République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 - Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de fin d'études Master

Filière : Informatique

Option : Système informatique

Thème :

Réalisation d'une approche d'optimisation pour un problème de transport

Encadré Par :

Dr. BENSEDIRA Badis

Présenté par :

CHEHIR Issam

Juillet 2019

Remerciement

Nous remercions après de tout cœur notre encadreur « Mr Bensedira Badis » pour son soutien, sa sympathie, ces encouragements, la confiance qu'il a nous témoignée en acceptant de diriger ce travail et pour avoir mis à notre disposition ses conseils pour une meilleure maitrise du sujet.

Nous remercions nos familles qui nous ont toujours donnés la possibilité de faire ce que nous voulions durant nos études et qui ont toujours cru en nous.

Enfin, nous remercions tous ceux qui ont contribués à ce travail par leurs remarques, leurs suggestions et leurs soutiens.

Dédicace

A mon père LAYACHI que Dieu lui habite son paradis éternel.

A ma mère NOURA qui représente pour moi la source de tendresse. Ta prière et ta bénédiction m'ont été d'un grand secours pour mener à bien mes études.

A Ma femme, tes sacrifices, ton soutien moral et matériel, ta gentillesse sans égal, ton profond attachement m'ont permis de réussir mes études.

A Mon fils MOHAMED, ma source de bonheur que dieu te protège.

A mon frères ABDELAAZIZ et ma sœur KHAWLA qui vous vous êtes dépensés pour moi sans compter.

A tous mes enseignants, mes amies, mes camarades en particulier ABIR et tous ceux du département d'Informatique

En reconnaissance de tous les sacrifices consentis par tous et chacun pour me permettre d'atteindre cette étape de ma vie. Avec toute ma tendresse.

A tous

ISSAM...

Résumé

L'optimisation combinatoire ou discrète permet de trouver la meilleure solution dans un ensemble de solutions réalisables.

Le problème des tournées sur arcs (Arc Routing Problem) consiste à déterminer les tournées associées aux véhicules qui servent un ensemble d'arcs et respectent certaine contraintes.

L'objectif est de trouver la meilleure combinaison de tournées qui réduit le cout de transport

Plusieurs méthodes ont été proposées pour résoudre ce problème mais ils sont limités de taille moyenne. Les applications d'ARP les plus répandus sont la collecte de la livraison des produits.

Les Metaheuristiques sont considérées comme étant un sauveur pour plusieurs problèmes de grande taille. Elles donnent des solutions de bonne qualité dans un temps raisonnable.

L'objectif de ce sujet est de proposer une approche basée sur les Métaheuristiques pour résoudre le problème de tournées sur arcs avec une étude comparative.

Mots clé : Métaheuristique, problème de tournées, Optimisation

Abstract

Combinatorial or discrete optimization makes it possible to find the best solution in a set of feasible solutions.

The problem of Arc Routing Problem is to determine the routes associated with vehicles that serve a set of arcs and respect certain constraints.

The goal is to find the best combination of tours that reduces the cost of transportation

Several methods have been proposed to solve this problem but they are limited in size. The most common ARP applications are collection of product delivery.

Metaheuristics are considered a savior for many large problems. It gives good quality solutions in a reasonable time.

The objective of this topic is to propose a Metaheuristics approach to solve the problem of arc turns with a comparative study.

Keywords: Metaheuristics, arc routing problem, Optimization.

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

1. Problématique :

Le problème de transport représente aujourd'hui une préoccupation primordiale dans la vie quotidienne. Les recherches sur l'optimisation du cout lié au problème du transport, notamment les recherches sur la construction des circuits de véhicules, vont dans le sens de nos obligations industriels et humains.

Les choix de tournées sont utilisées tant pour le transport des biens, de déchets, voire même des personnes. L'impact des solutions liées aux problèmes d'optimisation des tournées dans le secteur de transport a attiré de plus en plus les chercheurs académiques et les hommes d'affaires.

Beaucoup de recherches ont été mise en place par certaines entreprises afin d'optimiser leurs flux physiques et leurs flux d'informations, et d'optimiser les trois facteurs incontournables à leurs efficacité notamment la qualité de service, les délais et les coûts.

Le problème de tournées sur arc (ARP) est un problème d'optimisation combinatoire et de recherche opérationnelle. Il appartient à la catégorie des problèmes de transport connus à l'image de la livraison ou la collecte de produits auprès de plusieurs clients en optimisant la distance totale traversée, tout en en respectant le fait d'emprunter un court chemin dans un délai réduit.

En effet, le Problème de transport constitue un problème d'optimisation combinatoire NP-Difficile qui est lié La classe NP est une classe très importante de la théorie de la complexité. L'abréviation NP signifie « non déterministe polynomial », étudié depuis les années 50 et pour lequel de nombreuses méthodes de résolution ont été suggérées.

2. Objectif

Les Métaheuristiques sont considérées comme étant une solution pour plusieurs problèmes de grande taille. Elle donne des solutions de bonne qualité dans un temps raisonnable.

L'objectif de ce sujet est :

- Proposer une approche basée sur les Métaheuristiques/Heuristiques pour résoudre le problème de tournées d'arcs.
- Implémenter et expérimenter cette approche en effectuant une étude comparative pour choisir les meilleurs ajustements (paramètres et operateurs).

- Voir l'impact de lissage de l'espace de recherche sur le comportement de cette approche.

3. Organisation du mémoire

Nous commençons en premier lieu, à élaborer une perspective de littérature sur le problème de tournées sur arcs. Cette partie comporte deux chapitres :

- Le chapitre 1 présente les problèmes de tournés d'arcs. Ensuite, nous allons focaliser sur le problème de tournées avec contrainte de capacité, sa formulation mathématique et quelques variantes rencontrées dans la littérature. Ainsi,

- Le chapitre deux collecte les différentes méthodes de résolution du problème de tournées sur arcs.

Cependant la deuxième partie de ce travail présente notre contribution :

- Chapitre trois qui sera dédié pour la résolution du problème CARP avec LNS (Large Neighborhood search) ainsi que le Smoothing qu'on les verra par la suite.

- Chapitre quatre, nous traitons le CARP. L'objectif de ce dernier est d'étudier les tests et les résultats et éventuellement de déterminer des méthodes pratiques pour augmenter cette robustesse par rapport aux variations des données comme les taux de destructions.

La conclusion présente une synthèse des travaux que nous avons réalisés et propose quelques perspectives de recherches futures.

Chapitre I

Le problème de tournées sur arcs (CARP)

1. Introduction

De nombreuses études en recherche opérationnelle portent sur les problèmes de tournées de véhicules dans lesquels il faut desservir un certain nombre de clients dans un réseau urbain ou rural. Ce réseau est en général représenté par un graphe dont les sommets sont les intersections des rues et les arcs sont les portions de rues entre deux intersections. Par ailleurs, l'orientation des rues nous oblige à utiliser des graphes non orientés, orientés ou mixtes.

Les problèmes de tournées de véhicules se divisent en deux grandes catégories : les problèmes de tournées de véhicules (*VRP*) sur les sommets (*CARP*) et les problèmes de tournées de véhicules sur arcs (ou arêtes).

Comme ce travail est consacré aux problèmes de tournées sur arcs, l'état de l'art décrit dans ce chapitre, concerne uniquement ce genre de tournées. Nous allons présenter l'origine des problèmes de tournées d'arcs. Ensuite nous présenterons les différentes variantes de problèmes de tournées d'arcs sans capacité. Nous finissons par la présentation du problème CARP, sa formulation, ses principales variantes et le domaine d'application.

4. Historique de tournées sur arcs

L'étude de problèmes de tournées sur arc a commencé en 1735 par *Leonhard Euler* avec le problème des ponts de *Königsberg* (graphe eulérien).
Les problèmes de tournées sur les arcs origines du problème des ponts de *Konigsberg*, formulé par Euler en 1736. En mémoire de ce dernier, un graphe est dit Eulérien s'il contient un circuit (ou cycle) Eulérien qui passe exactement une fois par chacun des 8 arcs (ou arêtes) du graphe **[*1*]**.

Les conditions nécessaires et suffisantes pour qu'un graphe G connexe soit Eulérien dépendent du type de graphe :

- Si le graphe est non orienté, chaque sommet doit être de degré pair (i.e., le nombre d'arêtes incidentes à chaque sommet doit être pair).
- Si le graphe est orienté, le nombre d'arcs entrants doit être égal au nombre d'arcs sortants à chaque sommet. De plus, le graphe doit être fortement connexe.

- Si le graphe est mixte, chaque sommet doit être incident à un nombre pair d'arcs et d'arêtes.

De plus, pour chaque sous-ensemble 8 de sommets (*8-S*), la différence en valeur absolue entre le nombre d'arcs allant de 8 à V-S et le nombre d'arcs allant de V-S à 8 doit être inférieur ou égal au nombre d'arêtes entre 8 et V-S .

Plusieurs algorithmes pour trouver un cycle ou circuit Eulérien ont été publiés. On trouve par exemple l'algorithme "*end-pairing*" proposé par Edmonds et Johnson pour des graphes non orientés.

5. Le problème de tournées d'arcs sans contrainte de capacité

5.1. Le problème du postier chinois

Ce problème a été formulé par **[2]**. L'objectif est de trouver un cycle ou circuit de coût minimum, passant par toutes les arêtes ou arcs d'un graphe non nécessairement Eulérien.

Les algorithmes pour résoudre le problème du postier chinois consistent à augmenter le graphe en ajoutant des arêtes ou arcs à coût minimum afin d'obtenir un graphe Eulérien. Pour ce faire, il faut résoudre un problème d'augmentation minimale qui a pour objectif de minimiser la somme des longueurs des liens qui sont ajoutés.

5.1.1. Le problème du postier chinois non orienté

Le problème d'augmentation minimale est résolu sur un graphe qui ne contient que les sommets de degré impair du graphe original. Ce graphe contient également une arête entre chaque couple de sommets distincts, dont la longueur correspond à la longueur de la chaîne plus courte entre les deux sommets dans le graphe original. Le problème d'augmentation correspond alors à un problème de couplage de coût minimal.

5.1.2. Le problème du postier chinois orienté

Pour le cas orienté, le problème d'augmentation est défini sur les sommets dont le nombre d'arcs entrants est différent du nombre d'arcs sortants. Nous distinguons les sommets en déficit où le nombre d'arcs entrants est inférieur au nombre d'arcs sortants, et les sommets en surplus où le nombre d'arcs sortants est inférieur au nombre d'arcs entrants. L'objectif est d'ajouter des arcs à coût minimum entre les sommets en surplus et les sommets en déficit de façon à ce qu'ils soient en équilibre. Le problème d'augmentation correspond ici à un

problème de transport où les sommets en surplus sont des sources et sommets en déficit des puits.

5.1.3. Le problème du postier chinois mixte

Pour les graphes mixtes, le problème d'augmentation minimale est *NP-difficile*. Il existe pour le résoudre des méthodes exactes comme la méthode de *Nobert* et *Picard*.

5.2. Le problème du postier rural

Dans les problèmes de postiers chinois des sections précédentes, tous les liens étaient requis. Dans le problème du postier rural (*RPP ou Rural Postman Problem*), seul un sous-ensemble R de liens requis doit être obligatoirement traversé **[*3*]**.

Ce problème plus réaliste possède de nombreuses applications comme la collecte des ordures ménagères ou la livraison de lait.

5.2.1. Le problème du postier Rural non orienté

Le premier modèle linéaire publié est proposé dans **[*4*]**. Il suppose un prétraitement qui convertit tout URPP avec $V_R \neq V$ en une instance avec VR = V. On relie d'abord toute paire de nœuds de VR par une arête de coût égal au coût du plus court chemin dans G. Ensuite, on conserve une seule arête pour toute paire d'arêtes parallèles de même coût et on efface toute arête non requise e = [i, j] telle qu'il existe un noeud k avec $c_{ij} = c_{ik} + c_{kj}$.

5.2.2. Le problème du postier rural orienté

Un exemple de modèle linéaire pour le DRPP, étudié par **[*5*]** puis **[*6*]**. Comme pour l'URPP, la formulation suppose un prétraitement pour avoir $V_R = V$.

5.2.3. Le problème du postier rural mixte

Notons que le MRPP se réduit au UCPP si $A = \emptyset$ et R = E, au DCPP si $E = \emptyset$ et R = A et au MCPP si $R = E \cup A$ avec $A \neq \emptyset \neq E$. De plus, évidemment, le MRPP se réduit au URPP si $A = \emptyset$ et $R \subset E$ et au DRPP si $E = \emptyset$ et $R \subset A$.

Le MRPP contient donc en cas particulier tous les problèmes de tournées sur arcs sans capacités. Il est donc NP-difficile.

6. Le problème de tournées d'arcs avec contrainte de capacité

Le problème CARP consiste à trouver l'ensemble de tournées à coût minimal, qui sert à traiter (livrer ou collecter) chaque arête requise de l'ensemble R par un seul véhicule, de telle manière que chaque véhicule part et arrive au dépôt, chaque arête requise doit être traitée par un seul véhicule et une même arête de l'ensemble E peut être traversée par plusieurs véhicules.

6.1. Présentation du CARP

La définition la plus grossière du CARP est la suivante : il s'agit de la conception de routes optimales par une flotte de véhicules, basée en un dépôt, pour desservir un ensemble de rues (arrêtes ou arcs) dispersés géographiquement et ayant des demandes connues.

Cette définition met en évidence l'ensemble de paramètres qui caractérisent une variante du CARP :

- Le réseau : Le réseau routier qui pourrait être symétrique ou asymétrique ; en conséquence, le graphe associé G = (V, E) sera orienté ou non et les liaisons entre les sommets seront des arcs ou des arêtes. dont V représente les nœuds et E les arrêtes.

- La clientèle : La principale caractéristique de la clientèle est sa demande en marchandise. La livraison de celle-ci peut être contrainte à s'effectuer au cours de périodes de temps spécialisées, appelées fenêtres temporelles (time Windows). Ces contraintes peuvent être dures ou souples. Dans le cas de contraintes dures, une arrivée avant la fenêtre temporelle impose une attente, et les retards sont interdits ; alors que les contraintes souples peuvent être violées et induisent ainsi des pénalités **[*7*]**. Les clients peuvent être livrés en marchandises mais peuvent aussi en remettre aux véhicules. On parle alors de tournées de livraison/ramassage ou de service mixte. Les temps de livraison et de ramassage peuvent être non négligeables et sont ainsi pris en compte dans le calcul des durées de tournées. Finalement, l'accès à un client peut être illimité par rapport à tous les véhicules mais par contre un seul véhicule le desserve.

- La flotte de véhicules : Le nombre de véhicules disponibles peut être fixe ou non. Notons que dans le cas d'un seul véhicule, le problème de tournées reste tout de même différent d'un problème de voyageur de commerce (pvc). Les véhicules peuvent supporter une capacité maximale en termes de marchandises véhiculées (volume, poids… etc.).

➢ La fonction objective : Les buts les plus communs sont soit la minimisation du cout du transport soit la maximisation du bénéfice totale parcourue par les véhicules.

6.2. Formulation mathématique de base

Nous rappelons ici la première formulation pour le CARP, introduite dans **[*8*]**. Elle concerne un graphe non orienté. Les nœuds sont indicés de 1 (le dépôt) à n. Soit A l'ensemble des arcs obtenus en remplaçant chaque arête par deux arcs opposés, x_{ijk} une variable binaire égale à 1 si et seulement si le véhicule k traverse l'arête [i, j] de i vers j et l_{ijk} une variable binaire valant 1 si et seulement si le véhicule k traite [i, j] de i vers j.

$$Minimiser \sum_{k\in I} \sum_{(i,j)\in A} C_{ij} X_{ijk} \quad \text{.....(1)}$$

$$\forall i \in V, \forall k \in |I : \sum_{j\in\Gamma(i)} (X_{ijk} - X_{jik}) = 0 \quad \text{.....(2)}$$

$$\forall (i,j) \in A, \forall k \in I : X_{ijk} \geq l_{jik} \quad \text{.....(3)}$$

$$\forall (i,j) \in R: \sum_{k\in I} (l_{ijk} + l_{jik}) = 1 \quad \text{.....(4)}$$

$$\forall k \in I : \sum_{(i,j)\in A} l_{ijk}\, r_{ij} \leq W \quad \text{.....(5)}$$

$$\forall S \neq \emptyset, S \subset \{2, \dots, n\} \; et \; \forall k \in I : \begin{cases} \sum_{i,j\in S} X_{ijk} - n^2 y_s^k \leq |S| - 1 \\ \sum_{j\in S} \sum_{j\in S} X_{ijk} + u_s^k \geq 1 \\ u_s^k + y_s^k \leq 1 \end{cases} \quad \text{.....(6)}$$

$$u_s^k, y_s^k, x_{ijk}, l_{ijk} \in \{0, 1\}$$

- L'objectif consiste à minimiser la somme des coûts de parcours.
- La contrainte (2) garantit qu'un véhicule qui arrive en un nœud doit en repartir.
- La contrainte (3) garantit qu'un arc traité est aussi traversé.
- La contrainte (4) impose que chaque arête requise soit traitée, par un seul véhicule et dans un seul sens.
- La contrainte (5) assure le respect de la capacité des véhicules. Enfin l'interdiction des sous tours illégaux est imposée par la contrainte (6) qui nécessitent des variables binaires supplémentaires y_s k et u_s.

6.3. Variantes du problème CARP

Le problème CARP possède plusieurs variantes car des applications variées peuvent se modéliser sous forme d'un problème CARP.

6.3.1. Le problème CARP Périodique (PCARP)

Le PCARP est une extension du CARP auquel on ajoute des contraintes de périodicité des visites, c'est-à-dire, les arcs d'un réseau peuvent être visités plus d'une fois dans un horizon de temps. La demande de chaque arc dans le réseau est différente. Quelques arcs ont une demande très grande, d'autres ont une demande très petite **[9].**

La solution est de générer un horizon qui permet de laisser de côté les arcs à faible demande pour une ou plusieurs périodes et se concentrer sur les arcs à forte demande. Cependant, à la fin de l'horizon chaque arc sera servi au moins une fois. Certains seront servis plus d'une fois.

6.3.2. Le problème Multi-Dépôts CARP (MDCARP)

Cette variante est connue en anglais par le problème "*Multi-Depots CARP*" (MDCARP). La différence par rapport à la forme classique de problème CARP est l'existence de plusieurs dépôts dans le réseau. Au départ, les véhicules sont localisés dans les différents dépôts. La tournée de chaque véhicule commence et terminé dans le même dépôt où ce véhicule est localisé. Par contre, dans cette variante, les véhicules localisés dans un même dépôt ne sont pas tous identiques et peuvent avoir des capacités différentes.

6.3.3. Le problème CARP avec des sites intermédiaires (CARPIF)

Le problème CARPIF (*CARP with Intermdiate Facilities*) est une variante de problème CARP classique. Le problème CARP est un cas particulier de problème CARPIF où l'ensemble I est réduit à un seul dépôt, ce qui prouve que CARPIF est un problème NP-difficile.

6.3.4. Le problème Stochastique CARP (SCARP)

Le problème SCARP (*Stochastic CARP*) est très similaire au problème classique CARP, excepté que les demandes des arêtes ne sont plus des quantités fixes mais des variables aléatoires.

Le cas de ce problème SCARP existe dans plusieurs applications réelles comme le cas de collecte des déchets ménagers où la quantité exacte de demandes n'est pas connue à priori.

7. Domaines d'application

Les différentes versions des problèmes CARP modélisent un nombre important d'applications réelles qui peuvent être intéressantes dans les deux domaines publique et privé. Pour cette raison ce problème a été très étudié par les chercheurs surtout dans les dernières années.

Dans ce qui suit, nous citerons quelques applications que nous rencontrons souvent dans la littérature :

- La collecte des ordures : Il s'agit d'un problème de tournées sur les arcs avec contraintes de capacité.
- Le déneigement : Nous incluons ici le déglaçage et le déblaiement des rues et des trottoirs. Contrairement au problème de déglaçage, le déblaiement des rues et des trottoirs se modélise comme un problème de tournées sur les arcs, mais sans contraintes de capacité. Bien que ces problèmes aient été étudiés dans la littérature en tant que problèmes statiques, où toutes les données sur le problème sont supposées connues à l'avance, les aspects dynamiques de ces problèmes sont très importants en pratique.
- Le balayage des rues et la lecture des panneaux d'électricité : ce sont deux problèmes de tournées sur les arcs sans contraintes de capacité

Les problèmes de tournées sur les arcs décrits jusqu'à présent n'ont qu'un seul dépôt bien qu'on note souvent la présence de plusieurs dépôts dans les problèmes réels. Nous nous sommes intéressés dans la section précédente aux problèmes de base en tournées sur arcs avec contrainte de capacité. Beaucoup d'applications concernent les réseaux routiers.

Les demandes sont alors des quantités à collecter (ordures ménagères) ou à apporter (sablage de routes en cas de verglas), et les coûts sont souvent des distances ou des temps de parcours.

8. Conclusion

Dans ce chapitre, nous avons présentés les problèmes de tournées d'arcs (avec et sans capacité), origines.

Nous nous sommes focalisés sur le problème CARP en le présentant, montrer la Formulation mathématique et ses principales variantes et le domaine d'application.

Dans le prochain chapitre, on va décrire les approches de résolution et de l'optimisation du CARP.

Chapitre II

Méthodes de résolution du CARP

1. Introduction

Le problème de tournées sur arcs est un problème d'optimisation combinatoire et de recherche opérationnelle. Il fait partie de la catégorie des problèmes de transport, tout comme le problème du postier chinois. Dans ce problème relevant du domaine de la logistique, un ou plusieurs véhicules doivent couvrir un réseau de transport pour livrer charger et décharger des marchandises à des clients ou couvrir les routes de ce réseau. Divers algorithmes ont été proposés pour résoudre le problème dans un temps polynomial. Dans ce chapitre, nous présenterons une revue de littérature concernant les méthodes de résolution du CARP et de leurs classifications. Nous apporterons une attention aux méthodes approchées et leur classification.

La figure ci-dessous représente une classification des méthodes de résolution du CARP.

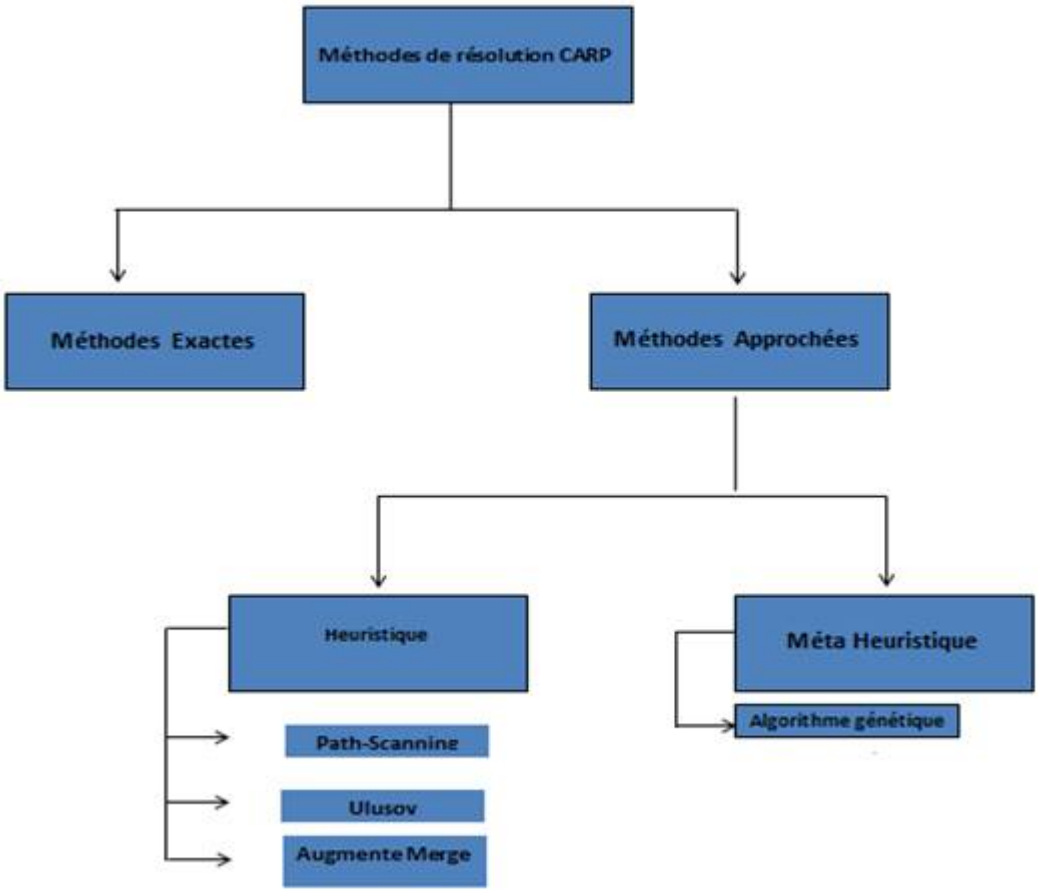

Figure 1 Classification des méthodes de résolution du CARP

2. Méthodes exactes

Les Méthodes de résolution exactes proposées pour les problèmes de genre CARP s'adossent sur un arbre de recensement des solutions, gère efficacement par la progression de bornes. Ces bornes sont souvent obtenues à partir de formulations linéaires en nombres entiers du problème. Ces méthodes, envisagent à déterminer la solution optimale d'un problème d'optimisation en explorant intégralement l'ensemble des solutions possibles.

Un algorithme exact vise à trouver la solution idéale. Or cela sollicite un temps de calcul considérable puisque elle consiste à dénombrer les meilleures solutions possibles tacitement.
Ainsi comme le temps de calcul s'accroit exponentiellement avec la taille du problème, ces méthodes s'avèrent inefficace lorsque la taille du problème s'élève. Donc les méthodes exactes se bornent à la résolution des problèmes de taille restreinte. Le balayage énumératif de toutes les solutions est la technique la plus basique mais elle reste inadéquate aux problèmes combinatoires.

2.1. Méthode Branch-and-Bound

La méthode du branch-and-bound est une méthode générale dont l'idée est de diviser le problème initial en sous-problèmes plus petits structurés sous forme d'arbre, et de couper des branches de cet arbre (afin de ne pas avoir besoin de les évaluer) grâce à un argument de majoration **[*10*]**.

On représente l'exécution de la méthode de branch-and-bound à travers une arborescence. La racine de cette arborescence représente l'ensemble de toutes les solutions du problème considéré. Dans ce qui suit, nous résumons la méthode de branch-and-bound sur des problèmes de minimisation. Pour appliquer la méthode de branch-and-bound, nous devons être en possession :

1. d'un moyen de calcul d'une borne inférieure d'une une solution partielle
2. d'une stratégie de subdiviser l'espace de recherche pour créer des espaces de recherche de plus en plus petits.
3. d'un moyen de calcul d'une borne supérieure pour au moins une solution. La méthode commence par considérer le problème de départ avec son ensemble de solutions, appelé la racine.

Des procédures de bornes inférieures et supérieures sont appliquées à la racine. Si ces deux bornes sont égales, alors une solution optimale est trouvée, et on arrête là. Sinon, l'ensemble des solutions est divisée en deux ou plusieurs sous-problèmes, devenant ainsi des enfants de la racine.

La méthode est ensuite appliquée récursivement à ces sous-problèmes, engendrant ainsi une arborescence. Si une solution optimale est trouvée pour un sous-problème, elle est réalisable, mais pas nécessairement optimale, pour le problème départ. Comme elle est réalisable, elle peut être utilisée pour éliminer toute sa descendance :

Si la borne inférieure d'un nœud dépasse la valeur d'une solution déjà connue, alors on peut affirmer que la solution optimale globale ne peut être contenue dans le sous-ensemble de solution représenté par ce nœud. La recherche continue jusqu'à ce que tous les nœuds sont soit explorés ou éliminés.

2.2. Méthode Branch-and-Cut

Baldacci et Maniezzo ont proposé une méthode exacte pour résoudre le CARP consistant à le transformer en CVRP, puis de résoudre le CVRP résultant **[*11*]**. Avec une légère adaptation de la méthode exacte mise en œuvre par **[*12*].** Rappelons que cette transformation produit un graphe avec nœuds, où chaque bord requis génère deux nœuds, et le bord auquel un cout est attribué. La motivation de ce coût est d'assurer qu'un itinéraire du CVRP la solution traversera le bord.

Cette méthode est un algorithme de type branche-and-cut basé sur une formulation du CVRP qui utilise une variable pour chaque bord et cela variable est égal à un si le bord correspondant est utilisé par la solution. Ensuite, Baldacci et Maniezzo exploite ce fait et évite d'utiliser le coût pour les arêtes en fixant les variables correspondantes à une dans la formulation de CVRP. Outre cette astuce, ils appliquer le découpage implémenté mis en œuvre par **[*12*]**, avec peu changements. Ils utilisent le paquet de routines développé par Lysgaard pour séparer les capacités inégalités, inégalités multistarts homogènes, grandes inégalités généralisées multistarts, encadré les inégalités en matière de capacités, les inégalités renforcées et les inégalités.

La stratégie de branchement adoptée est la stratégie de branchement fort CPLEX et le nœud. La règle de sélection est la meilleure première règle qui sélectionne le nœud dont la limite inférieure est la plus petite traité ensuite. Les auteurs ont testé leur méthode avec les ensembles d'instances vol et egl, et une première lié au coût de la meilleure solution

heuristique a été utilisé pour chaque instance. Vingt- huit des 34 instances de vol ont été résolues de manière optimale en deux heures. Ça signifie six optimums de plus que la méthode du plan de coupe de Belenguer et Benavent **[*13*]**.Néanmoins, les bornes inférieures obtenues au nœud racine sont légèrement pire en moyenne que ceux obtenus par Belenguer et Benavent. Le second ensemble d'instances contient 15 sur 24 instances egl, et la méthode branche and cut a pu prouver l'optimalité de la solution heuristiques correspondantes pour cinq instances. Aucune nouvelle meilleure solution n'a été trouvée pour toute instance.

3. Les heuristiques

Les algorithmes exacts ne permettant pas une résolution des problèmes traités de grande taille, le recours aux méthodes approchées s'impose. Des heuristiques et recherches locales sont d'abord présentées, elles sont à la base des méthodes de résolution plus sophistiquées détaillées. Les heuristiques sont des méthodes de résolution approchées, souvent basées sur le bon sens ou sur des observations empiriques, qui permettent d'obtenir des solutions réalisables de bonne qualité.

La construction d'une telle solution résulte en général de décisions élémentaires consécutives, chaque élément ajouté étant sélectionné de manière gloutonne. Cela signifie que l'attribut choisi parmi les différentes possibilités pour étendre la solution partielle est celui qui optimise un certain critère. La principale contribution concernant les heuristiques gloutonnes est la généralisation de l'algorithme de Clarke et Wright intégrant les aspects multi-dépôts et localisation (Extended Clarke & Wright Algorithm - ECWA).

L'algorithme classique mono-dépôt de Clarke et Wright commence par affecter une tournée à chaque client de façon à le relier à l'unique dépôt **[*15*]**. Dans une version multi-dépôts, les clients ne sont plus tous reliés à un site unique. Ils sont affectés au plus proche dépôt ayant encore suffisamment de capacité résiduelle pour les servir. Le résultat donne une solution initiale triviale ayant une allure de bouquet de marguerites. Ensuite, le but est de fusionner les tournées obtenues afin de réduire le coût de la solution courante.

3.1. Path_Scanning

Path Scanning **[*8*]** construit les tournées une à une en ajoutant successivement les clients à la fin de la tournée. On part d'une solution initiale dont aucun des nouveaux clients n'est servis ; on crée une nouvelle tournée avec le premier client (le premier client peut être par exemple le client le plus proche du dépôt tout en n'étant pas encore servi), puis on lui adjoint de nouveaux clients (le nouveau client à ajouter peut être par exemple le client le plus proche du précédent tout en n'étant pas encore servi).

Lorsqu'on ne peut plus ajouter de client (à cause d'une contrainte comme la capacité du véhicule par exemple), on ferme cette tournée pour en créer une nouvelle. On procède ainsi jusqu'à l'insertion de tous les clients.

Path Scanning originale est décrite de façon littéraire, sans vrai algorithme ni mention de la complexité. On construit les tournées une par une. La tournée courante θ part du dépôt σ et est prolongée à chaque itération vers une arête à traiter compatible avec la capacité résiduelle du véhicule. Pour une tournée parvenue en un nœud i, on donne la priorité aux arêtes non encore traitées partant de i. S'il n'y en a plus, on va au plus proche candidat ayant des arêtes non traitées. Les arêtes possibles [i,j] sont départagées par un critère F. Les auteurs en proposent 5, expliques plus loin. L'algorithme est exécuté 5 fois (une fois par critère) et on sort à la fin la meilleure solution.

Cet algorithme doit être exécuté en instanciant F par les fonctions simples suivantes :

- $F1(v) = d(e(v), \sigma)$, maximisation de la distance au dépôt,
- $F2(v) = d(e(v), \sigma)$, minimisation de la distance au dépôt,
- $F3(v) = r(v)/w(v)$, maximisation du rendement (quantité / coût de traitement),
- $F4(v) = r(v)/w(v)$, minimisation du rendement,
- $F5(v) =$ si $load(\theta) \leq Q/2$ alors $F5(v) := F1(v)$ sinon $F5(v) := F2(v)$ fin_si

Selon la façon de sélection de règles Il y'a autres versions de Path Scanning. Pearn (1989) a proposé de sélectionner la règle à appliquer selon une probabilité. Une autre version consiste à ignorer les règles et sélectionner une arête aléatoirement parmi les arêtes candidates **[*16*]**

3.2. Augment-Merge

Cette heuristique ressemble à la fameuse méthode de la marguerite proposée par **[*15*]** pour le VRP. Il s'agit encore d'un algorithme de Golden et Wong, pour lequel les auteurs donnent une description laborieuse et sans complexité. Voici son principe pour le CARP de base, non

orienté Elle part d'une solution triviale de τ tournées réduites chacune à une tâche. Cette méthode fonctionne en trois phases. L'initialisation crée un trajet pour chaque client. Dans la phase dite Augment, on cherche à éliminer les petites tournées dont l'unique tâche est traversée par une tournée plus grande : si la capacité résiduelle du camion le permet, la grande tournée peut collecter cette tâche en passant, absorbant ainsi la petite tournée.

La phase Merge concatène les grandes tournées restantes selon quatre configurations possibles : début-début, début-fin, fin-fin, fin-début.

Pour de meilleurs résultats, Golden et Wong préconisent de trier les tournées initiales par coûts décroissants puis, pour toute tournée Si dans l'ordre du tri, de tenter d'absorber Si+1, Si+2, ..., Sτ , si Si passe sur l'unique tâche-arête de ces tournées et si load(Si) le permet.

Ces absorptions ne changent pas le coût de Si puisque les coûts de traitement et les coûts de traversée sont égaux.

3.3. Algorithme d'Ulusoy

Ulusoy donne une description littéraire et sans complexité de son algorithme, avec une évaluation numérique réduite à quelques exemples **[*17*]**. Nous proposons donc notre propre analyse de cette heuristique qui, nous le verrons, est très efficace. L'heuristique d'Ulusoy nécessite une séquence θ contenant les τ tâches, qui peut être vue comme un « tour complet » effectué par un véhicule de capacité infinie. Ce tour complet est ensuite découpé optimalement en tournées réalisables pour le CARP.

La figure 2 illustre le principe de La méthode d'Ulusoy. Elle découpe optimalement θ en calculant un plus court chemin dans un graphe auxiliaire value H = (X, U, Z). X comprend τ + 1 nœuds indexés de 0 à τ . Le nœud 0 est un nœud de départ. Tout arc (i,j) de U représente une tournée réalisable traitant les tâches θ(i + 1) à θ(j) incluses, et la valeur z(i, j) sur l'arc est le coût d'une telle tournée. Un plus court chemin entre les nœuds 0 et τ dans H fournit un découpage de θ en tournées réalisables minimisant le coût total. On peut détailler les tournées en remontant le chemin et en remplaçant chaque arc par la sous-séquence correspondante de θ.

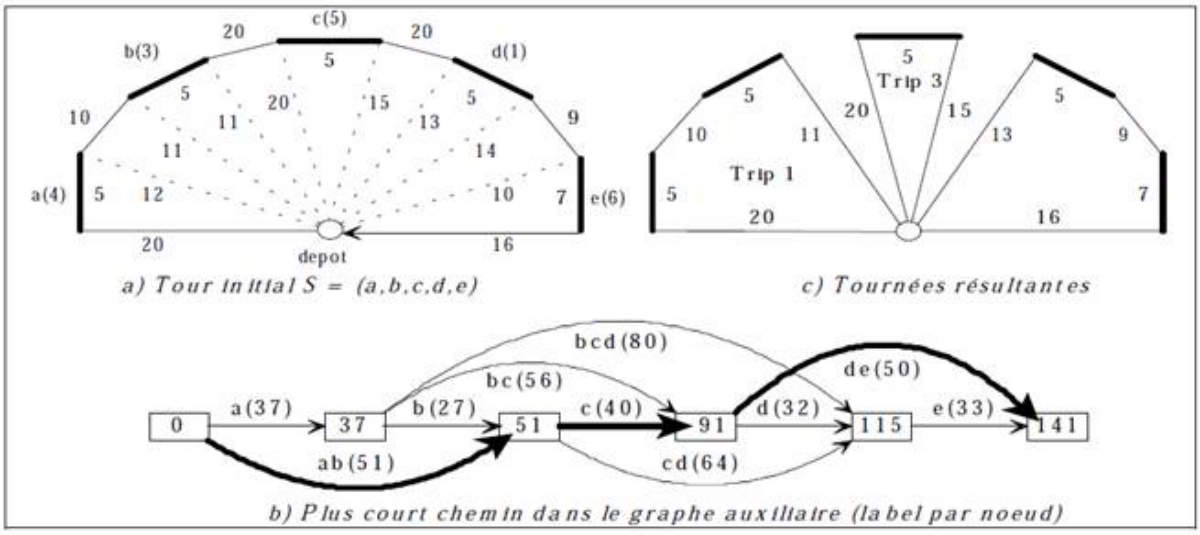

Figure 2 Principe de l'heuristique Ulusoy **[*17*].**

4. Métaheuristiques

Souvent plus performantes que des heuristiques couplées à des recherches locales, les métaheuristiques emploient des méthodes visant à éviter les minimums locaux. Après un bref rappel des méthodes de base, les versions plus performantes, développées pour les problèmes étudiés, sont présentées, à savoir des approches hybrides et/ou des Meta heuristiques.

Les métaheuristiques les plus classiques sont celles fondées sur l'exploration d'un voisinage, et forment la première famille. L'algorithme part d'une solution et la fait évolué à chaque itération sur l'espace de recherche, Une autre approche possible est basée sur l'utilisation d'une population de solutions et constitue la seconde famille. A chaque itération, la métaheuristique fait évoluer un ensemble de solutions en parallèle.

Une autre approche possible est basée sur l'utilisation d'une population de solutions et constitue la seconde famille. A chaque itération, la métaheuristique fait évoluer un ensemble de solutions en parallèle. Les algorithmes génétiques (Genetic Algorithms, GA) **[*18*]**, de recherche dispersée **[*19*]** (Scatter Search, SS) ou les algorithmes de colonies de fourmis (Ant Colony optimisation, ACO) **[*20*]** sont des exemples courants de ce type de méthodes. Dans le même esprit, on retrouve aussi l'algorithme des chemins reliants **[*19*]** (Path Relinking) qui est rarement utilisé seul, mais plutôt en complément des autres méthodes.

Une dernière catégorie, à mi-chemin entre les explorations de voisinage et les évolutions de population, contient les recherches locales évolutionnaires (Evolutionary Local Search, ELS) qui proposent un bon compromis. Basées sur le principe de la recherche locale itérative,

chaque itération ne fait évoluer qu'une seule solution dans l'espace de recherche par le biais de mutations suivies de recherches locales.

Les métaheuristiques classiques, bien que largement plus efficaces que les heuristiques constructives, sont souvent peu compétitives devant les dernières générations de méthodes hybrides. Dans les travaux développés, des versions hybridant divers composants sont proposées. La première hybridation est celle combinant une métaheuristique avec une recherche locale. Dans cette catégorie, l'exemple maître est certainement l'algorithme mémétique qui ajoute une recherche locale sur les solutions-enfants générées par un algorithme génétique. Cependant, ce n'est souvent pas suffisant.

Afin de mieux explorer l'espace de recherche, il est également possible d'avoir recours aux chemins reliant, soit en post-optimisation d'une méthode ou en interne, ou les deux. Cette hybridation est souvent faite avec les algorithmes de type GRASP (greedy randomized adaptive search procedure)car dans cette dernière approche, chaque itération de la méthode génère une solution indépendante des précédentes et le PR permet d'ajouter un lien entre les optimums locaux et ainsi mieux explorer l'espace des solutions.

5. Conclusion Générale

Dans ce chapitre, nous avons effectué une revue de littérature concernant les méthodes de résolution du problème CARP. On a donné une perspective générale sur les méthodes exactes, méthodes heuristique ainsi que les métaheuristiques. Le nombre de publications à ce jour est assez limité, on peut explorer d'avantage ce domaine en développant des nouvelles contributions ou en étendant et en améliorant les approches existantes. Dans le chapitre suivant, nous présenterons une nouvelle approche appliquée sur le problème CARP.

Chapitre III

Application d'une méthode de résolution pour CARP

1. Introduction

Dans ce chapitre nous allons nous focaliser sur un problème classique et connu dans la recherche opérationnelle qui est le problème de transport

Nous avons proposé une nouvelle contribution appelé Smoothing_LNS, dans cette contribution, la recherche est effectuée sur un espace lissé. L'objectif essentiel est de voir l'impact de cette dernière sur le problème CARP

Ainsi, nous présenterons les différentes étapes réalisées durant la modélisation de la contribution.

2. Algorithme de lissage

L'algorithme figure 3 décrit l'algorithme de base de lissage. Ce dernier est composé de trois étapes :

```
Procédure_smoothing ()
Début;
D : //Ensemble de données ;
/• Initialisation •/
Solution : = nouvelle-solution ();
α : = α0;// facteur de lissage
Tant que (α >1) faire
    /• Lissage de l'espace de recherche •/
    D : =Recalculer (D, α);
    /• Recherche •/
    Solution: = Recherche-locale (D, Solution);
    α: = α-1;
    Fin;
Fin;
```

Figure 3 Algorithme de lissage**[21]**

- **initialisation** : permet d'offrir la solution initiale à optimiser. Selon le problème à résoudre, on peut utiliser une solution générée aléatoirement ou bien utilisé une heuristique.
- **Recalculer les données** : dans cette étape, en recalcule les données qui influences sur la qualité de la solution
- **Amélioration :** après chaque calcul de donnée une solution en utilise une méthode d'amélioration (méthode de recherche locale, Méta heuristique).

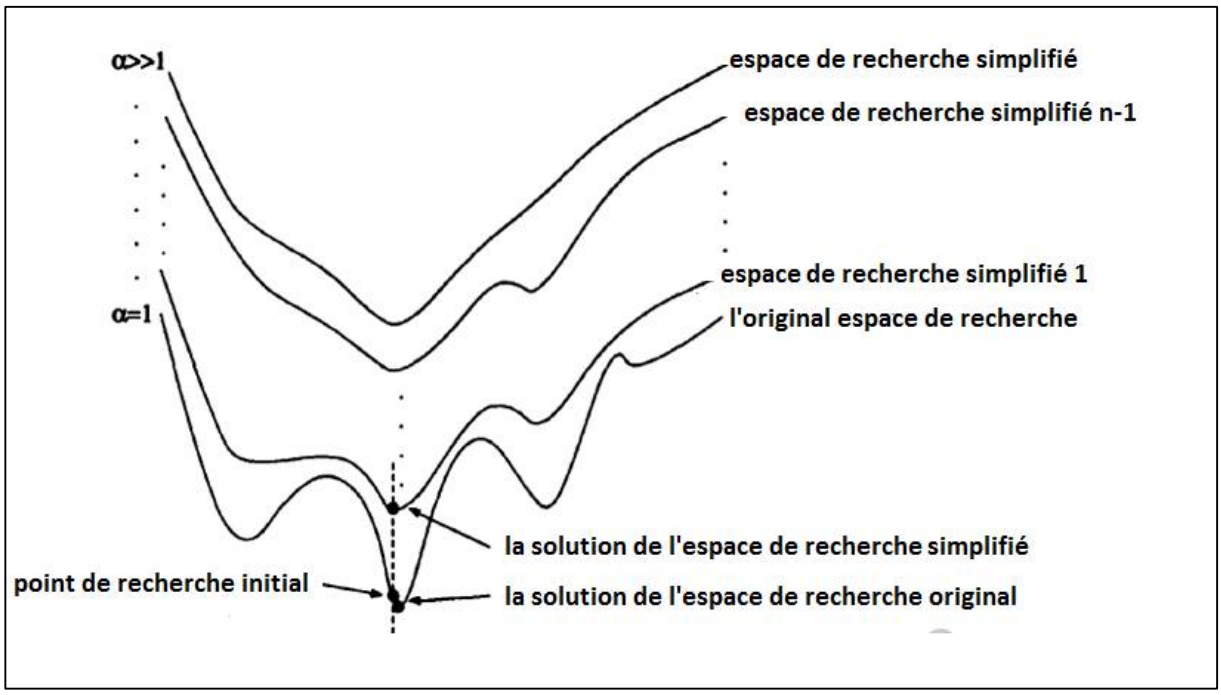

Figure 4 Lissage d'un espace de recherche **[*21*]**.

La figure 4 illustre le lissage d'un espace de recherche **[*21*]**. Cette méthode offre plusieurs avantages de l'utilisation. Premièrement, nous notons que le lissage de l'espace de recherche rend la recherche du point de solution initiale plus facile. Après une opération de lissage, le nombre de minimums locaux dans un espace de recherche est réduit. Ainsi, pour un processus de recherche, la probabilité d'être tombé sur un point local initial est minimisé et la chance de trouver un point optimal global est amplifiée. Dans le cas idéal, après une opération de lissage, il ne reste que le point initial, qui est l'optimum global dans l'espace de recherche. Deuxièmement, puisqu'un espace de recherche a qualitativement accumulé les informations de structure topologique de l'espace original de recherche, un processus de lissage facilite la recherche du point initial global dans l'espace d'origine.

Le rapport global du point initial dans l'espace de recherche peut être défini très proche du point initial global dans la recherche initiale.

Une opération de lissage à différents niveaux de puissance, résultante dans un espace de recherche avec un degré variable de lissage. On utilise le facteur α afin de caractériser le degré de lissage de l'espace initialement lissé si α=1 l'opération n'est pas applicable.si α >1 alors l'opération est applicable et l'espace résultant et plus plat, si α>>1 alors l'espace est presque plat. L'application de cette approche est souvent confrontée à une situation contradictoire, si on applique une opération de lissage plus faible, la topologie de la structure de l'espace de recherche est similaire à l'original. Les informations de guidage heuristique de l'espace de recherche d'origine est donc faible.

3. Recherhce locale avec un espace lissé pour CARP

L'algorithme figure 5 décrit l'algorithme de lissage proposé pour résoudre le problème CARP .

```
Smoothing_LNS ()
Début;
D // Matrice de distance entre les arrêtes //;
// Initialisation
3.1 Solution_Carp: = Path-scanning (D);
α : = α0; // Facteur de lissage
Tant que (α >1) faire
    Début
    // Améliorer la  solution CARP //
    Pour  i := 1 jusqu'à n faire
    Pour j:= 1 jusqu'à n faire
     // Recalculer la distance entre l'arc i et j //
    D [i,j] := Recalculer (D, α);
    // Recherche
    Solution: = Recherche voisinage large (D,Solution_Carp);
    α: = α-1;
    Fin;
Fin;
```

Figure 5 Algorithme de lissage pour CARP

3.1. Representation de la solution (initialisation)

On a utilisé l'heuristique *Path_Scanning* qui donne un meilleur résultat pour les problèmes CARP, la figure ci-dessous (Figure 6) représente une solution des tournées sur arcs, un arc peut être servi dans un sens positif ou négatif.

Le nombre d'arrêtes M=5 notre solution de départ est donc se compose de 3 tournées (5), (3,9) et (2,6) a noter que les arcs 6,9 sur la figure sont l'inverse des arcs 1 et 4 respectivement.

Donc il est trivial de proposer une formule pour localiser le numéro d'un arc inverse quelconque comme suite :

$A' = A + M$ avec A' le numéro d'arc inverse et A le numéro d'arc normal ainsi par exemple l'arc 1 son arc inverse est 6 qui vaut 1 +5.

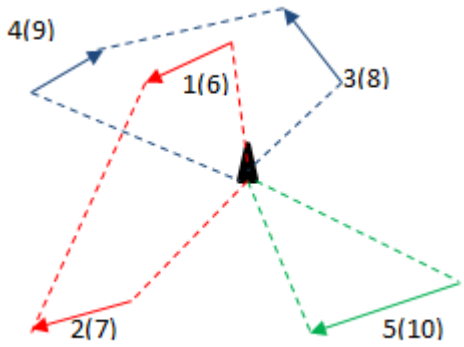

Figure 6 une solution CARP

L'objectif de l'optimisation dans le problème CARP est de minimiser les couts de transport. La formule (1) représente la fonction objective.

$$f\_obj(S) = Min \sum_{n=1}^{T} \left(\sum_{i=1}^{m_n} (D(i, i+1)) \right) \quad \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

Pour chaque tournée n qui contient mn arcs, on calcule le cout en faisant la somme des distances entre l'arc courant i et l'arc suivant i+1

La matrice de distance est calculée en deux étapes. La première étape consiste à calculer les distances entres les nœuds D_1. Cette matrice est utilisée dans la deuxième étape pour remplir la matrice de distances entres arcs D_2. La distance entre l'arc *i* et *j* est égale :

$$D_2(i,j) = D_1(End(i),Begin(j)) \text{ ……(8)}$$

End(i) est le nœud final de l'arc i et Begin(j) est le nœud initial de l'arc j

	Dépôt	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Dépôt	0	5	25	3	20	5	15	10	20	15	10
1	30	20	10	15	35	20	10	25	30	15	5
2	5	10	25	40	45	35	25	30	20	5	15
3	5	10	25	20	15	15	30	10	5	20	25
4	5	15	15	25	10	20	20	40	45	5	15
5	25	30	30	30	15	15	25	5	5	10	5
6	30	30	10	25	30	30	5	10	20	20	30
7	20	25	5	5	20	5	30	15	25	15	25
8	5	20	15	10	25	10	5	30	15	25	20
9	10	10	15	5	10	10	15	5	30	25	15
10	15	15	30	15	15	5	5	10	40	30	10

Tableau 1 Matrice de distance

3.2. Recalculer la matrice de distance

Pour calculer la matrice de distance on fait appel à la formule suivante :

$$\bar{d} = \frac{1}{m(m-2)} \sum_{\substack{i \neq j \\ i \neq j-n \\ i \neq j+n}} d_{ij} \text{……(9)}$$

Avec $\bar{d}$ est la moyenne de distance

m=2*n+1

m est le nombre d'arcs (le sens compte)

n est le nombre d'arrêtes et 1 est le dépôt.

De l'autre coté

$$d_{ij}(\alpha) = \begin{cases} \bar{d} + (d_{ij} - \bar{d})^{\alpha} \; if \; d_{ij} \geq \bar{d} \\ \bar{d} + (\bar{d} - d_{ij})^{\alpha} \; if \; d_{ij} < \bar{d} \end{cases} \text{……(10)}$$

$d_{i,j}$ est la distance entre les arcs i,j.

À signaler seulement que la variable entière α représente un facteur de lissage

Le tableau ci-dessous représente la matrice des distances recalculée de matrice (Tableau 1):

	Dépôt	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Dépôt	-293.69	-142.24	71.75	-293.69	23.19	-142.24	10.65	-40.80	23.19	10.65	-40.80
1	170.30	23.19	-40.80	10.65	318.85	23.19	-40.80	71.75	170.30	10.65	-142.24

2	-142.24	-40.80	71.75	517.41	765.96	318.85	71.75	170.30	23.19	-142.24	10.65
3	-142.24	-40.80	71.75	23.19	10.65	10.65	170.30	-40.80	-142.24	23.19	71.75
4	-142.24	10.65	10.65	71.75	-40.80	23.19	23.19	517.41	765.96	-142.24	10.65
5	71.75	170.30	170.30	170.30	10.65	10.65	71.75	-142.24	-142.24	-40.80	-142.24
6	170.30	170.30	-40.80	71.75	170.30	170.30	-142.24	-40.80	23.19	23.19	170.30
7	23.19	71.75	-142.24	-142.24	23.19	-142.24	170.30	10.65	71.75	10.65	71.75
8	-142.24	23.19	10.65	-40.80	71.75	-40.80	-142.24	170.30	10.65	71.75	23.19
9	-40.80	-40.80	10.65	-142.24	-40.80	-40.80	10.65	-142.24	170.30	71.75	10.65
10	10.65	10.65	170.30	10.65	10.65	-142.24	-142.24	-40.80	517.41	170.30	-40.80

Tableau 2 lissage de l'espace de recherche (α =2)

3.3. Recherche de voisinage large (Large Neighborhood Research)

Nous avons utilisé une recherche locale LNS qui se base sur trois éléments principaux et il est défini par une fonction de destruction «**destroy () »** et de réparation « **repair () »**. Une fonction de destruction détruit une partie de la solution courante cependant une fonction de réparation régénère la solution détruite auparavant. Le voisinage N(x) d'une solution X est alors caractérisé par l'ensemble de solutions qui peuvent être satisfaites en appliquant en premier lieu la fonction de destruction et puis la fonction de réparation.

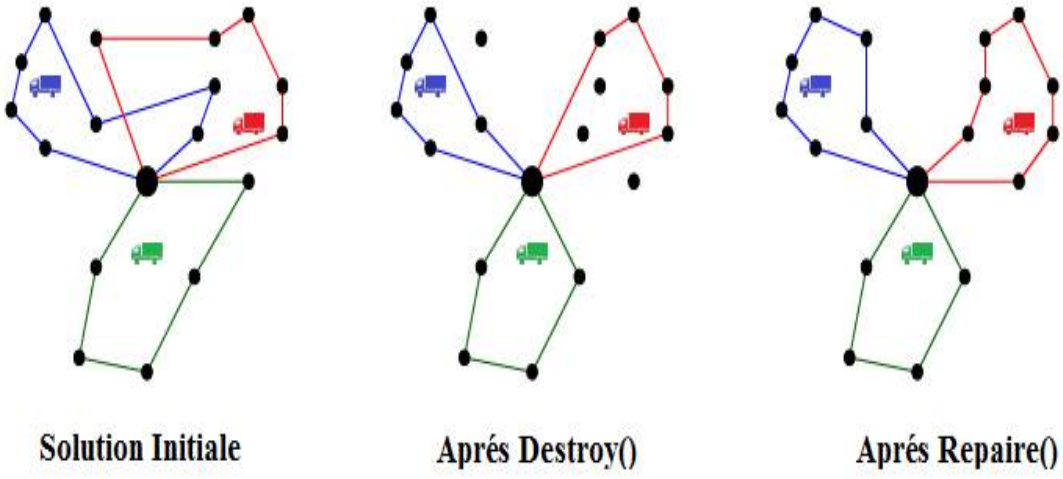

Figure 7 Destruction et réparation **[*22*]**

4. Recherche locale avec voisinage large

La figure 8 illustre un pseudo code de l'algorithme de recherche local (LNS) **[*22*]**.La solution initial x est détruit en générant une solution x' cette solution est réparé x'', à chaque itération on sauvegarde la meilleur solution x_b. On accepte le voisinage x'' comme un point de départ x' si le critère d'acceptation est concrétisé

```
Algorithme LNS

1 : Introduire une solution initiale x

2: xb ←x,

3: répéter

4: x'← détruire (xb);

5: x''←réparer (x')

5: si accepter(x'', x) alors

6: x← x'';

7: fin si

8: si c (xb) > c(x'') alors

9: xb ← x'' ;

10: fin si

11: jusqu'à ce que condition d'arrêt soit remplie

 Retourner xb.
```

Figure 8 Recherche voisinage large **[22]**

4.1. Operateur de Destruction

On prend la solution S et on choisit aléatoirement le degré de destruction Q puis à chaque itération On choisit un arc (a_i) à détruire de la solution. Un arc détruit est ajouté à la liste R, qui contient les arcs à insérer dans l'étape suivante (Voir figure 9).

La suppression aléatoire est l'heuristique de destruction la plus simple. Il supprime à plusieurs reprises les arrêtes de demandes (a_i) au hasard jusqu'à ce que le degré de destruction Q actuel soit atteint. C'est une heuristique très importante pour l'algorithme LNS car il peut supprimer toute arrête de la solution indépendamment des coûts. Par cela, le retrait aléatoire crée une diversification très importante lors de l'exploration d'un grand espace de recherche.

```
Fonction Destroy ()

Début;

Introduire une solution initiale S.

R : liste des arcs à détruire
Q : Degré de destruction (nombre d'arc à détruire).

    Pour  i := 1jusqu'à Q faire
    détruire un arc aléatoire a_i de S

    R :=R∪{a_i}

    Fin;

Fin;
```

Figure 9 Operateur de Destruction

Dans cet exemple on détruit les arcs (6,9). La solution détruit est :((5)(3)(2))

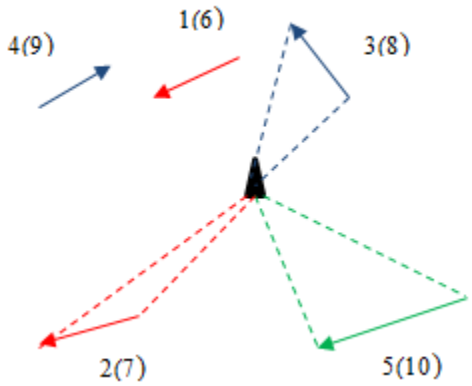

Figure 10 Destruction de la solution

4.2. L'operateur de réparation

Dans la réparation, on prend en considération les deux sens de l'arête à insérer. On adopte l'opérateur « *Basic Greedy Repair* ». Ce dernier détermine la position d'insertion la moins chère pour toute demande non traitée. Les tours de tous les véhicules sont pris en compte pour le calcul. Le classement dans la tournée est donc déterminé par l'insertion d'une arête d'une

demande r dans la tournée d'un véhicule v. Plus les coûts d'insertion ne sont bas, le plus élevé est la paire (r, v) classée.

```
Procédure Repair (R, S)

Début;
S : la solution déduit

R : liste des arcs à insérer

    Tant que |R| >0 faire

    Insérer l'arc a_i dans la meilleur position de S

    R :=R-{a_i}

    Fin Tant que

Fin
```

Figure 11 Operateur de Réparation

À chaque itération on choisit le meilleur emplacement pour un arc (ai) de la liste R pour le réinsérer dans la solution S, l'algorithme s'arrête dès que la liste R devient vide.

Dans l'exemple ci-dessous, La solution réparée est :((5) (3,1) (2,4)). On insère le sens positif d'arêtes 6 et 9 qui sont respectivement 1 et 4. Le meilleur emplacement de l'arête. L'arc 1 est insérer dans la deuxième tournée et l'arc 4 dans la troisième tournée.

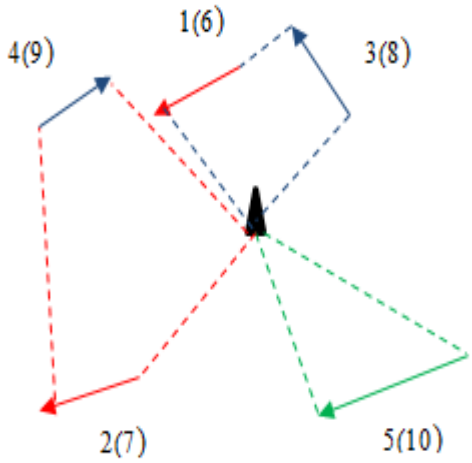

Figure 12 Réparation de la solution

5. Conclusion

Dans ce chapitre, nous avons essayé d'exposer l'ensemble des idées caractérisant l'algorithme proposé en se concentrant sur la conception et de l'application. Nous allons présenter l'étude empirique et comparative obtenue en appliquant l'algorithme sur le problème de tournées sur arc.

Chapitre VI

Implémentation et expérimentation

1. Introduction

L'objectif de ce chapitre est d'évaluer les performances de notre contribution pour la résolution du problème de tournées sur arc avec contrainte de capacité.

Afin de concrétiser notre objectif, nous avons implémenté la méthode en utilisant le langage orienté objet Java. Ensuite, nous l'avons exécuté sur des Benchmarks en vue d'une étude comparative.

2. Implémentation

2.1. Matériel utilisé

Nous avons utilisé un ordinateur DELL Le programme a été implémenté et exécuté sur un ordinateur portable Intel® Core(TM) i3-3217U CPU @ 2.30GHZ dotée de 4 Go de RAM Windows 8.1 édition intégral.

2.2. Langage utilisé

Java est un langage de programmation orienté objet et un environnement d'exécution, développé par Sun Microsystems. Il fut présenté officiellement en 1995. Le Java était à la base un langage pour Internet, pour pouvoir rendre plus dynamiques les pages (tout comme le JavaScript aujourd'hui). Mais le Java a beaucoup évolué et est devenu un langage de programmation très puissant permettant de presque tout faire.

2.3. Outils utilisé : Eclipse oxygène

Initié par IBM puis ouvert à la communauté Open Source, l'environnement de développement intégré (IDE) Eclipse pour développeurs Java présente l'énorme avantage de disposer d'un nombre impressionnant de plugins l'enrichissant dans tous les domaines de la programmation. Dorénavant, avec la version Photon d'Eclipse, un Marketplace Client propose de faciliter la vie des développeurs dans leurs recherches de plugins. Une interface dédiée rend leur intégration bien plus rapide. Fonctionnant en Java et avec la bibliothèque graphique SWT d'IBM, cet IDE "tourne" indépendamment du système d'exploitation, pourvu qu'une JVM soit installée. Des projets tels que Lotus Notes ou Symphony ont été développé grâce à cet atelier. L'IDE open source de référence supporte les projets EGit qui retrace tout l'historique des modifications effectuées sur le code. Et si Java fut l'un des premiers langages à avoir été

utilisé, il a depuis été enrichi par le C/C++, l'Ada, le Python, le Perl, le Ruby, le Cobol, le Pascal, le PHP, l'XML et HTML, l'Action Script, ColdFusion, etc.

3. Expérimentation

Les jeux de test de notre contribution sont effectués sur deux catégories des instances :

- **Kshs** : contient 6 instances utilisées dans la 1ére fois dans **[*22*]**. Elles contiennent 6-10 nœuds et 15 arêtes
- **Gdb** : contient 23 instances générées aléatoirement **[*23*]**. Avec 7-27 nœuds et 11-55 arêtes

Les tableaux 3 et 4 décrivent les caractéristiques des instances kshs et Gdb respectivement.

	Nombre de nœud	Nombre d'arêtes requises	Nombre d'arêtes non requises	Capacité de véhicule
Kshs1	8	15	0	150
Kshs2	10	15	0	150
Kshs3	6	15	0	150
Kshs4	8	15	0	150
Kshs5	8	15	0	150
Kshs6	9	15	0	150

Tableau 3 Caractéristiques des instances Kshs

	Nombre de nœuds	Nombre d'arêtes requises	Nombre d'arêtes non requises	Capacité de véhicule
Gdb1	12	22	0	5
Gdb2	12	26	0	5
Gdb3	12	22	0	5
Gdb4	11	19	0	5
Gdb5	13	26	0	5
Gdb6	12	22	0	5
Gdb7	12	22	0	5
Gdb8	27	46	0	27
Gdb9	27	51	0	27
Gdb10	12	25	0	10
Gdb11	22	45	0	50
Gdb12	13	23	0	35
Gdb13	10	28	0	41
Gdb14	7	21	0	21
Gdb15	7	21	0	37
Gdb16	8	28	0	24
Gdb17	8	28	0	41
Gdb18	9	36	0	37
Gdb19	8	11	0	27
Gdb20	11	22	0	27
Gdb21	11	33	0	27
Gdb22	11	44	0	27
Gdb23	11	55	0	27

Tableau 4 Caractéristiques des instances Gdb

3.1. Paramètres d'expérimentation

La bonne combinaison des paramètres de n'importe quelle contribution sert à atteindre les meilleurs résultats. Dans notre cas on a le nombre d'itérations et le taux de destruction. Le premier paramètre est fixé à 3000 itérations.

Pour voir l'influence de taux de destruction, on a exécuté chaque instance 30 fois. Le taux de destruction varie entre 10% et 35%.

Les tableaux 5 et 6 montrent une partie d'expérimentation, En analysant les résultats obtenus (afficher en gras), on remarque les meilleurs couts sont obtenus lorsque Q=30%

Q	10%	15%	20%	25%	30%	35%
kshs 1	14751,7	14761,5	14757,6	14769,8	**14739,6**	14752,1
kshs 2	9938	9898	9886	9899	**9883**	9876
kshs 3	9322	9321	9321	**9320**	**9320**	9324
kshs 4	12946,8	12511,8	12226,4	**12215,8**	12327	12297,9
kshs 5	12175	11719,4	11197,4	11466,5	11354,8	**11236,2**
kshs 6	10623,1	10540	10397,5	10321,5	**10234,3**	10534,5

Tableau 5 influence de taux de destruction Q sur kshs

Q	10%	15%	20%	25%	30%	35%
gdb 1	336,3	329,7	322,1	318,2	**317,3**	319,3
gdb 2	354,2	351,5	351,6	351,3	**348,6**	350,5
gdb 3	295,4	285,7	286,7	282,4	**281,53**	279,5
gdb 4	325,4	310,6	**290,2**	297,7	290,9	292,9
gdb 5	397,1	**388,2**	398,6	392,6	394,7	394,3
gdb 6	**307,9**	313,2	308,7	312,3	313,6	309,9
gdb 7	338	332	330,8	327,4	**325.1**	326,7
gdb 8	397	399,5	397,6	404,2	**396,3**	402,7
gdb 9	340,4	341,2	341,2	340,3	**340,1**	341,5
gdb 10	284,1	284	280,7	281,1	**280,3**	281,3
gdb 11	416,1	418,3	415,9	417,7	**415**	419,4
gdb 12	522,8	512,6	501,1	510,5	501,3	**498,5**
gdb 13	549,2	549,7	549,8	550,3	**549,1**	550,1
gdb 14	105	103	102,4	102,1	102,9	**101**
gdb 15	**59,2**	**59,2**	**59,2**	**59,2**	59,4	59,5
gdb 16	129,1	128,7	129	129	**128,4**	129,5
gdb 17	91	91	91	91	**91,2**	91,1
gdb 18	168,1	167,9	**167,8**	168,3	168,5	168,6
gdb 19	57	57	56	55,9	55,5	**55,4**
gdb 20	131,5	**125,2**	126,6	126,3	126,5	125,33
gdb 21	162,4	161,3	162,5	162,8	**160,7**	163.4
gdb 22	205	205,1	205	204,9	**204,1**	205.4
gdb 23	245	245,4	**244,6**	246,5	246,5	246.7

Tableau 6 influence de taux de destruction Q sur gdb

3.2. Etude comparative

Après le réglage des paramètres, on commence les différents jeux de test.

- **Teste 1** : l'influence des règles de Path Scanning :

Après l'application des règles sur nos benchmarks et la méthode LNS avec 30% de destruction et 3000 itérations consécutive, on a déduit que les bon résultats sont issus en utilisant «règle 1».

	régle1	régle2	régle3	régle4	régle5
kshs 1	**14750,5**	14871,3	14804,1	14914,3	14756,1
kshs 2	9884	**9863**	9892,5	9879,6	9877
kshs 3	**9320**	9457	9524,9	9451,9	9387
kshs 4	**12185,1**	12612	12332,3	12222,6	12300,6
kshs 5	11426,2	11564,5	11268,2	11380	**11111,3**
kshs 6	**10120,6**	10197	10520,9	10464,6	10300,6

Tableau 7 influence des règles sur Kshs

	régle1	régle2	régle3	régle4	régle5
gdb1	322,4	**317,4**	317,4	322,3	320,7
gdb2	352,2	354,3	350,4	349,2	**349**
gdb3	280,8	282,9	280,6	281,8	**278,4**
gdb4	294	291,4	292,5	297,1	**287,4**
gdb5	397,2	395,5	394,9	392,8	**392,5**
gdb6	**280,8**	310,16	311,1	313,8	311,4
gdb7	327,4	328,7	330,5	329,9	**326**
gdb8	**401,4**	406,1	402,8	403,8	402,1
gdb9	**340**	358,6	355	358,6	346,1
gdb10	**281,4**	284,6	281,9	293,5	281,9
gdb11	**415,7**	418,7	416	418,5	419,6
gdb12	509,7	500,9	**490,2**	493,2	499,1
gdb13	**549,4**	552,3	550	551,9	549,6
gdb14	**101,6**	103,2	102,2	103,6	101,9
gdb15	59,4	59,8	59,6	**59,3**	59,4
gdb16	**128,8**	131,6	131,5	131,2	130,6
gdb17	**91**	92,5	91,4	91,5	91,2
gdb18	**168,4**	170,2	168,5	170	169,1
gdb19	**55,6**	58,4	58,3	57,6	55,9
gdb20	126,7	124,5	123,9	124,3	**123,7**
gdb21	**162,5**	163,8	164,6	163,5	**162,5**
gdb22	**205**	207,6	207,2	207,3	207
gdb23	**246,8**	248,2	248,2	247,5	248,2

Tableau 8 influence des règles sur Gdb

Les deux tableaux 7 et 8 représentent les moyennes des couts de chaque instance exécutée avec un taux de destruction 30%.Les résultats affichés en gras représente les meilleurs couts obtenues pour chaque instance (chaque ligne), on remarque que la règle 1 en général nous offres le best.

Teste 2 : comparaison les deux versions de LNS

L'objectif de ce test est de voir l'influence de l'opérateur de réparation sur les résultats obtenus. On a implémenté deux méthodes de l'opérateur de réparation comme suit :

- LNS1 : utilise un opérateur de réparation qui consiste à insérer l'arc dans le meilleur emplacement de la 1ére tournée qui respecte la contrainte de capacité
- LNS2 : utilise un opérateur de réparation qui consiste à insérer l'arc dans le meilleur emplacement de tournées qui respectent la contrainte de capacité

	lns1	lns2
kshs 1	14572,4	**14263,9**
kshs 2	**9891,9**	10728,3
kshs 3	**9320**	9622,8
kshs 4	**11699,6**	14028,4
kshs 5	**11322,4**	12155
kshs 6	**10423,3**	12012,9

Tableau 9 influence de réparation sur kshs

	lns1	lns2
gdb 1	**320**	348,3
gdb 2	**348,1**	366,2
gdb 3	**349,2**	316,9
gdb 4	**276,7**	339,7
gdb 5	**295,2**	449,3
gdb 6	**386,7**	329,7
gdb 7	**303,3**	357,1
gdb 8	**326,6**	465,5
gdb 9	**340,3**	355
gdb 10	**282,8**	295
gdb 11	**419**	446,8
gdb 12	**501,2**	654,3
gdb 13	**549,4**	562,6
gdb 14	**101,5**	113,7
gdb 15	**59,5**	61,9
gdb 16	**128,8**	135
gdb 17	**91,06**	94,8
gdb 18	**168,5**	176,1
gdb 19	**56,1**	56,8

gdb 20	**125,6**	144,6
gdb 21	**162,9**	177,6
gdb 22	**205,1**	207,9
gdb 23	**246,3**	251

Tableau 10 influence de réparation sur gdb

Les deux tableaux 9 et 10 représentent les moyennes des couts de chaque instance exécutée avec un taux de destruction de 30%. Les résultats afficher en gras représente les meilleurs couts obtenue pour chaque instance (chaque ligne).

Dans ce test nous sommes arrivés à utiliser l'opérateur de réparation avec un taux de destruction de 30% et nombre d'itération qui vaut 3000, on a constaté que l'Algorithme LNS1 donne des bons résultats que LNS2. Cela justifie que LNS2 tombe dans l'optimum local

- **Teste 3** : comparaison LNS et LNS avec un espace lissé (LNS_Smoothing)
  L'objectif de ce test est de voir l'influence de la topologie de l'espace de recherche.

	Lns	Lns_smoothing	Meilleur cout
kshs 1	14762,8	**14668,4**	14661
kshs 2	9874	**9869**	9863
kshs 3	9322	**9322**	9321
kshs 4	12349,9	**11800**	11489
kshs 5	11106,1	**11241,2**	10957
kshs 6	10349,1	**10362,6**	10197

Tableau 11 comparaison entre LNS et LNS_Smoothing sur kshs

	lns	LNS_Smoothing	Meilleur cout
gdb1	319,1	**318**	316
gdb2	351	**348,7**	339
gdb3	282,4	**276,6**	275
gdb4	293,1	**293**	287
gdb5	396,1	**388,2**	377
gdb6	311,6	**302,2**	298
gdb7	325,1	**324,6**	325
gdb8	405,3	**381,8**	348
gdb9	340,9	**325,8**	303
gdb10	282	**276**	275
gdb11	419,4	**411**	395
gdb12	493,2	**477,8**	458
gdb13	550	**549**	536
gdb14	101,6	**101,2**	100
gdb15	59,3	**58,2**	58

gdb16	129,5	**128**	127
gdb17	91	**91**	91
gdb18	168,4	**164,8**	164
gdb19	55,4	**55.4**	55
gdb20	126	**124,5**	121
gdb21	163,3	**158,3**	156
gdb22	205,1	**202,9**	200
gdb23	246,7	**238,6**	233

Tableau 12 comparaison entre LNS et LNS_Smoothing sur gdb

Dans ce test nous sommes arrivés à des résultats plus performants toujours avec un taux de destruction de 30% et nombre d'itérations qui vaut 3000, Les deux tableaux ci-dessus représentent les moyennes de chaque instance exécutée avec les paramètres indiquées ci-dessus. Les résultats afficher en gras représente les meilleurs couts obtenue pour chaque instance (chaque ligne). On remarque que notre l'Algorithme LNS_Smoothing donne nous les meilleurs résultats à comparer avec L'Algorithme LNS notre contribution nous as permet de trouver des solutions considérable proche de l'optimal en comparant avec le meilleur cout obtenue jusqu'à présent sur les benchmarks (la colonne meilleur cout).

4. Conclusion

Dans ce chapitre, nous avons effectué une étude empirique et comparative de l'algorithme LNS_Smoothing par rapport aux Benchmarks utilisés sa permet d'évaluer et juger ses performances. Pour l'implémentation, nous avons adopté le langage Java sous l'environnement de développement Eclipse. Pour les expérimentations, nous avons effectué des tests sur les benchmark puis nous avons procédé à une étude comparative par rapport au résultat. Cette étude nous a permis de montrer que l'algorithme LNS_Smoothing est meilleur pour les 29 instances utilisée.

Conclusion générale

Les problèmes de tournées sur arcs est très présents autant dans la littérature que dans la réalité. Cet essai a permis de mieux connaître les problèmes de tournées sur arc et leurs applications dans le monde réel.

La contrainte de capacité associée au transport représente un tracas de plusieurs dirigeants d'entreprises et fait l'objet de plusieurs travaux de recherche.

Dans ce contexte, notre contribution était exploitée par un l'algorithme appliqué avec une méthode de lissage de l'espace (smoothing).

Notre objectif est de fournir un ensemble de tournées satisfaisant tous les clients, en respectant les contraintes de capacité et un coût minimal. Pour cela, on dispose d'une multitude de véhicules de même capacité.

L'algorithme déroule en trois étapes principales : la première étape consiste l'initialisation de la solution. Les deux étapes restantes sont itératives. Elles permettent de calculer la distance, générer et évaluer une solution voisine respectivement.

L'étape de recherche de voisinage comporte les deux phases élémentaires suivantes : appliquer une destruction et puis réparer la solution déjà détruit.

Les performances de l'algorithme de recherche voisinage avec smoothing sont évaluées sur un jeu de test qui contient 29 instances. Pour cela, des paramètres sont définies pour le test. Ensuite, une étude comparative des solutions obtenues est effectuée par rapport à la meilleure solution.

Comme perspectives, nous visons à poursuivre les tests sur les instances de grande taille afin d'évaluer davantage de la contribution.

Bibliographie

[1] Tagmouti, M. (2009). Étude d'un problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service dépendants du temps.

[2] Guan, M. (1962). Graphic programming using odd and even points. *Chinese Math.*, *1*, 237-277.

[3] Orloff, C. S. (1974). A fundamental problem in vehicle routing. *Networks*, *4*(1), 35-64.

[4] Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, *20*(1), 255-282.

[5] Christofides, N., Campos, V., Corberán, A., & Mota, E. (1986). An algorithm for the rural postman problem on a directed graph. In *Netflow at pisa* (pp. 155-166). Springer, Berlin, Heidelberg.

[6] Ball, M. O., & Magazine, M. J. (1988). Sequencing of insertions in printed circuit board assembly. *Operations Research*, *36*(2), 192-201.

[7] Kallehauge, B., Larsen, J., Madsen, O. B., & Solomon, M. M. (2005). Vehicle routing problem with time windows. In *Column generation* (pp. 67-98). Springer, Boston, MA.

[8] Golden, B. L., & Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, *11*(3), 305-315.

[9] Riquelme-Rodríguez, J. P., Langevin, A., & Gamache, M. (2014). Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints. *Networks*, *64*(2), 125-139.

[10] Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations research*, *14*(4), 699-719.

[11] Baldacci, R., & Maniezzo, V. (2006). Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, *47*(1), 52-60.

[12] Lysgaard, J., Letchford, A. N., & Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, *100*(2), 423-445.

[13] Belenguer, J. M., & Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, *30*(5), 705-728.

[15] Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, *12*(4), 568-581.

[16] Belenguer, J. M., Benavent, E., Lacomme, P., & Prins, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, *33*(12), 3363-3383.

[17] Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, *22*(3), 329-337.

[18] Lacomme, P., Prins, C., & Ramdane-Chérif, W. (2001, April). A genetic algorithm for the capacitated arc routing problem and its extensions. In *Workshops on Applications of Evolutionary Computation* (pp. 473-483). Springer, Berlin, Heidelberg.

[19] Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and cybernetics*, *29*(3), 653-684.

[20] Santos, L., Coutinho-Rodrigues, J., & Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, *44*(2), 246-266.

[21] Gu, J., & Huang, X. (1994). Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(5), 728-735.

[22] Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics* (pp. 399-419). Springer, Boston, MA.

[23] Kiuchi, M., Shinano, Y., Hirabayashi, R., & Saruwatari, Y. (1995). An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. In *Spring national conference of the operational research*

society of Japan (pp. 28-29).

[24] DeArmon, J. S. (1981). *A comparison of heuristics for the capacitated Chinese postman problem* (Doctoral dissertation, University of Maryland).