Nous espérons renforcer l'interaction des techniques de résolutions exactes avec les techniques de recherches locales proposées par la *PPC*. Ainsi, il nous sera possible de mieux informer les utilisateurs sur les solutions obtenues par nos outils d'aide à la décision, à travers des indicateurs tels que les bornes inférieures.

ANNEXES

A

MODÉLISATION DES DIFFÉRENTES CONTRAINTES

A.1 DONNÉES

A.1.1 *Ensemble*

— I^+ : ensemble des visites de collecte
— I^- : ensemble des visites de livraison
— I : ensemble des visites : $I = I^+ \cup I^-$
— V : ensemble des véhicules
— D_v : ensemble des localisations de départ du véhicule $v \in V$
— M_v : ensemble des localisations de fin du véhicule $v \in V$
— K : ensemble des jours de planification
— P_i : ensemble des successeur de la visite $i \in I^+$
— V_i : ensemble des véhicules pouvant visiter le noeud $i \in I$
— K_i : ensemble des jours ouvrés pour la visite $i \in I$

A.1.2 *Caractéristiques des visites*

Voici les différentes caractéristiques liée aux visites $i \in I$.
— q_i : quantité, $q_i \geqslant 0$ si $i \in I^+$ et $q_i \leqslant 0$ si $i \in I^-$
— $\mathtt{duration}_i$: durée de la visite
— $[A_i^{1,k}, B_i^{1,k}]$ et $[A_i^{2,k}, B_i^{2,k}]$: fenêtre de temps forte , $k \in K_i$
— $[a_i^{1,k}, b_i^{1,k}]$ et $[a_i^{2,k}, b_i^{2,k}]$: fenêtre de temps souple, $k \in K_i$ avec $A_i^{1,k} \leqslant a_i^{1,k} \leqslant b_i^{1,k} \leqslant B_i^{1,k}$ et $A_i^{2,k} \leqslant a_i^{2,k} \leqslant b_i^{2,k} \leqslant B_i^{2,k}$
— rc_i : coût de retard

A.1.3 *Caractéristiques des véhicules*

Voici les différentes caractéristiques liées aux véhicules $v \in V$ au jour $k \in K$.
— $c^{k,v}$: capacité du véhicule
— $fc^{k,v}$: coût fixe d'utilisation

- $fcn^{k,v}$: coût fixe de non utilisation
- $d^{k,v}$: distance maximum à parcourir
- $dc^{k,v}$: coût kilométrique
- $t^{k,v}$: temps de conduite maximum
- $[F^{k,v}, G^{k,v}]$: fenêtre de temps
- $W^{k,v}$: temps maximum de travail
- $w^{k,v}$: temps maximum de travail sans heures supplémentaires, $w^{k,v} \leqslant W^{k,v}$
- $wc^{k,v}$: coût horaire de travail
- $oc^{k,v}$: coût des heures supplémentaires
- $[LA^{k,v}, LB^{k,v}]$: fenêtre de temps des pauses déjeuners $[LA^{k,v}, LB^{k,v}]$
- $ld^{k,v}$: durée des pauses déjeuners
- $nc^{k,v}$: coût d'une nuitée à l'extérieur
- $mcno^{v}$: nombre maximum de nuitées à l'extérieur consécutives
- ac^{v} : coût d'attente

A.2 VARIABLES

- $x_{i,j}^{k,v} = 1$ si j est visité après i par v le jour k, 0 sinon
- $z_i^{k,v} = 1$ si i est visité par v le jour k, 0 sinon
- $s_i^{k,v}$: temps de début de la visite i si i est visité par v le jour k
- $wait_i^{k,v}$: temps d'attente du véhicule v avant la visite i.
- $S^{k,v}$: temps de début du tour (k, v)
- $E^{k,v}$: temps de fin du tour (k, v)
- $ls^{k,v}$: temps de début de la pause déjeuner du tour (k, v)
- $lp^{k,v} = 1$ si la pause déjeuner est effectué par le véhicule v le jour k, 0 sinon
- $load_i^{k,v}$: charge du véhicule v après la visite du client i
- $time_i^{k,v}$: temps de trajet cumulé du véhicule v après la visite i le jour k
- $dist_i^{k,v}$: distance parcourue par le véhicule v après la visite i le jour k
- $worktime_i^{k,v}$: temps de travail cumulé du véhicule v après la visite i le jour k
- $NO^{k,v} = 1$ si le véhicule v ne revient pas au dépôt à la fin du jour k et fait une nuitée à l'extérieur, 0 sinon
- $used^{k,v}$: 1 si le véhicule v est utilisé le jour k
- $\Delta_i^{k,v}$ 1 si $A_i^{1,k,v} \leqslant s_i^{k,v} \leqslant C_i^{1,k,v}$, 0 si $A_i^{2,k,v} \leqslant s_i^{k,v} \leqslant C_i^{2,k,v}$

A.3 CONTRAINTES

A.3.1 *Contraintes liées aux visites*

- Toutes les visites doivent être visitées : $\sum_{k\in K}\sum_{v\in V} z_i^{k,v} = 1, \forall i \in I$.
- Respect des fenêtres de temps : $(A_i^{1,k,v} \leqslant s_i^{k,v} \leqslant C_i^{1,k,v})$ et $(A_i^{2,k,v} \leqslant s_i^{k,v} \leqslant C_i^{2,k,v})$, si $z_i^{k,v} = 1$
- Compatibilité des tours (véhicules et/ou jour) : il existe $k \in K$ et $v \in V$ où $z_i^{k,v} = 0$

A.3.2 *Contraintes liées aux véhicules*

- Respect de la capacité : $0 \leqslant load_i^{k,v} \leqslant c_{k,v}$, $\forall v \in V, \forall i \in I \cup D^v, \forall k \in K$,
- Respect des fenêtres de temps : $F^{k,v} \leqslant S^{k,v} \leqslant E^{k,v} \leqslant G^{k,v}$
- Temps de travail maximum : $E^{k,v} - S^{k,v} \leqslant W^{k,v}$.
- Pause déjeuner
  - Respect des fenêtres de temps $LA^{k,v} \leqslant ls^{k,v} \leqslant LB^{k,v})$.
  - Durée non incluse dans le temps de travail si la pause déjeuner à lieu $E^{k,v} - S^{k,v} - ld^{k,v} \leqslant W^{k,v}$
  - La pause déjeuner à lieu si $E^{k,v} \geqslant LB^{k,v}$ et $S^{k,v} \leqslant LA^{k,v}$
- Respect de la distance maximum de conduite : $dist_i^{k,v} \leqslant d^{k,v}$, $\forall i \in I \cup M, \forall k \in K, \forall v \in V$
- Respect du temps de conduite maximum : $time_i^{k,v} \leqslant t^{k,v}$, $\forall i \in I \cup M, \forall k \in K, \forall v \in V$
- Respect du temps de travail maximum : $worktime_i^{k,v} \leqslant W^{k,v}$,$\forall i \in I \cup M, \forall k \in K, \forall v \in V$
- Retour au dépôt (si les nuitées extérieures sont autorisées) si les trois conditions suivantes sont respectées pour la dernière visite i du tour (v, k) :
  - $dist_i^{k,v} + d_{i,M^v} \leqslant d^{k,v}$
  - $time_i^{k,v} + t_{i,M^v} \leqslant t^{k,v}$
  - $worktime_i^{k,v} + t_{i,M^v} \leqslant W^{k,v}$
- Respect du nombre maximum de nuitées extérieures consécutives : $\sum_{k=h}^{h+mcno_v+1} NO^{k,v} \leqslant mcno_v, \forall h \in [1, |K| - mcno_v - 1], \forall v \in V$

A.3.3 *Collecte et livraison*

A.3.3.1 *Problème de tournées de véhicules avec rechargement*

- VRPCB : nous devons satisfaire une catégorie avant l'autre

  - Les collectes se font avant les livraisons : $s_i^{k,v} \leqslant s_j^{k,v}$, $\forall i \in I^+, \forall j \in I^-$
  - Les livraisons se font avant les collectes : $s_i^{k,v} \leqslant s_j^{k,v}$, $\forall i \in I^-, \forall j \in I^+$
- VRPSB : les visites ont une quantité de collecte (q_i^+) et une quantité de livraison (q_i^-) :
- Définition de la demande : $q_i = q_i^+ - q_i^-, \forall i \in I$

A.3.3.2 *Collecte et livraison*

- Toutes les requêtes de collecte et livraison doivent être effectuées par le même véhicule : $\sum_{j \in P_i} z_j^{k,v} = |P_i| z_i^{k,v}$, $\forall i \in D, \forall k \in K, \forall v \in V$
- La collecte se fait avant la livraison : $s_i^{k,v} \leqslant s_j^{k,v}$, $\forall i \in I, \forall j \in P_i, \forall k \in K, \forall v \in V$

A.4 COÛT

- Coût d'utilisation : $fc^{v,k}$ si v est utilisé le jour k, fcn^v sinon
- Coût de distance parcourue : $(\sum_{i \in I \cup D} \sum_{j \in I \cup M} d_{i,j} x_{i,j}^{k,v}) dc^{k,v}$, $\forall k \in K, \forall v \in V$
- Coût d'attente : $wait_i^{k,v} ac^{k,v}$ avec $wait_i^{k,v} = s_i^{k,v} - a_i^{k,v}$
- Coût de retard :
  - Si $C^{1,k,v} \geqslant s_i^{k,v} \geqslant B_i^{1,k,v}$, alors $(a_i^{k,v} - B_i^{1,k,v}) rc_i$
  - Si $a_i^{k,v} \geqslant B_i^{2,k,v}$, alors $(a_i^{k,v} - B_i^{2,k,v}) rc_i$
- Coût horaire de travail : si $(E^{k,v} - S^{k,v} - ld^{k,v} lp^{k,v}) \leqslant w^{k,v}$, alors $(E^{k,v} - S^{k,v}) wc^{k,v}$
- Coût des heures supplémentaires : si $(E^{k,v} - S^{k,v} - ld^{k,v} lp^{k,v}) \geqslant w^{k,v}$, alors $w^{k,v} wc^{k,v} + (E^{k,v} - w^{k,v}) oc^{k,v}$
- Coût des nuitées extérieures : si $NO^{k,v}$, alors $nc^{k,v}$

B

MODÈLE UML COMMUN AUX MODULES DE RÉSOLUTION

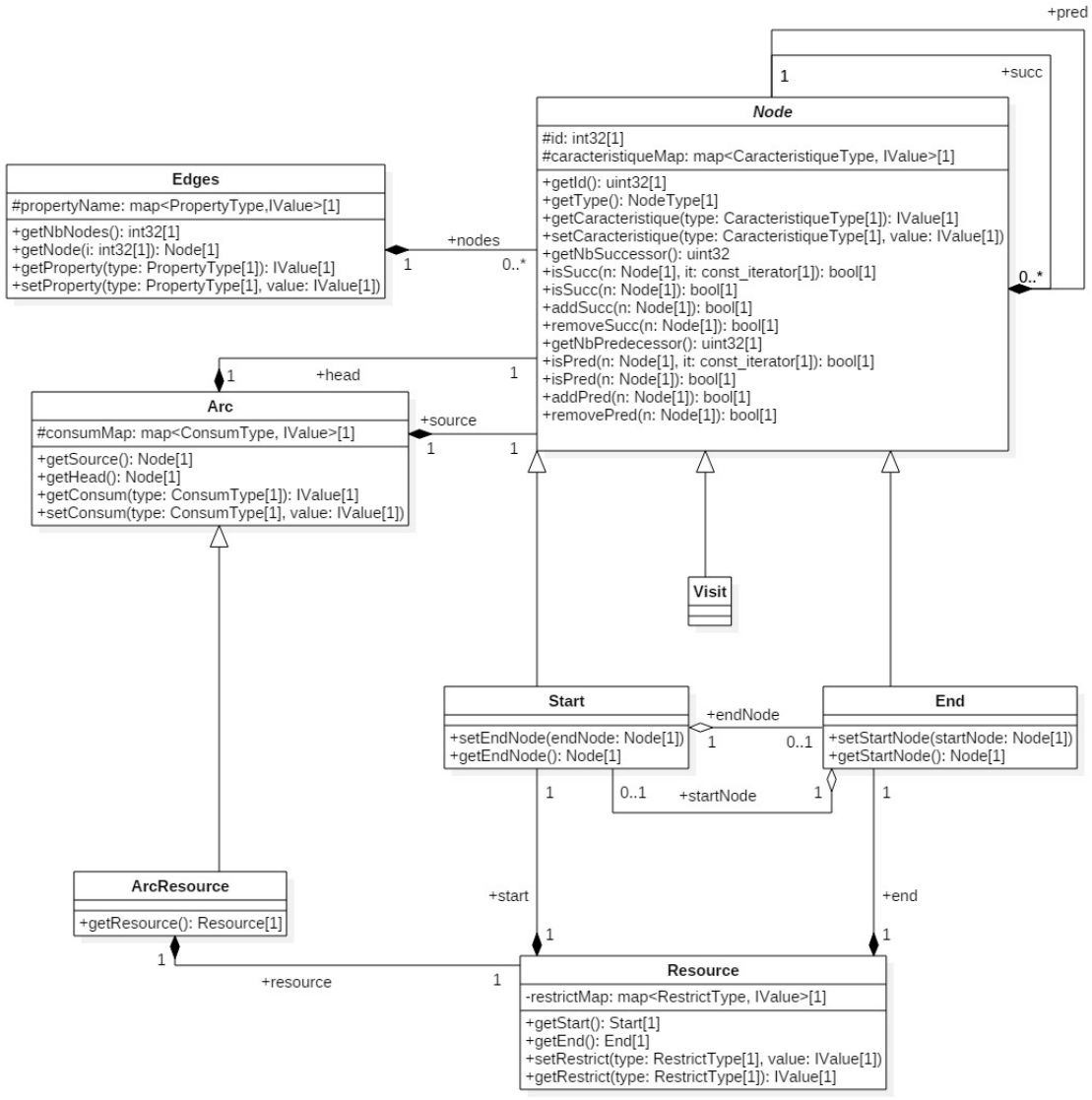


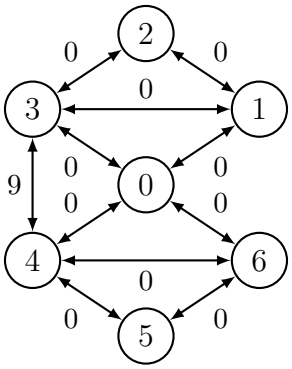
FIGURE 35 – Diagramme de classe UML

C

PREUVE DE L'OBSERVATION 3.2.5 SECTION 3.2.3

Observation : La valeur de la relaxation lagrangienne du sous-problème qui utilise la $n\text{-}path^{\text{no1Circuit}}$ $Max_\lambda(\mathbf{RL}_{18})$ dépend du nœud de départ choisi.

Démonstration. Reprenons le graphe G_4 étudié Section 3.3.2.



(a) Graphe G_4

FIGURE 36 – Graphe G_4

Si le nœud de départ de la relaxation lagrangienne qui utilise la $n\text{-}path^{\text{no1Circuit}}$ est le nœud 0, alors la valeur obtenue est 0. (Section 3.3.2).

Supposons que le nœud de départ est le nœud 5. Le dual lagrangien du programme défini par les équations du programme linéaire ($\mathbf{PL}_{25}$) possède les contraintes suivantes :

$$
\begin{aligned}
z^{rl}_{np} = \max\, & w \\
& w \leqslant 9 && && && && && (44a)\\
& w \leqslant 9 && && && && -2\lambda_4 + 2\lambda_6 && (44b)\\
& w \leqslant 9 && -2\lambda_0 + 2\lambda_1 && +2\lambda_2 && && -2\lambda_6 && (44c)\\
& w \leqslant 9 && -2\lambda_0 + 2\lambda_1 && +2\lambda_2 && -2\lambda_4 && && (44d)\\
& w \leqslant 9 && && +2\lambda_2 && && -2\lambda_6 && (44e)\\
& w \leqslant 9 && && +2\lambda_2 && -2\lambda_4 && && (44f)\\
& w \leqslant 9 && +2\lambda_1 && +2\lambda_2 && -2\lambda_4 && -2\lambda_6 && (44g)\\
& w \leqslant 18 && +2\lambda_0 && && -2\lambda_3 && -2\lambda_4 + 2\lambda_6 && (44h)\\
& w \leqslant 18 && && +2\lambda_2 && -2\lambda_3 && -2\lambda_4 + 2\lambda_6 && (44i)\\
& w \leqslant && -2\lambda_0 && +2\lambda_2 && && -2\lambda_4 + 2\lambda_6 && (44j)\\
& w \leqslant && -2\lambda_0 && +2\lambda_2 && && +2\lambda_4 - 2\lambda_6 && (44k)\\
& w \leqslant && -2\lambda_0 && +2\lambda_2 && && && (44l)\\
& w \leqslant && -2\lambda_0 + 2\lambda_1 && +2\lambda_2 + 2\lambda_3 && -2\lambda_4 - 2\lambda_6 && && (44m)
\end{aligned}
$$

Les différentes contraintes représentent les chemins possibles de 7 arcs commençant par le nœud 5. Il en existe 15 sur le graphe G_4, ce qui conduit à 13 contraintes dans le dual lagrangien. L'équation 44a correspond au tour optimal de coût 9 dans le graphe G_4 $(5-4-3-2-1-0-6-5)$. Les autres équations correspondent à différents plus courts chemins composés de 7 arcs partant du nœud 5. Par exemple, l'equation 44h correspond au chemin $(5-4-3-2-1-3-4-5)$ et l'équation 44e au chemin $(5-6-0-3-4-0-6-5)$. Si $\lambda_2 = 4.5$ et $\lambda_i = 0$, $\forall i \in \{0,1,3,4,6\}$, alors la valeur du dual lagrangien est égale à 9 (coût du tour optimal) si le nœud de départ est 5.

Il est possible donc d'améliorer la relaxation lagrangienne qui utilise la *n-path*$^{\text{no1Circuit}}$ en recherchant la meilleure valeur du nœud de départ. Cependant en pratique, il semble trop coûteux de rechercher le meilleur nœud de départ.

D

RÉSULTATS NUMÉRIQUES COMPLET DES COMPARAISON DES MODÈLES

D.1 PROPAGATION INITIALE

D.1.1 *TSP*

Benchs	Modèle	%Opti			Next		
		Moy	EcTy	Méd	Moy	EcTy	Méd
TSP<	$(\mathbf{MO}_{27})$	31.76	7.54	28	4.17	1.83	4.11
	$(\mathbf{MO}_{31})^{+}$	31.76	7.54	28	4.17	1.83	4.1
	$(\mathbf{MO}_{30})$	31.76	7.54	28	4.17	1.83	4.11
TSP>	$(\mathbf{MO}_{27})$	30.93	7.56	32,8	1.29	0.38	1,04
	$(\mathbf{MO}_{31})^{+}$	30.93	7.56	32,8	1.29	0.38	1,04
	$(\mathbf{MO}_{30})$	30.93	7.56	32,8	1.29	0.38	1,04
ATSP<	$(\mathbf{MO}_{27})$	42.02	35.29	23.85	5.01	7.15	2.5
	$(\mathbf{MO}_{31})^{+}$	42.02	35.29	23.85	5.01	7.15	2.5
	$(\mathbf{MO}_{30})$	42.02	35.29	23.85	5.01	7.15	2.5
ATSP>	$(\mathbf{MO}_{27})$	27.56	10.97	26.64	1.63	0.22	1.56
	$(\mathbf{MO}_{31})^{+}$	27.56	10.97	26.64	1.63	0.22	1.56
	$(\mathbf{MO}_{30})$	27.56	10.97	26.64	1.63	0.22	1.56

Tableau 26 – Comparaison des modèles au nœud racine des instances de *TSP*

D.1.2 *TSPTW*

Benchs	Modèle	%Opti			Next			Start		
		Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
D20	$(\mathbf{MO}_{27})$	32.42	13.96	31.65	48.66	17.43	41.31	24.05	24.64	5.18
	$(\mathbf{MO}_{31})^+$	19.69	12.04	22.87	73.97	15.9	74.04	43.91	20.93	38.34
	$(\mathbf{MO}_{30})$	18.98	11.42	20.54	74.55	15.21	74.49	43.86	20.96	38.34
D40	$(\mathbf{MO}_{27})$	39.85	8.47	42.38	43.94	9.23	41.06	14.61	16.57	41.06
	$(\mathbf{MO}_{31})^+$	26.64	10.32	25.91	77.52	13.14	76.89	36.64	17.21	76.89
	$(\mathbf{MO}_{30})$	26.41	10.05	25.91	78.69	11.91	77.6	36.62	17.17	77.6
D60	$(\mathbf{MO}_{27})$	38.05	9.36	37.73	45.08	7.16	44.7	13.77	14.35	13.22
	$(\mathbf{MO}_{31})^+$	25.93	9	25.31	80.68	11.09	84.15	34.47	16.09	32.22
	$(\mathbf{MO}_{30})$	25.61	8.77	25.31	81.73	10.14	84.85	34.46	16.09	32.22
D80	$(\mathbf{MO}_{27})$	38	10.79	41.72	47.11	6.5	45.28	12.96	14.1	6.94
	$(\mathbf{MO}_{31})^+$	25.56	8.84	26.35	85.97	7.21	87.54	35.16	12.13	32.99
	$(\mathbf{MO}_{30})$	25.31	8.6	26.35	86.7	6.58	87.86	35.13	12.13	32.99
D100	$(\mathbf{MO}_{27})$	35.32	7.45	36.59	49.01	3.63	47.62	18.97	10.48	17.45
	$(\mathbf{MO}_{31})^+$	24.11	6.92	22.77	89.71	5.32	89.9	37.56	12.3	31.78
	$(\mathbf{MO}_{30})$	23.69	6.68	22.77	90.2	4.9	90.42	37.56	12.3	31.78

Tableau 27 – Comparaison des modèles au nœud racine des instances de [Dum+95]

Benchs	Modèle	%Opti			Next			Start		
		Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
SPO	$(\mathbf{MO}_{27})$	27.67	6.42	27.79	21.01	12.35	22.79	1.36	0.93	1.38
	$(\mathbf{MO}_{31})^{+}$	24.97	5.31	25.64	31.73	22.95	26.19	10.49	6.52	7.42
	$(\mathbf{MO}_{30})$	24.64	5.19	25.2	34.34	23.44	28.53	10.46	6.55	7.42
SPE	$(\mathbf{MO}_{27})$	22.2	5.21	21.3	20.95	12.23	22.71	1.18	0.81	0.77
	$(\mathbf{MO}_{31})^{+}$	20.06	4.38	20.38	32	22.34	29.82	10.37	5.71	8.33
	$(\mathbf{MO}_{30})$	19.87	4.21	20.38	35.23	22.58	37.58	10.34	5.75	8.33
L<	$(\mathbf{MO}_{27})$	9.39	8.4	8.29	81.36	13.03	85.36	68.11	16.58	68.69
	$(\mathbf{MO}_{31})^{+}$	3.93	4.68	2.4	94.5	1.87	94.78	78.84	11.04	81.1
	$(\mathbf{MO}_{30})$	3.93	4.68	2.4	94.5	1.87	94.78	78.81	11.08	81.1
L>	$(\mathbf{MO}_{27})$	17.51	6.9	17.46	63.13	8.05	62.73	46.81	12.16	46.18
	$(\mathbf{MO}_{31})^{+}$	8.12	4.48	7.92	95.9	1.38	95.92	66.35	9.94	66.39
	$(\mathbf{MO}_{30})$	8.11	4.48	7.92	95.91	1.37	95.96	66.24	10.1	66.39
AFG<	$(\mathbf{MO}_{27})$	9.49	6.82	6.59	27.67	14.65	25.01	2.69	5.54	0.53
	$(\mathbf{MO}_{31})^{+}$	4.87	2.95	3.74	45.44	21.57	45.24	27.04	24.17	15.73
	$(\mathbf{MO}_{30})$	4.9	2.92	3.74	45.81	21.3	45.72	26.9	23.96	15.73
AFG>	$(\mathbf{MO}_{27})$	10.93	6.89	8.58	40.97	16.88	34.97	2.82	4.42	1.18
	$(\mathbf{MO}_{31})^{+}$	4.79	2.78	4.31	62.28	20.39	56.51	39.35	28.53	24.97
	$(\mathbf{MO}_{30})$	4.67	2.72	4.31	63.3	18.93	56.63	39.35	28.53	24.97

Tableau 28 – Comparaison des modèles au nœud racine des instances de *TSPTW*

D.2 RÉSOLUTION

D.2.1 *TSP*

Benchs	Modèle	Strat	Preuve	CPU (s)			Branches		
				Mo	EcTy	Méd	Mo	EcTy	Méd
TSP<	($\mathbf{MO}_{27}$)	Path	4	451.96	246.79	600	3142106.29	2037950.48	2986875
		MinDom	8	276.97	294.04	133.34	1066110.43	1176053.75	860564.5
		Pesant	8	280.21	291.55	148.02	1052508.29	1112598.7	878338
	($\mathbf{MO}_{31}$)$^+$	Path	4	478.59	220.4	600.02	1138610.29	947133.67	901984
		MinDom	7	324.15	296.34	451.49	277521.43	294212.55	231743
		Pesant	7	331.02	293.93	483.56	357562.21	344010.59	329016.5
	($\mathbf{MO}_{30}$)	Path	4	491.28	208.52	600.04	645918	871438.87	272135
		MinDom	6	350.33	299.85	600.04	85371.71	143059.52	35185.5
		Pesant	6	352.6	297.12	600.04	133168.43	188468.51	71962.5
ATSP<	($\mathbf{MO}_{27}$)	Path	0	600	0	600	4491167.5	3155039.05	3612795
		MinDom	2	494.51	198.4	600	2418879.63	1156689.06	2434915
		Pesant	2	475.17	231.31	600	2291450.75	1096658.14	2156735
	($\mathbf{MO}_{31}$)$^+$	Path	0	600.05	0.02	600.05	1329546.13	1807638.48	787744
		MinDom	1	569.22	87.2	600.05	900934.63	1134097.98	535329
		Pesant	2	536.01	118.64	600.05	945150.25	873196.17	510329.5
	($\mathbf{MO}_{30}$)	Path	0	600.21	0.12	600.22	695046.13	1625879.07	129905.5
		MinDom	0	600.16	0.08	600.18	406774.38	960127.78	66230.5
		Pesant	1	581.39	53.29	600.21	533248.5	833734.69	86709

Tableau 29 – Résolution des instances de *TSP*

D.2.2 *TSPTW*

Benchs	Modèle	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
D40	($\mathbf{MO}_{27}$)	Path	7	442.57	259.66	600.02	2501578.64	1856393.77	2425670
		MinDom	14	286.2	287.31	134.59	710181.68	730885.6	401876
		Pesant	14	284.59	288.4	131.75	693810.72	720050.16	378173
	($\mathbf{MO}_{31}$)$^+$	Path	10	396.36	278.17	600.05	589460.28	437820.15	698662
		MinDom	15	279.36	280.63	172.56	270868.16	295134.43	222247
		Pesant	15	275.95	282.57	176.62	266645.56	291163.31	223073
	($\mathbf{MO}_{30}$)	Path	9	401.24	278.48	600.15	494155.76	391176	487160
		MinDom	15	294.94	279.95	204.76	171205.96	188027.33	135404
		Pesant	15	288.75	281.12	205.24	172669.84	193061.1	163933
D60	($\mathbf{MO}_{27}$)	Path	2	580.4	74.85	600.05	2183906.64	1115674.33	2048990
		MinDom	5	510.41	200.3	600.05	735909.92	394479.32	728595
		Pesant	5	509.04	200.2	600.05	730140.64	370135.62	721816
	($\mathbf{MO}_{31}$)$^+$	Path	5	523.21	186.51	600.15	512301.2	238263.99	556648
		MinDom	6	484.05	222.03	600.14	253542.96	164986.84	234472
		Pesant	6	484.92	219.1	600.15	245751.56	148115.72	234789
	($\mathbf{MO}_{30}$)	Path	4	527.62	183.08	600.57	397499.8	232129.48	391184
		MinDom	6	492.78	217.21	600.57	152880.72	120357.89	105444
		Pesant	6	495.3	212.54	600.56	148917.76	98404.5	120232
D80	($\mathbf{MO}_{27}$)	Path	0	600.09	0.02	600.08	2147387.4	730057.0597	2080830
		MinDom	0	600.1	0.02	600.09	647660.2	216683.3503	608655.5
		Pesant	2	574.19	103.23	600.09	604169.55	208548.1096	548588.5
	($\mathbf{MO}_{31}$)$^+$	Path	1	579.87	91.19	600.26	362269.2	138802.338	345722.5
		MinDom	3	568.72	93.8	600.25	252512.05	97582.66489	243206
		Pesant	2	559.26	128.55	600.27	241883	121795.755	218206.5
	($\mathbf{MO}_{30}$)	Path	1	583.3	80.22	601.16	254788.1	142946.0235	194555
		MinDom	2	580.16	65.14	601.2	162216.35	121299.9254	126958.5
		Pesant	2	562.57	121.14	601.15	153944.05	106901.7628	117316

Tableau 30 – Résolution des instances de [Dum+95]

Benchs	Modèle	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
SPO	$(\mathbf{MO}_{27})$	Path	8	440.2	269.54	600.0085	2754729.47	2817203.96	2191990
		MinDom	16	337.16	277.25	450.7005	1289913.1	1079874.89	1283995
		Pesant	16	336.08	278.15	452.599	1397506.1	1150213.95	1688170
	$(\mathbf{MO}_{31})^{+}$	Path	14	348.46	288.77	600.0165	705210.67	657204.15	735181
		MinDom	15	331.74	282.67	472.062	544159.5	494089.89	521616
		Pesant	16	336.52	282.36	501.1085	600788.47	530593.92	621486.5
	$(\mathbf{MO}_{30})$	Path	14	350.68	290.43	600.0305	513106.13	539594.22	443044
		MinDom	15	335.99	285.86	504.112	367123.3	388741.14	247963.5
		Pesant	14	342.29	289.96	600.0305	448288.2	432987.15	345851
SPE	$(\mathbf{MO}_{27})$	Path	6	485.5	220.3	600.008	2620000.04	1602337.41	2548320
		MinDom	13	327.77	289.58	600.007	1184651.3	1159497.69	743959
		Pesant	13	324.4	292.42	600.007	1206729.07	1178431.38	664117
	$(\mathbf{MO}_{31})^{+}$	Path	11	377.01	280.38	600.019	612388.78	518731.94	717014
		MinDom	13	334.6	286.66	600.018	471986.78	501256.86	395722
		Pesant	13	332.48	286.9	600.019	521921.81	534250.54	445353
	$(\mathbf{MO}_{30})$	Path	11	379.93	279.9	600.039	440863.07	431134.81	429884
		MinDom	13	345.51	285.13	600.036	317373.56	403297.24	245668
		Pesant	13	340.88	284.39	600.037	374400.89	424312.98	272684

Tableau 31 – Résolution des instances de *TSPTW* symétrique

Benchs	Modèle	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
AFG<	($\mathbf{MO}_{27}$)	Path	7	488	218.49	600.003	5523951.7	2807559.92	6178510
		MinDom	14	341.61	285.22	600.003	2586705	2349130.49	2342645
		Pesant	14	342.21	283.46	600.003	2764228.6	2528793.09	2244170
	($\mathbf{MO}_{31}$)$^+$	Path	15	324.88	291.5	515.486	1466110.63	1391474.38	1570450
		MinDom	18	291.07	286.77	287.1665	1121331.8	1203314.72	898939
		Pesant	18	278.25	292.23	104.5905	1085885.13	1259192.83	323740
	($\mathbf{MO}_{30}$)	Path	15	323.82	290.6	486.833	1356054.7	1309222.08	1435105
		MinDom	17	297.23	290.94	286.346	961702.83	1074015.34	714616
		Pesant	17	285.77	297.05	118.814	937368.5	1112737.23	321884.5
AFG>	($\mathbf{MO}_{27}$)	Path	0	600.04	0.02	600.027	3600317.78	1324550.25	3716600
		MinDom	0	600.03	0.02	600.022	1772433.33	456539.62	1772670
		Pesant	0	600.03	0.02	600.023	2023465.56	665592.09	1826630
	($\mathbf{MO}_{31}$)$^+$	Path	0	600.09	0.06	600.058	1054123.33	344626.79	1114500
		MinDom	0	600.09	0.06	600.057	1035739.56	365746.35	966584
		Pesant	0	600.1	0.06	600.067	1032568	378980.93	940495
	($\mathbf{MO}_{30}$)	Path	0	600.35	0.3	600.181	794036.56	418844.52	798215
		MinDom	0	600.35	0.3	600.175	598476.89	366756.84	470063
		Pesant	0	600.34	0.29	600.167	669281.44	379251.94	541972

Tableau 32 – Résolution des instances de *TSPTW* asymétrique

E

RÉSULTATS NUMÉRIQUES COMPLET POUR LA COMPARAISON DES FILTRAGES DE LA WEIGHTEDCIRCUIT

E.1 TSP

Benchs	WC	%Opti			Next			Position		
		Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
TSP<	AP	17.18	4.77	17.68	22.01	14.24	19.91	0	0	0
	HK	0.21	0.45	0.04	89.74	3.49	90.68	0	0	0
	NP	3.47	2.75	3.47	68.01	18.8	71.65	25.93	33.71	8.28
TSP>	AP	19.69	7.25	19.5	6.12	8.01	1.72	0	0	0
	HK	0.8	0.82	0.61	73.24	33.74	88.94	0	0	0
	NP	5.97	3.15	6.1	45.68	25.22	44.1	5.04	9.8	2.46
ATSP<	AP	30.42	42.21	7.42	44.86	25.81	56.86	0	0	0
	HK	13.47	30.82	0.58	21.5	26.68	4.59	0	0	0
	NP	17.48	30.25	4.1	58.8	29	70.92	5.04	2.76	5.86
ATSP>	AP	8.5	4.65	9.44	35.22	14.63	35.44	0	0	0
	HK	8.5	10.66	7.08	18.63	33.44	1.92	0	0	0
	NP	5.63	4.19	4.77	51.06	28.76	61.19	1.97	1.43	1.59

Tableau 33 – Comparaison au nœud racine des différents filtrages de la WEIGHTEDCIRCUIT pour les instances de *TSP*

E.2 TSPTW

Benchs	wc	%Opti			Next			Position		
		Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
D20	AP	7.54	9.38	2.78	81.08	15.22	87.36	80.72	14.88	84.83
	HK	3.66	5.83	1.77	81.45	16.69	92.33	82.13	15.53	92.54
	NP	0.96	2.74	0	90.82	9.99	95.03	90.41	9.7	94.53
D40	AP	10.31	8.68	8.19	78.79	14.25	79.32	83.14	10.59	82.27
	HK	5.15	5.86	3.55	80.83	14.46	79.32	83.71	11.07	82.27
	NP	1.96	3.01	0	88.17	12.55	96.79	87.91	11.57	96.57
D60	AP	10.28	8.86	10.51	80.95	11.35	84.34	85.57	8.2	87.84
	HK	5.96	6.29	4.66	82.06	11.9	84.34	86.08	8.7	87.84
	NP	3	3.45	2.24	85.6	12.38	86.54	87.82	9.76	89.01
D80	AP	7.69	5.22	8.49	84.23	7.53	81.9	88.21	5.5	86.79
	HK	3.66	2.54	4.16	84.8	7.72	82.77	88.32	5.61	86.79
	NP	2.53	2.66	3.03	87.17	9.3	84.78	89.77	6.77	87.5
D100	AP	6.03	4.62	3.25	90.97	4.75	90.39	93.25	3.35	92.74
	HK	2.95	2.63	2.38	91.81	5.31	91.44	93.65	3.73	93.48
	NP	1.69	1.31	1.19	93.12	5.15	93.13	94.09	3.95	93.61

Tableau 34 – Comparaison au nœud racine des différents filtrages de la WEIGHTEDCIRCUIT des instances de [Dum+95]

Benchs	wc	%Opti			Next			Position		
		Moy	EcTy	Méd	Moy	EcTy	Méd	Moy	EcTy	Méd
SPO	AP	17.06	6.37	17.14	36.9	24.87	36.67	34.91	28.21	29.51
	HK	10.06	6.66	11.25	42.8	27.35	41.16	40.55	30.54	40.77
	NP	8.2	5.76	9.13	44.41	30.81	36.67	46.16	31.04	43.11
SPE	AP	12.87	5.09	1.74	38.29	23.35	37.86	38.87	25.44	40.71
	HK	6.31	4.88	0	48.86	25.58	47.81	38.84	25.38	40.71
	NP	5.25	3.95	0	51.55	27.43	49.28	44.68	28.83	50
L>	AP	0.83	1.19	0.2	97.09	1.56	97.89	97.25	1.18	97.76
	HK	0.55	0.79	0.13	97.09	1.51	97.92	97.28	1.18	97.99
	NP	0.17	0.47	0	97.97	0.88	98.31	97.99	0.67	98.25
AFG<	AP	0.79	1.14	0.22	75.06	12.26	73.8	59.77	24.84	67.6
	HK	4.54	2.98	4.22	45.21	21.6	43.78	48.77	25.32	51.75
	NP	0.45	0.73	0.2	76.75	13.76	77.48	65.89	24.09	71.95
AFG>	AP	0.57	0.22	0.48	66.39	17.55	56.51	68.63	18.84	61.09
	HK	4.48	2.63	3.47	62.28	20.39	56.51	68.56	18.77	61.09
	NP	0.31	0.25	0.25	72.09	17.35	84.89	69.52	18.23	61.93

Tableau 35 – Comparaison au nœud racine des différents filtrages de la WEIGHTED-CIRCUIT des instances de *TSPTW*

F

RÉSULTATS NUMÉRIQUES COMPLET POUR LES DIFFÉRENTES STRATÉGIES DE BRANCHEMENT

F.1 TSP

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
TSP<	AP	Path	9	270.72	289.06	77.5695	96888.36	169894.11	46934
		MinDom	13	147.71	250.95	3.2005	19884.21	33449.61	1637
		Pesant	12	139.28	251.86	1.73	18506.14	33511.41	722
	HK	Path	13	44.07	160.06	0.17	34268.5	129136.35	5.00
		NoGoods	13	43.87	159.55	0.13	34911.6	122689.35	3
		MinDom	13	43.58	160.18	0.19	8716.14	32315.82	7
		MinDomLag	13	43.16	160.29	0.08	14261.86	53068.97	5
		Pesant	13	43.49	160.2	0.21	9346	34792.73	8
		PesantLag	13	43.16	160.29	0.11	17945.64	66871.93	5
	NP	Path	11	173.33	239.82	67.17	103336.93	110977.11	79100
		NoGoods	11	166.26	241.91	46.34	147521.29	292159.26	11647.5
		MinDom	11	112.47	210.68	23.47	15650.36	32528.79	1350.5
		MinDomLag	12	99.51	212.82	17.4	14965.36	30685.79	1291.39
		Pesant	12	111.3	210.04	37.69	21705.43	28740.32	5160
		PesantLag	12	99.87	213.53	17.88	29855.43	23650.32	7097.49
TSP>	AP	Path	0	600.56	0.63	600.412	25539.79	20068.32	12970
		MinDom	2	568.14	104.99	600.4095	20608.36	13852.13	12291
		Pesant	2	543.42	153.67	600.401	17107.21	11206.36	10897
	HK	Path	2	518.51	208.73	600.41	79985.71	67931.45	65088
		NoGoods	2	511.62	217.24	600.44	125815.15	106300.84	107028
		MinDom	3	477.79	244.48	600.38	47864.07	27204.05	55466.5
		MinDomLag	4	465.93	244.79	600.34	57731.53	43066.11	52993
		Pesant	4	445.29	257.47	600.35	48547.86	40528.5	46062
		PesantLag	4	482.64	244.82	600.39	55527.8	48876.55	44140
	NP	Path	1	577.57	100.94	603.46	40067.71	52818.41	8020.5
		NoGoods	2	535.77	180.57	602.57	38667.71	59818.41	18026
		MinDom	3	497.8	218.38	602.94	22911.93	29322.81	3160
		MinDomLag	3	546.92	358.6	602.07	29811.93	32322.81	4111.64
		Pesant	3	496.51	218.14	603.17	21748.36	34957.77	18356
		PesantLag	3	568.26	342.09	602.16	28648.36	39657.77	24179.72

Tableau 36 – Résolution des instances de *TSP* symétrique

F.2 TSP ASYMÉTRIQUE

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
ATSP<	AP	Path	4	319.73	267.9	20.312	258308	593141.38	52766
		MinDom	7	126.05	254.78	24.169	33446	50077.01	1051
		Pesant	7	112.61	245.58	1.611	29574	46993.13	1249.5
	HK	Path	5	226.39	309.42	5.05	140672.63	195541.25	9529
		NoGoods	5	412.3	284.65	600.03	291458.86	226717.35	318136
		MinDom	6	206.54	274.11	37.84	93170.63	119791.15	24604
		MinDomLag	6	189.37	271.98	12.34	95398.38	135209.35	7670
		Pesant	5	245.26	298.74	80.83	112823.13	140275.26	51020
		PesantLag	6	199.93	277.95	12.53	93590.38	129098.68	7967
	NP	Path	5	326.52	243.21	251.79	146536	152004.15	110872
		NoGoods	6	244.12	234.12	140.14	10681.38	12800.01	56766.5
		MinDom	5	309.19	274.03	270.72	47265.25	51263.92	24757.5
		MinDomLag	7	310.67	263.61	308.15	43665.25	3266.36	22871.82
		Pesant	5	259.56	284.1	89.85	50670	59190.89	15277
		PesantLag	6	322.48	273.73	341.45	43960	51824.32	5687
ATSP>	AP	Path	0	600.1	0.03	600.098	46051.8	14344.36	118858
		MinDom	5	93.97	115.61	600.068	6019.4	6509.52	59445
		Pesant	5	86.24	121.22	600.103	4804.4	5585.61	118858
	HK	Path	1	480.15	268.29	600.13	107765.8	64165.67	130127
		NoGoods	1	450.13	299.93	600.08	138825.5	108726.65	146359
		MinDom	1	498.23	227.83	600.12	103035.8	55277.36	133727
		MinDomLag	1	539.98	134.57	600.13	79848.2	66682.42	96722
		Pesant	0	600.14	0.06	600.13	133551.4	56185.02	152958
		PesantLag	1	535.38	144.95	600.12	80488.4	66446.18	82030
	NP	Path	0	604.58	9.31	600.46	90833.8	61618.33	74070
		NoGoods	0	600.81	0.57	600.89	192233.8	224109.25	111306
		MinDom	0	601.68	0.71	601.78	44405.2	18481.64	44547
		MinDomLag	0	602.84	2.17	601.53	43264.6	17122.48	43402.76
		Pesant	2	583.37	32.08	602.91	62929.8	23542.98	61086
		PesantLag	2	588.59	28.49	601.06	61267.4	20101.48	39134

Tableau 37 – Résolution des instances de *TSP* asymétrique

F.3 TSPTW DUMAS

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
D40	AP	Path	14	268.15	300.41	32.68	78368.92	88922.21	8319
		MinDom	20	156.86	246.58	3.31	39277.68	62211.52	834
		Pesant	21	138.38	226.45	3.28	32852.04	53408.6	819
	HK	Path	15	252.22	290.85	56.1	180873.6	271851.86	55768
		NoGoods	15	257.17	287.17	57.2	209658.76	290377.11	55891
		MinDom	20	182.3	257.18	12.12	113615.4	170939.02	4622
		MinDomLag	20	183.4	255.9	12.48	13045.4	150396	4036
		Pesant	20	180.45	255.31	9.14	104822.36	158149.17	5594
		PesantLag	20	189.47	268.08	9.45	110063.48	166056.63	5962
	NP	Path	22	80.76	197.97	0.26	63687.48	150655.32	4
		NoGoods	22	82.46	202.13	0.27	65024.92	153819.08	4.08
		MinDom	23	56.94	164.87	0.3	5848.36	13835.19	4
		MinDomLag	23	58.14	168.33	0.31	5971.18	14125.73	4.08
		Pesant	23	57.07	164.93	0.27	9222.76	30989.81	5
		PesantLag	23	58.27	168.39	0.28	9416.44	31640.6	5.11
D60	AP	Path	11	375.16	267.1	600.12	36475.12	26150.38	49231
		MinDom	12	348.48	277.68	600.12	31022.64	24378.58	45139
		Pesant	13	348.51	274.81	446.51	30114.68	23682.42	44274
	HK	Path	9	402.21	274.49	600.12	208548.72	214807.51	90874
		NoGoods	9	420.38	275.06	600.12	246485.33	249434.72	95417.7
		MinDom	10	385.13	282.92	600.12	162869.12	131804.74	192819
		MinDomLag	10	390.63	236.5	600.12	203685.36	123604.78	202459.95
		Pesant	10	377.73	281.2	281.2	141614.12	125436.35	156445
		PesantLag	10	401.15	298.63	295.26	150394.2	133213.4	164267.25
	NP	Path	15	300.88	296.49	458.7	59399.16	87075.15	9210
		NoGoods	15	313.22	308.65	477.51	61834.53	90645.23	9587.61
		MinDom	17	241.2	272.99	51.33	42854.36	55440.42	10540
		MinDomLag	17	255.91	289.64	54.46	45468.48	58822.29	11182.94
		Pesant	17	239.27	270.84	66.7	37086.48	52191.24	9556
		PesantLag	17	249.08	281.94	69.43	38607.03	54331.08	9947.8

Tableau 38 – Résolution des instances de [Dum+95] D40 et D60

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
D80	AP	Path	5	476.51	227.35	600.24	20387.9	10076.41	22930
		MinDom	5	473.77	238.28	600.24	19852.3	10310.99	24287
		Pesant	5	474.69	239.67	600.24	19205.15	10096.53	23154.5
	HK	Path	3	531.14	183.09	600.23	260205.25	121430.55	274657
		NoGoods	3	496.25	202.55	600.23	244454.93	134049.39	285643.28
		MinDom	4	525.83	187.39	600.24	185350.65	85846.73	197620.5
		MinDomLag	4	555.59	198	600.23	195841.5	90705.65	205525.32
		Pesant	3	518.99	200.02	600.23	179993.7	84375.58	191016
		PesantLag	4	549.92	211.94	600.23	190721.32	89404.36	198656.64
	NP	Path	11	291.79	298.05	118.73	84429.85	97959.08	47491
		NoGoods	11	297.92	304.31	121.22	86202.88	100016.22	48488.31
		MinDom	9	357.53	299.13	120.63	84227.4	87567.37	58692
		MinDomLag	9	375.05	313.79	126.54	88354.54	91858.17	61567.91
		Pesant	8	360.45	301.34	108.59	77792.6	74969.25	76955.5
		PesantLag	9	386.04	322.74	116.3	83315.87	80292.07	82419.34
D100	AP	Path	1	565.63	134.78	600.42	14116.67	3631.55	14793
		MinDom	1	593.32	27.54	600.41	13779.93	1252.88	14105
		Pesant	1	584.32	62.54	600.44	13404.27	2079.32	13522
	HK	Path	0	600.41	0.03	600.38	221880.53	67058.17	239754
		NoGoods	0	600.41	0.03	600.38	313535.92	85711.54	246946.62
		MinDom	0	600.41	0.04	600.38	179548.87	61144.16	185002
		MinDomLag	0	600.41	0.03	600.38	198649.64	55369.14	190552.06
		Pesant	0	600.41	0.04	600.38	173003.4	52691.71	170168
		PesantLag	0	600.41	0.04	600.38	180926.96	55104.99	175273.04
	NP	Path	6	361.4	303	600.42	88771.4	81330.64	116460
		NoGoods	6	383.45	321.48	600.42	94186.46	86291.81	123564.06
		MinDom	4	440.57	274.35	600.42	101495.47	82780.83	98517
		MinDomLag	4	449.82	280.11	600.42	103626.87	84519.23	100585.86
		Pesant	4	440.59	274.31	600.48	87556.6	67158.28	117279
		PesantLag	4	458.65	285.56	600.48	91146.42	69911.77	122087.44

Tableau 39 – Résolution des instances de [Dum+95] D80 et D100

F.4 TSPTW SYMÉTRIQUE ET ASYMÉTRIQUE

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
SPO	AP	Path	17	284.46	288.75	94.3	184904.73	193580.7	139961.5
		MinDom	17	268.65	295.54	51.15	150456	173097.5	44553
		Pesant	17	268.25	295.69	53.31	141659.5	162324.19	47241
	HK	Path	17	279.87	287.94	124.58	250906.53	319764.18	103567
		NoGoods	17	284.99	291.5	147.52	220965.55	313394.32	76824
		MinDom	17	274.35	291.94	114.91	235470.33	269797.3	102647
		MinDomLag	17	279.81	288.17	139.37	234615.57	263740.46	142918
		Pesant	17	272.97	292.75	96.82	201604.2	246624.31	91638.5
		PesantLag	17	278.33	289.05	131.95	233688.53	263430.44	111893
	NP	Path	16	294.37	293.67	138.49	104240.6	122143.67	49040.39
		NoGoods	16	293.7	294.24	135.22	106325.41	124586.54	48951.32
		MinDom	17	278.73	288.59	104.33	76867.2	94928.41	28771.77
		MinDomLag	17	278.96	288.79	89.59	78404.54	96826.98	25179.34
		Pesant	17	279.18	288.23	98.06	85338.3	112284.05	29974.02
		PesantLag	17	281.23	287.59	102.32	87045.07	114529.73	31670.72
SPE	AP	Path	20	190.52	256.77	20.76	106891.11	153203.95	19094
		MinDom	19	211.71	264.79	92.39	103214.52	135137.19	54686
		Pesant	19	209.94	265.94	71.83	101606.33	136043.67	52062
	HK	Path	19	213.48	270.83	10.14	213533.74	272719.31	12416
		NoGoods	19	211.29	267.54	25.41	180412.38	228441.53	43122
		MinDom	19	200.5	273.7	10.05	153967	212815.75	7490
		MinDomLag	19	209.56	274.27	12.42	181974.78	247121.89	15544
		Pesant	20	191.24	266.05	13.15	140629.63	196019.57	15856
		PesantLag	20	208.61	273.7	10.7	179845.33	243975.55	12962
	NP	Path	19	194.53	271.1	31.33	142305.11	526411.56	22921.85
		NoGoods	19	193.46	272.03	20.15	145305.1	506148.66	15132.13
		MinDom	19	214.77	275.05	38.72	143107.89	525842.64	25801
		MinDomLag	19	215.05	279.1	35.39	145970.05	536359.49	24020.4
		Pesant	18	218.73	277.36	39.06	147274.78	526901.18	26302.48
		PesantLag	18	217.87	277.73	37.54	150220.27	537439.2	25880.88

Tableau 40 – Résolution des instances de *TSPTW* symétrique

Benchs	WC	Strat	Preuve	CPU (s)			Branches		
				Moy	EcTy	Méd	Moy	EcTy	Méd
AFG<	AP	Path	26	83.68	206.51	0.05	25466.47	70319.98	33
		MinDom	28	58.28	177.72	0.03	11317.77	34577.12	15.5
		Pesant	28	49.28	157.95	0.04	9642.33	31579	16
	HK	Path	18	280.19	283.83	151.89	164616.47	221531.7	59486.5
		NoGoods	18	277.63	297.26	67.77	297670.21	390449.4	26622
		MinDom	18	245.96	294.5	8.6	116509.93	199663.48	12084
		MinDomLag	18	264.31	298.96	15.37	188414.07	330812.39	11683.5
		Pesant	19	226.02	289.77	15.02	184699.63	338093.24	16509.5
		PesantLag	19	264.87	298.43	25.8	194512.53	323739.62	24152
	NP	Path	27	93.11	208.21	0.9	43515.13	128089.96	18
		NoGoods	27	75.02	193.15	0.77	40690.53	102042.49	21.5
		MinDom	27	65.55	182.28	0.77	32145.67	82928.95	14.5
		MinDomLag	27	60.82	182.87	0.7	31087.27	83967.93	14.3
		Pesant	27	80.96	207.11	0.74	47314.63	140068.57	15
		PesantLag	27	60.88	182.83	0.67	411084.07	133998.97	14.5
AFG>	AP	Path	0	600.2	0.15	600.16	59613	24167.91	70643
		MinDom	3	400.47	299.62	600.1	41135.89	38896.95	26165
		Pesant	3	400.52	299.57	600.1	38728.33	38528.14	19474
	HK	Path	0	600.19	0.11	600.15	146550.89	135173.83	86942
		NoGoods	0	600.18	0.14	600.13	271523.13	159973.27	304571
		MinDom	0	600.21	0.14	600.13	106905.33	90430.7	59515
		MinDomLag	0	600.17	0.12	600.11	104632.22	104145.05	71304
		Pesant	0	600.18	0.12	600.11	222705.11	117159.41	230549
		PesantLag	0	600.19	0.16	600.11	134565.22	123108.61	105483
	NP	Path	3	442.88	252.88	600.19	187397.11	185754.02	188672
		NoGoods	3	461.58	250.82	600.34	193206.42	191512.4	165457
		MinDom	3	404.33	294.07	600.12	153209	91852.56	165540
		MinDomLag	3	432.88	286.71	600.28	157958.48	94699.99	170671.74
		Pesant	3	404.14	294.54	600.16	187397.11	185754.02	108010
		PesantLag	2	432.59	287.15	600.14	193206.42	191512.4	111358.31

Tableau 41 – Résolution des instances de *TSPTW* asymétrique

BIBLIOGRAPHIE

[AB93] Abderrahmane Aggoun et Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling* 17.7 (1993), p. 57–73.

[App+11] David L Applegate, Robert E Bixby, Vasek Chvatal et William J Cook. *The traveling salesman problem : a computational study*. Princeton university press, 2011.

[Asc95] Norbert Ascheuer. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Thèse de doct. Technische Universität Berlin, 1995.

[BB08] Genevieve Benoit et Sylvia Boyd. Finding the exact integrality gap for small traveling salesman problems. *Mathematics of Operations Research* 33.4 (2008), p. 921–931.

[BC81] Egon Balas et Nicos Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming* 21.1 (1981), p. 19–46.

[BMR11] Roberto Baldacci, Aristide Mingozzi et Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* 59.5 (2011), p. 1269–1283.

[BMR12a] Roberto Baldacci, Aristide Mingozzi et Roberto Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* 24.3 (2012), p. 356–371.

[BMR12b] Roberto Baldacci, Aristide Mingozzi et Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* 218.1 (2012), p. 1–6.

[Bel+07] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey et Thierry Petit. Global constraint catalogue : Past, present and future. *Constraints* 12.1 (2007), p. 21–62.

[Ben+12] Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau et Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints* 17.3 (2012), p. 205–233.

[CMT81a] Nicos Christofides, Aristide Mingozzi et Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming* 20.1 (1981), p. 255–282.

[CMT81b] Nicos Christofides, Aristide Mingozzi et Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* 11.2 (1981), p. 145–164.

[CT80] Giorgio Carpeneto et Paolo Toth. Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science* (1980), p. 736–743.

[Car82] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research* 11.1 (1982). Third {EURO} {IV} Special Issue, p. 42 –47.

[Chv83] V. Chvatal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman, 1983.

[DCP15] Sylvain Ducomman, Hadrien Cambazard et Bernard Penz. Etude de modeles de programmation par contraintes pour le probleme du voyageur de commerce avec fenêtres de temps. *JFPC*. 2015.

[DCP16] Sylvain Ducomman, Hadrien Cambazard et Bernard Penz. Alternative Filtering for the Weighted Circuit Constraint : Comparing Lower Bounds for the TSP and Solving TSPTW. *AAAI Conference on Artificial Intelligence* (2016).

[DDS92] Martin Desrochers, Jacques Desrosiers et Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* 40.2 (1992), p. 342–354.

[DFJ54] George Dantzig, Ray Fulkerson et Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America* 2.4 (1954), p. 393–410.

[DR59] George B Dantzig et John H Ramser. The truck dispatching problem. *Management Science* 6.1 (1959), p. 80–91.

[DVCS15] Cyrille Dejemeppe, Sascha Van Cauwelaert et Pierre Schaus. The Unary Resource with Transition Times. *International Conference on Principles and Practice of Constraint Programming*. Springer. 2015, p. 89–104.

[Dan90] George B Dantzig. *Origins of the simplex method*. ACM, 1990.

[Dan98] George Bernard Dantzig. *Linear Programming and extensions*. Princeton University Press, 1998.

[Der15] Alban Derrien. Cumulative scheduling in constraint programming : energetic characterization of reasoning and robust solutions. Theses. Ecole des Mines de Nantes, nov. 2015.

[Dum+95] Yvan Dumas, Jacques Desrosiers, Eric Gelinas et Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research* 43.2 (1995), p. 367–371.

[FD94] Daniel Frost et Rina Dechter. Dead-end Driven Learning. *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*. AAAI '94. Seattle, Washington, USA : American Association for Artificial Intelligence, 1994, p. 294–300.

[FLM02] Filippo Focacci, Andrea Lodi et Michela Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing* 14.4 (2002), p. 403–417.

[FLM99] Filippo Focacci, Andrea Lodi et Michela Milano. Cost-based domain filtering. *International Conference on Principles and Practice of Constraint Programming*. Springer. 1999, p. 189–203.

[Fis94] Marshall L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research* 42.4 (1994), p. 626–642.

[GT10] Michel Gendreau et Christos D Tarantilis. *Solving large-scale vehicle routing problems with time windows : The state-of-the-art*. CIRRELT, 2010.

[Gab+86] Harold N Gabow, Zvi Galil, Thomas Spencer et Robert E Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6.2 (1986), p. 109–122.

[Gen+08] Michel Gendreau, Jean-Yves Potvin, Olli Bräumlaysy, Geir Hasle et Arne Løkketangen. Metaheuristics for the vehicle routing problem and its extensions : A categorized bibliography. *The vehicle routing problem : latest advances and new challenges*. Springer, 2008, p. 143–169.

[Geo10] Arthur M Geoffrion. Lagrangian relaxation for integer programming. *50 Years of Integer Programming 1958-2008*. Springer, 2010, p. 243–281.

[HE80] Robert M Haralick et Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14.3 (1980), p. 263–313.

[HK70] Michael Held et Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research* 18.6 (1970), p. 1138–1162.

[HK71] Michael Held et Richard M Karp. The traveling-salesman problem and minimum spanning trees : Part II. *Mathematical Programming* 1.1 (1971), p. 6–25.

[HKW72] Michael Held, Richard M Karp et Philip Wolfe. Large scale optimization and the relaxation method. *Proceedings of the ACM annual conference-Volume 1*. ACM. 1972, p. 507–509.

[Hoe01] Willem-Jan van Hoeve. The alldifferent constraint : A survey. *arXiv preprint cs/0105015* (2001).

[JV83] Roy Jonker et Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters* 2.4 (1983), p. 161–163.

[Kru56] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7.1 (1956), p. 48–50.

[Kuh10] Harold W Kuhn. The hungarian method for the assignment problem. *50 Years of Integer Programming 1958-2008*. Springer, 2010, p. 29–47.

[LMN86] Gilbert Laporte, Hélène Mercure et Yves Nobert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks* 16.1 (1986), p. 33–46.

[LND85] Gilbert Laporte, Yves Nobert et Martin Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research* 33.5 (1985), p. 1050–1073.

[Lan+93] André Langevin, Martin Desrochers, Jacques Desrosiers, Sylvie Gélinas et Fraňlois Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks* 23.7 (1993), p. 631–640.

[Lap+00] Gilbert Laporte, Michel Gendreau, J-Y Potvin et Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research* 7.4-5 (2000), p. 285–300.

[Lau78] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence* 10.1 (1978), p. 29–127.

[Lem01] Claude Lemaréchal. Lagrangian relaxation. *Computational Combinatorial Optimization*. Springer, 2001, p. 112–156.

[MT90] Silvano Martello et Paolo Toth. *Knapsack problems : algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[Mon74] Ugo Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences* 7 (1974), p. 95–132.

[Net+07] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck et Guido Tack. MiniZinc : Towards a standard CP modelling language. *International Conference on Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg. 2007, p. 529–543.

[OMT89] A. Oplobedu, J. Marcovitch et Y. Tourbier. CHARME : Un langage industriel de programmation par contraintes, illustré par une application chez Renault. *the Ninth International Workshop on Expert Systems and their Applications : General Conference 1*. 1989, p. 55–70.

[OPF14] Nikolaj van Omme, Laurent Perron et Vincent Furnon. *or-tools user's manual*. Rapp. tech. Google, 2014.

[OW06] AJ Orman et H Paul Williams. A survey of different integer programming formulations of the travelling salesman problem. *Optimisation, Economics and Financial Analysis. Advances in Computational Management Science* 9 (2006), p. 93–106.

[PB96] Jean-Yves Potvin et Samy Bengio. The vehicle routing problem with time windows part II : genetic search. *INFORMS Journal on Computing* 8.2 (1996), p. 165–172.

[Pes+98] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin et Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science* 32.1 (1998), p. 12–29.

[Pri04] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research* 31.12 (2004), p. 1985–2002.

[Pri57] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal* 36.6 (1957), p. 1389–1401.

[RVBW06] Francesca Rossi, Peter Van Beek et Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[Ree93] Colin R Reeves. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

[Refo4] Philippe Refalo. Impact-based search strategies for constraint programming. *Principles and Practice of Constraint Programming–CP 2004*. Springer, 2004, p. 557–571.

[Repa] *Chiffres clés du transport*. Rapp. tech. Ministère de l'Ecologie, du développement durable, des Transports et du logement, 2012.

[Repb] *Chiffres clés du transport*. Rapp. tech. Ministère de l'Ecologie, du développement durable, des Transports et du logement, 2015.

[Rég+10] Jean-Charles Régin, Louis-Martin Rousseau, Michel Rueher et Willem-Jan van Hoeve. The weighted spanning tree constraint revisited. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2010, p. 287–291.

[Rég02] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints* 7.3-4 (2002), p. 387–405.

[Rég08] Jean-Charles Régin. Simpler and incremental consistency checking and arc consistency filtering algorithms for the weighted spanning tree constraint. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2008, p. 233–247.

[Rég11] Jean-Charles Régin. Global constraints : A survey. *Hybrid optimization*. Springer, 2011, p. 63–134.

[S.61] Vajda S. *Mathematical Programming*. London : Addison-Wesley, 1961.

[SD88] Marius M Solomon et Jacques Desrosiers. Survey Paper-Time Window Constrained Routing and Scheduling Problems. *Transportation Science* 22.1 (1988), p. 1–13.

[Selo2] Meinolf Sellmann. An arc-consistency algorithm for the minimum weight all different constraint. *International Conference on Principles and Practice of Constraint Programming*. Springer. 2002, p. 744–749.

[Sol87] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35.2 (1987), p. 254–265.

[TV02] Paolo Toth et Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics* 123.1 (2002), p. 487–512.

[TV97] Paolo Toth et Daniele Vigo. An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science* 31.4 (1997), p. 372–385.

[Tar82] Robert Endre Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters* 14.1 (1982), p. 30–33.

[VHC88] Pascal Van Hentenryck et Jean-Philippe Carillon. Generality versus Specificity : An Experience with AI and OR Techniques. *AAAI*. 1988, p. 660–664.

[VHSD95] Pascal Van Hentenryck, Vijay Saraswat et Yves Deville. Design, implementation, and evaluation of the constraint language cc (FD). *Constraint Programming : basics and trends*. Springer, 1995, p. 293–316.

[Vid+13] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau et Christian Prins. Heuristics for multi-attribute vehicle routing problems : A survey and synthesis. *European Journal of Operational Research* 231.1 (2013), p. 1–21.

[Vid+14] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau et Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research* 234.3 (2014), p. 658–673.

[Vil04] Petr Vilím. O (nlog n) filtering algorithms for unary resource constraint. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2004, p. 335–347.

[Wil90] David Paul Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. Thèse de doct. Massachusetts Institute of Technology, 1990.

RÉSUMÉ

Les problèmes de tournées de véhicules sont des problèmes d'optimisation combinatoire épineux avec des enjeux économiques et environnementaux importants au sein de la chaîne logistique. Le problème fondamental est de desservir des clients avec un ensemble de véhicules de façon à minimiser la distance totale parcourue. En pratique, il y a une grande variété d'objectifs et de contraintes additionnelles, liées à la législation et à la diversité des domaines d'applications. Ces problèmes de tournées sont très fréquents pour de nombreuses industries et la conception d'approches de résolution génériques est devenue une question de recherche importante.

Cette thèse porte sur la conception et le développement d'un nouveau moteur de résolution pour les logiciels de tournées de véhicules proposés par l'entreprise GEOCONCEPT. Le solveur mis au point s'appuie sur la programmation par contraintes (PPC) pour améliorer la flexibilité (prise en compte de contraintes additionnelles), la déclarativité et la maintenance qui sont les limites des solveurs actuels de GEOCONCEPT fondés sur la recherche locale.

Dans un premier temps, un modèle de graphe est établi pour la représentation unifiée des données et de nombreuses contraintes métiers. La résolution s'effectue par des approches à base de voisinage large disponibles dans les solveurs de PPC modernes. On peut ainsi traiter des instances de très grandes tailles efficacement tout en conservant une approche déclarative pour exprimer une classe très large de problèmes de tournées de véhicules. Dans un second temps, des modèles PPC s'appuyant sur des représentations redondantes du problème sont proposés afin de renforcer le filtrage. Nous nous intéressons en détails aux mécanismes de filtrage c'est-à-dire aux processus d'élimination des valeurs infaisables ou sous-optimales dans les domaines des variables. Ces algorithmes permettent de simplifier rapidement le problème et de fournir des bornes inférieures afin d'évaluer la qualité des solutions obtenues. Les bornes inférieures sont obtenues en résolvant des relaxations du plus célèbre des problèmes de la Recherche Opérationnelle : le problème du voyageur de commerce (TSP). Ce problème est le cœur de la contrainte globale WEIGHTEDCIRCUIT permettant de modéliser les problèmes de tournées en PPC. Nous proposons de nouveaux mécanismes de filtrage pour cette contrainte s'appuyant sur trois relaxations du TSP. Ces relaxations sont comparées sur les plans théorique et expérimental. L'originalité de ce travail est de proposer un nouvel algorithme de filtrage permettant de raisonner à la fois sur les successeurs directs d'un client et sur sa position dans la tournée. Ces raisonnements sont particulièrement utiles en présence de contraintes de fenêtres de temps, très communes dans les problèmes industriels.

Le nouveau moteur de résolution offre d'excellentes performances sur des problèmes académiques et industriels tout en proposant des bornes inférieures informatives à des problèmes industriels réels.