Modèles de résolution approchée et efficace pour les problèmes des réseaux de transport et de télécommunication

Ibrahim Moussa

UNIVERSITÉ de Picardie Jules Verne
www.u-picardie.fr

Eco-procédés, optimisation et aide à la décision
eproad

UNIVERSITÉ DE PICARDIE JULES VERNE D'AMIENS

ÉCOLE DOCTORALE : Sciences, Technologie et Santé (ED 547)

Spécialité : Informatique

THÈSE DE DOCTORAT

soutenue le 15 Juin 2015

par

Ibrahim MOUSSA

Modèles de résolution approchée et efficace pour les problèmes des réseaux de transport et de télécommunication

Membres du jury :

M. KACEM	Imed	Professeur	Rapporteur
M. SAUBION	Fréderic	Professeur	Rapporteur
M. GIANNAKOS	Aristotelis	Maître de Conférences	Examinateur
M. HAO	Jin-Kao	Professeur	Examinateur
M. HIFI	Mhand	Professeur	Directeur
M. SAADI	Toufik	Maître de Conférences	Co-Encadreur

Remerciement

Ce travail s'est déroulé au sein de l'équipe $\mathcal{ROAD}$ de l'unité de recherche ($\mathcal{EPROAD}$), dans une bonne ambiance. Au moment où j'achève ce travail, je pense avant tout à ceux qui m'ont soutenu et accompagné et je tiens à adresser mes remerciements les plus sincères au professeur Mhand HIFI, directeur de cette thèse et directeur du $\mathcal{EPROAD}$, pour m'avoir accueilli au sein de son équipe. J'ai particulièrement apprécié sa confiance, ses conseils et son suivi, mais je ne le remercierai jamais assez pour les remarques éclairées qu'il m'a prodiguées tout au long de cette thèse. Je remercie également M. SAADI Toufik, co-encadrant de cette thèse pour ses encouragements, ses conseils, ses connaissances dans le domaine et surtout, son soutient dans les moments de doute, ont été un réel apport.

Je remercie les professeurs Frédéric SAUBION et Imed KACEM d'avoir accepté de rapporter ma thèse. Je remercie également le professeur Jin-Kao HAO et M. Aristotelis GIANNAKOS d'avoir accepté de faire partie du jury.

Je profite de cette occasion pour remercier chaleureusement tous les membres de l'unité de recherche $\mathcal{EPROAD}$ et quelques membres de l'unité de recherche $\mathcal{LTI}$, pour avoir su créer une ambiance agréable et multi-culturelle. Je remercie particulièrement Sagvan SALEH pour son aide dans la programmation `C++` et pour sa grande disponibilité durant mes problèmes de débogage.

Finalement, tout mon respect va à mes parents pour leurs encouragements et ce malgré l'éloignement, à mon frère Oumar, sa femme Awa et ma petite nièce Kamila pour leur presence chaleureuse.

Je ne peux m'empêcher en ce moment de remercier la république du Niger pour le soutien financier qu'elle m'a accordé tout au long de cette thèse.

Résumé

Titre : Modèles de résolution approchée et efficace pour les problèmes des réseaux de transport et de télécommunication.

Le transport des personnes et des marchandises soulève un grand nombre de problèmes difficiles à résoudre. En général, l'objectif des compagnies de transport est souvent de visiter un ensemble de points (représentant des clients) à un moindre coût. Ces clients peuvent être considérés ponctuels comme un site bien précis, une station ou même un numéro de rue. Les délocalisations des sites de production, de distribution, de commercialisation et l'ouverture des marchés augmentent de plus en plus, l'intérêt des entreprises de transport à minimiser les coûts. En effet, pour rester compétitif, les professionnels du transport doivent réduire des coûts d'exploitation (carburant, péage, location, etc) et contrôler l'empreinte écologique engendrée par leurs activités. Ceci revient alors, à optimiser le nombre de véhicules opérationnels et le nombre de trajets pour chaque véhicule, tout en respectant les contraintes liées à l'activité de l'entreprise (délais des livraisons, horaires des livraisons, temps de travail réglementaire des chauffeurs, type de marchandises transportées, nature et handicape éventuel des personnes à transporter, etc).

Aujourd'hui, la recherche opérationnelle sur ce type de problèmes s'avère très importante car elle permet de concevoir des systèmes d'informations essentiels dans la prise de décision. En effet, ces systèmes permettent de modéliser et de traiter les flux d'informations de l'entreprise dans le but d'aider à la prise de décision. Notons ainsi que le but final est de satisfaire les clients tout en respectant les contraintes à un moindre coût.

Cette thèse porte sur la résolution approchée de deux problèmes de l'optimisation combinatoire bien connus en recherche opérationnelle. C'est problèmes trouvent de larges champs d'application dans le domaine de transport des personnes ou de marchandises et dans le domaine de la télécommunication. La première partie de la thèse est consacrée au problème d'orientation d'équipe qui est une variante du célèbre problème de tournées de véhicules. La deuxième partie de la thèse s'attaque au problème de K-clusters dans un graphe biparti. Ce dernier est utile pour décomposer et faciliter la résolution d'un problème combinatoire.

`Mots clés` : Biclique, Cluster, Greedy, Heuristique, K-CmBCP, Logistique, Optimisation combinatoire, Recherche opérationnelle, TOP, Transport, VRP.

Abstract

Title : Approached and effective resolution models for problems of transport and telecommunication networks.

The transport of people and goods raises a large number of problems that are difficult to solve. In general, the purpose of the transport companies is to visit at low cost, a set of points representing clients. These customers can be considered as a one-off specific site, a station or even a house number. Relocation of production sites, distribution, marketing and market openness increase more and more the interest of transport companies to minimize costs. In order to stay competitive, transport professionals need to reduce operating costs (fuel, tolls, location, etc) and check the ecological footprint generated by their activities to avoid any surcharges. This amounts to optimize the number of operational vehicles and the number of trips per vehicle, while respecting the constraints associated with the company's business (delays in deliveries, schedules of deliveries, regular working hours of drivers, such freight traffic, nature and possible handicaps of transported people, etc).

Nowadays, operational research on such problems is very important because it allows the design of critical information systems in decision-making. Indeed, these systems are used to model and treat the company's information flow in order to help decision making knowing that the ultimate goal is to satisfy customers, within the constraints and, at lower cost.

This thesis focuses on the approximate resolution of two well known combinatorial problems in operations research, and, that find several applications in the field of transportation of people or goods and in filed of telecommunication : the first part of the thesis is devoted to the team orienteering problem that is a variante of the famous vehicle routing problem and the second part of the thesis addresses the K-clusters minimum biclique completion problem in a bipartite graph. This problem is useful to decompose and help for solving a combinatorial problem.

`Keywords` : Biclique, Cluster, Combinatorial Optimization, Greedy, Heuristics, K-CmBCP, Logistics, Operations Research, TOP, Transportation, VRP.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Introduction générale

Les enjeux économiques et environnementaux renforcent l'importance de l'optimisation combinatoire dans les domaines de informatique et la recherche opérationnelle. C'est pour cela que de nos jours, l'optimisation des transports et la performance logistique sont des enjeux économiques très importants pour les entreprises. Cela repose notamment sur l'organisation et l'efficacité des tournées de véhicules. Ainsi, la plupart des solutions commerciales proposant une optimisation des tournées n'aboutissent pas souvent sur de véritables outils d'optimisation mais proposent plutôt, des solutions organisationnelles. Ces solutions sont souvent exploitées pour rationaliser les coûts des livraisons, des distributions ou encore des collectes.

Dans cette thèse, nous nous sommes intéressés à deux problèmes de l'optimisation combinatoire. Dans la première partie (partie I), nous traitons le problème d'orientation d'équipe qui est historiquement la problématique la moins étudiée dans la catégorie des problèmes de tournées de véhicules. Dans la deuxième partie (partie II), nous nous intéressons au problème de regroupement dans un graphe biparti.

Plus précisément, la première partie de cette thèse comporte trois chapitres. Cette partie présente le problème d'orientation d'équipe comme un cas particulier du célèbre problème de tournées de véhicules plus connu sous sa nomination anglophone "Vehicles Routing Problem (VRP)".

La deuxième partie de cette thèse composée de deux chapitres. Elle entreprend l'étude du problème de regroupement dans un graphe biparti.

Néanmoins, avant d'aborder plus spécifiquement les dites parties, cette introduction fournit une vue panoramique sur les sujets de recherche principaux dans le domaine du transport et de la logistique. Cette introduction présente également les contributions de la thèse et énonce le plan de ce mémoire.

Présentation

Dans un contexte économique toujours plus concurrentiel entre les entreprises, l'optimisation des ressources occupe une place importante dans les prises de decision. Ainsi, on assiste à l'apparition des problèmes d'optimisation de plus en plus complexes que des méthodes classiques sont incapable de résoudre de façon optimale.

Le problème de tournées de véhicules est un problème de l'optimisation combinatoire parmi les plus connus de la littérature. Il consiste à déterminer une séquence de clients ou de segments routiers pour construire les itinéraires.

D'une manière générale, on parle d'un problème d'optimisation combinatoire lorsque ce dernier est constitué des variables de décision, d'une fonction objectif et des contraintes à satisfaire. En effet, les nouvelles contraintes liées à la demande des clients génèrent plusieurs nouveaux problèmes de tournées de véhicules, en général plus complexes à résoudre.

La première partie de cette thèse considère une classe dérivant du problème de tournées de véhicules à savoir, le problème d'orientation d'équipe (TOP [1]) qui un problème très peu traité dans la littérature. Plus précisément, ce problème est de nature bi-objectifs, étant donné qu'il consiste à construire une ou plusieurs tournées en maximisant le gain total récolté chez les clients et en minimisant la longueur des tournées. La nature bi-objectifs induit plusieurs variantes qui peuvent figurer dans cette classe en jouant sur la considération des deux objectifs déjà mentionnés. Parmi les variantes de ce problème, nous avons traité dans cette thèse, le cas qui consiste à maximiser le gain récolté par tous les véhicules et à limiter la longueur de chaque tournée par une longueur maximale. Cela, revient donc à construire plusieurs tournées afin de servir un certain nombre de clients. A noter que, dans le TOP, la capacité des véhicules n'est pas prise en considération car on fournit un service aux clients et on récolte un gain chez chaque client servi en contre-partie de ce service. Ainsi, chaque véhicule doit partir d'un dépôt et revenir à une autre dépôt après avoir visité un ensemble de clients et sans dépasser la longueur maximale autorisée sachant que chaque client ne peut être servi que par un seul véhicule et une seule fois. La caractéristique principale du TOP, qui le distingue des autres problèmes de transport, est qu'une solution peut avoir des clients non servis. Ceci, est dû éventuellement d'une part à la limitation du nombre de véhicules et d'autre part à la limitation de chaque tournée par une longueur maximale.

Cette thèse étant une contribution dans le domaine de la recherche opérationnelle et l'aide à la décision, nous avons proposé dans la première partie, une méthode de résolution approchée pour le TOP.

Dans à la deuxième partie de cette thèse, nous abordons le problème de K-clusters dans un graphe biparti, qui est aussi un problème de l'optimisation combinatoire très peu étudié dans la littérature. Dans ce problème, l'objectif consiste à répartir un nombre de services sous formes de clusters [2], où chaque cluster peut contenir des services et des clients. Chacun des services doit satisfaire au moins un client. Ce problème peut être représenté sous forme d'un graphe biparti, où les services représentent les sommets du côté gauche du graphe et les clients représentent les sommets du côté droit du graphe biparti.

Ainsi, afin de contribuer à la résolution de ce problème de l'optimisation combinatoire, nous avons proposer une méthode de résolution approchée permettant le meilleur partitionnement des services capable de satisfaire les clients à moindre coût.

Organisation du manuscrit

La suite de cette thèse est organisée comme suit : Dans le chapitre 1, nous rappelons quelques variantes du problèmes de tournées de véhicules. Pour chaque variante, nous présentons un bref état de l'art. Puis pour la variante qui nous intéresse particulièrement dans cette thèse, à savoir le problème d'orientation d'équipe,

1. TOP : Team Orienteering Problem
2. Un cluster est groupe de noeuds

nous fournissons un état de l'art détaillé. Le chapitre 2 est consacré aux différentes méthodes utilisées le plus souvent dans la littérature permettant de résoudre les problèmes de tournées de véhicules et ses principales variantes d'une manière générale. Dans le chapitre 3, nous présentons une méthode approchée de résolution du problème d'orientation d'équipe que nous avons appelé : la méthode de recherche en deux phases. Cette approche a été inspirée d'une heuristique de résolution du problème de tournées de véhicules avec contraintes de capacités. Nous avons testé cette heuristique sur les instances de la littérature et elle a fournit des résultats compétitifs par rapport aux meilleurs résultats de la littérature.

Dans le chapitre 4, nous présentons d'abord un état de l'art pour notre seconde problématique de recherche qui est le problème de K-clusters dans un graphe biparti et dont la nomination anglophone est : *The K-Clustering minimum Biclique Completion Problem : K-CmBCP*, ensuite, nous citons quelques méthodes de résolution utilisées dans la littérature pour résoudre le problème de regroupement dans un graphe biparti et nous donnons quelques exemples d'applications du K-CmBCP dans le monde industriel. En fin de ce chapitre, nous présentons les modèles mathématiques du problème de regroupement dans un graphe biparti. Dans le chapitre 5, nous proposons de résoudre le problème du K-clusters dans un graphe biparti en appliquant une heuristique qui s'appuie sur le principe de la coopération entre une recherche gloutonne et une recherche par diversification. Cette approche utilise trois phases complémentaires : la première phase permet la construction d'une solution de départ réalisable, la deuxième phase améliore la qualité de la solution de départ et enfin, la troisième et dernière phase introduit la *stratégie de diversification* dans le but d'atteindre certains sous-espaces d'une manière efficace.

Les résultats obtenus par cette approche, en particulier, lorsqu'ils sont comparés aux meilleurs résultats de la littérature ainsi qu'aux résultats obtenus par le solveur Cplex, montrent des améliorations sur certains types d'instances. Globalement, cette approche reste compétitive.

Nous terminons ce manuscrit par une conclusion, dans laquelle nous discutons et analysons les résultats obtenus dans nos travaux de recherche et précisons nos perspectives.

Première partie

Le problème d'orientation d'équipe

Chapitre 1

Etat de l'art

Sommaire

Dans ce chapitre, nous commençons par un bref état de l'art du problème de tournées de véhicules [1], suivi de l'état de l'art de sa variante qui nous intéresse particulièrement à savoir le problème d'orientation d'équipe [2]. Ensuite, nous proposons une classification des différentes variantes du problème de tournées de véhicules et du problème d'orientation d'équipe que nous avons eues à étudier durant cette thèse.

1.1 Introduction

De nos jours, le problème de transport occupe une place importante dans la vie économique de notre société. Historiquement, les problèmes de transport les plus connus restent le problème du voyageur de commerce [3] et le problème du postier chinois [4]. Le problème de tournées de véhicules est la combinaison de plusieurs problèmes de voyageur de commerce. Il a fait l'objet de nombreux travaux et classifications de ses variantes dans la littérature.

Dans un contexte où les problématiques commerciales et industrielles traitées par les entreprises et les collectivités sont de plus en plus complexes car les modèles opérationnels permettant la construction des tournées sont remplacés par des modèles de planification stratégique. L'ambition des fournisseurs de logiciels est de proposer des outils d'aide à la décision pour les grandes étapes du processus décisionnel. Ainsi, parmi les étapes du processus décisionnel, nous citons les trois principales :

— L' étude prévisionnelle qui permet d'estimer les demandes.
— L' étude stratégique qui permet de planifier les services.

1. Vehicles Routing Problem (noté : VRP)
2. Team Orienteering Problem (noté : TOP)
3. Traveling Salesman Problem (noté : TSP)
4. Chinese Postman Problem (noté : CPP)

— Lé étude opérationnelle qui permet l'optimisation de ces services et la rationalisation de ces services.

Le problème d'orientation d'équipe (équipe de véhicule) est un modèle qui se situe entre le niveau stratégique et opérationnel. Ainsi, nous savons par définition que le problème d'orientation d'équipe est une extension d'un problème connu en optimisation qui est celui de la course d'orientation [5]. Le problème de course d'orientation est défini de la manière suivante : Soit un ensemble de clients, chaque client ayant un score associé, un point de départ D et un point d'arrivée A. Chaque client a des coordonnées et il y a donc un coût pour passer d'un client à un autre. L'objectif de cette course d'orientation est d'arriver à obtenir le chemin ayant le plus grand score possible (qui est la somme des scores des clients appartenant au chemin) tout en satisfaisant une contrainte de coût, c'est à dire que la somme des coûts du chemin ne doit pas dépasser une valeur prédéfinie.

Dans le reste de ce chapitre, nous décrirons le problème de tournées de véhicules et le problème d'orientation d'équipe, puis nous proposons une classification selon les objectifs à optimiser, les contraintes spécifiques et le type de solutions recherchées.

Afin de faciliter la lecture, nous commençons par donner une terminologie relative aux problèmes de transport qui sera utilisée tout au long de ce chapitre et du chapitre suivant.

Terminologie

— Le système de transport est défini par une flotte de M véhicules basés dans un ou plusieurs dépôts. Chaque véhicule peut avoir ses propres caractéristiques : capacité, équipement, vitesse de déplacement, coût d'utilisation, durée d'utilisation maximale, ..., etc.
— Un tronçon est un itinéraire continu, sans arrêt, entre deux services et identifiable sur le réseau routier.
— Fondamentalement, une tournée est un plan exact qui peut être défini comme étant un ensemble de tronçons empruntés par un véhicule et qui peut contenir des heures de passages éventuellement.

1.2 Etat de l'art sur le problème de tournées de véhicules

Le problème de tournées de véhicules est une extension du problème du voyageur du commerce. Il a été introduit pour la première fois par Dantzig et al.[21], sous le nom de *Truck Dispatching Problem* et a depuis fait l'objet de nombreuses études qui ont permis de le modéliser et de le résoudre.

La multitude d'applications du problème de tournées de véhicules dans de nombreux domaines, dont celui du transport et de la distribution, a fait naître dans la littérature de nombreuses variantes de ce problème.

5. Orienteering Problem (noté : OP)

Dans cette section, nous présentons les plus connues des variantes du problème de tournées de véhicules :

Ainsi, dans sa version la plus fondamentale dite problème de tournées de véhicules avec contraintes de capacité [6], on considère une flotte de M véhicules de même capacité Q (pour chaque véhicule) basée dans un dépôt D et qui doit assurer des tournées entre N clients ayant demandé chacun une quantité de marchandises q_i. L'ensemble des clients visités par un véhicule désigne alors la tournée de celui-ci. Chaque client doit être servi une et une seule fois et chaque tournée commence et se termine au dépôt D. L'objectif du problème de tournées de véhicules avec contraintes de capacité [7] est de minimiser le coût total C, c'est à dire la somme des distances ou des temps de parcours des tournées, tout en respectant la contrainte de capacité Q du véhicule qui assure la tournée. Autrement dit, la quantité de marchandises q_i des clients, livrée sur une tournée R, ne doit pas dépasser la capacité du véhicule Q qui assure cette tournée : $\sum_{i=1}^{N} q_i \leq Q$.

Le problème de tournées de véhicules avec fenêtres de temps [8], quant à lui, spécifie que pour chaque client $i \in N$, on impose une fenêtre de temps dans laquelle la livraison doit être effectuée. La fenêtre de temps est un intervalle de temps $[e_i\ l_i]$ au cours duquel son service (par exemple le chargement ou le déchargement de marchandises) doit être accompli. A noter que l'intervalle de temps au niveau du dépôt D est $[e_D\ l_D]$. L'objectif du VRPTW est alors de minimiser le nombre de véhicules et la distance totale parcourue pour servir les clients sans violer les contraintes de fenêtres de temps pour chaque client. Par exemple, quand b_i représente le debut de service au niveau d'un client i, alors pour qu'une route $R = (i_0, i_1, ..., i_N, i_N + 1)$ soit réalisable, il faut que $e_i \leq b_i \leq l_i$, $1 \leq R \leq N$, et $(b_N + \delta_N + e_N, 0) \leq l_D$.

Le problème de tournées de véhicules avec fenêtres de temps est dit à contraintes dures [9], lorsqu'il est impossible de servir un client i en dehors de son intervalle de temps $[e_i\ l_i]$ et il est dit à contraintes souples [10], lorsque les véhicules peuvent servir le client en dehors de sa fenêtre de temps mais au prix d'une certaine pénalité. Ainsi, cette variante du VRP avec fenêtres de temps est la plus répandue dans la littérature, néanmoins, il existe d'autres variantes du VRP des fenêtres de temps, notamment, lorsque les fenêtres de temps sont relatives aux horaires d'ouverture du dépôt ou aux horaires de travail du personnel (voir Ombuki et al.[64]).

Le problème de tournées de véhicules avec retour [11], comporte deux types de clients : les receveurs et les livreurs. Toutes les marchandises livrées sont prises au dépôt D et toutes les marchandises récupérées sont retournées au dépôt D. Ce modèle général donne trois catégories de VRPB (voir Parragh et al.[67]) :

— *VRP with Clustered Backhauls* (VRPCB), où, toutes les livraisons sont ef-

6. Les véhicules ont une capacité limitée (quantité, taille, poids, etc.).
7. *Capacited Vehicle Routing Problem (noté : CVRP)*
8. *Vehicle Routing Problem with Time Windows* (VRPTW)
9. Hard time window constraints
10. Soft time windows constraints
11. *Vehicle Routing Problem with Backhauls* (VRPB)

fectuées avant le premier ramassage, comme expliqué dans l'article de Brandao [9].

— *VRP with Mixed Linehauls and Backhauls* (VRPMB), où, les ramassages et les livraisons peuvent être mêlés dans une même tournée, comme dans l'article de Crispim et Brandao [18].

— *VRP with Simultaneous Delivery and Pickup* (VRPSDP), où, chaque client i peut être receveur et livreur en même temps (voir Hoff et Løkketangen [42]).

Le problème de ramassage et de livraison [12], est presque similaire au VRPB à la différence que, ici, les marchandises livrées peuvent être prises, soit au dépôt, soit chez les livreurs, mais pas seulement au dépôt. Ce modèle comprend deux catégories de problèmes (voir Parragh et al. [68]) : la première catégorie [13],dans laquelle, les demandes des clients sont indépendantes, ce qui signifie que chaque unité ramassée (au dépôt ou chez un livreur) peut être utilisée pour livrer n'importe quel client receveur. Généralement, ce problème est désigné par *Pick-up and Delivery VRP* (PDVRP) et il est mono produit, cela veut dire que, toutes les marchandises transportées sont du même type, comme par exemple dans l'article de Chalasani et Motwani [13]. Dans la deuxième catégorie [14], les demandes des clients sont liées, ce qui signifie que, chaque transport doit faire un lien entre une origine et une destination précises. Dans la littérature, deux types de ce modèle coexistent : le *Pickup and Delivery Problem* (PDP) et le *Dial-A-Ride Problem* (DARP). Ainsi, le PDP porte sur le transport des marchandises (voir Ambrosini et al. [3]), tandis que le DARP porte sur le transport de personnes (voir Aldaihani et Dessouky [2]).

Le problème de tournées de véhicules stochastique [15], est un VRP dans lequel au moins un élément du problème est aléatoire. Les trois catégories du SVRP, les plus connus (voir Cordeau et al. [16]) sont :

— *VRP with Stochastic Customers* (VRPSC), où, P_i représente la probabilité qu'un client i fasse une demande.

— *VRP with Stochastic Demands* (VRPSD), dans lequel, la demande d'un client i est une variable aléatoire.

— *VRP with Stochastic Travel Times* (VRPSTT), dans lequel, le temps de service t_i d'un client i et le temps de trajet t_{ij} (entre un client i et un autre client j) sont des variables aléatoires.

Le problème de tournées de véhicules multi-périodique [16], considère une planification à M périodes. Chaque client i doit être visité k fois ($1 \leq k \leq M$), et les demandes quotidiennes sont fixes (voir Lacomme et al. [52]). A noter que, si $M = 1$, le PVRP devient alors un problème de tournées de véhicules classique.

Le problème de tournées de véhicules avec plusieurs dépôts [17], comme son nom l'indique, comporte plusieurs dépôts dans lesquels sont localisés les véhicules. Chaque

12. *Vehicle Routing Problem with Pick-up and Delivery* (VRPPD)
13. Nommée : unpaired pickup and delivery points.
14. Nommée : paired pickup and delivery points
15. *Stochastic Vehicle Routing Problem* (SVRP)
16. *Periodic Vehicle Routing Problem* (PVRP)
17. *Multi-Depot Vehicle Routing Problem* (MDVRP)

tournée d'un véhicule doit commencer et finir au même dépôt. Ainsi, chaque client doit être visité exactement une fois par l'un des véhicules situés dans les dépôts indiqués par ce client (voir Mingozzi [59]). Sachant que la demande totale de marchandises doit être servie à partir de plusieurs dépôts, l'objectif du MDVRP, est de minimiser la flotte de véhicules et la somme du temps de tournées.

Le problème d'orientation d'équipe[18], est une variante du VRP dans laquelle on est pas obligé de visiter tous les N clients. Ainsi, le TOP consiste à trouver un ensemble de M chemins disjoints partant tous d'un même dépôt D et ayant la même destination A. L'objectif du TOP, est de maximiser le score total P correspondant à la somme des scores récoltés par chaque véhicule au niveau des clients visités sur les M chemins tout en respectant une longueur de temps maximale T_{max} pour chaque véhicule (voir[14]).

Afin d'apporter notre contribution à la résolution de cette variante du problème de tournées de véhicules qui est très peu étudiée dans la littérature, nous élaborons son état de l'art dans la section suivante.

La figure 1.1 montre un exemple du problème de tournées de véhicules, avec 16 clients résolu en utilisant 3 véhicules partant d'un dépôt D.

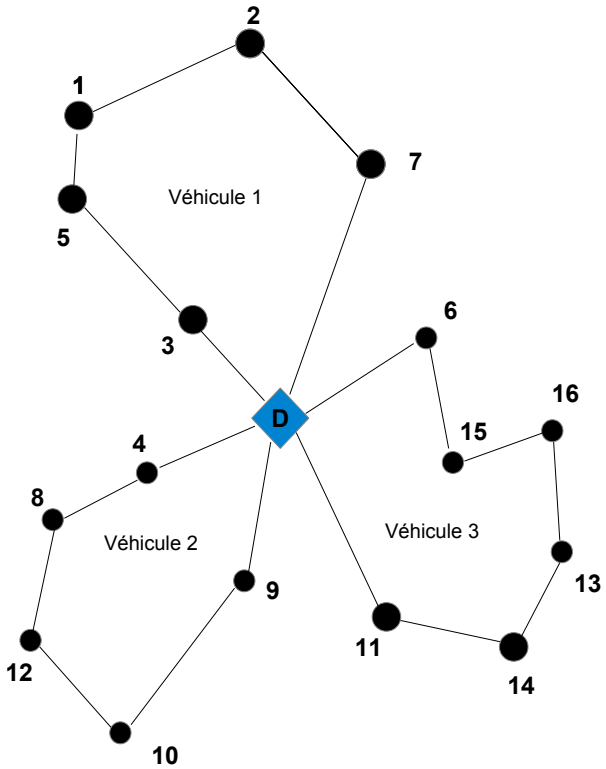

FIGURE 1.1 – Exemple de VRP avec 16 clients résolu avec 3 véhicles.

18. *Team Orienteering Problem* (TOP)

1.3 Etat de l'art sur le problème d'orientation d'équipe

Dans cette section, nous parlerons du problème d'orientation d'équipe que nous avons défini comme un problème de tournées de véhicules pour lequel chaque véhicule n'est pas forcé de visiter tous les clients qui lui sont affectés. Sachant qu'un client est visité au plus une seule fois par les véhicules d'une même équipe et que la durée de chaque chemin est limitée par une longueur de temps maximale, le but du problème d'orientation d'équipe, est de maximiser le score total récolté par chacun de membres d'une équipe au niveau des clients.

La figure 1.2 montre un exemple de problème d'orientation d'une équipe résolu avec 3 véhicules qui doivent partir du dépôt D vers l'arrivée A en maximisant le score récolté lors de la visite d'un client. Dans notre exemple, on a 16 clients au total, mais certains d'entre eux ne seront pas visités, cela, est dû non seulement à la limitation du temps de parcours, mais aussi, par le fait que chaque client doit être visité au plus une seule fois.

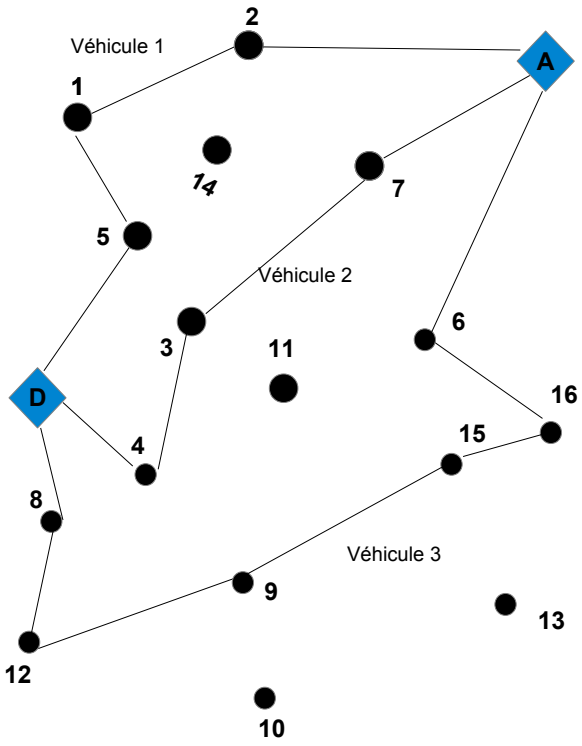

FIGURE 1.2 – Exemple de TOP avec 16 clients résolu avec une équipe de 3 véhicles.

Ce problème aurait été introduit en premier lieu par Butt et al. [10] sous forme de "TSP- sélectif" mais n'aurait acquis son nom actuel (*Team Orienteering Problem* : TOP) et sa formalisation que dans Chao et al. [44]. Dans ce article, les auteurs ont proposé, une première heuristique pour résoudre ce problème. Cette heuristique construit dans un premier temps un ensemble de chemins entre lesquels des échanges

de clients vont être faits en appliquant un algorithme de re-séquencement.

Il existe de nombreuses variantes du problème d'orientation d'équipe (TOP). Parmi ces variantes nous pouvons citer :

Le TOP avec contraintes de capacité [19], qui est la variante du problème d'orientation d'équipe pour laquelle, les véhicules n'ont pas la même capacité ou bien pour laquelle la somme des demandes d'une tournée ne doit pas dépasser une certaine quantité [20].

Le TOP avec fenêtres de temps [21], qui est l'extension du problème d'orientation d'équipe, avec une contrainte qui limite le temps pendant lequel un client peut être servi [22].

Le TOP avec fenêtres de temps et contrainte de capacité [23], est une variante de la classe des problèmes de tournées avec scores. Ainsi, l'objectif est de maximiser le score total collecté par les M véhicules de la flotte, sachant que la durée totale de chaque tournée est limitée par une durée maximale T_{max}, la somme des demandes d'une tournée ne doit pas dépasser la capacité Q, chaque client doit être satisfait au plus une seule fois et en respectant sa fenêtre de temps.

Le TOP avec fenêtres de temps et une multitude de contraintes [24], introduit par Garcia et al. [1], est une généralisation du CTOPTW, dans laquelle des contraintes supplémentaires ont été imposées. Les auteurs ont ajouté plusieurs autres contraintes et à chaque client, ils ont associé plusieurs fenêtres de temps.

Survol des méthodes proposées dans la littérature afin de résoudre le TOP

En effet, quelques autres approches ont vu le jour pour tenter de résoudre au mieux le problème d'orientation d'équipe et ses variantes, ainsi on sait que :

Labadi et al. [51] ont proposé une heuristique basée sur la recherche locale pour résoudre le problème d'orientation d'équipe avec fenêtres de temps. Dans cette heuristique, l'algorithme de la recherche locale remplace un arc d'un chemin par des noeuds qui ne figurent pas dans ce chemin et qui offrent plus de profit. Pour cela, les auteurs ont résolu le problème d'affectation associé au TOPTW, en se basant sur la solution obtenue, l'algorithme décide quels arcs, il doit insérer dans ce chemin.

Certaines approches proposées, sont des adaptations des métaheuristiques comme l'approche des colonies de fourmis proposée par Liangjun Ke et al. [50] qui fournit de bons résultats en un temps court, l'adaptation d'un algorithme génétique proposée par Bouly et al. [33], des recherches tabous par Tang et al. [76] ou par Archetti et al. [12].

19. *Capacited Team Orienteering Problem* (CTOP)
20. La capacité peut être limitée sur les vehicles ou sur l'ensemble des demandes.
21. *Team Orienteering Problem with Time Windows* (TOPTW)
22. C'est à dire que pour chaque client, on définit une fenêtre de temps qui limite le temps du service.
23. *Capacitated Team Orienteering Problem with Time Windows* (CTOPTW)
24. *Multi Constrained Team Orienteering Problem with Time Windows* (MCTOPTW)

D'autres encore ont proposé des approches différentes pour résoudre le problème d'orientation d'équipe et ses variantes, nous pouvons citer, par exemple :

Butt et al. [10] qui ont utilisé une approche basée sur la génération de colonnes ou encore Boussier et al. [72] qui ont proposé une méthode de *branch and price*.

Garcia et al. [1] qui ont proposé une méthode hybridant la recherche locale et une méthode de recherche aléatoire, qu'ils ont appelé le *Greedy Randomized Iterated Local Search (noté : GRILS)*, afin de résoudre le *Multi Constrained Team Orienteering Problem with Time Windows (MCTOPTW)*

Fahim et al. [25] qui ont proposé une heuristique à deux phases, appelée : *Insert* et *shake*. La première phase de cette heuristique, consiste à construire une solution réalisable à l'aide d'une méthode d'insertion qui évalue chaque possibilité selon une fonction d'évaluation gloutonne, tandis-que, la deuxième phase, améliore la solution donnée par la première phase en faisant des substitutions.

Pisinger et al.[71] ont proposé une méthode notée : LNS [25], dans laquelle, la solution courante est perturbé de façon significative et un grand nombre de voisins sont explorés en une seule itération. Bien que le temps de calcul soit généralement plus long que l'heuristique de recherche locale, cette méthode fonctionne très bien pour divers problèmes de tournées de véhicules.

Dans les sections qui suivent, nous ferons une classification du problème de tournées de véhicules et du problème d'orientation d'équipe en fonction des objectifs à optimiser, des contraintes spécifiques et de la qualité de solutions recherchées.

1.4 Classification des variantes du problème de tournées de véhicules

Dans cette section, nous offrons une vision globale des problèmes de tournées de véhicules. Cependant, nous ne ferons pas un état l'art exhaustif des méthodes de résolution pour le problème de tournées de véhicules, mais plutôt, nous présentons des des variantes qui permettront la bonne compréhension de la classification de ce problème. Pour une étude approfondie des problèmes de tournées de véhicules, nous recommandons l'ouvrage de Toth et Vigo [78].

Fondamentalement, le VRP est une généralisation du TSP à plusieurs voyageurs. D'un point de vue terminologique, on parlera de véhicules au lieu de voyageurs. En effet, il s'agit de visiter chaque point d'un graphe une et une seule fois avec d'une flotte de véhicules, partant tous d'un même dépôt.

Soit G = {V, E} un graphe, avec l'ensemble $V = \{0, ..., N\}$, comportant le dépôt [26] et les N noeuds du graphe qui représentent les clients. L'ensemble E quant à lui, représente les arcs ou les tronçons dans les chemins ou les tournées. Ainsi, nous savons qu'un grand nombre de problèmes réels peuvent être modélisés sous forme de graphe afin de simplifier leurs résolution. Ainsi, cette modélisation dépend

25. LNS : Large Neighborhood Search
26. Le dépôt est toujours représenté par le noeud 0 et noté D.

généralement des qualités de solutions recherchées, des objectifs à optimiser et des contraintes spécifiques.

Nous notons que plus souvent, l'objectif à optimiser, pour les problèmes de tournées de véhicules peut être soit :

— Minimiser les distances de parcours.
— Minimiser les temps de travail.
— Minimiser les coûts des tournées à planifier.
— Minimiser le nombre de véhicules à utiliser.
— Maximiser le gain à récolter.

Selon que l'on optimise un ou plusieurs objectifs, la qualité de solutions recherchées peut être différente. Ainsi, lorsqu'on traite des objectifs multiples, alors, on peut : soit définir un objectif agrégé, comme la combinaison linéaire des différents objectifs à optimiser, soit faire une optimisation multicritères avec un ensemble de solutions Pareto-optimales[46]. Les contraintes spécifiques peuvent être différentes, en fonction de l'entité (client, dépôt ou véhicule) du problème de tournées de véhicules. Ainsi, ces contraintes peuvent être : la capacité des véhicules, la satisfaction des demandes des clients, le respect des dépôts d'affectation, etc.

Une classification des problèmes de tournées de véhicules est souvent basée sur la distinction entre : les problèmes statiques pour lesquels toutes les données sont connues d'avance, les problèmes stochastiques où les données sont connues par une fonction de probabilité et les problèmes dynamiques pour lesquels des nouvelles données peuvent apparaitre pendant le traitement. Cependant, afin de classifier les différentes variantes étudiées durant cette thèse, nous les avons résumés dans le tableau suivant :

	Dépôt	Type de la flotte	Taille de la flotte	Nature de demande	Période	Coûts	Fenêtres de temps	Capacité
VRP	unique	homogène	plusieurs véhicules	déterministe	une	fixes		
CVRP	unique	homogène/hétérogène	plusieurs véhicules	déterministe	une	fixes		variable
VRPTW	unique	homogène	plusieurs véhicules	déterministe	une	fixes	dures/souples	
VRPB	Multiple	homogène	plusieurs véhicules	déterministe	une	fixes		
VRPPD	Multiple	homogène	plusieurs véhicules	déterministe	une	fixes		
SVRP	unique	homogène	plusieurs véhicules	stochastique	une	variables		
PVRP	unique	homogène	plusieurs véhicules	déterministe	plusieurs	fixes		
MDVRP	Multiple	homogène	plusieurs véhicules	déterministe	une	fixes		

TABLE 1.1 – Tableau des classifications du VRP et ses variantes

1.5 Classification des variantes du problème d'orientation d'équipe

Comme le problème de tournées de véhicules, le problème d'orientation d'équipe et ses variantes, sont classés en trois catégories : l'objectif à optimiser, les contraintes spécifiques, la qualité des solutions recherchées.

• D'une manière générale, l'objectif à optimiser pour le problème d'orientation d'équipe, est soit :

— Minimiser les coûts totaux de la distribution qui peuvent être : des coûts fixes (par exemple : les coût des véhicules, les salaires du personnel, les autres charges éventuelles telles que l'assurance, les taxes, licences de véhicules, etc.) ou des coûts variables (par exemple : les coûts qui varient selon : le gain récolté par les véhicules, la distance parcourue, la satisfaction des clients).
— Minimiser la distance totale de la tournée.
— Minimiser la durée totale de la tournée.
— Minimiser le nombre de véhicules utilisés.
— Maximiser le score total récolté lors de la tournée.

• Les contraintes sont multiples et varient en fonction des objectifs, néanmoins, nous citons ci-dessous les principales contraintes rencontrées dans le problème d'orientation d'équipe et ses variantes :

— Contraintes liées au véhicule (par exemple : la capacité du véhicule, localisation du véhicule, etc).
— Contraintes liées au client (par exemple : la priorité des clients lors d'un service, les fenêtres de temps du client, le score associé aux clients, etc.).
— Contraintes liées aux tournées (par exemple : la longueur maximale de la tournée en temps ou en distance).

• La qualité de solutions recherchées est généralement relative aux nombres des objectifs à optimiser.

Le problème d'orientation d'équipe peut être classifié en plusieurs types et catégories qui diffèrent en quelques aspects liés au type d'opération, le type de charge, au type de la flotte, la localisation des clients, au type de contraintes, au type de la fonction objectif et aux nombreux autres facteurs. Dans cette classification, nous nous considérons les paramètres suivants : le nombre de chemins ou la taille de la flotte, le nombre des dépôts, la nature des demandes, les contraintes, le type de la flotte de véhicule, l'horizon considéré, etc. Ainsi, nous avons résumé la classification du problème d'orientation d'équipe et ses variantes dans la table suivante :

	Dépôt	Type de la flotte	Taille de la flotte	Nature de demande	Période	Coûts	Fenêtres de temps	Capacité
TOP	Multiple	homogène	plusieurs véhicules	déterministe	une	variables		
CTOP	Multiple	homogène/hétérogène	plusieurs véhicules	déterministe	une	variables		variable
TOPTW	Multiple	homogène	plusieurs véhicules	déterministe	une	variables	dures/souples	
CTOPTW	Multiple	homogène/hétérogène	plusieurs véhicules	déterministe	une	variables	dures/souples	variable
MCTOP	Multiple	homogène/hétérogène	plusieurs véhicules	déterministe	une/plusieurs	variables		variable
MCTOPTW	Multiple	homogène/hétérogène	plusieurs véhicules	déterministe	une/plusieurs	variables	dures/souples	variable

TABLE 1.2 – Tableau des classifications du TOP et ses variantes

1.6 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art des problèmes de tournées de véhicules et des problèmes d'orientation d'équipe.

L'étude bibliographique que nous avons mené, essentiellement, en début de thèse, nous a permis dans un premier temps, de voir la diversité des variantes des problèmes de l'optimisation combinatoire, en particulier, le problème de tournées de véhicules et le problème d'orientation d'équipe. Ensuite, dans un second temps, de les classifier.

Ainsi, les deux classifications que nous avons proposé dans ce chapitre sont basées sur notre étude bibliographique et resteront certainement perfectibles par les nombreuses autres variantes et applications.

D'autre part, cette étude bibliographique, nous a aussi permis d'étudier plusieurs approches de résolution, surtout pour le problème de tournées de véhicules et le problème d'orientation d'équipe. En effet, dans le chapitre qui suit, nous présenterons de façon générale, les méthodes de résolution des problèmes de l'optimisation combinatoire.

Chapitre 2

Chapitre 2

Méthodes de résolution

Sommaire

Dans ce chapitre, nous rappelons d'abord certaines notions sur les graphes et la programmation linéaire. Ensuite, nous abordons les méthodes de résolution des problèmes d'optimisation combinatoire qui peuvent être classées en deux catégories : Les méthodes exactes qui garantissent l'optimalité de la solution et les méthodes approchées (heuristiques et métaheuristques) qui donnent des solutions approximatives.

Le principe de base d'un algorithme exact est en général, l' énumération de l'ensemble des solutions dans l'espace de recherche de manière implicite. Souvent, ce type d'algorithmes permet de résoudre que des problèmes de petite taille. Autrement, le temps de calcul augmente exponentiellement avec la taille du problème.

Ainsi, les méthodes approchées sont considérées comme une alternative intéressante pour résoudre les problèmes d'optimisation de grande taille étant donné que dans ce cas l'optimalité n'est pas primordiale. Parmi ces méthodes, nous citons les heuristiques et les métaheuristiques représentées essentiellement par des méthodes de voisinage (la recherche locale, le recuit simulé, la recherche tabou) et les algorithmes évolutionnaires (les algorithmes génétiques, les colonies de fourmis, etc.).

2.1 Introduction

En optimisation combinatoire, deux types de représentations peuvent être suivies pour la modélisation des problèmes de transport avant de les résoudre : une représentation sous forme d'un graphe ou une autre représentation sous forme d'un programme linéaire. Dans ce qui suit, nous rappellerons des définitions et des notions utiles pour la compréhension de la suite de cette thèse.

2.1.1 Notions sur les graphes

" Un petit dessin vaut mieux qu'un grand discours " Napoléon.

Bien souvent, la théorie des graphes essaie de mettre en pratique ce dicton.

En effet, c'est un réflexe naturel, d'abstraire une situation donnée en traçant, sur une feuille ou sur un tableau, des points pouvant représenter des villes, des individus ou des localités et les réliés par des flèches ou des arcs, symbolisant leurs relations. Ainsi, cette représentation figurative de la réalité par un dessin, présente plusieurs avantages :

— Elle permet de mettre en évidence la structure d'une situation donnée,
— D'un point de vue pratique, elle fournit une vision globale du problème, ce qui facilite le raisonnement et laisse libre court à l'intuition.
— Elle permet aussi, de faciliter l'implementation de l'ensemble des structures de données associées aux problèmes modélisés.

Un graphe $G = (V, E)$ est défini par deux ensembles finis, un ensemble V non vide de sommets ou noeuds et un ensemble E de couple de noeuds (i, j) alors appelés *arcs* dans le cas d'un graphe orienté ou *artes* dans le cas d'un graphe non-orienté (voir Berge [7]).

Ainsi, il est très pratique de dessiner les graphes sur un plan en représentant les noeuds par des sommets et les arêtes par des segments qui relient les points extrémités concernés. Aussi, il est possible d'associer des valeurs, entières ou réelles, positives ou négatives, aux sommets ou aux arêtes, dans ce cas, on parle alors, de graphe valué qui est généralement noté : $G = (V, E, C)$.

Concerne les problèmes de transport de façon générale, un sommet correspond à un dépôt ou à un client, tandis qu'une arête représente le trajet entre deux sommets.

Lorsqu'un trajet entre les deux sommets est représenté par un arête, cela veut dire que ce trajet, peut être effectué dans les deux sens, comme sur une voie à deux sens. En revanche, si le trajet est représenté par un arc (flèche), cela veut dire qu'il est à sens unique. D'autre part, les valeurs associées aux clients peuvent représenter la demande à satisfaire, la durée de service chez ce client, l'heure d'arrivée chez ce client, etc. Les valeurs associées aux arêtes peuvent correspondre aux durées de trajets entre les deux extremites, le coût du trajet, la probabilité qu'un incident survienne sur ce trajet, etc.

Dans les problèmes de transport présenté sous forme d'un graphe, on parle de tournées sur sommets quand le service doit être effectué sur les sommets (par exemple, lors de la livraison de colis chez les clients). Cependant, on parle de tournées sur arêtes lorsque le service est à effectuer le long d'une arête du graphe (par exemple, lors des salages ou nettoyages des routes).

Dans un graphe non-orienté, une chaine est une suite d'arêtes consécutives. Un cycle est équivalent a une chaine fermée. Ainsi, on dit qu'un cycle, est élémentaire lorsqu'il ne passe pas deux fois par un même sommet. Nous notons que la tournée effectuée par un véhicule correspond souvent à un cycle élémentaire.

2.1.2 Notions sur la programmation linéaire

On dit qu'un programme mathématique est présenté sous forme d'un programme linéaire (noté : PL) lorsque sa fonction-objectif et ses contraintes sont linéaires.

Un problème de programmation linéaire consiste à minimiser (ou à maximiser) une fonction linéaire sous certaines contraintes linéaires. Ainsi, une forme d'un programme linéaire dans le cas d'une minimisation, est la suivante :

$$(P)\begin{cases} \min\ f(x) \\ \\ \textit{sous contraintes} : \\ \mathrm{g}_i(\mathrm{x}) \leq 0,\ i = 1, ..., M \\ \mathrm{x} \in (x_1, x_2, ..., x_N) \in R \end{cases} \tag{2.1}$$

où les fonctions f et g_i sont des fonctions linéaires avec des variables x_i,..., x_N. La fonction f est appelée fonction-objectif ou fonction-économique et les conditions posées représentent les contraintes du problème (P). Une solution du problème (P) est un vecteur qui permet de satisfaire toutes les contraintes. Cependant, une solution peut être réalisable ou admissible, si elle permet de satisfaire une majorité de contraintes du problème (P). Le coût d'une solution est la valeur de la fonction-objectif f.

Une solution optimale du problème (P) est la solution qui minimise $f(x)$ parmi l'ensemble de toutes les solutions admissibles. Dans ce cas, on parle d'un optimum global.

D'autre part, on dit qu'un programme mathématique est présenté sous forme d'un programme linéaire en nombres entiers (noté : PLNE) lorsque les valeurs des variables du programme linéaire sont des nombres entiers. A noter qu'un programme mathématique peut aussi être présenté sous la forme d'un programme linéaire mixte (noté : PLM). Ce dernier, est obtenu lorsque certaines de variables du problème sont entières et d'autres pas.

Ainsi, modéliser un problème de tournées de véhicules sous forme d'un programme linéaire ou sous forme d'un programme linéaire en nombres entiers, revient à définir :

- Les variables des décisions à prendre, par exemple le choix du véhicule qui visite chaque client, l'ordre de visite des clients, etc.
- La fonction-objectif exprimée en fonction des variables à maximiser ou à minimiser.
- Les contraintes exprimées en fonction des variables à prendre en compte lors de la construction de solutions admissibles. Ces contraintes représentent souvent, les conditions ou les limites (physiques, logiques, temporelles, etc.)

2.2 Les Méthodes de résolution classiques

Les méthodes de résolution des problèmes d'optimisation comme les problèmes de transport sont généralement scindées en deux catégories : les méthodes exactes et les méthodes approchées (heuristiques et métaheuristiques).

Conçues pour trouver l'optimum d'un problème, les méthodes exactes ont souvent, des durées de calcul qui augmentent exponentiellement avec la taille des problèmes qu'elles essaient de résoudre. Elles doivent donc être arrêtées prématurement lors de la résolution. Dans ce cas, les méthodes exacte ne produisent que des minorants appelés bornes inférieures ou des majorants appelés bornes supérieures.

A l'inverse, les méthodes approchées sont conçues pour produire des solutions admissibles de bonne qualité avec un temps de résolution raisonnable, mais pas nécessairement optimales. Ainsi, les méthodes approchées comprennent les heuristiques constructives, les recherches locales qui donnent une suite de solution de coûts décroissants (ou croissants) jusqu' à l'optimum local, et enfin, les métaheuristiques qui utilisent divers mécanismes pour se sortir des optima-locaux.

Les méthodes exactes et les méthodes approchées sont complémentaires, car leurs résultats peuvent permettre de les évaluer mutuellement : ainsi, un faible écart entre les bornes inférieures et bornes supérieures signifie que l'optimum est proche,alors qu'un écart nul prouve que les valeurs atteintes sont optimales. Cette complémentarité entre les méthodes exactes et les méthodes approchées, fait qu'il est courant d'utiliser une de ces méthodes pour améliorer la performance de l'autre.

La figure 2.1 présente un organigramme classant les méthodes de résolutions des problèmes d'optimisation combinatoire en deux catégories à savoir les méthodes exactes et les méthodes approchées (ou approximatives).

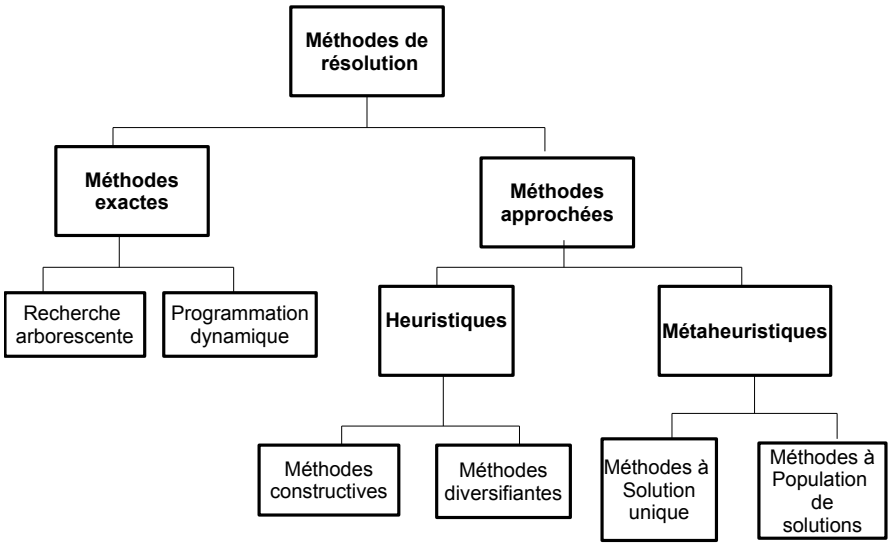

FIGURE 2.1 – Organigramme des méthodes de résolution

2.2.1 Les méthodes exactes

La plupart des problèmes de transport peuvent être présentés sous forme d'un PLNE, dans le but de minimiser (ou maximiser) une fonction linéaire à variables entières en respectant un ensemble de contraintes linéaires.

Le programme linéaire résultant de la relaxation continue d'un programme linéaire en nombres entiers peut être facilement résolu en utilisant l'algorithme du simplexe. Cependant, arrondir une solution continue pour obtenir une solution entière conduit souvent à une solution moins bonne qualité. Il est d'ailleurs facile de construire des exemples où l'optimum continu est très éloigné de l'optimum entier.

Ainsi, résoudre un PLNE nécessite l'application de méthodes dediées, telles que des méthodes de type séparation et évaluation [1][54, 63], des méthodes de coupes [2][66, 70], des méthodes par génération de colonnes[22, 80], la programmation dynamique[6, 8], la relaxation Lagrangienne[55], etc.

Ainsi, certaines des méthodes dediées à la résolution d'un PLNE peuvent être très efficaces dans le cas où, une borne supérieure de bonne qualité leurs est préalablement fournie, en général a l'aide de métaheuristiques efficaces.

Dans la suite de cette section, nous procédons à un petit état de l'art de quelques méthodes de résolution exactes comme la méthode de recherche arborescente par séparation et évaluation, la méthode par génération de colonnes et la programmation dynamique pour les problème de transport.

La méthode de recherche arborescente par séparation et évaluation

Les méthodes de recherche arborescente telles que l'algorithme de séparation et évaluation introduit par Land et Doig [53] sont basées sur l'idée d'énumerer implicitement les solutions possibles d'un problème présenté sous forme d'un PLNE pour trouver la meilleure solution. Cependant, l'énumération explicite demande beaucoup trop de temps : comme le montre Michel Minoux [60], un ordinateur capable d'évaluer une solution en 10^{-9} secondes aura besoin de 30 ans pour énumérer l'ensemble des solutions des $2n$ solutions potentielles d'un programme lineaire en nombres entiers avec $n = 60$ variables. Par conséquent, il est impératif de rechercher un principe algorithmique permettant de déterminer une solution optimale sans avoir à énumérer explicitement l'ensemble des solutions possibles. Pour cela, la procédure de séparation divise le problème en un certain nombre de sous problèmes qui ont chacun leur ensemble de solutions réalisables de telle sorte que tous ces ensembles forment un recouvrement de l'ensemble des solutions. Ce principe de séparation peut être répété pour chacun des ensembles de solutions obtenus tant qu'il y a des ensembles contenant plusieurs solutions. Ainsi, les ensembles de solutions construits peuvent être considérés comme des branches ou des noeuds d'un arbre. Cet arbre de l'ensemble de solutions est appelé arbre de recherche.

La procédure d'évaluation à pour objectif, la réduction de la taille de l'arbre de

1. Algorithme de *Branch and Bound*
2. Algorithme de *Branch and Cut*

recherche, en bloquant l'expansion des noeuds qui ne contiennent pas l'optimum. Ainsi, à chaque itération de la procedure de séparation, on calcule une borne supérieure ou inférieure, selon que l'on maximise ou l'on minimise le coût de la meilleure solution en chaque noeud. Cette borne est généralement obtenue en résolvant une relaxation (linéaire, Lagrangienne ou agrégée) du problème correspondant au noeud. En effet, lorsqu'on a de bonnes procédures de calcul de bornes, alors on peut savoir si le noeud correspondant contient l'optimum ou pas. Dans le cas où, le noeud ne contiendrait pas l'optimum, il serait donc inutile de continuer à explorer ce noeud. Donc, avoir de bonnes procédures de calcul de bornes permet d'accélérer la recherche de l'optimum.

Notons que l'exploration de l'arbre de recherche peut être éffectuée soit en largeur, soit en profondeur ou en donnant la priorité d'abord au meilleur noeud.

Des exemples de procédures de séparation et évaluation sur les noeuds pour des problèmes de tournees de vehicules et ses variantes sont disponibles dans le livre de Toth et Vigo [78], ainsi que, dans l'article de Fischetti et al.[28].

La méthode par génération de colonnes

La génération de colonnes est une méthode efficace permettant de résoudre des programmes linéaires ayant un nombre de variables (ou colonnes) très grand au point où, il est impossible d'énumérer ces variables de façon explicite. Ainsi, le principe de la méthode par génération de colonnes est de résoudre un tel problème en tenant compte d'un nombre réduit de variables, puisque de toute manière à l'optimum plusieurs variables seront hors base et nulles, c'est-a-dire que seul un ensemble de variables comptera pour résoudre le problème, et au pire, cet ensemble est égale au nombre de contraintes.

Une méthode par génération de colonnes initialise le programme linéaire avec un ensemble de colonnes de petite taille, qu'on appelle le plus souvent, le problème-maitre, qui sera résolu afin d'obtenir une solution. Ainsi, deux situations peuvent alors se présenter :

- Soit, toutes les colonnes ignorées ont un coût réduit positif ou nul : cela veut dire la solution obtenue sur le problème-maitre est réalisable et que son coût est l'optimum de la relaxation linéaire du problème original, donc même si, on avait pris en compte les colonnes ignorées, il n'y aurait pas d'impact sur la fonction-objectif et la solution serait la même.
- Soit, il existe des colonnes avec un coût réduit négatif parmi celles qui ont été ignorées : cela veut dire que si elles avaient été prises en compte, la solution obtenue serait différente et de coût inférieur à celui de la solution courante qui n'est donc pas optimale pour le problème original. Dans ce cas, des colonnes ayant un coût réduit négatif doivent être inserées dans le problème-maitre et ce dernier doit être résolu à nouveau.

D'une façon générale, l'efficacité d'une méthode par génération de colonnes depend sur de celle des deux procédures suivantes :

• La procédure permettant de résoudre chaque problème-maitre. Cette procédure doit être rapide étant donné qu'elle sera répétée à chaque itération.

• La procédure permettant de générer les colonnes ayant un coût réduit négatif. Cette procédure doit fonctionner sans expliciter toutes les colonnes ignorées.

La programmation dynamique

La programmation dynamique inventée par Bellman [6], qui l'a défini comme étant une méthode d'optimisation opérant par séquences et dont l'efficacité est basée sur le principe d'optimalité de Bellman, selon lequel une séquence optimale est composée de sous-séquences optimales. Ainsi, ce principe donne une méthode de résolution ascendante qui détermine la solution optimale d'un problème à partir des solutions de tous ses sous-problèmes : on débute par les plus petits sous-problèmes et on remonte vers les sous-problèmes de plus en plus difficiles en tirant profit des résultats des problèmes déjà obtenus pour que les calculs à effectuer restent simples et rapides, grâce à une formule de récurrence.

2.2.2 Les méthodes approchées

Les méthodes approchées ont pour objectif de produire des solutions réalisables de bonne qualité, pas nécessairement optimales pour des problèmes d'optimisation combinatoire. Les durées de calcul des méthodes approchées sont souvent très inférieures à celles des méthodes exactes. Ce qui rend les méthodes approchées plus attractives pour la résolution de problèmes industriels, c'est le fait qu'elles sont capable de traiter des instances de grande taille. Ainsi, l'efficacité des heuristiques est jugée de deux façons :

— De façon empirique, en comparant son résultat (c'est à dire : le temps de calcul et la valeur de la solution obtenue) avec la meilleure solution connue ou avec une borne inférieure ou supérieure (selon l'objectif du problème).
— De façon mathématique, en calculant le ratio dans le pire cas entre l'optimum et la valeur de la solution obtenue.

Dans ce qui suit, nous récapitulons, les méthodes les méthodes approchées les plus fréquents dans la littérature, qui ont été utilisé pour résoudre le problème de tournées en général.

Les méthodes constructives

Une heuristique constructive a pour objectif de fournir en un temps polynomial une bonne solution à un problème d'optimisation mais sans garantie d'optimalité. Les heuristiques plus connues pour les problèmes de tournées de véhicules et ses variantes, sont celles qui construisent une solution avec des principes simples. Nous pouvons citer :

L'heuristique du plus proche voisin qui construit une seule tournée à la fois et dont le principe est que la tournée courante commence au dépôt, puis à chaque ité-

ration, on ajoute le client disponible le plus proche. Ainsi, la tournée ne retourne au dépôt que lorsqu'aucun client supplémentaire ne peut être ajouté à la tournée. Dans ce cas, une nouvelle tournée peut commencer. Cette heuristique est adaptée aux instances dans lesquelles, les clients sont regroupés par groupe (ou cluster).Cependant, elle est moins efficace sur des instances où, les clients sont dispersés tout autour du dépôt.

L'heuristique de Clarke et Wright [15] qui construit les tournées à partir d'un seul client pour tous les autres clients, puis, elle fusionne ces tournées deux par deux, tant que cela permet de réduire le coût de la solution courante et que chaque nouvelle tournée respecte les contraintes du problème. Dans cette heuristique, on commence par fusionner les tournées qui réduisent le plus de coûts. Cette heuristique présente un double avantage, celui de réduire à la fois le nombre de véhicules utilisés et le coût total. C'est pourquoi, elle est efficace sur la plupart des instances.

Il existe des heuristiques constructives à deux phases : celles du type *route first - cluster second* qui correspond à construire d'abord les chemins et former les groupes après, ou, celles du type *cluster first - route second* qui équivaut à former d'abord les groupes et construire les chemins après.

Les premières construisent un tour géant contenant tous les clients, puis découpent ce tour en tournées réalisables, c'est à dire des tournées qui respectent les contraintes du problème. C'est le cas, par exemple, dans l'heuristique d'Ulusoy [79].

Les secondes quant-à elles regroupent d'abord les clients en groupes pouvant être servis par un seul véhicule avant de construire la meilleure tournée pour chacun des groupes. Le regroupement de clients peut être selon plusieurs critères par rapport au dépôt ou par rapport à la compatibilité avec la capacité des vehicules. Néanmoins, on peut aussi utiliser d'autres techniques pour regrouper les clients, par exemple, la technique de K-Means. Pour plus de details sur la technique de K-Means, le lecteur peut se référer à l'article de Hartigan *et al.*[34].

Les méthodes diversifiantes

Les méthodes diversifiantes sont des recherches dans des voisinages proches ou lointains permettant d'améliorer la solution courante. Ainsi, dans le cas où la recherche est effectuée dans un voisinage proche, la méthode est appelée : la recherche locale et est notée, LS (*Local Search*) et dans le cas où la recherche est effectuée dans un voisinage plus large, la méthode est appelée : la recherche dan un large voisinage et elle est notée, LNS (*Large Neighborhood Search*). Néanmoins, on peut aussi, mixer tous les voisinages effectuée une recherche, dans ce cas, on parle de recherche dans un voisinage variable, notée : VNS (*Variable Neighborhood Search*). Pour plus de details sur les méthodes diversifiantes les plus répandues, le lecteur peut se référer au livre de Siarry *et al.*[74].

D'une manière générale, lorsqu'on veut améliorer la solution obtenue par une méthode diversifiante, il est possible soit :

— D'exécuter plusieurs méthodes diversifiantes et garder la meilleure solution obtenue.

— De rendre aléatoire une composante de la méthode et appliquer cette dernière plusieurs fois, en gardant à chaque fois la meilleure solution obtenue.
— D'appliquer une autre heuristique en utilisant la meilleure solution obtenue par la précédente méthode, comme une solution initiale.

La plupart des méthodes diversifiantes sont basées sur le principe la recherche locale qui est appliquée différents voisinages. Par consequent parler des méthodes diversifiantes revient tout simplement à parler de la recherche locale qui a pour but de vérifier, s'il existe de meilleures solutions dans le voisinage d'une solution donnée. Le voisinage est defini comme l'ensemble des solutions qui peuvent être obtenues en appliquant une transformation à la solution courante. Ainsi, pour éviter l'enumération complète de l'espace des solutions, la recherche locale applique des permutions ou des mouvements qui réduisent le coût total et elle s'arrête dès qu'il n'existe plus de mouvement ou permutation permettant d'améliorer la solution.

Les principales procédures des permutations ou mouvement utilisées dans la recherche locale sont : la recherche du meilleur d'abord (appelée : *Best-first Search*), cela veut dire qu'on construit l'arbre de recherche en choisissant à chaque fois le premier voisin qui permet d'améliorer la solution courante et la recherche du meilleur voisin (appelée : *Depth-first search*) qui construit l'arbre de recherche en choisissant à chaque fois le meilleur parmi tous les voisins du même niveau.

Dans la procédure du *Best-first Search*, le premier mouvement améliorant trouvé est immediatement appliqué, tandis- que dans la procédure du *Depth-first search*, l'ensemble des mouvements possibles à chaque itération sont évalues avant que le meilleur d'entre eux ne soit choisi et appliqué.

Ainsi, notons que la meilleure solution obtenue à la fin d'une recherche locale est appelée optimum local. Selon quel'on minimise ou on maximise, l'optimum local peut être appelé : minimum local ou maximum local.

De nombreuses autres techniques d'exploration ont été proposées dans la littérature, les plus classiques sont les techniques de k-optimisations [4, 35].

Les métaheuristiques

Comme les heuristiques, les métaheuristiques ont pour objectif de donner des solutions admissibles de bonne qualité et pas forcement optimales pour les problèmes d'optimisation combinatoire.

Cependant, les métaheuristiques sont des heuristiques plus puissantes qui sont capables de s'extraire de minima locaux, ceci les rend plus performantes que de simples heuristiques. Ainsi, de façon générale les métaheuristiques reposent sur deux mécanismes : l'intensification qui est le mécanisme qui leurs permet de trouver l'optimum local de la solution courante (l'intensification se ramène souvent à une recherche locale) et la diversification qui est le mécanisme permettant aux métaheuristques de sortir du voisinage de ce minimum local (la diversification varie d'un algorithme à l'autre).

Les meilleures métaheuristiques permettent d'obtenir d'excellents résultats sur des problèmes de grandes tailles avec des temps de calculs raisonnables contraire-

ment aux méthodes exactes. Nous avons remarqué deux grandes familles de métaheuristiques : les métaheuristiques qui manipulent une solution à la fois, appelées métaheuristiques à "solution unique" et les métaheuristiques qui manipulent une multitude de solutions, appelées métaheuristiques à "population de solutions".

Les métaheuristiques à solution unique [3]

Les métaheuristiques à solution unique les plus connues dans la littérature[74] sont : la procédure de recherche gloutonne randomisée, notée : GRASP (*Greedy Randomized Adaptive Search Procedure*), la recherche locale itérative, notée : ILS (*Iterated Local Search*), la recherche à voisinage variable, notée : VNS (*Variable Neighborhood Search*), le recuit simulé, noté : SA (*Simulated Annealing*) et la recherche taboue, notée : TS (*Tabu Search*). Nous résumons les principes de ces quelques métaheuristiques à solution unique dans les lignes suivantes :

Le GRASP qui est introduit dans Feo et al. [27], crée une nouvelle solution à chaque iteration à l'aide d'un algorithme glouton aléatoire et améliore cette solution avec une recherche locale. La métaheuristique (GRASP) s'arrête après un nombre prédefini d'itérations et la meilleure solution trouvée lors des différentes itérations est sauvegardées comme résultat. Il s'agit donc d'un échantillonnage aléatoire de l'espace des optima-locaux.

En ce qui concerne la recherche locale itérative (ILS), décrite par exemple dans l'article de Cowling et Keuthen [17], le principe est similaire à celui du GRASP, sauf que la solution initiale de chaque itération est obtenue en appliquant une perturbation aléatoire à la meilleure solution trouvée jusque-là. L'idée est de garder certaines caractéristiques de la meilleure solution au cours d'une marche aléatoire dans l'espace des solutions de façon à obtenir d'autres solutions encore meilleures.

La recherche à voisinage variable (VNS) présentée dans Mladenovic et Hansen [61], est également inspirée du GRASP, mais utilise une série de k voisinages. Le tirage au sort de la nouvelle solution ainsi que la recherche locale s'effectuent alternativement sur les différents voisinages. Sachant que plusieurs variables d'une solution localement optimale gardent la même valeur dans une solution globalement optimale, alors explorer différents voisinages permet de remettre en cause un ensemble de variables à la fois, dans le but de retrouver l'optimum global.

Le recuit simulé (SA) [74], est inspiré du processus de recuit thermique utilisé en métallurgie dans le but de réduire l'energie E d'un matériau en alternant les phases de refroidissement lent et les phases de réchauffage (recuit). Il a été introduit en optimisation combinatoire comme métaheuristique dans Kirkpatrick et al. [49].

La recherche taboue (TS)[74], est une métaheuristique qui consiste à parcourir tout le voisinage de la solution courante, puis à lui appliquer des interdictions sur le retour, même si cette dernière dégrade la fonction-objectif. Cette métaheuristique à été introduite dans Glover et Laguna [30].

3. Mono-solutions

Les méthodes à population de solutions [4]

Les métaheuristiques à population de solutions les plus connues sont : la recherche dispersée, notée : SS (*Scatter Search*), l'optimisation par essaims de particules, notée : PSO (*Particle Swarm Optimization*), l'optimisation par les colonies de fourmis , notée : ACO (*Ant Colony Optimization*) et l'algorithme génétique, noté : GA (*Genetic Algorithm*). (Pour plus de details sur ces métaheuristiques, voir Siarry*et al.* [74]) Dans ce qui suit, nous rappellerons les principes de certaines métaheuristiques dites à population de solutions :

La recherche dispersée (SS), est une méthode à population de solutions qui utilise la recherche locale et d'autres opérateurs permettant une combinaison de solutions (voir Marì et al. [57]). Ainsi, à chaque itération, un sous-ensemble de solutions dont les éléments répondent à des critères de qualité et diversité est sélectionné et, est appelé ensemble de référence. Les solutions des différents ensembles de référence sont alors combinées entre elles pour donner de nouvelles solutions, qui sont ensuite améliorées par la recherche locale. Ces nouvelles solutions remplacent des éléments moins bons de l'ensemble de référence, si elles permettent de satisfaire des critères de qualité et de diversité. Si aucune nouvelle solution n'est admise dans l'ensemble de référence, alors une re-génération partielle de cet ensemble est effectuée et une nouvelle itération commence. L'algorithme s'arrete après un nombre maximum d'itérations et la meilleure solution ayant fait partie de l'ensemble de référence est alors sauvegardé comme résultat.

L'algorithmes d'optimisation par essaim de particules (PSO), est inspiré des essaims d'insectes et de leurs mouvements coordonnés (voir Siarry*et al.*[74]). De la même manière que ces oiseaux se déplacent en groupe pour trouver de la nourriture ou pour éviter les prédateurs, les algorithmes à essaim de particules recherchent des solutions pour un problème d'optimisation. Les individus de l'algorithme représentent des solutions et sont appelés particules, tandis-que la population entière est appelée essaim. L'algorithme PSO a été introduit dans Eberhart et Kennedy [24], dans cet algorithme, une particule décide de son prochain mouvement en fonction de sa propre expérience qui est dans ce cas la mémoire de la meilleure position qu'elle a occupée et en fonction de son meilleur voisin. Le principe de cet algorithme dits à essaim de particules, est qu'à partir d'optima-locaux, l'ensemble des particules va converger vers la solution optimale globale du problème traité. L'algorithme du PSO s'arrête alors, dès qu'une convergence est trouvée, bien que l'optimum ne soit pas garanti. Ainsi, l'algorithme d'optimisation par essaim de particules peut s'appliquer aussi bien à des données discrètes qu'à des données continues.

L'algorithme de colonies de fourmis (ACO), est inspiré du comportement des fourmis qui cherchent de la nourriture. Lorsqu'elles en trouvent, elles marquent le chemins y conduisant en laissant des quantités de phéromones [5] dépendantes de l'éloignement de la source trouvée. Ainsi, les autres fourmis sont averties et alors attirées dans les directions les plus prometteuses. Avec le temps, les parcours condui-

4. Multi-solutions
5. Substances chimiques produites par les fourmis

sant aux meilleures sources de nourriture sont de plus en plus fréquentés et donc les taux de phéromones augmente à chaque passage de fourmis. Cependant, les chemins peu fréquentés voient leur taux de phéromones diminuer par évaporation. L'algorithme de colonies de fourmis, introduits dans Dorigo et al. [23], consiste à assimiler les trajets réalisés par les fourmis à des parcours de graphe et la qualité de la source à la valeur de la fonction-objectif. L'initialisation du graphe se fait en attribuant des taux de phéromones nuls aux arcs. Les fourmis sont représentées par des agents construisant la solution. A chaque itération, ils avancent dans le graphe en effectuant des choix soumis à des probabilités sur les arcs à traverser. La construction de leur chemin est donc faite en favorisant les arcs fortement marqués de phéromones. Ensuite, selon la valeur de la fonction-objectif obtenue, les taux de phéromones sont mis à jour. L'algorithme s'arrete après un nombre prédéfini d'itérations. Pour plus de details sur les fourmis artificielles, (voir le livre de Siarry *et al.*[74]).

L'algorithme génétique (GA), présenté dans Holland [43], est inspiré de la génétique et de l'évolution naturelle des espèces qui utilisent des croisements, des selections et des mutations pour faire évoluer vers l'optimum, une population de solutions représentées sous forme de chromosomes (pour plus de details sur GA, le lecteur peut se référer au livre [74]). Dans l'algorithme GA, à chaque itération deux chromosomes parents de la population sont sélectionnés en favorisant les plus prometteurs, puis croisés pour obtenir un ou deux chromosomes enfants combinant les caractéristiques des parents. Cependant, des mutations peuvent être appliquées aux enfants à titre de diversification afin d'éviter une convergence prématurée de la population. L'algorithme génétique s'arrête après un nombre prédéfini d'itérations. Pour les problèmes de tournées de véhicules et ses dérivées, l'algorithmes génétique est moins efficace et donc, il doit être combiné avec une recherche locale pour intensifier la recherche. Ainsi, lorsqu'on combine l'algorithme génétique avec une recherche locale, on obtient un algorithme hybride appelé algorithme mémétique, noté : MA (*Memetic Algorithm*). Pour plus de détailles sur les algorithmes mémétiques, (voir le livre de Moscato *et al.*[62]).

2.3 Conclusion

Dans ce chapitre, nous avons présenté les principales méthodes de résolution des problèmes d'optimisation combinatoire en nous basant sur l'organigramme de la figure 2.1. Ainsi, nous avons souligné l'évolution remarquable au cours des dernières décennies, des méthodes de résolution des problèmes de l'optimisation combinatoire en général, et des problèmes de transport en particulier.

Les modèles et algorithmes présentés sont des tremplins sur lesquels de améliorations dans la résolution des problèmes d'optimisation combinatoire peuvent s'appuient afin de toujours trouver de meilleures solutions possibles.

Dans le chapitre suivant, nous avons appliqué avec succès une combinaison de quelques méthodes diversifiantes pour résoudre approximativement le problème d'orientation d'équipe.

Chapitre 3

Chapitre 3

Une heuristique en deux phases pour le problème d'orientation d'équipe

Sommaire

L'essentiel de ce chapitre est consacré à la description de la méthode que nous proposons pour résoudre le problème d'orientation d'équipe (TOP [1]).

L'approche proposée ici est une combinaison de deux phases complémentaires : La première phase est appliquée pour construire une solution de départ considérée comme chemin élite tandis que la seconde phase est utilisée pour construire une solution réalisable pour le problème d'orientation d'équipe en améliorant la solution fournie par la première phase. Nous avons testé cette heuristique sur des instances du problème d'orientation d'équipe connues dans la littérature. Ces instances peuvent être vues comme les variantes de celles du célèbre problème de tournée de véhicule.

3.1 Introduction

Le problème d'orientation d'équipe, noté TOP, est un problème d'optimisation combinatoire de la famille du problème de tournées de véhicules qui a été introduit pour la première fois dans Chao et al. [44].

Ainsi, le problème d'orientation d'équipe est une extension naturelle du problème d'orientation simple pour une équipe (voir Chao et al. [44]). Ce problème appartient

1. TOP := Team Orienteering Problem

au domaine du sport de compétition dans lequel un concurrent quitte un point de départ spécifique et tente de visiter autant de points de contrôle que possible, avant d'atteindre un point d'arrivée spécifique ; en respectant la contrainte de temps qui lui est impartie.

Sachant que chaque point de contrôle visité est caractérisé par un score, le but du jeu est que chaque concurrent cherche à maximiser son score tout en respectant la contrainte de temps.

La suite de ce chapitre est organisée en cinq sections respectivement dans lesquelles, nous rappellerons le principe de problème d'orientation d'équipe ; nous donnerons un bref état de l'art qui contiendra quelques applications pratiques de ce problème et quelques approches utilisées dans la littérature pour résoudre ce problème ; nous présenterons un modèle mathématique pour le problème d'orientation d'équipe ; nous décrirons l'algorithme à deux phases que nous proposons afin de résoudre le problème ; dans la partie expérimentale, nous évaluerons les performances de cet algorithme en comparant sa solution à celle des meilleurs algorithmes de la littérature ; et enfin, nous terminerons ce chapitre par une conclusion dans laquelle nous analyserons les résultats expérimentaux de cette approche et donnerons quelques pistes d'amélioration pour l'heuristique proposée.

3.2 Le problème d'orientation d'équipe

Comme déjà indiqué, les problèmes de tournées sont parmi les problèmes les plus étudiés dans l'optimisation combinatoire.

Dans la pratique, les problèmes de tournées considèrent une flotte de véhicules pour visiter un ensemble de clients. Dans la plupart des versions de cette famille de problèmes, tous les clients doivent être visités exactement une fois. Cependant, dans de nombreuses situations du monde réel, il y'a des contraintes qui nous obligent à choisir les clients à visiter. Le problème d'orientation d'équipe est un problème de tournées qui modélise une de ces situations.

Dans le problème d'orientation d'équipe, chaque client i se voit affecter un score p_i et chaque tournée k a une durée maximale T_{max}. Voilà pourquoi, le choix des clients est réalisé en fonction du rapport entre son score p_i et le respect de la contrainte de temps (durée de la tournée).

Ce problème peut être rencontré dans plusieurs situations concretes et a d'importantes applications. Ainsi, Tang et Miller-Hooks [76] ont décrit une application du problème d'orientation d'équipe comme la tournée des techniciens pour servir des clients. Sachant que, chaque chemin représente la visite d'un technicien et ce dernier ne peut travailler que pour un certain nombre d'heures par jour, alors, tous les clients qui ont besoin de services ne pourront pas être inclus dans les fenêtres horaires des techniciens.

Dans un autre contexte, Butt et Cavalier [10] ont proposé une application du problème d'orientation d'équipe pour le recrutement des athlètes dans les écoles secondaires. Le recruteur doit visiter plusieurs écoles dans un nombre donné de

jours. Ainsi, les scores sont assignés à chaque école, en tenant compte du potentiel de recrutement. Le recruteur ayant un nombre limité de jours pour recruter, doit donc choisir de visiter les écoles ayant un grand potentiel afin de maximiser la probabilité de recrutement.

Parmi les approches utilisées dans la littérature pour résoudre le problème d'orientation simple et le problème d'orientation d'équipe, on distingue les approches exactes et les approches approximatives qui nous avons présenté dans le chapitre 2 de cette thèse.

La section suivante présente la formulation mathématique du problème d'orientation d'équipe que nous proposons. Cette formulation est basée sur les modèles présentés dans [81], [72] et [11].

3.3 Formulation

Le problème d'orientation d'équipe étant la généralisation du problème d'orientation d'équipe simple [2] où l'équipe est composée de M véhicules (voir [44]).

Ainsi, le but est de construire M chemins qui maximisent le score (gain) p_i en respectant la contrainte de temps de chaque chemin k (voir [44, 76]).

Formellement, le problème d'orientation d'équipe peut se representer sous forme d'un graphe $G = (V, E)$, où

— V est l'ensemble des sommets qui représentent les clients.

— E est l'ensemble des arcs (i, j), $\forall$i$\in V$ et $\forall$j$\in V$.

Comme considéré dans l'article de Bouly et al. [33] :

— Chaque client i possède un score(profit) p_i.

— Chaque chemin k est représenté par une liste de $|N|$ clients ordonnés, tel que : $k = (k_1, ...k_{|N|})$.

— Chaque chemin commence au sommet de départ et finit au sommet d'arrivée. Nous notons que les sommets de départ "D" et d'arrivée "A" sont bien identifiés, sachant que $D \in V$ et $A \in V$.

Ainsi, le score (ou profit) $P(k)$ récolté sur un chemin k est noté par :

$$P(k) = \sum_{i \in k} p_i \tag{3.1}$$

et le coût du chemin k est :

$$C(k) = C_{D,k_1} + \sum_{i=1}^{|N|-1} C_{k_i,k_{i+1}} + C_{k_{|N|},A} \tag{3.2}$$

Dans le cas d'une flotte de M véhicules, la solution sera un ensemble de chemins dans lequel chaque client reçoit une seule visite. Pour chaque chemin k, la solution

2. On parle de problème d'orientation d'équipe simple lorsque l'équipe est composé d'un seul véhicule

est réalisable si et seulement si :

$$C(k) \leq T_{max} \tag{3.3}$$

Où, T_{max} représente le temps maximum du trajet.

Les variables de decision permettant de modéliser le problème d'orientation d'équipe sont les suivantes :

$$x_{ijk} = \begin{cases} 1 & \text{si le client } i \text{ est visité avant le client } j \text{ dans le chemin } k \\ 0 & \text{sinon} \end{cases} \tag{3.4}$$

$$y_{ik} = \begin{cases} 1 & \text{si le } i\text{-ième client est visité dans le chemin } k \\ 0 & \text{sinon} \end{cases} \tag{3.5}$$

$$u_{ik} = \begin{cases} 0 & \text{la position du client } i \text{ est dans le chemin } k \\ 1 & \text{sinon} \end{cases} \tag{3.6}$$

Le modèle mathématique pour le problème d'orientation d'équipe peut être formulé comme suit :

$$\max \sum_{k=1}^{M} \sum_{i=1}^{N} p_i y_{ik} \tag{3.7}$$

sous contraintes :

$$\sum_{k=1}^{M} \sum_{j=1}^{N} x_{1jk} = \sum_{k=1}^{M} \sum_{i=1}^{N} x_{iNk} = M; \tag{3.8}$$

$$\sum_{k=1}^{M} y_{lk} \leq 1; \qquad \forall\ l = 1, ..., (N); \tag{3.9}$$

$$\sum_{i=1}^{N-1} x_{ilk} = \sum_{j=2}^{N} x_{ljk} = y_{lk};\ \forall l = 2, ..., N-1; \forall k = 1, ..., M; \tag{3.10}$$

$$\sum_{i=1}^{N} \{T_i y_{ik} + \sum_{j=1}^{N} t_{ij} x_{ijk}\} \leq T_{max};\ \forall k = 1, ..., M; \tag{3.11}$$

$$1 \leq u_{ik} \leq N;\ \forall i = 1, ..., N;\ \forall k = 1, ..., M; \tag{3.12}$$

$$u_{ik} - u_{jk} + 1 \leq (N-1)(1 - x_{ijk});\ \forall i, j = 1, ..., N;\ \forall k = 1, ..., M; \tag{3.13}$$

$$x_{ijk}, y_{ik} \in \{0, 1\};\ \forall i, j = 1, ..., N;\ \forall k = 1, ..., M; \tag{3.14}$$

— La fonction objective (3.7) cherche à maximiser le gain total récolté par l'équipe.
— La contrainte (3.8) garantie que chaque chemin commence au sommet de depart et finit au sommet d'arrivée.
— La contrainte (3.9) assure que chaque client soit visité au plus une seule fois.
— La contrainte (3.10) garantie la connectivité de chaque chemin.
— La contrainte (3.11) donne la longueur maximale pour chaque chemin.
— Les contraintes (3.12) et (3.13) sont utilisées pour éviter les cycles.
— La contrainte (3.14) montre l'intégrité des variables de decision.

3.4 Une heuristique pour le problème d'orientation d'équipe

L'algorithme que nous proposons dans ce chapitre (noté : GAL [3]) afin de résoudre le problème d'orientation d'équipe s'inspire de deux algorithmes classiques qui ont déjà été appliqués avec succès à de nombreux autres problèmes d'optimisation combinatoire. Ces deux algorithmes sont : l'algorithme de recherche par faisceaux, noté : BSA [4](voir Hifi et Saadi [41]) et l'algorithme de séparation et évaluation, noté : B&B [5](voir Hifi [36]).

Cette approche a pour objectif, d'éviter l'énumération exhaustive en effectuant une recherche partielle de l'espace de solution. Ainsi, dans cette recherche partielle, seul un sous-ensemble de clients est sélectionné pour une ramification plus loin. Les autres clients sont simplement "ignorés [6]" et aucun retour en arrière ne sera effectué. Le nombre de clients sélectionnés à chaque niveau correspond à la largeur de l'espace de recherche partielle exploré. Les clients sélectionnés sont ceux ayant un fort potentiel pour aboutir à la meilleure solution. Le potentiel d'un client est évalué par une fonction d'évaluation dont le rôle est de fournir un bon mécanisme de séparation des clients à chaque niveau de l'espace développé. Cette fonction d'évaluation correspond à l'élément intelligent de ce type de méthodes de résolution.

3.4.1 Définition d'une solution partielle

Nous rappelons que le problème d'orientation d'équipe consiste à définir M chemins correspondant aux M membres de l'équipe et sachant que chaque chemin est

3. GAL := Global Algorithm
4. BSA := Beam Search Algorithm
5. B&B := Branch and Bound Algorithm
6. Un client est dit "ignoré" dans un arbre de recherche, si aucune ramification ne sera faite à partir de ce client

Algorithm 1 L'algorithme *GAL*

Entrées: Une instance du TOP.
Sorties: Une bonne solution du TOP.

1: **Tant que** (le critère d'arrêt sur le temps de résolution n'est pas atteint) **faire**
2: Lancer la première phase (voir Algorithme 2).
3: Lancer la deuxième phase (voir Algorithme 3).
4: Lancer la procédure d'affectation des clients aux chemins.
5: **fin du « Tant que »**
6: **Retourner** Une solution S pour le TOP.

limité par une contrainte de temps. Le but du problème est de maximiser la somme des scores collectés par toute l'équipe.

Dans ce contexte, une *solution partielle* du problème d'orientation d'équipe est représentée par :

• Une paire de clients (i, j) qui signifie que les clients i et j se trouvent successivement sur le même chemin, le score p_{ij} de la solution partielle qui correspond au score du client j, et enfin, le temps t_{ij} qui correspond au temps de déplacement entre le client i et le client j.

Dans notre cas, la *solution partielle* du problème est représentée par une paire de clients et la première *solution partielle* commence toujours à partir du client de départ (noté : D) identifié dans l'instance du problème et la dernière *solution partielle* se termine toujours par le client d'arrivée (noté : A) identifié aussi dans l'instance du problème d'orientation d'équipe.

Ainsi, nous définissons un chemin (ou solution) réalisable comme une séquence de paires de clients.

Afin de faciliter la compréhension de nos algorithmes, nous introduisons quelques notations :

— La première solution partielle représentant le debut du chemin est notée $PathStart$ et la dernière solution partielle représentant la fin du chemin est noté $PathEnd$.
— Les solutions partielles entre $PathStart$ et $PathEnd$ correspondent à un ensemble ordonné de clients qui est noté $GeneralNodesSet$. Ce dernier représente l'ordre de la tournée d'un client vers le suivant (ce que nous avons aussi appelé : connectivité du chemin solution).
— L'ensemble contenant les meilleures solutions partielles est noté $BestSet$.
— La procedure permettant l'affectation des clients aux différents chemins dans nos algorithmes est notée $AffecNodeProcedure$.

3.4.2 Première phase de l'algorithme

La première étape de cette méthode nous permet de construire un chemin unique. La procédure qui permet de construire ce chemin est détaillée dans l'algorithme 2.

En effet, cette phase permet à l'algorithme d'éviter l'énumération exhaustive en effectuant une recherche gloutonne dans l'espace de recherche. Ainsi, à chaque étape

de l'algorithme, un sous-ensemble de solutions partielles est choisi en utilisant une stratégie de sélection gloutonne. Ensuite, le branchement suivant est réalisé en se basant sur le sous-ensemble de solution, considéré comme élites.

En effet, la sélection de la solution partielle prometteuse est évaluée par une fonction d'évaluation dont le rôle est de fournir une meilleure solution partielle incrémentée d'un client.

Nous décrivons ici les principales étapes utilisées par la première phase de l'algorithme pour évaluer les solutions partielles prometteuses.

La première phase de la méthode est donc basée sur les deux étapes suivantes :

— Etape 1. Sélection d'un sous-ensemble de solutions partielles : cette étape effectue une comparaison entre les scores générés par chaque client visité et elle sélectionne les sous-ensembles de solutions partielles qui respectent la contrainte de temps.

— Etape 2. Sélection de la meilleure solution partielle : cette étape considère toutes les solutions partielles générées par l'étape 1 et sélectionne la meilleure solution à développer.

Dans l'algorithme 2 qui illustre la première phase de l'algorithme proposé, nous montrons comment cette première phase de la méthode permet de fournir un chemin unique. La fonction d'évaluation prend en compte les critères suivants :

(a) : Le score du prochain client à visiter (critère1).

(b) : Le temps nécessaire pour atteindre le prochain client (critère2).

(c) : Un rapport entre le score du prochain client et le temps à consommer pour atteindre le prochain client à visiter (critère1 / critère2).

3.4.3 Deuxième phase de l'algorithme

Dans cette deuxième étape, nous construisons une solution réalisable finale pour le problème d'orientation d'équipe.

Nous rappelons qu'une solution est dite réalisable lorsque la construction de tous les chemins est achevée et qu'aucun chemin ne viole la contrainte de temps.

Dans une solution réalisable pour le problème d'orientation d'équipe, un client ne peut appartenir qu'à un et un seul chemin et la valeur de la solution réalisable est la somme des scores de tous les chemins.

Notons que cette deuxième phase de la méthode utilise l'ensemble des solutions partielles fournies par l'algorithme 2 pour construire une solution de départ. Par ailleurs, cette solution réalisable [7] est représentée par un ensemble de chemins indépendants.

Sachant que les clients de départ et d'arrivée appartiennent à tous les chemins. Dans cette phase, l'algorithme commence par construire une solution réalisable à partir de la solution partielle de départ contenant le client de départ.

A chaque itération du processus de résolution, l'algorithme effectue les deux étapes de sélection de la solution partielle. Ensuite, il effectue une affectation des

7. C'est à dire le chemin unique

Algorithm 2 Première phase de l'algorithme : Construction du chemin unique.

Entrées: Une instance du TOP représenté par un graphe $G = (V, E)$;
Sorties: Un chemin Unique k.

Mettre $Z = 0$ (Profit) et $T = 0$ (Temps) ;
Initialisation
$BestSet = \emptyset$ et $Set = V$;
si Une solution initiale réalisable S_k est disponible
alors
$Z =$ Z (S_k) et T = T(S_k) ;
sinon
Mettre $Z = 0$ et $T = 0$;
finsi
Sélectionner la meilleure *solution partielle* qui maximise le profit
Créer la *solution partielle* de départ ($PathStart$) ;
$BestSet \leftarrow PathStart$
Supprimer $PathStart$ de l'ensemble Set ;
Sélectionner la meilleure $PathStart$ dans l'ensemble $BestSet$
Considerer cette meilleure solution partielle comme $PathStart$ courante à explorer ;
Supprimer la $PathStart$ courante de l'ensemble $BestSet$;
Etapes itératives
Tant que $BestSet \neq \emptyset$
faire
Effectuer une sélection des *solutions partielles* courantes et déterminer la meilleure ;
Insérrer la nouvelle *solution partielle* dans $BestSet$
Mettre à jour Z et T.
fin du « Tant que »
Résultat
Retourner Un chemin k avec le profit $Z(k)$ et le temps $T(k)$.

clients à chaque chemin en cours de construction. Cette affectation dépend du score généré par chaque solution partielle et le temps nécessaire pour se rendre de la dernière solution au client à ajouter à celle-ci.

L'obtention de la solution réalisable présentée dans l'algorithme 3 crée une liste de solutions partielles visitées par chaque chemin. Ceci est utilisé pour gérer l'interaction entre les chemins.

A l'initialisation de l'algorithme, la liste des clients utilisée (noté : *UsedCustomersSet*) est vide.

Ainsi, la construction de l'arbre de recherche commence au client de départ afin de choisir les clients qui seront alloués pour chaque chemin. Cette allocation est basée sur le score associé à chaque client, ainsi que la distance entre le client de départ et le client sélectionné. Nous notons que ce choix de critère est facilement modifiable et paramétrable.

La liste de l'ensemble des clients utilisés (noté : *UsedCustomersSet*) est mise à jour après chaque sélection d'un nouveau client.

Après la phase d'initialisation, l'algorithme entre en phase itérative. Ainsi, à chaque iteration, il crée au plus M chemins. Les principales étapes dans chaque itération sont :

1. Pour chaque chemin à construire, effectuer la première phase de la méthode présentée dans la section 3.4.2.

2. L'algorithme met le résultats dans la liste commune à tous les chemins qui est notée *GeneralNodesSet*.
3. L'algorithme genère une affectation des clients se trouvant dans *GeneralNodesSet* à chaque chemin selon deux critères :
   (a) : Le profit généré par chaque client.
   (b) : Le temps consommé lors du déplacement du dernier client de chaque chemin au prochain client sélectionné (cette procédure est notée : *AffecNodeProcedure*).
4. Selon l'affectation précédente, l'algorithme passe à la création d'un nouveau niveau dans l'arbre de recherche et effectue une mise à jour de la liste *UsedCustomersSet* .
5. La liste *GeneralNodesSet* est réinitialisée pour préparer l'itération suivante.

Nous illustrons cette deuxième phase de l'algorithme dans l'algorithme 3 qui décrit les principales étapes de cette phase.

Algorithm 3 Deuxième phase de l'algorithme : Obtention d'une solution réalisable finale.

Entrées: Une instance du TOP modélisée par un graphe $G = (V, E)$, V
Sorties: Une solution réalisable pour le TOP

Initialisation :
Mettre $NodeSet = V$;$VisitedSet = \emptyset$; $GeneralNodesSet = \emptyset$;
si Une solution réalisable initiale S est disponible
alors
  - Mettre $Z = profit(S)$;
  - Insérer les clients visités dans $VisitedNodesSet$;
  - Mettre $Z(k) = profit(S_k)$ et $T(k) = T(S_k)$, $\forall\ k = 1, ..., M$;
sinon
  - Mettre $(k) = 0$ et $T(k) = 0$, $\forall\ k = 1, ..., M$;
finsi
Etape Itérative :
Tant que $GeneralNodesSet \neq \emptyset$ **faire**
  - Effectuer $AffecNodeProcedure$ utilisant $GeneralNodesSet$ et mettre $GeneralNodesSet = \emptyset$;
  - Mettre à jour $Z(k)$ et $T(k)$, $\forall\ k = 1, ..., M$, selon $AffecNodeProcedure$;
  - Mettre à jour $VisitedSet$ selon $AffecNodeProcedure$;
  - Pour chaque chemin k, effectuer l'étape de selection en utilisant le client précédent et insérer le nouveau client dans $GeneralNodesSet$;
fin du « Tant que »
Résultat
Retourner M chemins réalisables et la valeur du profit total $Z(M)$;

La section 3.4.4 suivante décrit la coopération des deux phases de l'algorithme proposé et elle présente la procédure d'affectation des clients entre les chemins.

3.4.4 Procédure d'affectation des clients aux chemins

Comme décrit dans les sections précédentes, l'algorithme construit M chemins en parallèle en utilisant les clients disponibles dans l'espace de recherche.

Il est important de noter que le nombre de chemins générés à l'étape 2 de la deuxième phase[8] peut être inférieur au nombre M de chemins à construire. Dans ce cas, la procédure $AffecNodeProcedure$ ferme certains chemins pour permettre la construction d'une solution réalisable. L'algorithme ne s'arrête que si tous les chemins sont fermés ou si le nombre d'itérations fixé est atteint.

L'algorithme 4, présenté dans la suite de ce chapitre, montre comment la procedure $AffecNodeProcedure$ effectue l'affectation des meilleurs clients aux différents chemins obtenus et l'intègration de cette phase dans la méthode de résolution globale.

Algorithm 4 Procedure d'affection de clients aux M chemins.

Entrées: Les meilleurs clients et les M chemins réalisables ;
Sorties: Meilleure solution pour le TOP ;

1: Etant donné $BestSet \neq \emptyset$; et M chemins partiels réalisables ;
2: **si** $|BestSet| = M$ **alors**
3: **pour** Chaque chemin partiel k' **faire**
4: **pour** Chaque client $i \in BestSet$ **faire**
5: - Calculer : $T(i, LastC_{k'})$ et $Z(i, LastC_{k'})$;
6: **si** $(T(k') + T(i, LastC_{k'}) \leq T_{max})$ **alors**
7: $Eval_{(i,k')} = Z(i, LastC_{k'}) + Z(k')/T(i, LastC_{k'}) + T_{k'}$;
8: - Affecter le client $(LastC_{k'}, i)$ au chemin partiel k' qui maximise la valeur $Eval_{(i,k')}$;
9: - Mettre à jour le profit $Z(k')$ et le temps $T(k')$ du chemin partiel k' ;
10: **sinon**
11: - Supprimer le client i de la liste $BestSet$;
12: - Mettre à jour le profit $Z(k')$ et le temps $T(k')$ du chemin partiel k' ;
13: **finsi**
14: **fin pour**
15: **fin pour**
16: **sinon**
17: - Fermer les chemins qui restent ;
18: **finsi**

Dans la section suivante, nous testons notre algorithme sur les instances de la littérature et nous comparons les solutions obtenues aux meilleures résultats de la littérature.

3.5 Les résultats numériques

Le but de cette section est double. Tout d'abord, nous évaluons la qualité des solutions fournis par notre l'algorithme d'approximation qui est noté GAL. Puis, nous évaluons ses performances à travers les temps nécessaires pour atteindre les solutions finales.

L'algorithme GAL a été programmé en `C++` et exécuté sur un ordinateur portable sous Windows7 (2.8 GHz, 4 Gb), avec un temps de résolution (CPU) limité à 900 secondes.

8. Voir le section 3.4.3

L'algorithme a été testé sur les instances par Chao et al. dans [14] [9].

Ces instances sont constituées par un ensemble de 387 instances scindées en sept (7) séries ($p1$, $p2$, $p3$, $p4$, $p5$, $p6$ et $p7$) avec une variation du nombre de clients N. Chaque client i appartenant à une série est caractérisé par des coordonnées et un score fixes (x_i, y_i et p_i). Pour chaque série, le nombre de véhicules M varie entre 2 et 4 ($M = 2$, $M = 3$ et $M = 4$) et la contrainte de temps maximale T_{max} qui varie aussi d'une instance à l'autre.

La table 3.1 résume les caractéristiques des sept séries d'instances. Ainsi, dans la colonne 1 de cette table, le premier chiffre désigne le numéro de la série, par exemple $p1.2$, $p1.3$ et $p1.7$ appartiennent à la série 1 et le deuxième chiffre indique le nombre de véhicules, par exemple, $p1.3$ implique qu'il y'a trois véhicules (c'est aussi le nombre de chemins possibles à construire). La colonne 2, représente le nombre de clients. La colonne 3 quant à elle, nous donne le nombre d'instances qui composent chaque série. Enfin, la colonne 4 donne la variation de T_{max} entre les instances d'un groupe [10]. Par exemple, les instances ($p1.2.a$,..., $p1.2.r$) ont un T_{max} qui varie de 2.5 à 42.5.

Instance	Nb de clients	Nb d'instances	T_{max}
p1.2	32	18	2.5–42.5
p1.3	32	18	1.7–28.3
p1.4	32	18	1.2–21.2
p2.2	21	11	7.5–22.5
p2.3	21	11	5.0–15.0
p2.4	21	11	3.8–11.2
p3.2	33	20	7.5–55.0
p3.3	33	20	5.0–36.7
p3.4	33	20	3.8–27.5
p4.2	100	20	25.0–120.0
p4.3	100	20	16.7–80.0
p4.4	100	20	12.5–60.0
p5.2	66	26	2.5–65.0
p5.3	66	26	1.7–43.3
p5.4	66	26	1.2–32.5
p6.2	64	14	7.5–40.0
p6.3	64	14	5.0–26.7
p6.4	64	14	3.8–20.0
p7.2	102	20	10.0–200.0
p7.3	102	20	6.7–133.3
p7.4	102	20	5.0–100.0

TABLE 3.1 – Description de la caractéristique des instances.

9. Ces instances sont téléchargeables sur : http ://www.mech.kuleuven.be/en/cib/top
10. Un groupe := l'ensemble des instances ayant le même nombre de véhicule et appartenant à la même série.

Dans la suite de ce chapitre, nous comparerons notre approche de résolution, notée *GAL* dans toutes les tables (voir Hifi et al. [37]), avec les 12 algorithmes ci-après :

1. TMH : the tabu search algorithm de Tang et Miller-Hooks (2005) [76].
2. GTH : the tabu search with penalty strategy de Archetti et al. (2007) [12].
3. GTF : the tabu search with feasible strategy de Archetti et al. (2007) [12].
4. FVF : the fast variable neighborhood search de Archetti et al. (2007) [12].
5. SVF : the slow variable neighborhood search de Archetti et al. (2007) [12].
6. SEQ : the sequential algorithm de Ke et al. (2008) [50].
7. DET : the deterministic-concurrent algorithm de Ke et al. (2008) [50].
8. RAN : the random-concurrent algorithm de Ke et al. (2008) [50].
9. SIM : the simultaneous algorithm de Ke et al. (2008) [50].
10. PSOMA [11] : the PSO-based memetic algorithm de Dang et al. (2011) [19].
11. PSOiA : the effective PSO-inspired algorithm de Dang et al. (2013) [20].
12. ALNS [12] : An augmented large neighborhood search de Kim et al. (2013) [48].

La table 3.2, rapporte les solutions obtenues par l'algorithme *GAL* que nous proposons et les solutions obtenues par les 12 algorithmes de la littérature cités précédemment. Ainsi, dans la dernière ligne de cette table, nous avons fait la moyenne des solutions pour chaque algorithme.

11. Avec $\omega = 0,07$
12. Avec $I_{max} = 5000$

Inst.	GAL	TMH	GTP	GTF	FVF	SVF	SEQ	DET	RAN	SIM	PSOMA	PSOiA	ALNS
P1.2	128.1	148.8	149.1	149.1	149.1	149.1	149.1	149.1	149.1	149.1	149.1	149.1	149.1
P1.3	124.7	124.7	125.0	125.0	125.0	125.0	125.0	125.0	125.0	125.0	125.0	125.0	125.0
P1.4	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0	101.0
P2.2	205.0	190.0	190.5	190.5	190.5	190.5	190.5	190.5	190.5	190.5	190.5	190.5	190.5
P2.3	200.5	135.9	136.4	136.4	136.4	136.4	136.4	136.4	136.4	136.4	136.4	136.4	136.4
P2.4	97.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5	94.5
P3.2	465.0	492.0	494.5	496.0	496.0	496.0	496.0	496.0	496.0	496.0	496.0	496.0	496.0
P3.3	448.0	408.0	411.5	411.5	411.5	411.5	411.5	411.5	411.5	411.5	411.5	411.5	411.5
P3.4	436.5	335.0	336.5	336.5	336.5	336.5	336.5	336.5	336.5	336.5	336.5	336.5	336.5
P4.2	915.6	895.1	904.9	908.5	914.0	915.9	915.6	908.4	909.5	911.8	916.9	917.1	917.1
P4.3	856.2	844.3	845.5	852.5	853.0	855.6	853.8	847.7	848.4	848.2	856.1	856.2	856.2
P4.4	802.3	784.6	800.1	802.3	801.7	801.9	798.1	795.9	791.4	795.2	803.6	804.1	804.1
P5.2	897.8	886.8	892.6	897.4	895.8	896.8	897.6	896.4	896.2	896.2	897.6	897.8	897.8
P5.3	783.6	775.8	781.4	783.6	783.6	783.6	782.8	780.4	781.2	781	783.6	783.6	783.6
P5.4	708.8	699.0	707.5	708.8	708.8	708.8	708.8	707.7	706.3	705.6	708.5	708.8	708.8
P6.2	819.3	818.2	813.8	818.7	819.3	819.3	819.3	818.7	818.7	819.3	819.3	819.3	819.3
P6.3	792.8	783.0	792.8	792.8	792.8	792.8	792.8	790.5	790.5	791.3	792.8	792.8	792.8
P6.4	714.2	712.8	714.0	714.0	714.0	714.0	714.0	714.0	714.0	714.0	714.0	714.0	714.0
P7.2	502.2	633.5	639.6	641.4	640.6	642.6	642.7	641.5	641	641.2	642.8	642.8	642.8
P7.3	599.4	592.5	596.7	597.7	597.1	599.2	599.9	599.4	598.6	599.2	599.8	600.0	600.0
P7.4	519.1	514.6	517.2	516.9	516.9	518.9	519.1	518.2	518.4	518.4	518.9	519.1	519.1
Avg.	**529.41**	**522.39**	**525.96**	**527.39**	**527.53**	**528.09**	**527.86**	**526.63**	**526.41**	**526.76**	**528.30**	**528.39**	**528.39**

TABLE 3.2 – L'algorithme *GAL* vs 12 algorithmes de la littérature (Comparaison de *solutions*).

La table 3.3 nous permet de faire une comparaison, entre la solution de notre algorithme *GAL* et la meilleure solution des 12 algorithmes de la littérature (notée *Best_Litt* dans la table 3.3). Ainsi, la première colonne de cette table indique le numero de l'instance, la deuxième colonne montre la meilleure solution des 12 algorithmes et la troisième colonne affiche la solution obtenue par notre algorithme *GAL* et enfin la quatrième colonne de cette table donne l'écart ou la différence (notée : Gap) entre la solution de l'algorithme *GAL* et *Best_Litt*.

Toujours dans la table 3.3, on observe que l'algorithme *GAL* améliore la solution pour 6 instances sur 21 instances soit un pourcentage de 28.57% d'amélioration.

Globalement, l'écart moyen atteint par *GAL* est égale à +1.02. Cela signifie qu'il est en mesure d'améliorer certaines instances et égaler les meilleures solutions pour la majorité d'instances mais qu'il ne parvient pas à améliorer d'autres séries d'instances.

Instance	Best_Litt	GAL.	Gap.
p1.2	149.10	128.06	-21.04
p1.3	125.00	124.70	-0.30
p1.4	101.00	101.00	0.00
p2.2	190.50	205.00	+14.50
p2.3	136.40	200.45	+64.05
p2.4	94.50	97.50	+3.00
p3.2	496.00	465.00	-31.00
p3.3	411.50	448.00	+36.50
p3.4	336.50	436.50	+100.00
p4.2	917.10	915.60	-1.50
p4.3	856.20	856.20	0.00
p4.4	804.10	802.30	-1.80
p5.2	897.80	897.80	0.00
p5.3	783.60	783.60	0.00
p5.4	708.80	708.80	0.00
p6.2	819.30	819.30	0.00
p6.3	792.80	792.80	0.00
p6.4	714.00	714.20	+0.20
p7.2	642.80	502.20	-140.60
p7.3	600.00	599.40	-0.60
p7.4	519.10	519.10	0.00
Av.	**528.39**	**529.41**	**+1.02**

TABLE 3.3 – Différence entre l'algorithme *GAL* et Best_Littérature.

La table 3.4 rapporte le temps d'exécution pour chaque instance testée : la première colonne représente le numéro de l'instance tandis-que la deuxième colonne affiche le temps d'exécution (CPU) mesuré en secondes. Nous donnons également la moyenne du temps d'exécution (moyenne du CPU) pour chaque série d'instances.

La table 3.4 n'indique pas les temps de résolution des 12 algorithmes de la littérature, car ces temps pour chaque instance ne sont pas indiqués dans les références que nous avons utilisées. Néanmoins, les auteurs ont fournis les moyennes des temps d'exécution ou de calcul pour chaque série d'instance.

Inst.	CPU_GAL.
p1.2	0.01
p1.3	0.02
p1.4	0.02
Av. P1	0.02
p2.2	0.01
p2.3	0.01
p2.4	0.00
Av. P2	0.01
p3.2	0.03
p3.3	0.02
p3.4	0.03
Av. P3	0.02
p4.2	45.34
p4.3	0.34
p4.4	0.21
Av. P4	15.30
p5.2	0.14
p5.3	0.10
p5.4	0.09
Av. P5	0.11
p6.2	0.09
p6.3	0.06
p6.4	0.04
Av. P6	0.06
p7.2	0.30
p7.3	0.19
p7.4	0.16
Av. P7	0.22

TABLE 3.4 – Temps d'exécution (CPU) de l'algorithme *GAL*.

Dans la table 3.5, nous rapportons les moyennes de temps de calcul de l'algorithme *GAL* et celles des 12 algorithmes de la littérature cités plus haut. Nous notons que certains temps de résolution pour la l'algorithme TMH (colonne 3) sont introuvables dans la littérature (voir Tang et Miller-Hooks [76]).

Concrètement, il est difficile de comparer directement le temps de calcul de notre algorithme *GAL* avec celui des algorithmes de la littérature dès lors que la vitesse des machines est différente. Néanmoins, dans la dernière ligne de la table 3.5, nous faisons les moyennes de temps de calcul pour notre algorithme *GAL* et celles des 12 algorithmes de la littérature. Ainsi, la moyenne des temps de calculs pour l'algorithme *GAL* est égale à 2.25 secondes. Ce temps peut être considéré comme un temps d'exécution très intéressant, par rapport aux temps d'exécution des algorithmes de la littérature. Nous pouvons conclure que les résultats [13] obtenus par notre algorithme *GAL* sont compétitifs par rapport à ceux des meilleures algorithmes de la littérature .

13. Les résultats sont toujours représentés par les solutions et le temps d'exécution *CPU*.

.Inst.	GAL	TMH	GTP	GTF	FVF	SVF	SEQ	DET	RAN	SIM	PSOMA	PSOiA	ALNS
p1	0.02	–	4.70	1.60	0.10	7.80	5.80	5.20	4.90	5.20	0.20	2.20	0.80
p2	0.01	–	0.00	0.00	0.00	0.00	3.20	3.00	2.80	3.00	0.00	0.40	0.00
p3	0.02	–	6.00	1.60	0.20	10.20	6.50	6.00	5.70	6.00	0.50	3.20	1.40
p4	15.3	184.50	105.30	282.90	22.50	457.90	36.80	31.80	30.70	32.00	78.50	218.60	34.70
p5	0.11	16.50	69.50	26.60	34.20	158.90	17.40	15.10	14.30	15.10	12.90	49.50	10.40
p6	0.06	14.10	66.30	20.20	8.70	147.90	16.10	14.10	13.50	14.20	4.00	47.10	6.10
p7	0.22	84.30	159.00	256.80	10.30	309.90	30.40	24.70	23.30	24.70	54.50	97.50	31.70
Avg.	**2.25**	**74.85**	**58.69**	**84.24**	**10.86**	**156.09**	**16.60**	**14.27**	**13.60**	**14.31**	**21.51**	**59.79**	**12.16**

TABLE 3.5 – L'algorithme *GAL* vs 12 algorithmes de la littérature (Comparaison de *CPU*).

3.6 Conclusion

Dans ce chapitre, nous avons proposé de résoudre le problème d'orientation d'équipe (TOP) en utilisant une heuristique basée sur une recherche en deux phases.

L'approche proposée est un algorithme qui combine deux phases qui sont complémentaires. Ainsi, la première phase est appliquée afin de construire rapidement un chemin de départ qui est considéré comme un chemin élite et la deuxième phase itérative est utilisée pour obtenir une solution réalisable pour le problème d'orientation d'équipe, par des itérations d'amélioration successives.

Les solutions obtenues sur les instances de la littérature montrent que l'approche proposée reste concurrentiel par rapport aux meilleures algorithmes de la littérature et améliore 28.57% des instances testées.

Cette approche a fait l'objet d'une publication dans le *proceding* de la conférence internationale IEEE " *The 2nd International Conference on Future Internet of Things and Cloud (FiCloud 2014)* [37]".

Deuxième partie

Le problème de K-clusters dans un graphe biparti

Chapitre 4

Etat de l'art

Sommaire

Ce chapitre présente un état de l'art du problème de K-clusters dans un graphe biparti, qui consiste à trouver un nombre K de *clusters* minimisant le nombre d'arêtes à ajouter pour obtenir K *bicliques*.

Ce problème est plus connu sous le nom de K-Clustering minimum Biclique Completion Problem (noté : K-CmBCP) et il est prouvé NP-difficile (voir Gualandi et al. [31]).

Dans ce chapitre, nous commençons d'abord par rappeler quelques notions de la théorie des graphes afin d'établir la notation qui sera utilisée tout au long de cette partie de la thèse .

Ensuite, la section 4.2 donnera une vue d'ensemble sur le problème de K-clusters dans un graphe bipartis. La section 4.3 présentera les méthodes de résolution utilisées dans la littérature pour résoudre ce problème et quelques exemples d'applications du K-CmBCP. La section 4.4 détaillera les principaux modèles mathématiques. Et enfin, la section 4.5 résumera ce chapitre et introduira la méthode de résolution que nous proposons pour ce problème.

4.1 Notions sur les graphes

De façon formelle, un *graphe* est un ensemble noté généralement $G = (V, E)$ où V est un ensemble fini de sommets et E est un ensemble fini de paires de sommets $(i, j) \in V * V$.

Rappelons qu'un *graphe* peut être orienté ou non :

— Dans un *graphe orienté*, les paires $(i,\ j) \in E$ sont orientées, c'est à dire que $(i,\ j)$ est une paire ordonnée, où i est le sommet initial, et j le sommet terminal. Dans ce cas, une paire $(i,\ j)$ est nommée un *arc*, et est représentée graphiquement par ($i \longrightarrow j$).
— Dans un *graphe non orienté*, les paires $(i,\ j) \in E$ ne sont pas orientées, c'est à dire que $(i,\ j)$ est équivalent à $(j,\ i)$. Dans ce cas, une paire $(i,\ j)$ est nommée une *arête*, et est représentée graphiquement par (i ____ j).

A partir de maintenant, nous nous intéressons qu'aux *graphes non orientés*, puisque ce travail est basé sur cette notion.

Terminologie

— Deux sommets sont *adjacents* ou *voisins*, s'ils sont joints par une arête. C'est à dire qu'un sommet i est dit adjacent (ou voisin) à un autre sommet j, s'il existe une arête entre i et j. Dans un graphe non-orienté $G = (V, E)$, le *voisinage* d'un sommet $i \in V$, est noté $N_G(i)$ [1] $= \{j : (i, j) \in E\}$.
— L'*ordre* d'un graphe est le nombre de ses sommets $|V|$.
— Le *degré* $d(i)$ d'un sommet i est le nombre des arêtes reliant ce sommet aux autres sommets voisins.
— $G' = (V', E')$ est dit *sous-graphe* d'un graphe non-orienté $G = (V, E)$, si $V' \subseteq V$ et $E' \subseteq E$.
— Un graphe non-orienté $G = (V, E)$ est dit *connexe*, si quels que soient les sommets i et j de V, il existe une chaîne de i vers j. C'est-à-dire, s'il existe une suite d'arêtes permettant d'atteindre j à partir de i (ou i à partir de j).
— Un graphe non-orienté est dit *complet*, si pour toutes les paires de sommets distincts i et j, il existe une arête entre i et j. On peut aussi dire qu'un graphe est complet, si tous ses sommets sont adjacents.
— Un graphe non-orienté $G = (V, E)$ peut aussi être représenté par la matrice d'adjacence A définie par :

$$A\left[i, j\right] = \begin{cases} 1 & \text{Si } (i,j) \in E \\ 0 & \text{Sinon} \end{cases} \tag{4.1}$$

— Une *clique* dans un graphe non-orienté $G = (V, E)$ est un sous-ensemble de sommets de ce graphe dont le sous-graphe induit est complet, c'est-à-dire que deux sommets quelconques de la clique sont toujours adjacents. La taille d'une clique est égale au nombre de sommets qu'elle contient.
— Un graphe $G = (V, E)$ est dit graphe *biparti* si l'ensemble de ses sommets V peut être partitionné en deux sous-ensembles non vides S et T tel que, $V = S \cup T$ et $S \cap T = \emptyset$. Le graphe *biparti* G est noté alors $G = (S, T, E)$ où, S et T, dans notre cas, sont appelés respectivement *services* et *clients*. A noter que, dans un graphe biparti, il ne doit pas y avoir d'arête réliant deux sommets de S ou deux sommets de T.

1. N pour neighborhood qui signifie voisinage en français.

— Le graphe *biparti complet* $G = (S, T, E)$, est un graphe *biparti* qui contient le nombre maximal d'arêtes. Cela veut dire que, $\forall\, i \in S$ et $\forall\, j \in T$, il existe une arête entre i et j. Un graphe *biparti complet* est aussi appelé une *biclique*.

— Dans le problème du K-clusters dans un graphe biparti, un *cluster* est un sous-ensemble de services $S_1 \subset S$, tel que $|S_1| \geq 1$.
Si l'on considère un graphe biparti $G = (S \cup T, E)$, où S et T sont les deux ensembles de sommets du graphe biparti et E est l'ensemble d'arêtes entre S et T. Le cluster $\{S_1, T_1\}$ avec $S_1 \subset S$ et $T_1 \subset T$ est une *biclique* si le sous-graphe induit de G par $S_1 \cup T_1$ est complet.

Pour transformer K-clusters en K-bicliques, on peut ajouter ou retirer des arêtes. Dans le cas du problème du K-clusters, l'objectif est de trouver le nombre d'arêtes minimal à ajouter pour former K bicliques.

4.2 Le problème de K-clusters dans un graphe biparti

Le problème que nous traitons dans cette partie de la thèse, est celui du regroupement minimum de K-bicliques, noté : K-CmBCP. L'objectif de la résolution de ce problème consiste à grouper les services en K-clusters, de tel sorte que le nombre des arêtes à ajouter pour former K-bicliques soit minimum.

Le graphe considéré étant non orienté, les clusters doivent induire d'une partition des services. En d'autres termes, chaque fois qu'un *cluster* est identifié, nous considérons toutes les arêtes reliant les services [2] appartenant à ce cluster, avec les clients correspondants. La figure 4.1 illustre le problème avec 2 clusters : 2-CmBCP.

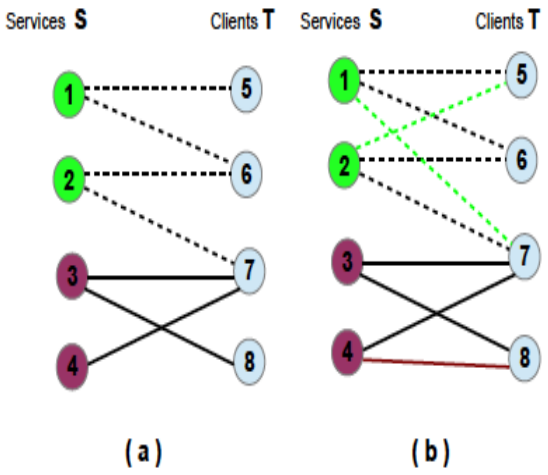

FIGURE 4.1 – Exemple d'un problème de 2-clusters dans un graphe biparti.

2. Dans notre cas, les sommets du coté gauche du graphe biparti, sont appelés les *Services* (S) et les sommets du coté droit du graphe biparti, sont appelés les *Clients* (T)

Ainsi, La figure 4.1 est composée de deux parties : la partie (a), montre un graphe G bipartite avec les services $(S) = \{1, ..., 4\}$ et clients $(T) = \{5, ..., 8\}$, et deux possibilités de cluster $S_1 = \{1, 2\}$ et $S_2 = \{3, 4\}$. Les *arêtes* en pointillées appartiennent au premier *cluster* induit par S_1 et les *arêtes* en gras quant à elles, appartiennent au second *cluster* induit par S_2 et enfin, la partie (b), représente le graphe complémentaire G' permettant de former 2-bicliques.

La pénalité de ces 2-clusters est égale à *trois*, ceci est donné par *deux arêtes* manquantes dans le premier *cluster* et une *arête* manquante dans le deuxième *cluster*. Ainsi, les *arêtes* manquantes pour les deux *clusters* sont respectivement, les *arêtes* : $(1, 7)$, $(2, 5)$, et $(4, 8)$

Nous notons qu'une *biclique* exige que toutes ses arêtes soient présentes dans le graphe, ainsi, les arêtes manquantes dans une solution réalisable sont considérées comme des pénalités.

L'objectif de la résolution du problème du k-clusters, est alors de trouver une façon de partitionner les services, permettant d'avoir la plus faible pénalité.

Dans la littérature, il existe plusieurs problématiques concernant les *bicliques* dans les graphes bipartis. Certaines utilisent une seule biclique, c'est le cas du problème consistant à trouver la biclique maximale en nombre d'arêtes (noté : MEB [3]) qui a été prouvé NP-complet dans Peeters [69].

Cependant, d'autres problématiques prennent en compte plusieurs *bicliques* dans le graphe biparti. Par exemple, le problème de couverture par un minimum de *bicliques*, plus connu sous sa nomination anglophone "*minimum biclique cover*" qui consiste à trouver un ensemble de *bicliques* de G telles que chaque arête $e \in E$ appartient à au moins une *biclique*. Le problème de couverture par un minimum de *bicliques*, est aussi NP-complet (voir Orlin [65]).

Dans la section suivante, nous faisons un survol des méthodes de résolution et des applications pour le K-CmBCP.

4.3 Les méthodes de résolution et les applications pour le K-CmBCP

Dans cette section, nous donnerons d'abord les quelques rares méthodes de résolution que nous avons trouvées dans la literature afin de traiter le problème de k-cluster dans un graphe biparti. Ensuite nous donnerons les domaines d'application de ce problème.

Dans cette section, nous ne donnerons pas les méthodes de résolution des problèmes de l'optimisation combinatoire de manière générale car nous l'avons fait précédemment dans notre chapitre 2 de la partie I. Néanmoins, nous donnerons plutôt des méthodes ayant été utilisées dans la littérature pour la résolution du problème de K-clusters dans un graphe biparti.

A notre connaissance, le K-CmBCP est très peu étudié dans la littérature, seules quelques papiers le concernant sont disponibles dans la littérature à ce jour. Parmi

3. MEB := Maximum Edge Biclique problem

ces papiers, nous citons :

Le papier de Faure *et al.* [26] dans lequel les auteurs ont proposé un modèle de programmation linéaire pour résoudre les instance de petites tailles à l'optimalité et dans le même papier, les auteurs ont également proposé une heuristique basée sur la méthode de génération de colonnes, où certaines instances de grandes tailles ont été résolues.

Gualandi [31] a abordé le problème du K-CmBCP en utilisant une approche hybride qui combine la programmation par contraintes et la programmation semi-définie. A noter que même si, la méthode proposée dans ce papier sort du champ de notre étude, il fait partie des rares travaux sur ce problème, qu'il convient de citer.

A notre connaissance, le dernier papier en date dans la littérature est celui de Gualandi *et al.* [32]. Les auteurs ont conçu une méthode basée sur le principe du *Branch and Price* afin d'accélérer le processus d'amélioration de la qualité des bornes obtenues par le solveur Cplex sur le modèle mathématique pour la programmation linéaire en nombre entiers du K-CmBCP.

Concernant les domaines d'application, le problème de K-clusters dans un graphe biparti est à la base plusieurs applications industriels qui ont attiré l'attention des chercheurs. Ainsi, l'une des applications qui a fait connaitre le K-CmBCP auprès des chercheurs est le problème d'agrégation des sessions multicast en télécommunication. Dans ce cas pratique, une session multicast est définie comme un sous ensemble de clients qui demandent la même information. Aussi, chaque client peut demander plusieurs sessions multicast. Comme le réseau de télécommunication ne pouvait pas gérer plusieurs sessions multicast en même temps, il devenait alors nécessaire de grouper les sessions en un nombre limité de clusters. Le problème consistait alors de faire une agrégation des sessions sous forme de clusters afin de limiter le nombre d'informations inutiles envoyées aux clients.

La variante que nous traitons dans cette partie de la thèse requiert des clusters pour définir une partition sur l'ensemble des sessions. Cette variante est inspirée des problèmes comme le *Multicast Partition Problem : MPP* (voir Suh *et al.* [75]).

La figure 4.2 illustre un exemple d'application du *MPP* dans lequel nous utilisons la même représentation que le K-CmBCP. Ainsi, sur le côté gauche du graphe biparti, il y'a trois sessions A, B et C, qui envoient des informations multicasts aux clients 1, 2, 3 et 4 (sur le côté droit du graphe biparti). Chaque client demande un certain nombre de sessions qui sont mentionnées sur sa droite.

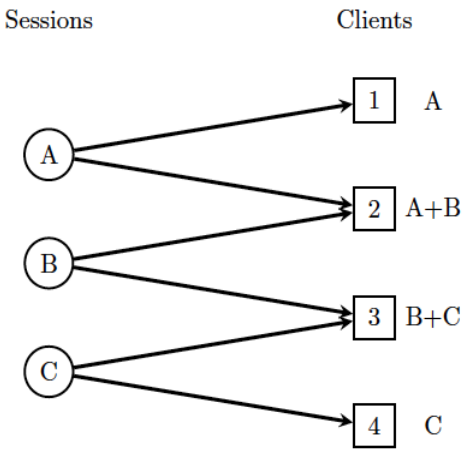

FIGURE 4.2 – Un exemple d'application du MPP.

La figure 4.3 affiche la même instance du MPP, en regroupant les sessions A et B ensembles et qui deviennent maintenant une source unique multicast permettant d'envoyer toutes les informations que les sessions A et B envoyaient séparément. Ainsi, le client 1 reçoit la session B bien qu'il ne la demande pas. La même chose pour le client 3 avec la session A. On remarque que ce regroupement, a permis le gain d'une liaison tout en respectant les demandes des clients.

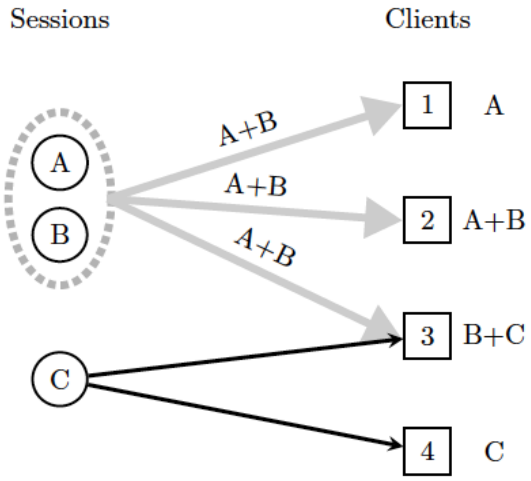

FIGURE 4.3 – Un cluster pour l'exemple d'application du MPP.

En dehors du problème MPP, d'autres applications du K-CmBCP existent dans des domaines très variés à savoir : la biologie, la sécurité des réseaux informatique ou le *data mining* pour lequel le *clustering* ou *regroupement* est très important.

La section suivante donne les principales formulations mathématiques pour le problème de K-clusters dans un graphe biparti.

4.4 Modèles mathématiques pour le K-CmBCP

Cette section est consacrée à la description des différentes modèles mathématiques du problème de cluster dans un graphe biparti (voir Magni *et al.* [56]).

Le problème de K-clusters dans un graphe biparti est un problème de l'optimisation combinatoire ayant plusieurs types de modèles mathématiques :

1. Modèles d'affection : dans lesquelles les clusters sont indiqués de façon explicite et les sommets sont affectés à ces clusters.
2. Modèles représentatif : où chaque cluster est représenté par un noeud qui est noté "head-imi" et les autres sommets sont donc affectés à ce sommet.
3. Modèles de génération de colonne : où les configurations des clusters sont prises en compte et le nombre de variables est exponentiel.

Dans les sections suivantes de ce chapitre, nous allons présenter cinq modèles mathématiques pour le problème de K-clusters dans un graphe biparti.

Pour l'ensemble des modèles mathématiques, nous considérerons : Un graphe biparti $G = (S, T, E)$ où S représente l'ensemble des services i et T représente l'ensemble des clients j. E représente l'ensemble des arêtes (i, j) du graphe biparti, tandis-que $\overline{E}$ représente l'ensemble des arêtes du graphe biparti complémentaire, c'est à dire que $\overline{E} = \{S * T\} \backslash E$ et enfin, $K = \{1, ..., k\}$ représente l'ensemble des indices des K-clusters prédéfinis dans le problème.

Pour un soucis de clarté, nous avons choisi d'utiliser la terminologie utilisée dans Magni *et al.* [56].

4.4.1 Première formulation

Ce premier modèle est certainement le plus évident et le plus utilisé dans la littérature (voir [26], [31], [32]). Il est quadratique [4]. Rappelons qu'en optimisation mathématique, un problème d'optimisation quadratique est un problème d'optimisation dans lequel on minimise (ou maximise) une fonction quadratique sur un polyèdre convexe. Les contraintes peuvent donc être décrites par des fonctions linéaires ou par des fonctions non-linéaires.

Nous présentons dans cette section le premier modèle mathématique sous une formulation quadratique, avec des contraintes non linéaires et dont les variables binaires sont :

$$x_{ik} = \begin{cases} 1 & \text{Si le service } i \text{ est dans le cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{4.2}$$

4. Une forme quadratique est un polynôme homogène de degré deux avec un nombre quelconque de variables [77].

$$y_{jk} = \begin{cases} 1 & \text{Si le client } j \text{ est dans le cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{4.3}$$

Pour $i \in k$ et $j \in k$, si $x_{ik} = y_{jk} = 1$ et l'arête $(i,j) \notin E$ (ou $(i,j) \in \overline{E}$), alors, l'arête (i,j) est comptabilisée comme une pénalité.

Ainsi, la fonction-objectif est obtenue en faisant la somme exprimée dans l'équation (4.4).

$$\min \sum_{(i,j)\in\overline{E}} \sum_{k\in K} x_{ik} y_{jk} \tag{4.4}$$

sous contraintes :

$$\sum_{k\in K} x_{ik} y_{jk} = 1 \qquad \forall\ (i,j) \in E \tag{4.5}$$

$$\sum_{k\in K} x_{ik} = 1 \qquad \forall\ i \in S \tag{4.6}$$

$$\sum_{k\in K} x_{ik} \geq 1 \qquad \forall\ k \in K \tag{4.7}$$

$$x_{ik}, y_{jk} \in \{0,1\} \qquad \forall\ i \in S,\ j \in T,\ k \in K \tag{4.8}$$

La première contrainte (4.5) permettra de couvrir toutes les demandes. Ainsi, pour chaque arête (i,j), on doit s'assurer que le service i et le client j appartiennent à un cluster au plus.

La deuxième contrainte (4.6) est requise pour la partition des services dans les clusters. cette contrainte exige qu'un service i appartienne à un seul cluster à la fois.

La troisième contrainte (4.7) oblige l'utilisation de tous les K-clusters en interdisant qu'un cluster soit vide et enfin la quatrième et dernière contrainte (4.8) désigne les variables binaires citées précédemment.

Malheureusement, un modèle de ce type avec des contraintes non linéaires est très difficile à résoudre (pour plus de details, voir le livre de Bazaraa et al.,[5]).

Cependant, en gardant la contrainte (4.6) et en combinant les contraintes (4.5) et (4.7), le modèle peut être transformé en modèle linéaire.

Aussi, en utilisant l'associativité de la multiplication, nous pouvons simplifier l'écriture de la fonction objectif par l'introduction d'une troisième variable z qui exprimera le produit $x_{ik}y_{jk}$. Dans ce cas, la nouvelle variable binaire sera :

$$z_{ijk} = \begin{cases} 1 & \text{Si le service } i \text{ et le client } j \text{ sont dans le cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{4.9}$$

Et le modèle mathématique peut être réécrit sous une formulation quadratique, avec des contraintes linéaires et une fonction-objectif simplifiée, comme suit :

$$\min \sum_{(i,j)\in\overline{E}} \sum_{k\in K} z_{ijk} \tag{4.10}$$

sous contraintes :

$$y_{jk} \geq x_{ik} \qquad \forall\ (i,j) \in E,\ \mathrm{k} \in K \tag{4.11}$$

$$\sum_{k\in K} x_{ik} = 1 \qquad \forall\ \mathrm{i} \in S \tag{4.12}$$

$$z_{ijk} \geq x_{ik} + y_{jk} - 1 \qquad \forall\ \mathrm{i} \in S,\ \ j \in T,\ \ k \in K \tag{4.13}$$

$$x_{ik},\ y_{jk},\ z_{ijk} \in \{0,1\} \qquad \forall\ \mathrm{i} \in S,\ \ j \in T,\ \ k \in K \tag{4.14}$$

Dans cette dernière formulation, la contrainte (4.11) permet de couvrir toutes les demandes. La contrainte (4.12) oblige un service i à appartenir à un seul cluster à la fois. La contrainte (4.13) est utilisée pour présenter l'équation linéaire et enfin la dernière contrainte (4.14) désigne les variables du problème.

4.4.2 Deuxième formulation

Ici, nous présentons une formulation non-linéaire différente et qui interprète le problème comme un modèle *min-max* (voir Meka [58]). Ce modèle introduit la variable z_{ij} ayant le sens suivant :

$$z_{ij} = \begin{cases} 1 & \text{Si l'arête } (i,j) \text{ est ajoutée (comptabilisée comme une pénalité)} \\ 0 & \text{Sinon} \end{cases} \tag{4.15}$$

Sachant qu'une arête (i,j) est ajoutée ou est comptabilisée comme une pénalité, si et seulement si, il existe au moins un service l, relié à un client j et que le service l est dans le même cluster k que le service i.

D'autre part, si $N(j)$ représente le voisinage du clients j tel que $N(j) = \{i \mid (i,j) \in E\}$, alors, la contrainte suivante peut être introduite :

$$z_{ij} = \max_{l\in N(j)} \{x_{ik}\ x_{lk}\} \tag{4.16}$$

Cette contrainte peut être réécrite comme un ensemble de contraintes $z_{ij} \geq x_{ik}\ x_{lk}$, qui sont linéarisable sous-forme de : $z_{ij} \geq x_{ik} + x_{lk} - 1$.

Ainsi, la formulation linéaire du problème de K-clusters dans un graphe biparti en utilisant un modèle *Min-Max* est :

$$\min \sum_{(i,j)\in\overline{E}} z_{ij} \tag{4.17}$$

sous contraintes :

$$\sum_{k\in K} x_{ik} = 1 \qquad \forall \text{ i} \in S \tag{4.18}$$

$$z_{ij} \geq x_{ik} + x_{lk} - 1 \qquad \forall \text{ (i, j)} \in \overline{E},\ \ (l,j) \in E,\ \ k \in K \tag{4.19}$$

$$x_{ik},\ z_{ij} \in \{0,1\} \qquad \forall \text{ i} \in S,\ \ j \in T,\ \ k \in K \tag{4.20}$$

La contrainte (4.18) exprime la repartition des services, la contrainte (4.19) est linéarisation de la contrainte *min-max* (4.16) et enfin, la contrainte (4.20) donne les variables du problème.

4.4.3 Troisième formulation

La troisième modèle utilise le modèle *représentatif*, c'est à dire que le cluster est représenté par un sommet. Ainsi, tous les services appartenant au même cluster sont représentés par un seul service représentant le cluster auquel ils adressent leurs affectations. Dans ce modèle, le premier service de chaque cluster devient son représentant. La construction se fait dans un ordre quelconque.

Par ailleurs, ce modèle permet d'éliminer les symétries. Les variables binaires pour ce modèle sont :

$$x_{il} = \begin{cases} 1 & \text{Si le service } i \text{ est le représentant du service } l \\ 0 & \text{Sinon} \end{cases} \tag{4.21}$$

$$z_{ij} = \begin{cases} 1 & \text{Si l'arête } (i,j) \text{ est ajoutée} \\ 0 & \text{Sinon} \end{cases} \tag{4.22}$$

La fonction-objectif est très simple grace à la variable z. Mais, on a besoin d'autres contraintes : une première contrainte permettant d'assurer l'identification des K clusters, une deuxième contrainte permettant l'affectation de chaque service au *service représentant* de son cluster, une troisième contrainte permettant de lier correctement les variables x entre elles (c'est à dire que si un service l est affecté à un *représentant* i, alors i doit être le *représentant* et donc ne peut pas être affecté) et enfin, une dernière contrainte permettant de mettre à 1 la variable binaire z_{ij}, lorsque l'arête (i,j) est ajoutée pour obtenir une *biclique*.

$$\min \sum_{(i,j)\in\overline{E}} z_{ij} \tag{4.23}$$

sous contraintes :

$$\sum_{i\in I} x_{ii} \ = \mathrm{K} \tag{4.24}$$

$$\sum_{i\in I,\ i\in l} x_{il} \ = 1 \qquad\qquad \forall\ \mathrm{l} \in S \tag{4.25}$$

$$x_{il} \leq x_{ii} \qquad\qquad \forall\ \mathrm{i} \in S \text{ et } l \in S \tag{4.26}$$

$$z_{ij} \geq x_{hi} + x_{hl} - 1 \qquad\qquad \forall\ (\mathrm{i},\ \mathrm{j}) \in \overline{E},\ \ h \in S \text{ et } (l, j) \in E \tag{4.27}$$

$$x_{il},\ z_{ij} \in \{0, 1\} \qquad\qquad \forall\ \mathrm{i}\ ,\mathrm{l} \in S \text{ et } j \in T \tag{4.28}$$

La contrainte (4.24) assure que nous groupons les services en K clusters. la contrainte (4.25) assure que chaque service a un seul *représentant*. La contrainte (4.26) force le service représentant un autre service d'être aussi son propre représentant. La contrainte (4.27) force la création d'une biclique et enfin, la dernière contrainte (4.28) donne les variables du problème.

4.4.4 Quatrième formulation

Dans ce modèle, les clusters sont considérés comme des objets combinatoires.

Soit $H = \{(S_t, T_t)\}$ un ensemble de clusters, tel que S_t et T_t sont respectivement, l'ensemble des services et l'ensemble des clients appartenant au cluster t (c'est à dire : $S_t \subseteq S$ et $T_t \subseteq T$).

t représente le vecteur caractéristique d'un cluster, c'est à dire : $t_i = 1$, si $i \in S_t$, et $t_i = 0$ sinon.

Soit $c_t = \mid (\mathrm{S}_t * T_t) \bigcap \overline{E} \mid$, la pénalité d'un cluster t et $a_{it} = 1$, si i $\in S_t$, t $\in \{1, .., \mid H \mid\}$.

Soit, la variable suivante :

$$\lambda_t = \begin{cases} 1 & \text{Si le cluster } t \text{ est selectionné} \\ 0 & \text{Sinon} \end{cases} \tag{4.29}$$

Le modèle linéaire, utilisant la génération de colonnes est le suivant :

$$\min \sum_{t\in H} c_t \lambda_t \tag{4.30}$$

sous contraintes :

$$\sum_{t \in H} a_{it} \lambda_t \ = 1 \qquad \forall\ i \in S \tag{4.31}$$

$$\sum_{t \in H} \lambda_t \ = \mathrm{K} \tag{4.32}$$

$$\lambda_t \in \{0, 1\} \qquad \forall\ t \in H \tag{4.33}$$

La contrainte (4.31) donne la partition des services. La contrainte (4.32) assure que le nombre de clusters utilisés est exactement K. Et enfin, la contrainte (4.33) présente la variable binaire du problème.

4.4.5 Cinquième formulation

Ce modèle est basé sur les variables entières et les numéros des clusters varient de 1 à k. Parmi tous les modèles mathématiques cités, celui-ci est le plus difficile à linéariser. Par conséquent, pour linéariser ce modèle mathématique, des techniques dites " *ad-hoc*" sont utilisées. Ces dernières permettent de réduire ce problème à une formulation moins bonne mais similaire à celle du troisième modèle mathématique (voir section 4.4.3). Ici, le numero du cluster auquel le service i appartient est noté x_i et la variable binaire z_{ij} est la suivante :

$$z_{ij} = \begin{cases} 0 & \text{Si l'arête (i, j) est ajoutée à la solution} \\ 1 & \text{Sinon} \end{cases} \tag{4.34}$$

Ce modèle teste la presence de deux services i et j dans le même cluster k en faisant la difference de leurs variables x respectives (x_i et x_j). Si la difference ($x_i - x_j$) est nulle, alors les deux services i et j partagent le même cluster. La formulation est la suivante :

$$\min \sum_{(i,j) \in \overline{E}} (1 - z_{ij}) \tag{4.35}$$

sous contraintes :

$$z_{ij} \leq \mid x_i - x_l \mid \qquad \forall\ (i, j) \in \overline{E},\ (\mathrm{l},\ \mathrm{j}) \in E \tag{4.36}$$

$$x_i \in \{1, ..., k\} \qquad \forall\ \mathrm{i} \in S \tag{4.37}$$

$$z_{ij} \in \{0, 1\} \qquad \forall\ \mathrm{i} \in S,\ \ j \in T \tag{4.38}$$

Où, la contrainte (4.36) permet de mettre z_{ij} à zero lorsque l'arête (i, j) doit être ajoutée pour former la *biclique*, la contrainte (4.37) permet de garder la variation de x dans l'intervalle de [1 à k] clusters. La dernière contrainte (4.38) présente la variable binaire z_{ij} donnée dans (4.34).

4.5 Conclusion

Dans ce chapitre, nous avons d'abord, rappelé quelques notions sur la théorie des graphes afin de poser les bases de la problématique abordée dans cette deuxième partie de la thèse.

Ensuite, nous avons donné une vue d'ensemble sur le problème de K-clusters dans un graphe biparti. Ceci nous a permis de détailler certains aspects de cette problématique et surtout d'introduire l'approche que nous proposons pour résoudre approximativement ce problème.

L'avant dernière section de ce chapitre, nous a permis d'une part, de citer quelques travaux de recherche de la littérature ayant traité le problème du K-clusters dans un graphe biparti et d'autre part, de citer des applications du K-CmBCP dans le monde industriel.

Dans la dernière section de ce chapitre, nous nous sommes intéressés aux principaux modèles mathématiques. Ainsi, nous notons que ces modèles représente un socle de base pour quelques caractéristiques importantes dans la littérature (voir Magni *et al.* [56]) et que nous avons choisi d'énumérer pour conclure ce chapitre :

• Propriété 1. La solution optimale est celle qui utilise toujours tous les K clusters.

• Propriété 2. La pénalité totale de K-clusters qui se chevauchent [5] est supérieur ou égale à celle de K-clusters qui ne se chevauchent pas.

• Propriété 3. La relaxation continue du premier modèle mathématique génère une solution entière.

• Propriété 4. La solution du premier modèle est entière, lorsque, les variables y_{jk} et z_{ijk} sont continues sur l'intervalle [0 1].

• Propriété 5. La solution du cinquième modèle mathématique est entière, lorsque, la variable z_{ij} est continue sur l'intervalle [0 1].

Dans le chapitre suivant, nous avons proposé une recherche par perturbation de voisinage pour résoudre approximativement le problème de K-clusters dans un graphe biparti.

5. On dit que deux clusters se chevauchent, lorsqu'ils ont une partie de services en commun.

Chapitre 5

Chapitre 5

Une recherche par voisinage pour le problème de K-clusters dans un graphe biparti

Sommaire

Ce chapitre présente une méthode de recherche par voisinage pour résoudre approximativement le problème du K-clusters dans un graphe biparti. Cela, consiste à trouver un nombre K de sous-graphes minimisant le nombre d'arêtes à considérer pour obtenir K bicliques.

L'approche proposée combine trois phases complémentaires : la première phase permet la construction d'une solution initiale réalisable. Cette phase utilise un algorithme glouton et l'algorithme de Johnson (voir [47]. [29]). La deuxième phase est une étape d'intensification et enfin, la troisième et dernière phase de notre approche est une diversification.

L'heuristique proposée est évaluée sur des instances de la littérature et des instances de grande taille que nous avons généré. Les résultats numériques obtenus sont comparés aux meilleurs résultats de la littérature et aux résultats fournis par le solveur Cplex12.5 pour un modèle mathématique linéaire basé sur la cinquième formulation présentée dans le chapitre précédent.

5.1 Le problème de K-clusters et ses applications

Le but principal de ce travail est de développer une méthode de résolution permettant de résoudre le problème de régroupement dans un graphe biparti.

Nous rappelons qu'une instance du K-CmBCP est définie par un graphe biparti $G(V, E)$, où V est l'ensemble de N sommets tel que : $V = S \cup T$ et $S \cap T = \emptyset$ et E représente l'ensemble des arêtes. Nous rappelons aussi que, si (S, T) est une biclique, alors chaque sommet de S (respectivement de T) est relié à tous les sommets de T (respectivement de S). Par consequent, si le graphe est formé de K clusters : $\Big((S_1, T_1), (S_2, T_2), \ldots, (S_k, T_k)\Big)$, alors tous les sommets de chaque couple (S_k, T_k), sont interconnectés $\forall\ k = \{1, \ldots, K\}$.

Etant donné que le graphe biparti $G(V, E)$ considéré est non-orienté. chercher K-clusters revient à chercher K sous-graphe biparti de G avec l'ajout d'un minimum d'arêtes qui n'appartiennent pas à E. De façon similaire, nous pouvons dire que le but de ce problème est de trouver le meilleur partitionnement de l'ensemble S en K-clusters avec un minimum d'arêtes à ajouter.

Comme décrit dans Gualandi *et al.* [32], le K-CmBCP a de nombreuses applications réel comme le partitionnement des canaux pour les transmissions multi-diffusions [1]. Dans cette application, étant donné un ensemble de services demandé par les clients, l'objectif est de determiner les K sessions multicast permettant le partitionnement de l'ensemble des demandes. Ainsi, chaque service considéré doit appartenir à une session multicast tandis que chaque client peut être dans plusieurs sessions.

La suite de ce chapitre est organisé comme suit : dans la section 5.2.2, nous déroulerons un exemple de 2-CmBCP et nous présenterons la formulation du modèle mathématique linéaire en nombres entiers que nous résoudrons dans la partie expérimentale en utilisant le solveur Cplex. La section 5.3 détaillera notre heuristique de recherche par voisinage pour le K-CmBCP. Ainsi, dans cette section, nous commençerons d'abord par construire la solution initiale selon une procédure gloutonne combinée avec l'algorithme de Johnson, ensuite, nous améliorerons cette solution initiale en utilisant les stratégies d'intensification qui comportent à la fois des techniques de recherche locale et des permutations. Et enfin, nous évaluerons notre approche en la comparant aux meilleurs résultats de la littérature et aussi aux résultats du solveur Cplex lorsqu'on lui fournit le modèle ILPM (voir les equations (5.2) à (5.7)). La dernière section conclura ce chapitre en résumant l'apport de cette contribution et donnera quelques perspectives pour la résolution du K-CmBCP.

1. La transmissions multi-diffusion peut-être aussi appelée la transmission multicast en télécommunication

5.2 Le modèle mathématique utilisé

Dans cette section, nous construisons d'abord un exemple de deux solutions réalisables pour un K-CmBCP, ensuite, nous présentons la formulation mathématique en nombres entiers (ILPM) qui sera résolue dans la partie expérimentale en utilisant le solveur Cplex12.5.

5.2.1 Exemple avec deux clusters ($K = 2$)

La figure 5.1 donne un graphe biparti représentant une instance du K-CmBCP. Ainsi, dans ce graphe biparti les services (sommets du côté gauche) sont représentés par l'ensemble $\{i_1, i_2, i_3 \; et \; i_4\}$ et les clients (sommets du côté droit) sont représentés par l'ensemble $\{j_1, j_2, j_3, j_4, j_5 \; et \; j_6\}$.

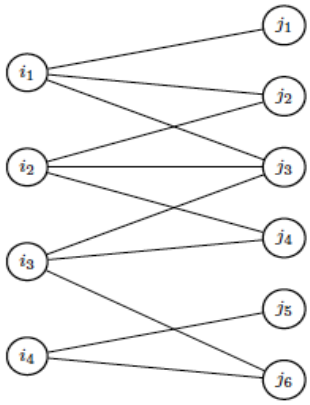

FIGURE 5.1 – Représentation graphique d'un 2-CmBCP.

Afin de permettre une bonne compréhension de la figure 5.2, nous rappelons que l'objectif du 2-CmBCP est de diviser l'ensemble des services [2] S en 2-clusters disjoints, tel que, le nombre des arêtes à ajouter pour former les 2 bicliques, soit minimum.

La figure 5.2, montre deux solutions possibles pour l'instance précédente. Ainsi, nous formons au hasard 2-clusters et pour chaque cluster, nous complétons les arêtes manquantes afin de former une biclique avec les sommets correspondants.

Deux cas ont été considérés pour montrer à quel point le partitionnement peut influencer la pénalité totale. Ainsi, dans la figure 5.2, les 2-clusters sont représentés par des blocks entourant les sommets tandis que les arêtes à ajouter pour former les 2 bicliques sont représentés par des flèches discontinues (en rouges) et donc la pénalité de chaque cluster est la somme de ses flèches discontinues.

La pénalité totale du 2-CmBCP pour chaque exemple de partitionnement [3] de la figure 5.2 est la somme des pénalités de l'ensemble de ses clusters (voir la figure 5.2 (a) et (b)).

2. Ici représentés par les sommets du côté gauche : $\{i_1, i_2, i_3 \; et \; i_4\}$
3. Partitionnement (a) et (b) de la figure 5.2

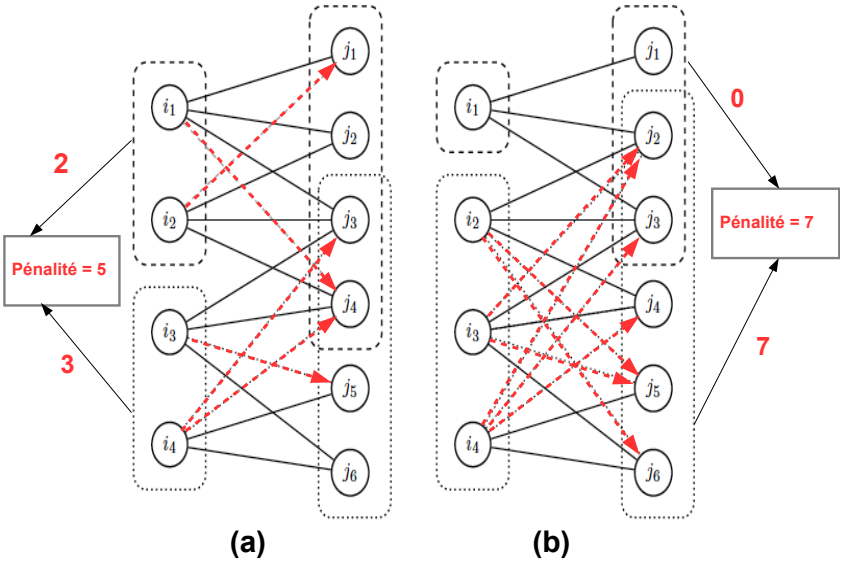

FIGURE 5.2 – Construction de deux solutions réalisables pour un 2-CmBCP.

5.2.2 Le modèle mathématique

Dans la suite, nous donnons un modèle mathématique issu du premier modèle présenté en section 4.4.1 du chapitre 4, ayant une formulation linéaire en nombre entier.

Ainsi, le modèle mathématique du problème de K-clusters dans un graphe biparti peut être exprimé comme suit :

soit z_{ijk} une variable binaire égale à 1, lorsque $x_{ik} = 1$ et $y_{jk} = 1$.

$$z_{ijk} = \begin{cases} 1 & \text{Si les noeuds } i \text{ et } j \text{ sont tous les deux affectés au cluster } k \\ 0 & \text{Sinon} \end{cases} \tag{5.1}$$

En utilisant les techniques de linéarisation sur le premier modèle mathématique, on obtient la formulation suivante :

$$(ILPM) : \quad \min \sum_{k \in K} \sum_{(i,j) \in \overline{E}} z_{ijk} \tag{5.2}$$

sous contraintes :

$$x_{ik} + y_{jk} \leq 1 + z_{ijk} \qquad \forall (i,j) \in \overline{E},\ \forall\ \mathrm{k} \in \mathrm{K} \tag{5.3}$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall\ \mathrm{i} \in S \tag{5.4}$$

$$x_{ik} \leq y_{jk}, \qquad \forall \text{ (i, j)} \in E, \ \forall \text{ k} \in K \tag{5.5}$$

$$x_{ik}.y_{jk} \in \{0,1\}, \qquad \forall \text{ i} \in S, \ \forall \text{ j} \in T, \ \forall \text{ k} \in K \tag{5.6}$$

$$z_{ijk} \in \{0,1\}, \qquad \forall \ (i,j) \in \overline{E}, \ \forall \text{ k} \in K \tag{5.7}$$

Où :

La contrainte 5.2 représente la fonction-objectif qui minimise la pénalité totale de l'ensemble des K clusters. La contrainte 5.3 est la contrainte qui lie la variable z_{ijk} aux variables x_{ik} et y_{jk} en mettant z_{ijk} à 1 lorsque les deux autres le sont aussi. La contrainte 5.4 est la contrainte permettant d'attribuer chaque service $i \in S$ à un et un seul cluster. La contrainte 5.5 force chaque client $j \in T$ affecté au cluster k d'être dans le même cluster qu'un service $i \in S$ lorsque j est adjacent à i dans E. Et enfin, les contraintes 5.6 et 5.7 sont des variables binaires du problème.

Malheureusement, la relaxation linéaire du modèle ci-dessus est très faible car il est à moitié entier et fournit une limite inférieure égale à zéro (voir Faure *et al.*[26]).

5.3 Une heuristiques pour le problème de K-clusters dans un graphe biparti

Dans cette section, nous proposons de résoudre approximativement le K-CmBCP en utilisant une heuristique de recherche par voisinage. L'approche proposée se base sur trois étapes complémentaires. Une telle approche, peut être vue comme une variante de la méthode proposée dans Hifi et Michrafy [39] pour la résolution de quelques problèmes de type Sac à dos et dans Shaw [73], où l'auteur a proposé une méthode plus générale de recherche dans un large voisinage pour la résolution des problèmes combinatoires.

Dans ce travail. nous proposons de résoudre le K-CmBCP par application d'une heuristique qui s'appuie sur le principe de la coopération entre une recherche gloutonne, une amélioration par intensification et une recherche par diversification. Cette approche utilise trois phases complémentaires :

- Phase 1 : La première phase permet la construction d'une solution de départ réalisable en utilisant une stratégie gloutonne combinée avec l'algorithme de Johnson.
- Phase 2 : La deuxième phase tente d'améliorer la qualité de la solution de départ en utilisant une *stratégie d'intensification.*
- Phase 3 : La troisième et dernière phase introduit une *diversification* dans le but d'atteindre certains sous-espaces d'une manière efficace : dans cette phase des dégradations et reconstructions sont utilisées pour améliorer la qualité des solutions. Le processus utilisé dans cette phase est basé sur le principe de la recherche à large voisinage (voir Shaw [73]).

5.3.1 Construction d'une solution de départ pour le K-CmBCP

Il existe plusieurs façons de construire une solution de départ. Ici, nous utilisons une simple procedure gloutonne basée sur le principe de l'algorithme de Johnson (voir Johnson [47], Garey et Johnson [29]).

Pour un problème de K-clusters dans un graphe biparti, qui peut être vu comme un cas particulier du problème de recouvrement d'ensemble (voir Orlin [65]), une adaptation de l'algorithme de Johnson peut être décrite comme suit :

(i) : $S' \longleftarrow S$.

(ii) : Ordonner tous les services appartenant S' par ordre décroissant de leurs degrés de liaisons.

(iii) : Sélectionner le service $i \in S'$ ayant le plus grand degré et le mettre dans le cluster courant.

(iv) : Définir un graphe réduit G' en supprimant le service i précédemment sélectionner de l'ensemble S'.

(v) : Stop, si $S' = \emptyset$, sinon, répéter les étapes (ii)-(iv).

A partir d'un classement bien défini des services, selon des critères liés aux degrés de chacun des services, une affectation des services vers des clusters est réalisée. Cette attribution réalise une solution admissible pour le problème K-CmBCP, où le contenu de l'ensemble des clusters est presque équilibré.

Bien entendu, la procédure ci-dessus est utilisée pour le partitionnement de l'ensemble S du graphe biparti G. Alors, afin de fournir la pénalité de la solution de départ atteinte par la procedure gloutonne, une étape supplémentaire est utilisée pour compléter chaque cluster créé avec des arêtes supplémentaires pour former les K-bicliques du graphe biparti G.

5.3.2 Amélioration de la qualité des solutions : phase d'intensification

Généralement l'amélioration de la qualité des solutions est réalisée par application de certaines procédures standards comme la recherche locale qui s'appuie sur des permutations des sommets et dont le but est d'effectuer une recherche sur des séries de voisinages. Dans notre cas. la phase d'intensification introduite est basée sur la stratégie de permutation de k éléments à la fois. Cette stratégie est notée par k-opt dans la suite de ce chapitre.

La stratégie de permutation est utilisée entre certains services des différents clusters que forme l'ensemble S. Ainsi, ces permutations peuvent être généralisées pour un δ-voisinage, où δ représente un sous-ensemble de services voisins, qui peuvent être déplacées par groupe (ou individuellement) d'un cluster vers d'autres clusters. A partir de la meilleure solution obtenue dans le voisinage, ce processus peut être réitéré. Ce dernier s'arrête après avoir exploré un certain nombre de voisinages ou après un nombre d'itérations paramétrable.

La procedure k-opt, considère plusieurs permutations entre les clusters.

Algorithm 5 Amélioration des solutions : une recherche de voisinage.

Entrées: Une solution initiale F avec sa pénalité (notée e_{nb}), $Iter\_local$.
Sorties: Une nouvelle solution réalisable F'.

1: $ok \leftarrow$ `faux` ;
2: **Tant que** (pas (ok) et $(Iter\_local < 20)$) **faire**
3: **pour tout** $k_j \in K$ **faire**
4: **pour tout** $s_i \in k_j$ **faire**
5: **pour tout** $k_\ell \in K.\ \ell \neq j.$ **faire**
6: Mettre s_i (issu de k_j) dans k_ℓ ;
7: Soit F' une nouvelle solution avec sa pénalité correspondante e^{ℓ}_{nb} ;
8: **si** $(e^{\ell}_{nb} < e_{nb})$ **alors**
9: $F \leftarrow F'$ et $ok =$`vrai` ;
else
10: Sortir avec la meilleure solution F' du voisinage, lorsque, $Iter\_local = 20$.
11: **finsi**
12: **fin pour**
13: **fin pour**
14: **fin pour**
15: **fin du « Tant que »**

L'algorithme 5 décrit les principales étapes de la procedure d'intensification qui est utilisée pour améliorer la qualité des solutions. Ainsi, la donnée à l'entrée de l'algorithme est la solution initiale obtenue avec la procedure qui combine l'algorithme de Johnson et une méthode gloutonne (voir la section 5.3.1). La boucle `while` (ligne 2) est utilisée pour stopper la résolution lorsque la recherche locale est capable de trouver une nouvelle solution réalisable avec une moindre pénalité (dans ce cas, le booléen *ok* est appliqué afin de stopper la boucle). Bien entendu, le critère d'arrêt est aussi consolidé avec la limite du nombre d'itérations qui encadre d'exploration du voisinage. La phase de permutation est considérée dans les lignes 3 à 6, Ici, deux cas peuvent être considérés :

1. Un service appartenant à un cluster peut être permuté avec un autre service qui appartient à un autre cluster ; la solution est mise à jour lorsque la nouvelle solution améliore la qualité de la meilleure solution du voisinage.
2. Lorsque l'itération précédente ne parvient pas à fournir une meilleure solution, alors une procédure k-opt est introduite comme suit :
   (a) : Sélectionner k services d'un cluster donné et les déplacer vers d'autres clusters (lorsque c'est possible).
   (b) : Réconfigurer les clusters pour fournir une solution réalisable.

enfin, la solution réalisable trouvée est mise à jour (ligne 9), si la pénalité de la solution réalisable courante améliore la qualité de la solution dans le voisinage visité. On note que le nombre de services à permuter est également paramétrable.

5.3.3 Application de la phase de diversification des solutions

Dans cette section, nous allons montrer comment appliquer le principe de diversification pour rechercher une série de nouvelles solutions. La phase d'intensification peut améliorer les solutions quand elle explore une série de voisinages très proches. Cependant, la réconfiguration de l'espace de recherche peut permettre l'exploration de nouveaux voisinages et donc l'emergence de meilleures solutions. La variante de la recherche dans un large voisinage introduite ici, est basée sur *la stratégie de dégradation* et *la stratégie de reconstruction* (comme utilisé dans Hifi et Michrafy [39]). Après avoir appliqué les deux stratégies, la phase d'intensification est appelée de nouveau, afin d'améliorer la qualité de la solution courante. Dans la suite, nous allons montrer comment combiner les deux stratégies pour rechercher une série de nouvelles solutions.

Soit F la solution réalisable courante obtenue à la première phase (ou par la recherche itérative interne). Soit α une constante, tel que $\alpha \in [0, 100]$. dénotant le pourcentage de services à supprimer de l'ensemble S, c'est à dire, α services sont supprimés de la solution courante F. Ainsi, la phase de diversification peut être résumée aux deux procédures suivantes :

- *Procédure de dégradation* : Considérons $S^{(\alpha)}$ comme l'ensemble des services supprimés dans la solution courante F. Soit $\overline{S}^{(\alpha)}$ l'ensemble complémentaire de l'ensemble de services $S^{(\alpha)}$. Cette procedure permet de supprimer $\alpha\%$ des services de la solution réalisable courante F, afin d'obtenir une nouvelle solution partielle dégradée $F^{(\alpha)}$.
- *Procédure de reconstruction* : Sert à compléter la solution partielle dégradée $F^{(\alpha)}$, obtenue après la procédure de dégradation en appliquant l'algorithme de Johnson pour reconstruire une nouvelle solution réalisable courante F' en utilisant $\overline{S}^{(\alpha)}$.

Ces deux procédures sont appelés successivement pour tenter d'améliorer la qualité de la solution courante avec la possibilité de réaliser des intensifications régulières au cours des itérations. Les principales étapes de la phase de diversification sont détaillées dans l'algorithme 6. Ce dernier, montre le principe de la diversification en utilisant les procedures de *dégradation* et *reconstruction*. Au début. l'algorithme 6 commence en mettant la solution initiale égale à la solution obtenue par l'algorithme 5 (bien sûr, la solution fournie peut également être atteinte à chaque boucle interne). La boucle principale décrit le principe de la phase de diversification en combinant à la fois les procédures de *dégradation* et de *reconstruction*. Ainsi, elle commence toujours avec la meilleure solution courante obtenue. ensuite, la procédure de dégradation est appelée pour supprimer de façon aléatoire $\alpha\%$ des services de la solution courante. Et après. essayer d'améliorer la qualité de la solution par la procédure de reconstruction. A chaque étape de la boucle locale, la nouvelle solution fournie est améliorée en appelant l'algorithme d'intensification (lignes 8 et 9). Enfin, le processus est répété en utilisant la meilleure solution courante. Le nombre d'itérations est encadré par des paramètres qui représentent respectivement, le nombre d'itérations global et le nombre d'itérations local (notés : $Iter\_global$ et $Iter\_local$).

Algorithm 6 Une variante de la recherche par voisinage

Entrées: Une solution initiale réalisable F' de pénalité $V(F')$; $Iter\_global$ et $Iter\_local$.
Sorties: Une meilleure solution F_{global} de pénalité $V(F_{global})$.

$F_{best} \leftarrow F'$;
Tant que ($Iter\_global < 50$) **faire**
  $F_{global} \leftarrow F_{best}$;
  Tant que ($Iter\_local < 20$) **faire**
    $F_{local} \leftarrow F_{best}$;
    Appliquer la `stratégie de dégradation` sur la solution courante F_{local} ;
    Appeler la `stratégie de reconstruction` pour compléter la solution partielle ;
    Soit F_{global} la nouvelle solution obtenue ;
    Appeler l'`algorithme` 5 pour améliorer la solution courante F_{global} ;
    si ($V(F_{global}) < V(F_{best})$). **alors** $F_{best} = F_{global}$;
  fin du « Tant que »
fin du « Tant que »

5.4 Les résultats numériques

Cette section examine l'efficacité de l'approche proposée (notée : ANS). Celle-ci a été testée sur les instances de Gualandi *et al.* [32] et sur quelques instances de grandes tailles que nous avons généré.

Les résultats obtenus par notre heuristique codée en `C++` et exécutée sur une machine Core Duo 2.9 Ghz. sont comparés avec les meilleurs résultats disponibles de la littérature et avec les résultats du solveur Cplex[45] version 12.5 (limité à une heure d'exécution), basé sur le modèle de la section 5.2.2.

Lors des tests numériques. nous avons constaté certaines améliorations sur certains types d'instances. Globalement, notre approche reste compétitive.

5.4.1 Description des instances du K-CmBCP

Deux types d'instances peuvent être considérés : des instances *symétriques*, où le nombre de services [4] $|S|$ et le nombre de clients [5] $|T|$ sont égaux et, des instances *non-symétriques*, où parfois le nombre de clients est plus grand que le nombre de services (ou, vice versa).

A noter que, dans cette thèse, nous avons considéré des instances *symétriques*. La table 5.1 montre les caractéristiques des deux groupes d'instances que nous avons considéré : le premier groupe ($I1$ à $I12$) représente les instances extraites de Gualandi *et al.* [32] et le deuxième groupe représente les instances que nous avons généré en utilisant le même générateur que Gualandi *et al.* [32]. Dans les deux groupes de la table 5.1, la colonne 1 indique le label de l'instance, les colonnes de 2 à 4 donnent

4. Sommets du côté gauche du graphe biparti
5. Sommets du côté droit du graphe biparti

respectivement le nombre de services $|S|$, le nombre de clients $|T|$ et le nombre de clusters $|K|$ et enfin, la colonne 5 donne la densité d du graphe biparti associé.

Les instances du K-CmBCP sont composées de plusieurs sous-groupes selon la variation du nombre de services $|S|$ (ou, nombre de clients $|T|$), du nombre de clusters $|K|$ et une variation de la densité d ($d = 0.3$, $d = 0.5$, $d = 0.7$).

Inst.	S	T	k	d	Inst.	S	T	k	d
I1.1	15	15	2	0.3	I13.1	120	120	5	0.3
I1.2	15	15	2	0.5	I13.2	120	120	5	0.5
I1.3	15	15	2	0.7	I13.3	120	120	5	0.7
I2.1	15	15	5	0.3	I14.1	120	120	10	0.3
I2.2	15	15	5	0.5	I14.2	120	120	10	0.5
I2.3	15	15	5	0.7	I14.3	120	120	10	0.7
I3.1	18	18	2	0.3	I15.1	120	120	15	0.3
I3.2	18	18	2	0.5	I15.2	120	120	15	0.5
I3.3	18	18	2	0.7	I15.3	120	120	15	0.7
I4.1	18	18	5	0.3	I16.1	150	150	5	0.3
I4.2	18	18	5	0.5	I16.2	150	150	5	0.5
I4.3	18	18	5	0.7	I16.3	150	150	5	0.7
I5.1	20	20	2	0.3	I17.1	150	150	10	0.3
I5.2	20	20	2	0.5	I17.2	150	150	10	0.5
I5.3	20	20	2	0.7	I17.3	150	150	10	0.7
I6.1	20	20	5	0.3	I18.1	150	150	15	0.3
I6.2	20	20	5	0.5	I18.2	150	150	15	0.5
I6.3	20	20	5	0.7	I18.3	150	150	15	0.7
I7.1	50	50	5	0.3	I19.1	200	200	5	0.3
I7.2	50	50	5	0.5	I19.2	200	200	5	0.5
I7.3	50	50	5	0.7	I19.3	200	200	5	0.7
I8.1	50	50	10	0.3	I20.1	200	200	10	0.3
I8.2	50	50	10	0.5	I20.2	200	200	10	0.5
I8.3	50	50	10	0.7	I20.3	200	200	10	0.7
I9.1	80	80	5	0.3	I21.1	200	200	15	0.3
I9.2	80	80	5	0.5	I21.2	200	200	15	0.5
I9.3	80	80	5	0.7	I21.3	200	200	15	0.7
I10.1	80	80	10	0.3	I22.1	300	300	5	0.3
I10.2	80	80	10	0.5	I22.2	300	300	5	0.5
I10.3	80	80	10	0.7	I22.3	300	300	5	0.7
I11.1	100	100	5	0.3	I23.1	300	300	10	0.3
I11.2	100	100	5	0.5	I23.2	300	300	10	0.5
I11.3	100	100	5	0.7	I23.3	300	300	10	0.7
I12.1	100	100	10	0.3	I24.1	300	300	15	0.3
I12.2	100	100	10	0.5	I24.2	300	300	15	0.5
I12.3	100	100	10	0.7	I24.3	300	300	15	0.7

TABLE 5.1 – Description de la caractéristique de quelques instances du K-CmBCP.

5.4.2 Paramétrages

Dans les résultats numériques, deux principaux paramétrages doivent être pris en compte pour notre heuristique : le critère d'arrêt et le pourcentage de suppression des services appartenant à la solution courante. Afin de maintenir la diversité des solutions, nous mettons le `critère d'arrêt local à 20 itérations et le critère d'arrêt global à 50 itérations`.

Dans nos résultats numériques, quatre valeurs du paramètre α sont considérées : $\alpha \in \{2\%;\ 5\%;\ 10\%;\ 15\%\}$.

Dans la table 5.2 : la colonne 1 montre le label de l'instance. Les colonnes 2 à 9, donnent respectivement la solution (UB) et le temps de résolution (CPU) obtenus avec $\alpha = 2\%$, $\alpha = 5\%$, $\alpha = 10\%$ et $\alpha = 15\%$. Les colonnes 10 à 11 donnent le meilleur résultats de la littérature (voir Gualandi *et al.*[32]). Et enfin, les colonnes 12 à 13 affichent le résultats du solveur Cplex 12.5 pour le modèle utilisé en limitant le temps de résolution à 3600 secondes. La dernière ligne de cette table donne, la valeur moyenne des solutions et des temps de résolution obtenus par *ANS* et par le solveur Cplex.12.5. Cette table permet aussi de comparer nos résultats aux meilleurs résultats de la littérature, notés : UB_{GMM} et CPU_{GMM}.

	Variation de α pour les instances de la littérature vs Meilleurs résultats de la littérature et Cplex.										
Inst.	$UB_{2\%}$	$CPU_{2\%}$	$UB_{5\%}$	$CPU_{5\%}$	$UB_{10\%}$	$CPU_{10\%}$	$UB_{15\%}$	$CPU_{15\%}$	UB_{GMM}	CPU_{GMM}	UB_{Cplex}
I7.1	1319.0	20.00	1319.0	19.10	1321	17.50	1324	15.23	1321	107	1423
I7.2	1078.0	01.74	1072.0	01.55	1070	01.33	1072	0.79	1072	11	1091
I7.3	672.0	02.51	671.0	02.12	673	01.87	675	00.91	672	16	676
I8.1	938.0	17.90	938.0	17.41	938	16.53	940	10.83	938	137	1135
I8.2	876.0	07.46	875.4	07.24	877	06.89	880	05.02	876	50	930
I8.3	577.0	05.61	575.0	05.39	578	04.85	582	03.15	577	43	587
I9.1	3820.2	199.83	3815.0	199.45	3888	198.75	3824	120.37	3819	872	4166
I9.2	2862.0	29.91	2862.0	29.60	2895	29.49	2866	19.96	2862	167	2931
I9.3	1769.2	09.87	1768.4	09.13	1769	08.98	1774	07.06	1769	46	1775
I10.1	3202.0	74.22	3202.0	71.80	3202	71.20	3231	52.74	3202	780	3735
I10.2	2571.0	33.60	2571.0	33.10	2575	32.15	2574	20.16	2571	400	2675
I10.3	1618.4	20.26	1618.0	19.96	1619	19.64	1632	13.00	1618	144	1634
I11.1	6248.0	87.31	6248.0	85.20	6281	83.20	6248	75.20	6248	452	6645
I11.2	4655.6	78.41	4650.0	71.71	4669	71.33	4669	65.41	4658	198	4716
I11.3	2842.0	18.01	2842.0	17.53	2844	17.13	2866	13.62	2842	87	2854
I12.1	4298.0	202.10	4298.0	200.31	4297	199.17	4298	170.49	4298	893	6119
I12.2	5498.4	586.50	5427.2	583.89	5442	583.55	5455	486.52	5428	5427	9111
I12.3	2644.0	146.01	2644.0	146.00	2645	144.70	2644	112.56	2644	441	2704
Moy.	2638.27	85.61	**2633.11**	84.47	2643.50	83.79	2641.89	**66.28**	2634.17	570.61	3050.39

TABLE 5.2 – Effet de α sur la qualité des solutions (UB) et les temps de résolution (CPU).

Nous avons tracé les diagrammes (voir la figure 5.3), afin de voir l'évolution moyenne de nos résultats expérimentaux (solutions (UB) et temps de résolution (CPU)) par rapport à la variation de α.

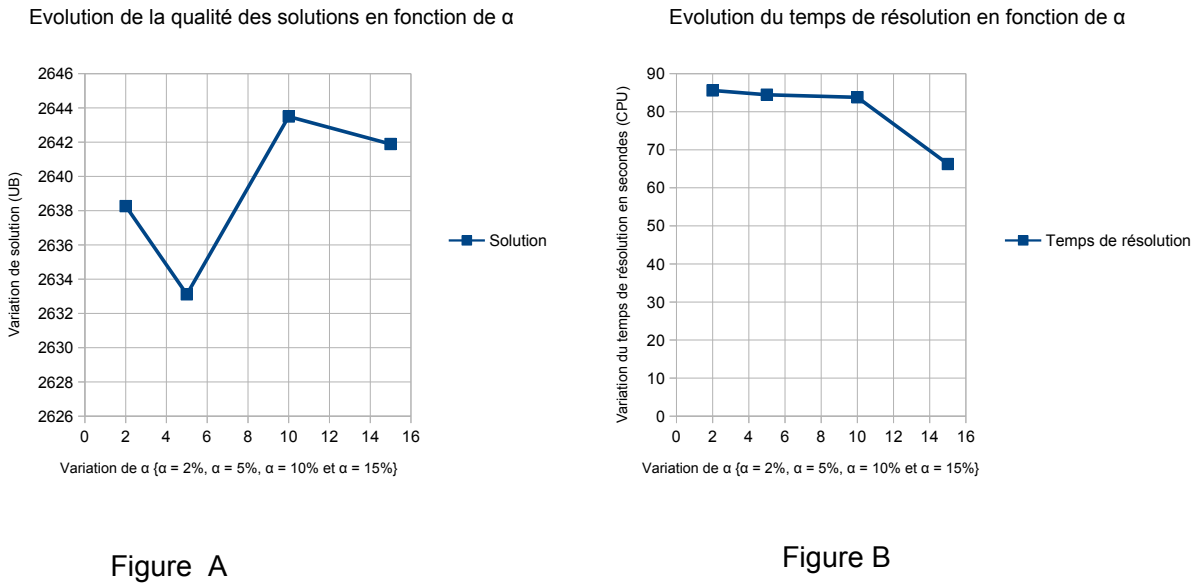

Figure A

Figure B

FIGURE 5.3 – Evolution du résultat de notre approche en fonction de α.

Dans la figure 5.3, la figure A, montre que lorsque la valeur de $\alpha = 5\%$, la qualité des solutions obtenues pour certaines instances devient meilleure par rapport à celle de la littérature et la figure B, montre que notre approche demande plus de temps de résolution pour améliorer certaines instances de la littérature lorsque $\alpha = 2\%$ et moins de temps de résolution quand $\alpha = 15\%$. Dans cette table, on remarque globalement, que plus la valeur de α augmente, plus, le temps de résolution est meilleur.

En ce qui concerne les instances de Gualandi *et al.* [32], nous avons constaté qu'avec la valeur de $\alpha = 5\%$, nous obtenons la meilleure valeur moyenne (notée : $Moy.$) qui est compétitive par rapport à la littérature (voir la table 5.2).

Dans la suite de nos expérimentations pour les instances de Gualandi *et al.* [32], nous nous sommes intéressés au paramètre $\alpha = 5\%$, étant donné que globalement la qualité des solutions obtenues est meilleure lorsque nous fixons α à 5%.

5.4.3 Effet des procedures de dégradation et reconstruction

Cette sous-section évalue l'effet des procedures de dégradation et reconstruction sur notre approche. Rappelons que l'approche proposée fonctionne comme suit : d'abord, on degrade la solution initiale ou solution courante en supprimant $\alpha\%$ des variables fixées. Ensuite, on reconstruit la solution partielle obtenue en utilisant la procédure gloutonne et on l'améliore en utilisant la phase d'intensification. Enfin, on répète la même résolution sur la dernière solution jusqu'à ce qu'il n'y ait pas de meilleure solution ou jusqu'à ce que le nombre maximum d'itérations soit atteint.

			Les solutions du ANS avec $\alpha = 5\%$						
#Inst	ILPM V_{Cplex}	GMM V_{GMM}	1	2	3	4	5	*Moy.*	Best
I7.1	1423 .0	1321.0	1319.0	1319.0	1319.0	1319.0	1319.0	1319 .0	1319.0
I7.2	1091.0	1072.0	1072.0	1072.0	1072.0	1072.0	1072.0	1072.0	1072.0
I7.3	676.0	672.0	671.0	671.0	671.0	671.0	671.0	671.0	671.0
I8.1	1135.0	938.0	938.0	938.0	938.0	938.0	938.0	938.0	938.0
I8.2	930.0	876.0	875.0	876.0	876.0	875.0	875.0	875.4	875.0
I8.3	587.0	577.0	575.0	575.0	575.0	575.0	575.0	575.0	575.0
I9.1	4166.0	3819.0	3815.0	3815.0	3815.0	3815.0	3815.0	3815.0	3815.0
I9.2	2931.0	2862.0	2862.0	2862.0	2862.0	2862.0	2862.0	2862.0	2862.0
I9.3	1775.0	1769.0	1768.0	1769.0	1768.0	1768.0	1769.0	1768.4	1768.0
I10.1	3735.0	3202.0	3202.0	3202.0	3202.0	3202.0	3202.0	3202.0	3202.0
I10.2	2675.0	2571.0	2571.0	2571.0	2571.0	2571.0	2571.0	2571.0	2571.0
I10.3	1634.0	1618.0	1618.0	1618.0	1618.0	1618.0	1618.0	1618.0	1618.0
I11.1	6645.0	6248.0	6248.0	6248.0	6248.0	6248.0	6248.0	6248.0	6248.0
I11.2	4716.0	4658.0	4650.0	4650.0	4650.0	4650.0	4650.0	4650.0	4650.0
I11.3	2854.0	2842.0	2842.0	2842.0	2842.0	2842.0	2842.0	2842.0	2842.0
I12.1	6119.0	4298.0	4298.0	4298.0	4298.0	4298.0	4298.0	4298.0	4298.0
I12.2	9111.0	5428.0	5427.0	5428.0	5427.0	5427.0	5427.0	5427.2	5427.0
I12.3	2704.0	2644.0	2644.0	2644.0	2644.0	2644.0	2644.0	2644.0	2644.0
Moyenne	3050.39	2634.17	2633.06	2633.22	2633.11	2633.06	2633.11	2633.11	2633.06

TABLE 5.3 – Performance du ANS vs $ILPM$ résolu avec Cplex & GMM algorithm.

La table 5.3 montre les résultats obtenus par notre approche et les meilleurs résultats de la littérature extraits de Gualandi *et al.* [32] (noté GMM) et aussi les résultats obtenus par le modèle mathématique ILPM (voir la section 5.2.2), lorsqu'on le résout en utilisant le solveur Cplex 12.5 (à noter que nous avons limité le temps de résolution du solveur Cplex à une heure). La colonne 1 de cette table représente le label de l'instance ; la colonne 2 affiche les solutions obtenues par le solveur Cplex sur le modèle mathématique ILPM ; la colonne 3 donne les meilleures solutions de la littérature (Gualandi *et al.*[32]) ; les colonnes de 4 à 8 affichent les cinq solutions obtenues par la méthode ANS et finalement, les colonnes 9 et 10 montrent, respectivement, les valeurs moyennes des solutions sur les cinq exécutions et la meilleure solution fournie par ANS pendant les cinq exécutions. L'analyse de la table 5.3 montre que :

1. On peut facilement observer l'infériorité du modèle mathématique ILPM résolu avec le solveur Cplex. étant donné qu'il n'améliore pas les solutions de la littérature.

2. Les meilleures solutions de la littérature (noté : GMM) améliore 10 meilleures solutions sur 18, cela représente un pourcentage de 55.56% des meilleures solutions. En effet, GMM domine les résultats du solveur Cplex sur toutes les instances, mais, ses solutions restent inférieures à celle obtenues par *ANS* (voir la dernière colonne de la table 5.3).
3. Parmi les cinq exécutions réalisées pour *ANS*. On peut observer que les valeurs moyennes (voir la dernière ligne de la table 5.3) sont meilleures que les valeurs moyennes du GMM. Ainsi, *ANS* est en mesure de réaliser une valeur moyenne de solutions, variant de 2633.22 (exécution 2) à 2633.06 (exécutions 1 et 4) tandis-que GMM réalise une valeur moyenne de 2634.17.
4. En ce qui concerne les meilleures solutions réalisées par *ANS* (voir colonne 10), nous pouvons conclure que l'approche *ANS* est capable d'obtenir 8 nouvelles solutions par rapport aux solutions de Gualandi *et al.* [32] sur les instances de la littérature, ce qui représente un pourcentage de plus de 44% des instances testées.

Concernant les instances de grandes tailles générées :

Comme indiqué précédemment, nous avons généré des instances de grandes tailles afin de tester l'efficacité de notre approche (notée : ANS). Ainsi, nous avons comparé les solutions obtenues par ANS à celles obtenues par le solveur Cplex 12.5.

La table 5.4 affiche les résultats (solutions et temps de résolution) trouvés lorsqu'on applique ANS sur les grandes instances : les colonnes 2 et 3 donnent respectivement les solutions et les temps de résolution obtenus par ANS avec $\alpha = 5\%$, les colonnes 4 et 5 donnent respectivement les solutions et les temps de résolution obtenus par ANS avec $\alpha = 10\%$ et les colonnes 6 et 7 donnent respectivement les solutions et les temps de résolution obtenus par ANS avec $\alpha = 15\%$. Etant donné que, ces grandes instances n'ont pas été testées dans la littérature, nous ne pouvons comparer nos résultats qu'avec ceux obtenus par le solveur Cplex 12.5 avec une limite du temps de résolution à 1 heure (voir les colonnes 8 et 9).
L'analyse de la table 5.4 montre que :

1. Les solutions fournies par notre approche ANS sont meilleures que celles obtenues par le solveur Cplex 12.5 (voir colonne 2, colonne 4, colonne 6 et colonne 8).
2. L'effet de la variation de α, concernant, le temps de résolution sur les grandes instances est presque le même que sur les instances de Gualandi*et al.* [32], c'est à dire : lorsque α augmente, le temps de résolution pour certaines instances s'améliore. On remarque aussi que, pour toutes les instances de grande taille que nous avons testées, les solutions obtenues par ANS avec $\alpha = 5\%$ et $\alpha = 10\%$ sont meilleures ou égales à celles obtenues par ANS avec $\alpha = 15\%$. Cependant, nous observons aisément que globalement la qualité des solutions obtenues pour les instances de grande taille, est meilleure lorsque, nous fixons α à 10%.

	Variation de α pour les grandes instances générées vs Cplex.						
#Inst	$UB_{5\%}$	$CPU_{5\%}$	$UB_{10\%}$	$CPU_{10\%}$	$UB_{15\%}$	$CPU_{15\%}$	UB_{Cplex}
I13.1	8415.00	201.48	8415.00	198.95	8418.00	163.32	8420.00
I13.2	6019.00	81.29	6019.00	75.51	6019.00	54.46	6106.00
I13.3	4264.00	105.63	4264.00	97.45	4264.00	87.93	4344.00
I14.1	8195.40	237.92	8195.00	201.41	8201.00	179.94	8205.00
I14.2	5821.00	179.50	5821.00	175.63	5822.00	50.05	5841.00
I14.3	4063.00	92.55	4063.00	88.42	4078.00	72.18	4141.00
I15.1	7808.00	201.42	7808.00	199.85	7861.00	179.85	8859.00
I15.2	5728.00	173.37	5728.00	167.13	5739.00	133.52	5908.00
I15.3	3913.00	75.53	3913.00	68.51	3927.00	54.51	4388.00
I16.1	14541.00	217.35	14541.00	204.55	14544.00	195.62	14547.00
I16.2	2005.60	144.35	2005.00	126.33	2101.00	98.66	2136.00
I16.3	8480.00	104.42	8480.00	101.79	8480.00	88.74	8594.00
I17.1	14095.00	246.39	14095.00	225.46	14146.00	201.61	14821.00
I17.2	1528.00	172.68	1528.00	114.74	1548.00	108.55	1687.00
I17.3	8127.00	103.25	8127.00	99.68	8130.00	68.92	8810.00
I18.1	15312.00	199.37	15312.00	198.74	15371.00	176.50	16102.00
I18.2	11082.00	198.82	11082.00	187.95	11406.00	143.51	11583.00
I18.3	8971.00	167.82	8971.00	154.63	8994.00	105.20	9596.00
I19.1	29318.00	257.72	29318.00	244.66	29329.00	223.17	29353.00
I19.2	19456.00	231.46	19456.00	214.34	19456.00	198.54	19518.00
I19.3	19457.00	219.34	19457.00	209.80	19457.00	167.10	19829.00
I20.1	28374.00	293.31	28374.00	263.31	28383.00	247.92	29701.00
I20.2	18989.00	187.28	18989.00	177.42	18995.00	173.35	19931.00
I20.3	19002.50	99.74	19002.00	98.45	19004.00	59.65	19932.00
I21.1	41861.00	278.95	41861.00	272.23	42074.00	248.94	45385.00
I21.2	21547.00	242.38	21547.00	214.70	21557.00	200.35	22443.00
I21.3	20526.00	204.63	20526.00	196.63	21497.00	173.45	22377.00
I22.1	68219.00	204.35	68219.00	222.46	69270.00	310.21	72059.00
I22.2	56219.00	152.69	56219.00	148.55	56219.00	155.42	56694.00
I22.3	56463.00	161.22	56463.00	122.25	56463.00	99.87	56938.00
I23.1	72008.00	302.10	72008.00	298.82	72213.00	286.35	73806.00
I23.2	55764.00	296.53	55764.00	277.56	55768.00	244.23	56694.00
I23.3	55008.00	184.42	55008.00	165.81	55995.00	129.46	56938.00
I24.1	72861.00	371.14	72861.00	335.96	73909.00	315.84	75321.00
I24.2	55239.00	326.82	55218.00	301.45	55239.00	278.94	58026.00
I24.3	55508.00	298.75	55508.00	288.95	55540.00	276.19	56713.00
Moy.	25116.32	200.44	**25115.69**	187.22	25261.58	**165.33**	25992.94

TABLE 5.4 – Performance de ANS vs Cplex sur des grandes instances générées.

3. D'une manière générale, les résultats numériques montrent que même pour les instances de grandes tailles, notre heuristique donne de meilleurs résultats par rapport au solveur Cplex.12.5 pour le modèle mathématique utilisé, en utilisant le temps de résolution à 3600 secondes.

5.5 Conclusion

Dans ce chapitre nous avons proposé une variante de la recherche par voisinage afin de résoudre approximativement le problème du K-clusters dans un graphe biparti. Cette approche est basée sur les procedures de dégradation et reconstruction. Ainsi, la première procedure est utilisée pour diversifier le processus de recherche et la seconde procedure permet d'améliorer la solution partiellement dégradée. Après

les deux procedures, notre approche fait appel à la phase d'intensification qui est une recherche locale qui tentera d'améliorer la qualité d'une série de solutions.

Les résultats numériques ont montré que l'approche proposée donne des résultats encourageants en améliorant la qualité de 44% des instances connues de la littérature et de 100% les solutions fournies par le solveur Cplex.12.5 sur le modèle utilisé (voir section 5.2.2) pour les instances de grandes tailles.

Cependant, d'autres points d'amélioration restent possibles, notamment sur l'aspect non-symétrique des instances, qui est une perspective de recherche.

L'heuristique *ANS* à fait l'objet d'un *proceding* dans une conférence internationale *Modelling, Computation and Optimization in Information Systems and Management Sciences, Proceedings of MCO 2015 [40]* et a été présentée lors du 16ème conférence de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF 2015 [38].

Conclusion générale et perspectives

Chapitre 6

Conclusion générale et perspectives

Dans cette thèse, nous avons étudié deux problème de l'optimisation combinatoire : un problème d'orientation d'équipe qui est plus connu sous sa nomination anglophone "*Team Orienteering Problem : TOP*" et un problème de K-clusters dans un graphe biparti, lui aussi connu comme "*The K-Clustering minimum Biclique Completion Problem : K-CmBCP*".

Notre travail est essentiellement basé sur la proposition de méthodes approchées permettant de résoudre efficacement les deux problématique que nous abordons.

Dans la première partie, nous avons élaboré un état de l'art et une classification des variantes pour les problèmes de tournées de véhicules en général, et particulièrement pour le problème d'orientation d'équipe. Ensuite, après un rappel sur les graphes et la programmation linéaire, nous avons énuméré les différentes méthodes classiques pour la résolution des problèmes de l'optimisation combinatoire de façon générale. Dans le chapitre 3, nous avons proposé une approche pour la resolution du problème d'orientation d'équipe. Ainsi, cette approche combine deux phases complémentaires : la première phase permet la construction d'un chemin de départ appelé *chemin elite* et la deuxième phase est utilisée pour construire une solution réalisable en améliorant la solution obtenue à la première phase. Nous avons testé cette approche sur des instances du problème d'orientation d'équipe et nous avons constaté que notre algorithme permet d'avoir de meilleurs résultats par rapport à la littérature.

Dans la deuxième partie de cette thèse, nos travaux de recherche ont porté sur la résolution du problème de K-clusters dans un graphe biparti. Dans le chapitre 4, après un survol de l'état de l'art du problème de K-clusters dans un graphe biparti, nous avons présenté cinq modèles mathématiques pour le K-CmBCP, à partir desquels nous avons obtenu le modèle mathématique linéaire en nombre entiers (ILPM) qui a été ensuite utilisé dans le chapitre 5 pour obtenir des résultats numériques avec le solveur Cplex version 12.5 pour les instances du K-CmBCP issues de la literature et pour les instances de grandes tailles que nous avons générées aléatoirement.

Dans le chapitre 5, nous avons présenté une méthode de recherche par voisinage permettant une résolution approchée du problème de K-clusters dans un graphe biparti. Cette approche est utilise trois étapes : la première étape combine deux algorithmes : l'algorithme de Johnson [47, 29] et un algorithme glouton ; la deuxième

étape de notre approche applique une stratégie d'intensification qui utilise des techniques de recherche locale et permutations ; enfin, la troisième et dernière étape est une diversification qui se base sur deux procedures : la procedure de dégradation et la procedure de réconstruction.

Nous avons vu dans les résultats numériques (section 5.4, du chapitre 5) que notre approche est capable d'obtenir des meilleurs résultats que ceux publiés dans la littérature.

Parmi les perspectives à court terme, il nous parait intéressant d'introduire le parallélisme dans l'approche que nous avons proposé pour le problème d'orientation d'équipe et d'étudier le problème d'orientation d'équipe avec fenêtres de temps.

Ensuite, Améliorer notre algorithme, proposé pour la résolution approchée du problème de K-clusters dans un graphe biparti : (i) en développant un meilleur mécanisme de dégradation et réconstruction ; (ii) en reconsidérant le sens de cluster.

Bibliographie

[1] P. Vansteenwegen W. Souriau O. Arbelaitz M. T. Linaza A. Garcia, . Solving multi constrained team orienteering problems to generate tourist routes. *4OR*, Psinger, Sep. 2010. (Cité en pages 15 et 16.)

[2] Majid Aldaihani and Maged M Dessouky. Hybrid scheduling methods for paratransit operations. *Computers & Industrial Engineering*, 45(1) :75–96, 2003. (Cité en page 12.)

[3] Marco Ambrosini, Thomas Caruso, Sara Foresti, and Giovanni Righini. A grasp for the pickup and delivery problem with rear loading. *Information Technology Department, Technical Report*, (65), 2004. (Cité en page 12.)

[4] Gilbert Babin, Stéphanie Deneault, and Gilbert Laporte. Improvements to the or-opt heuristic for the symmetric travelling salesman problem. *Journal of the Operational Research Society*, 58(3) :402–407, 2007. (Cité en page 33.)

[5] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan Marakada Shetty. *Nonlinear programming : theory and algorithms*. John Wiley & Sons, 2013. (Cité en page 66.)

[6] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10) :767, 1956. (Cité en pages 29 et 31.)

[7] Claude Berge. *Théorie générale des jeux à n personnes*, volume 138. Gauthier-Villars Paris, 1957. (Cité en page 26.)

[8] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995. (Cité en page 29.)

[9] José Brandao. A new tabu search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research*, 173(2) :540 – 555, 2006. (Cité en page 12.)

[10] S. E. Butt and T. M. Cavalier. A heuristic for the multiple tour maximum collection problem. *Comput. Oper. Res*, vol. 21, no. 1, pp. 101-111, Jan. 1994. (Cité en pages 14, 16 et 40.)

[11] Steven E Butt and David M Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4) :427–441, 1999. (Cité en page 41.)

[12] M. G. Speranza C. Archetti, A. Hertz. Metaheuristics for the team orienteering problem. *J Heuristics*, vol. 13, no. 1, pp.49-76, Feb. 2007. (Cité en pages 15 et 50.)

[13] Prasad Chalasani and Rajeev Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6) :2133–2149, 1999. (Cité en page 12.)

[14] I Chao, Bruce L Golden, Edward A Wasil, et al. The team orienteering problem. *European journal of operational research*, 88(3) :464–474, 1996. (Cité en pages 13 et 49.)

[15] G. Clarke and J.W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12 :568–581, 1964. (Cité en page 32.)

[16] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. Vehicle routing. *Transportation, handbooks in operations research and management science*, 14 :367–428, 2006. (Cité en page 12.)

[17] Peter I Cowling and Ralf Keuthen. Embedded local search approaches for routing optimization. *Computers & operations research*, 32(3) :465–490, 2005. (Cité en page 34.)

[18] José Crispim and José Brandão. Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. *Journal of the Operational Research Society*, 56(11) :1296–1302, 2005. (Cité en page 12.)

[19] Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. A pso-based memetic algorithm for the team orienteering problem. In *Applications of Evolutionary Computation*, pages 471–480. Springer, 2011. (Cité en page 50.)

[20] Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229(2) :332–344, 2013. (Cité en page 50.)

[21] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1) :pp. 8091,, 1959. (Cité en page 10.)

[22] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2005. (Cité en page 29.)

[23] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system : optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on*, 26(1) :29–41, 1996. (Cité en page 36.)

[24] Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995. (Cité en page 35.)

[25] Hassan EL Fahim and Brahim Aghezzaf. The capacitated team orienteering problem with time windows. (Cité en page 16.)

[26] Nathalie Faure, Philippe Chrétienne, Eric Gourdin, and Francis Sourd. Biclique completion problems for multicast network design. *Discrete Optimization*, 4(3) :360–377, 2007. (Cité en pages 63, 65 et 79.)

[27] Thomas A Feo, Mauricio GC Resende, and Stuart H Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5) :860–878, 1994. (Cité en page 34.)

[28] Matteo Fischetti, Paolo Toth, and Daniele Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5) :846–859, 1994. (Cité en page 30.)

[29] Michael R Garey and David S Johnson. Computer and intractability. *A Guide to the Theory of NP-Completeness*, 1979. (Cité en pages 75, 80 et 95.)

[30] Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1999. (Cité en page 34.)

[31] Stefano Gualandi. k-clustering minimum biclique completion via a hybrid cp and sdp approach. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 87–101. Springer, 2009. (Cité en pages 59, 63 et 65.)

[32] Stefano Gualandi, Francesco Maffioli, and Claudio Magni. A branch-and-price approach to k-clustering minimum biclique completion problem. *International Transactions in Operational Research*, 20(1) :101–117, 2013. (Cité en pages 63, 65, 76, 83, 85, 87, 88 et 89.)

[33] D.-C. Dang H. Bouly and A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR-Q J Oper Res*, vol.8, no. 1, pp. 49-70, Mar. 2010. (Cité en pages 15 et 41.)

[34] John A Hartigan and Manchek A Wong. Algorithm as 136 : A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979. (Cité en page 32.)

[35] Keld Helsgaun. *An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic*. PhD thesis, Roskilde University. Department of Computer Science, 2006. (Cité en page 33.)

[36] Mhand Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25(11) :925–940, 1998. (Cité en page 43.)

[37] Mhand Hifi, Moussa Ibrahim, and Toufik Saadi. A two-stage resolution search-based heuristic for the team orienteering problem. In *2014 2nd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 442–447. IEEE, 2014. (Cité en pages 50 et 55.)

[38] Mhand Hifi, Moussa Ibrahim, and Saleh Sagvan. Une recherche par perturbation de voisinage pour le problème d'affectation de transactions dans des services. In *16ème conférence de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF*. Marseille 2015, 2015. (Cité en page 91.)

[39] Mhand Hifi and Mustapha Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6) :718–726, 2006. (Cité en pages 79 et 82.)

[40] Mhand Hifi, Ibrahim Moussa, Toufik Saadi, and Sagvan Saleh. An adaptive neighborhood search for k-clustering minimum bi-clique completion problems. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 15–25. Springer, 2015. (Cité en page 91.)

[41] Mhand Hifi and Toufik Saadi. Résolution hybride du problème de découpe à deux dimensions et à deux niveaux. (Cité en page 43.)

[42] Arild Hoff and Arne Løkketangen. Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European Journal of Operations Research*, 14(2) :125–140, 2006. (Cité en page 12.)

[43] John H Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975. (Cité en page 36.)

[44] B. L. Golden I. M. Chao and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, vol. 88, no. 3, pp. 464-474, Feb. 1996. (Cité en pages 14, 39 et 41.)

[45] IBM ILOG. Inc. cplex 12.5 user manual, 2012. (Cité en page 83.)

[46] Alexis Jacquemin and Henry Tulkens. *Fondements d'économie politique*. 1986. (Cité en page 17.)

[47] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1) :61–68, 1954. (Cité en pages 75, 80 et 95.)

[48] Byung-In Kim, Hong Li, and Andrew L. Johnson. An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems with Applications*, 40(8) :3065 – 3072, 2013. (Cité en page 50.)

[49] Scott Kirkpatrick, MP Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598) :671–680, 1983. (Cité en page 34.)

[50] Z. Feng L. Ke, C. Archetti. Ants can solve the team orienteering problem. *Computers and Industrial Engineering*, vol. 54, no. 3, pp. 648-665,, Apr. 2008. (Cité en pages 15 et 50.)

[51] Nacima Labadie, Renata Mansini, Jan Melechovskỳ, and Roberto Wolfler Calvo. The team orienteering problem with time windows : An lp-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1) :15–27, 2012. (Cité en page 15.)

[52] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif. Evolutionary algorithms for periodic arc routing problems. *European Journal of Operational Research*, 165(2) :535–553, 2005. (Cité en page 12.)

[53] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica : Journal of the Econometric Society*, pages 497–520, 1960. (Cité en page 29.)

[54] Eugene L Lawler and David E Wood. Branch-and-bound methods : A survey. *Operations research*, 14(4) :699–719, 1966. (Cité en page 29.)

[55] N Maculan. Relaxation lagrangienne : le problème du knapsack 0-1. *Canadian Journal of Operational Research and Information Processing*, 21 :315–327, 1983. (Cité en page 29.)

[56] Claudio Patrizio Magni. Biclique completion problem : models and algorithms. *Master Project, Politecnico di Milano*, 2009. (Cité en pages 65 et 71.)

[57] Rafael Martí, Manuel Laguna, and Fred Glover. Principles of scatter search. *European Journal of Operational Research*, 169(2) :359–372, 2006. (Cité en page 35.)

[58] Anand Meka. *Distributed algorithms for summarizing and querying spatio-temporal data in sensor networks*. ProQuest, 2007. (Cité en page 67.)

[59] Aristide Mingozzi. The multi-depot periodic vehicle routing problem. In *Abstraction, Reformulation and Approximation*, pages 347–350. Springer, 2005. (Cité en page 13.)

[60] Michel Minoux. *Programmation mathématique : théorie et algorithmes*, volume 1. Dunod Paris, 1983. (Cité en page 29.)

[61] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, 1997. (Cité en page 34.)

[62] Pablo Moscato, Carlos Cotta, and Alexandre Mendes. Memetic algorithms. In *New optimization techniques in engineering*, pages 53–85. Springer, 2004. (Cité en page 36.)

[63] Patrenahalli M Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *Computers, IEEE Transactions on*, 100(9) :917–922, 1977. (Cité en page 29.)

[64] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1) :17–30, 2006. (Cité en page 11.)

[65] James Orlin. Contentment in graph theory : covering graphs with cliques. In *Indagationes Mathematicae (Proceedings)*, volume 80, pages 406–424. Elsevier, 1977. (Cité en pages 62 et 80.)

[66] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1) :60–100, 1991. (Cité en page 29.)

[67] Sophie N Parragh, K Doerner, and Richard F Hartl. A survey on pickup and delivery models part i : Transportation between customers and depot. Technical report, Working paper, Chair of Production and Operations Management, University of Vienna, 2006. (Cité en page 11.)

[68] Sophie N Parragh, K Doerner, and Richard F Hartl. A survey on pickup and delivery models part ii : Transportation between pickup and delivery locations. Technical report, Working paper, Chair of Production and Operations Management, University of Vienna, 2006. (Cité en page 12.)

[69] René Peeters. The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3) :651–654, 2003. (Cité en page 62.)

[70] Dilson Lucas Pereira, Michel Gendreau, and Alexandre Salles da Cunha. Branch-and-cut and branch-and-cut-and-price algorithms for the adjacent only quadratic minimum spanning tree problem. *Networks*, 2015. (Cité en page 29.)

[71] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010. (Cité en page 16.)

[72] D. Feillet S. Boussier and M. Gendreau. An exact algorithm for team orienteering problems. *4OR*, vol. 5, no. 3, pp.211-230, Sep. 2007. (Cité en pages 16 et 41.)

[73] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998. (Cité en page 79.)

[74] Patrick Siarry. *Métaheuristiques : Recuits simulé, recherche avec tabous, recherche à voisinages variables, méthodes GRASP, algorithmes évolutionnaires, fourmis artificielles, essaims particulaires et autres méthodes d'optimisation.* Editions Eyrolles, 2014. (Cité en pages 32, 34, 35 et 36.)

[75] Jeong-Jun Suh, Shan Guo Quan, Jong-Kuk Park, Jong Hyuk Park, and Young Yong Kim. Multicast partition problem for smart home applications. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, volume 2, pages 284–291. IEEE, 2006. (Cité en page 63.)

[76] H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Comput. Oper. Res*, vol. 32, no.6, pp. 1379-1407, Jun. 2005. (Cité en pages 15, 40, 41, 50 et 53.)

[77] Jacques Tits. Formes quadratiques, groupes orthogonaux et algebres de clifford. *Inventiones mathematicae*, 5(1) :19–41, 1968. (Cité en page 65.)

[78] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001. (Cité en pages 16 et 30.)

[79] Gündüz Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3) :329–337, 1985. (Cité en page 32.)

[80] François Vanderbeck and Laurence A Wolsey. An exact algorithm for ip column generation. *Operations research letters*, 19(4) :151–159, 1996. (Cité en page 29.)

[81] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European journal of operational research*, 196(1) :118–127, 2009. (Cité en page 41.)