République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ MOHAMMED SEDDIK BENYAHIA JIJEL

FACULTÉ DE SCIENCES EXACTES ET D'INFORMATIQUE

Département d'Informatique

MEMOIRE DE MASTER

Présentée pour l'obtention du diplôme de :

MASTER En INFORMATIQUE

Option : INTELLIGENCE ARTIFICIELLE

Thème :

Résolution de problème de voyageur voleur TTP à base des heuristiques à voisinage variable.

Présenté par:

Mettai Oussama
Bouhanna Houssem

Encadré par :

Lemouari ali

Année universitaire 2020-2021

Remerciements

Avant toute personne, nous tenons à remercier ALLAH le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce modeste travail.

Nous tenons à exprimer notre sincère gratitude à notre encadreur Mr.Lemouari Ali pour ses conseils et ses encouragements tout au long de ce projet.

Nous tenons à exprimer toute notre gratitude à tous les membres de jury, pour avoir bien voulu juger notre travail.

Nous voulons aussi adresser nos sincères remerciements à tous les enseignants de département de l'informatique qui ont contribué à notre formation.

Aussi nos parents qui nous ont toujours soutenus et encouragés.

Par crainte d'avoir oublié quelqu'un, que tous ceux et toutes celles dont nous sommes redevables sont ici vivement remerciés.

Dédicace

Avec une profonde gratitude et des mots sincères, je dédie cet modeste travail à mes chers et respectueux parents, qui ont sacrifié leur vie pour ma réussite, et qui m'ont donné leur amour éternel et inconditionnel, leur soutien, et leurs prières, jour et nuit, qui ont éclairé mon chemin et m'ont aidé dans ma vie, en espérant qu'un jour, je serai leur rendre un peu de ce qu'ils ont fait pour moi. Que Dieu leur accorde le bonheur, la santé et une longue vie.

À

Toute ma famille dont je suis fier. À Mon cher frère Ilyes, et mes chères sœurs,

À

Toutes mes chères amies,

À

Tous ceux que j'aime et ce qui m'aiment,

Une dédicace très spéciale À mon bras droit Ismail Mebirouk (Ami Ali) Pour leur aide et support dans les moments difficiles. Et sans oublier surtout surtout 1908.

Bouhanna Houssem.

Dédicace

Je dédie ce modeste travail à Mes chers parents que nulle dédicace ne puisse exprimer mes sincères sentiments pour leur aide, leur encouragement et leur patience illimitée.

À

Mes sources de fierté, mes chers frères, Sifou et Houssem.

À

Toute ma grande famille et spécialement mon cousin Ilyes, le petit Tamim et ma princesse Halouma.

À

Mon plus proche amie Fares Boubidi, mon troisième frère.

À

Tout mes chers amies chaqu'un a son nom.

Mettai Oussama.

Résumé

De nombreux problèmes du monde réel sont composés de plusieurs sous-problèmes en interaction. Cependant, peu d'études ont été menées pour examiner la façon d'aborder ce type de problèmes avec des méta-heuristiques. Le problème du voyageur voleur (Traveling Thief Problem, TTP) est un nouveau problème NP-difficile avec deux composantes interdépendantes qui visent à fournir un modèle de référence pour mieux représenter cette catégorie de problèmes. Dans ce mémoire, le TTP est étudié de manière théorique et pratique. Deux algorithmes sont proposés pour résoudre le problème: SA et ALNS.

Mots clé: Optimisation combinatoire, Problème de voyageur voleur, Problème de voyageur de commerce, Problème de sac à dos, Heuristique, Méta-heuristique, Recherche local, Recherche par large voisinage, Recuit simuler,

Abstract

Numerous real-world problems are composed of several interacting sub-problems. However, few studies have been conducted to examine how to address such problems with meta-heuristics. The Traveling Thief Problem (TTP) is a new NP-hard problem with two interdependent components that providing a benchmark model to better represent this class of problems. In this thesis, the TTP is studied in a theoretical and practical way. Two algorithms are proposed to solve the problem: SA and ALNS.

Keywords: Combinatory optimization, Traveling thief problem, Traveling salesman problem, Knapsack problem, Heuristic, Meta heuristic, Local search, Large neighborhood search, Simulated annealing.

Table des matières

Liste des figures :

Liste des tableaux

Liste des abréviations

-TSP : Travelling Salesman Problem

-KP : Knapsack Problem

-VRP : Vehicle Routing Problem

-TTP : Travelling Thief Problem

-ALNS : adaptative large neigborhood search

-SA : Simulated annealing

-TSPLIB : Travelling Salesman Problem libreary

-CPU : central processing unit

-LK : Lin-Kernighan

-LKH : LK de Helsgaun

-EAX : le croisement par assemblage

-GPX : le croisement de partition généralisé

-CX : le croisement des cycles

-PMX : le croisement partiellement mappé

-PX : le croisement des positions

-OX : le croisement d'ordre

-ERX : le croisement par recombinaison d'arêtes

-COMBO : un algorithme de programmation dynamique

-PWT : Packing While Traveling

-SH : l'heuristique simple

-RLS : recherche locale aléatoire

-DH : Density-based Heuristic

-JNB : Joint N1-BF

-J2B : Joint 2opt-BF

-MA2B,MA : algorithme mimétique

-OPX : le croisement une-point

-MMAS : Un algorithme d'optimisation

-LLH : heuristiques de bas niveau.

-ABC : Artificial Bee Colony

-MTTP : le problème du voyageur voleur multiple

CHAPITRE 1

INTODUCTION.

1.1 Contexte et motivation

Ces jours là, les chercheurs à travers le monde donnent une grande importance aux problèmes d'optimisation dans plusieurs domaines d'études : en mathématiques appliquées, en recherche opérationnelle, dans les différentes applications de l'intelligence artificielle, en électronique et automatique, .etc. La cause de cette importance est le besoin de trouver des solutions à des problèmes réels, souvent similaires [1].

Les problèmes d'optimisation peuvent être divisés en deux catégories selon que les variables sont continues ou discrètes [2], on s'intéresse ici à l'optimisation combinatoire avec des variables discrètes, l'objectif est de trouver une meilleure solution parmi un très grand nombre de possibilité [3].

La plupart des problèmes étudiés appartiennent à la classe des problèmes connus sous le nom des problèmes NP-difficiles, de sorte que la résolution de grandes instances de ces problèmes jusqu'à l'optimalité n'est pas possible. ce type de problèmes comprend comme exemple : le problème de voyageur de commerce (TSP-Travelling Salesman Problem) , le problème de sac a dos (KP-Knapsack Problem) , le problème de la tournée de véhicules (VRP-Vehicle Routing Problem) ...etc [1].

Plusieurs méthodes exactes pour résoudre ces problèmes NP-difficiles ont été proposées, par ex. branch and bound, programmation dynamique, méthode de génération des colonnes. Bien que ces méthodes exactes aient remportés quelques succès dans la résolution de ce type des problèmes mais aucune de ces méthodes n'était efficace dans le temps, pour les instances du problème de grande taille.

Il est donc souvent nécessaire d'utiliser des méthodes qui fournissent une solution de bonne qualité (approché a l'optimal) dans un temps raisonnable, on parle ici d'heuristiques, par ex. la méthode de Lin-kernighan utilisée pour le problème de voyageur de commerce [4], la méthode de Pirkul's utilisée dans la résolution du problème de sac a dos [5], etc.

Ces méthodes sont très efficaces en terme de temps de résolution, dont leurs inconvénients est: la particularité, la méthode doit être adaptée aux problèmes envisagés, il est donc impossible de trouver des solutions à des problèmes Np -difficile qui sont très variés, alors il est important de concevoir des méthodes plus général, les méta-heuristiques, parmi ceci la méthode de recherche tabou, le recuit simulé, les algorithmes génétiques, les colonies de fourmis et les algorithmes évolutionnaires[1].

Ce type de méthodes ne nécessite pas des connaissances particulières sur le problème à optimiser pour fonctionner. Le fait de pouvoir associer une (ou plusieurs) valeurs à une solution est la seule information nécessaire.

1.2 Problème étudié

Dans ce mémoire, on va s'intéresser à la combinaison des deux problèmes combinatoires NP-difficiles largement étudiés, qui sont indépendants l'un de l'autre : le problème de voyageur de commerce (Travelling Salesman Problem - TSP) et le problème de sac à dos binaire (0-1 Knapsack Problem - KP). Cette combinaison conduit à un nouveau problème qui lui aussi NP-difficiles : le problème du voyageur voleur (Travelling Thief Problem - TTP).

Nous cherchons dans le cadre de ce mémoire de présenter les travaux qui ont été proposés pour ce problème et ensuite nous proposons le recuit simulé (SA), comme méthode de résolution, des heuristiques de recherche de voisinage ont été réalisés afin de trouver les meilleurs voisins cherchés, dont un but d’améliorer les résultats des solutions recherchés avons utilisé le SA comme étape intermédiaire avant de passer à la méthode de voisinage adaptés (ALNS).

1.3 Organisation de mémoire

Ce mémoire est structuré en quatre chapitres. Dans ce qui suit nous détaillons le contenu des différents chapitres.

- Le premier chapitre c’est une introduction dont Nous présentons la vision générale du problème étudié.

- Nous présentons dans le chapitre 2 le problème de voyageur voleur (TTP), sa définition, les sous-problèmes liés, sa représentation formelle et d'autres détailles sur le problème.

- Nous fournissons dans le chapitre 3 des méthodes de résolution de TTP, utilisées dans la littérature.

- Dans le chapitre 4, Nous proposons l'ALNS et le SA comme des méthodes de résolution pour le TTP.

- Nous présentons dans le chapitre 5 notre application sur les méthodes proposées (ALNS et SA), les résultats de l'application, et les instances de référence utilisées pour les testes.

CHAPITRE 2

Le problème de voyageur voleur.

2.1 Introduction

Récemment, un nouveau problème de référence appelé le problème du voyageur voleur a été introduit [1] dans une tentative de fournir une abstraction des problèmes comportant de multiples composants interdépendants.

Dans ce chapitre, nous présentons le problème voyageur voleur (TTP), sa définition, ces sous problème et l'interdépendance entre eaux, une représentation mathématique du problème et finalement l'origine de la variante du problème étudier dans ce mémoire.

2.2 La définition de problème de voyageur voleur

Le problème du voyageur voleur (Travelling Thief Problem - TTP) est une combinaison de deux sous-problèmes NP-difficiles bien connus interdépendants l'un de l'autre qui sont: le problème de voyageur de commerce (Travelling Salesman Problem - TSP) et le problème de sac à dos binaire (0-1 Knapsack Problem - KP).

2.3 Les sous-problèmes de TTP

Le TTP est une combinaison de deux sous-problèmes: le problème de voyageur de commerce et le problème de sac à dos binaire.

2.3.1 Le problème de voyageur de commerce

Le problème du voyageur de commerce est l'un des problèmes d'optimisation les plus étudiés [6], Sa popularité est due à sa nature: facile à comprendre mais difficile à résoudre. Il a également une grande variété d'applications dans de nombreux domaines: la logistique, la

cartographie des génomes, la visée des télescopes, le guidage des machines industrielles, l'ordonnancement des tâches et bien d'autres encore [7].

Dans le problème du voyageur de commerce [8]on considère N villes et la matrice D des distances entre chaque paire de ces villes tel que : $D_{i,j} = D_{j,i}$ (i , j ∈ {1,...,N}), on vise à trouver le chemin d'un vendeur qui visite tout les villes exactement une fois et retourne au ville de départ de telle sorte que la distance parcourue soit minimale.

Min F(X) $= D_{xn,x1} + \sum_{i=1}^{N-1} D_{xi,xi+1}$, X = $\{x_1, x_2, \dots, x_n\}$ (x_i ∈ {1,...,N}) représente l'ordre de la visite des villes.

2.3.1.1 Méthodes exacte pour le TSP

Une amélioration importante, mais encore peu pratique, a été apportée par l'algorithme de Held-Karp qui résoudre le TSP par programmation dynamique et présente une limite de temps améliorée de O ($n^2 2^n$) [9]. Cela prend encore trop de temps, même pour les plus petites instances.

L'état de l'art de l'utilisation d'une méthode exacte pour résoudre le TSP est la programmation linéaire en utilisant la méthode de branch and cut, L'implémentation la plus efficace de cette méthode est celle de Concorde [10], Concorde est un logiciel disponible gratuitement qui contient 130.000 lignes de code et se trouve au cœur de la recherche effectuée par Applegate et al [6].

Il comprend de nombreuses méthodes qui ont été accumulées au cours de plus de 60 ans de recherche LP sur le TSP. Concorde peut résoudre des instances jusqu'à 85900 nœuds, par exemple l'instance pla85900 de TSPLIB [11]. Le calcul se fait cependant au coût d'une consommation de plus de 136 CPU pendant des années sur un ensemble d'ordinateurs. Les instances de petite taille, c'est-à-dire 1000 nœuds, ne prennent que quelques minutes. La programmation linéaire est l'approche TSP exacte la plus réussie proposée.

2.3.1.2 Les algorithmes de construction

En raison de la difficulté du problème, de nombreuses recherches sur le TSP ont visé des algorithmes heuristiques. Parmi les premières tentatives, on peut citer l'algorithme du plus proche voisin, une tournée est créée en suivant une règle simple, aller vers la ville la plus proche. Cet algorithme naïf fonctionne raisonnablement bien pour les premières villes, mais il

nécessitera par la suite de longs trajets pour revenir aux villes non visitées. En fait, Gutin, Yeo et Zverovich [12] ont montré qu'il peut renvoyer la pire route possible.

D'autres algorithmes de construction sont proposés donnent une garantie sur la qualité de la solution : les algorithmes d'approximation. Parmi les meilleurs algorithmes d'approximation à ce jour est l'algorithme de Christodes [13].

L'algorithme de Christodes trouve la tournée avec la procédure suivante : Tout d'abord, un arbre minimal T est créé. Ensuite, un appariement parfait de poids minimum M dans les sommets de degré impair de T est trouvé. Ensuite, un tour d'Euler est construit dans T U M. En supprimant les sommets en double dans ce tour d'Euler donne un tour TSP avec une qualité qui n'est pas inférieure à 3/2 de la solution optimale.

En pratique, la tournée est beaucoup plus proche que 3/2 de l'optimum, mais il est toujours surpassé par un simple algorithme d'escalade (hill climber) utilisant 2-opt [14].

2.3.1.3 L'heuristique de Lin-Kernighan

Des succès plus pratiques ont été obtenus par diverses applications de l'heuristique Lin-Kernighan développée par Lin et Kernighan [17]. est une généralisation de la recherche à 2-opt et constitue l'épine dorsale de nombreuses méta-heuristiques qui résolvent les TSP [6].

LK améliore itérativement la tournée en recherchant une séquence d'échanges d'arêtes telle que chaque sous-séquence initiale a une chance d'améliorer la tournée. Si elle améliore effectivement le tour, la séquence d'échanges d'arêtes est exécutée. Si aucune séquence ne peut être trouvée, un optimum local a été atteint et l'algorithme s'arrête.

Pour trouver une meilleure solution, vous pouvez exécuter LK plusieurs fois (MLS) ou perturber la solution et appliquer à nouveau la recherche locale (ILS). C'est exactement ce que fait l'heuristique Lin-Kernighan chaînée.

Elle expulse une solution hors du bassin d'attraction de son optimum local par un mouvement non séquentiel appelé le double pont. Le double pont est un échange de 4 arêtes tel qu'une recherche de Lin-Kernighan ne peut trouver l'ensemble de retournements nécessaires pour annuler les arêtes échangées. À ce jour, l'heuristique de Lin-Kernighan chaînée reste l'une des meilleures approches pour les très grands ensembles de données (n > 25.000.000) [7].

Une implémentation assez efficace de cette heuristique LK est celle de Helsgaun, appelée LKH [18]. LKH comporte différentes optimisations qui accélèrent l'algorithme ainsi que des changements significatifs qui modifient la structure de l'algorithme.. Au lieu d'utiliser l'échange à 2-opt comme mouvement de base, il utilise un échange à 5-opt, en considérant 10 arêtes en même temps.

Un autre changement est que LKH ne considère que les arêtes prometteuses pour l'échange. Les arêtes prometteuses sont celles qui sont susceptibles de se trouver dans la solution optimale. Il trouve ces bords par une mesure appelée α-mesure qui est définie comme la distance au minimum correspondant.

Un arbre couvrant avec un bord supplémentaire. Il a été prouvé empiriquement qu'un arbre minimal contient entre 70 et 80 % des arêtes d'une tournée optimale [18].LKH détenait le record pour plusieurs instances TSP et avait déjà trouvé l'optimum global de pla85900 avant Applegate et al. Mais en beaucoup moins de temps [6].

2.3.1.4 Algorithme génétique pour le TSP

Bien que les premiers algorithmes génétiques pour le TSP n'aient pas été couronnés de succès [7], des algorithmes génétiques avec des nouveaux croisements de permutation comme EAX et GPX donnent résultats comparatifs avec l'heuristique de Lin-Kernighan enchaînée et LKH [24, 25].

Un aspect important d'un algorithme génétique est son opérateur de recombinaison : Trouver un bon croisement pour les permutations n'est pas trivial. L'utilisation d'un croisement traditionnel, tel que le croisement en 1-point ou le croisement uniforme n'est pas une option.

Figure 1 : Exemple de cas où le croisement en un point donne une solution invalide, extrait de R.H.Wuijts [80].

Au cours des 30 dernières années, de nombreux opérateurs de croisement ont été définis pour la représentation par permutation de TSP. Les différents croisements se concentrent sur différents aspects du problème. Certains valorisent la position absolue, comme le croisement des cycles (CX), le croisement des positions (PX), le croisement partiellement mappé (PMX) et dans une certaine mesure, le croisement d'ordre (OX).

D'autres croisements valorisent l'ordre relatif, comme le croisement à conservation maximale (MPX) et OX. Le dernier groupe évalue l'information d'adjacence, comme le croisement par recombinaison d'arêtes (ERX), le croisement de partition (PX), le croisement de partition généralisé (GPX) et le croisement par assemblage d'arêtes (EAX). Les informations d'adjacence et l'ordre relatif sont des informations importantes pour TSP, mais pas la position absolue [23].

En général, pour le problème TSP, il s'avère que CX < PMX < OX < ERX [26, 27].

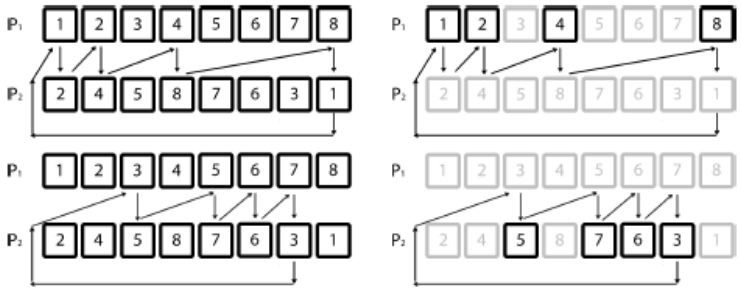

Figure 2 : Exemple de deux cycles (à gauche) et d'un croisement de cycles (à droite) , extrait de R.H.Wuijts [80].

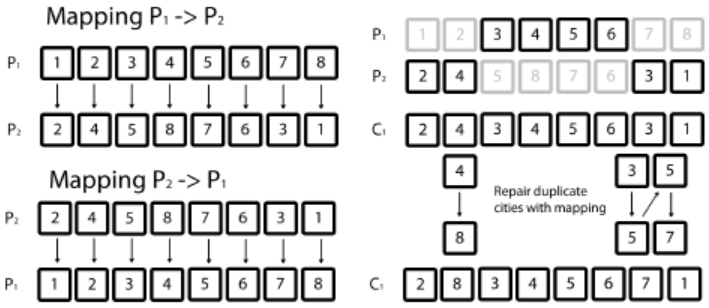

Figure 3 : Exemple de mappage entre un parent (à gauche) et un croisement partiellement mappé (à droite) , extrait de R.H.Wuijts [80].

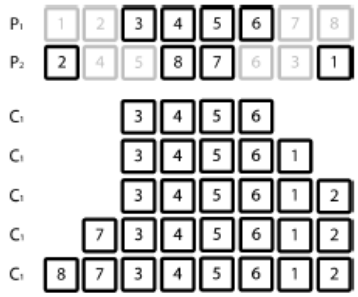

Figure 4 : Exemple de mappage entre un parent (à gauche) et un croisement partiellement mappé (à droite) , extrait de R.H.Wuijts [80].

2.3.1.5 Le TSP comme un sous-problème de TTP

Comme mentionné précédemment, le problème du voyageur de commerce fait partie du problème du voyageur voleur. En fait, si aucun objet n'est ramassé ou si le tarif de location R est fixé à une valeur extrêmement élevée, le sous-problème devient un TSP.

Une différence majeure est que la distance entre les villes n'est pas la seule à compter dans la fonction objective, mais que le temps de trajet total de l'ensemble de la tournée est important. Le temps de trajet dépend de la vitesse du voleur et donc des objets qui sont ramassés. Il dépend également de l'ordre dans lequel ils sont ramassés.

En général il est préférable de ramasser les objets lourds à la fin de la tournée. Une autre différence entre TSP et TTP est qu'il existe une ville initiale. Par conséquent, l'ordre relatif et la position absolue ne jouent pas un rôle dans la TSP, mais ils sont importants pour TTP.

L'utilisation du temps de parcours au lieu de la distance présente un autre inconvénient: les petits changements affectent l'ensemble de la tournée et l'évaluation incrémentale de l'aptitude n'est donc pas possible. Il s'agit d'un inconvénient majeur car il multiplie la complexité temporelle de presque toutes les heuristiques TSP par un facteur O(n).

Cela implique également que les algorithmes, comme LK, et les opérateurs, comme EAX ou GPX, ne sont pas trivialement traduits en TTP, LK prend des décisions basées sur la longueur des bords et EAX recombine les solutions partielles d'une manière avide.

La question reste de savoir si ces opérateurs de recombinaison peuvent être utilisés d'une manière efficace et effective.

2.3.2 Le problème de sac a dos binaire

Le problème KP a été étudié en profondeur par Martello, Pisinger et Toth [28]. Leur travail porte sur le 0-1 et proposent un algorithme de programmation dynamique exact appelé COMBO pour le résoudre. De plus, ils proposent et étudient un test de référence riche en moyens pour générer des données KP.

COMBO adopte un ensemble d'inégalités valides ainsi qu'un nouveau problème de base initial qui lui permet de résoudre des instances comportant jusqu'à 10 000 objets en moins de 0,2 seconde.

Le problème KP est un problème d'optimisation NP-Difficile. Dans ce problème, on considère qu'il y a un voleur qui veut remplir son sac à dos en prenant certains objets, il y a M objets, chaque objet k (k ∈ {1,..., M}) a une valeur $Z_k > 0$ et un poids $P_k > 0$, La capacité du sac à dos est limitée (C).

L'objectif du voleur est de choisir les objets pour maximiser leur valeur totale mais a condition de ne dépasse pas la capacité du sac C. Le problème est modélisé comme suit :

$$\text{Max } F(X) = \sum_{i=1}^{M} Z_i X_i$$

$$\sum_{i=1}^{M} P_i X_i \leq C$$

$$x \in \{0,1\}$$

2.3.2.1 Méthodes exacte pour le KP

Les algorithmes de sac à dos performants sont généralement basés sur deux approches, la méthode de branch-and-bound et la programmation dynamique [20]. La plupart d'entre eux utilisent la solution entière de Dantzig en tant que solution initiale ou utilisent une limite supérieure de la solution continue.

Des limites plus étroites sont trouvées en calculant la cardinalité maximale. Des accélérations supplémentaires peuvent être obtenues en multipliant les coefficients.

Une façon de résoudre le sac à dos est d'utiliser des algorithmes de base qui ne prennent en compte que les éléments intéressants appelés le noyau. Le noyau est défini comme l'ensemble des éléments qui entourent l'élément de rupture.

Tous les éléments dont le rapport est supérieur à tous les éléments du noyau qui sont inclus dans le sac à dos et tous les éléments dont le rapport est inférieur à tous les éléments du noyau qui sont exclus. Si le noyau est choisi correctement, le problème est résolu de manière optimale. il peut être utilisé comme une limite supérieure. Le choix d'un noyau correct peut être difficile.

Pisinger [22, 21] résout ce problème en trouvant le noyau correct à la volée en ajoutant ou en supprimant des éléments de manière itérative. Le noyau lui-même peut être résolu soit par une ILP, soit par une programmation dynamique.

2.3.2.2 La difficulté du KP

La plupart des instances de sac à dos étudiées dans la littérature sont générées artificiellement. La difficulté d'une instance de sac à dos dépend en partie de la corrélation choisie entre le profit et le poids. Une instance où le profit et le poids sont choisis indépendamment est dites non corrélée et elle est facile à résoudre [29].

Cela vient du fait que les limites supérieures peuvent facilement identifier quel élément ne contribuera pas à une solution optimale. Les cas où le profit dépend fortement du poids, $Z_i = P_j + \in$, sont beaucoup plus difficiles à résoudre. Deux observations de Pisinger [30] expliquent pourquoi elles sont plus difficiles:

1) Il y a un grand écart entre la solution continue et la solution entière du problème.

2) Pour tout petit intervalle d'objet commandés, il y a une variation limitée dans les poids, ce qui rend difficile de remplir le sac à dos à la capacité maximale.

L'utilisation de coefficients plus élevés augmentera également le temps de calcul de la programmation dynamique, Martello, Pisinger et Toth [29] suggèrent que les instances dont le coefficient croît de manière exponentielle resteront difficiles à résoudre en raison de la NP-Difficulté du problème.

2.3.2.3 Le KP comme un sous-problème de TTP

Le problème de sac a dos est un sous-problème du TTP. Si le tarif de location R ou les distances entre les villes sont fixées à zéro, alors le TTP devient un KP. La principale différence entre le problème du sac à dos et le problème du sac à dos dans le TTP est que la décision de ramasser un article ne dépend pas seulement du poids et du bénéfice, mais aussi du temps total de déplacement.

Cela rend la propriété du rapport poids-profit beaucoup moins importante que dans le problème normal du sac à dos, qui est la clé de presque tous les algorithmes efficaces du sac à dos.

Le gain d'un élément dépend de la tournée mais aussi de l'état des autres objets. La situation où la tournée est maintenue fixe est un problème à part entière appelé Packing While Traveling (PWT) [31]. Polyakovskiy et Neumann [19] résolvent le PWT en utilisant d'abord un schéma de prétraitement qui diminue le nombre des objets en incluant ou en supprimant directement des objets.

Ensuite, ils résolvent les instances réduites avec l'une des deux approches exactes l'une utilisant la programmation par contraintes avec la méthode branch-and-cut et l'autre utilisant la programmation mixte en nombres entiers.

Neumann et al. [32] ont par la suite proposé une approche exacte plus efficace avec la programmation dynamique qui résout le PWT en temps pseudo-polynomial. Dans le même article, ils ont proposé un schéma d'approximation en temps entièrement polynomial pour une variante de PWT.

Les premières tentatives pour traduire ces résultats en TTP ont été faites [33]. et il reste intéressant de voir si la programmation dynamique de Neumann et al. [32] peut être utilisée dans le cadre d'un algorithme de résolution de TTP.

2.3.3 L'interdépendance entre les sous problèmes

L’une des principales complexités des problèmes du monde réel est l'interdépendance entre les sous-problèmes, ce qui rend inefficaces de nombreuses approches conventionnelles. Par conséquent, même si chaque sous-problème a été étudié de manière approfondie, la question de l'intégration des solutions partielles de haute qualité reste ouverte.

Comment intégrer les solutions partielles de haute qualité pour les sous-problèmes afin d'obtenir un optimum global ou au moins une solution de haute qualité pour le problème global, il est donc important d'étudier comment traiter l'interdépendance entre les sous-problèmes [15].

Dans le TTP, la combinaison de TSP et de KP augmente le nombre de variables de décision, ce qui rend le problème plus difficile que les problèmes de référence ordinaires. Cependant, l'interdépendance rend le problème encore plus difficile à résoudre en augmentant la complexité de l'évaluation, Ainsi, la conception d'un algorithme fort qui traite intelligemment les défis informatiques du TTP exige plus d'efforts [16].

De plus, en combinant ces deux problèmes, les contraintes d'un sous-problème influencent la région de faisabilité des solutions pour l'autre sous-problème. Par exemple, dans le TSP original, toute permutation des villes était une solution réalisable. Cependant, dans le TTP, si la solution pour KP devient infaisable, toute la Solution et les permutations de la tournée deviennent également infaisables [1].

Bonyadi, Michalewicz et Barone [1] ont montré dans l'article original qu'un optimum global pour l'un des deux sous-problèmes n'est pas nécessairement un optimum global du problème entier. En effet, pour certaines instances, des solutions avec des tournées beaucoup plus longues ont de meilleures valeurs de fitness que celles qui ont une tournée qui se situe plus près des solutions optimales de TSP

Mei, Li et Yao [15] ont étudié l'interdépendance entre les deux sous-problèmes et ont déclaré que l'interdépendance non linéaire des sous-problèmes rend difficile, ou ' impossible' de décomposer le problème en sous-problèmes indépendants.

2.4 La représentation formelle du problème

dans le TTP on considère N villes et la matrice D des distances entre chaque paire de ces villes tel que : $D_{i,j} = D_{j,i}$ $(i , j \in \{1,...,N\})$,on considère aussi M objets distribué sur les N villes ,chaque objets k ($k \in \{1,...,M\}$) est situé a une ville l_k ($l_k \in \{2,...,N\}$) ,a une valeur $Z_k > 0$ et un poids $P_k > 0$, le voleur doit visiter chacune des villes exactement une fois ,ramasser des objets et revenir a la première ville visité avec une condition de ne pas dépasser la

capacité de sac C lors de la collection des objets ,il a une vitesse V ($V_{min} < V < V_{max}$) dépend de la charge de sac et sa capacité C ,le voleur doit payer pour chaque unité de temps qu'il prend dans sa tourné un tarif de location R.

une solution est représenté par 2 vecteurs X , Y tel que :

- $X = \{x1, x2, \ldots, xn\}$ ($xi \in \{1, \ldots, N\}$) représente l'ordre de la visite des villes
- $Y = \{y1, y2, \ldots, ym\}$ ($yk \in \{0,1\}$) représente le choix des objets, la valeur y_k=1 exprime que l'objet k est choisi.

le but est de trouver un ordre des villes pour la tourné X et un choix des objets Y pour maximiser le gain $G(X,Y)$ tel que:

$$G(X,Y) = B(Y) - R * T(X,Y)$$

$B(Y)$ Représente le bénéfice c'est a dire le total des valeurs des objets choisi :

$$B(Y) = \sum_{i=1}^{M} Z_K * Y_K$$

$T(X,Y)$ Représente le temps que le voleur prend pour la tourné :

$$T(X,Y) = \frac{Dxn,x1}{V(xn)} + \sum_{i=1}^{N-1} \frac{D_{xi,xi+1}}{V(xi)}$$

$V(xi)$ Représente la vitesse de voleur dans la ville x_i :

$$V(xi) = Vmax - \frac{(Vmax-Vmin)}{C} * W(xi)$$

W (x_i) représente le poids chargé dans le sac lorsque le voleur soit dans la ville (x_i)

2.5 D'autre variante de problème de voyageur voleur

Dans l'article original de Bonyadi, Michalewicz et Barone [7], ils proposent deux variantes du problème de voyageur voleur, TTP1 et TTP2.

Le TTP2 est un problème d'optimisation bi-objectif, minimiser le temps de déplacement total et maximiser le profit, avec une caractéristique supplémentaire que la valeur d'un objet diminue avec le temps. Cette variante a reçu peu d'attention dans les recherches.

TTP1 est la variante la plus étudiée dans les recherches mais Il est important de noter que la définition de TTP1 est légèrement différente de la définition du TTP communément utilisée. Dans l'article original le voleur pouvait prendre le même objet à différents villes alors que la plupart des recherches actuelles concentre sur la variante de Polyakovskiy [19] , dans lequel les objets sont limités à une seule ville

2.6 Conclusion

Dans ce chapitre, le problème voyageur voleur (TTP), sa définition, ces sous problème et l'interdépendance entre eaux sont détaillé avec une représentation formel du problème et finalement l'origine de la variante du problème étudier dans ce mémoire.

Dans le prochain chapitre Nous fournissons des méthodes de résolution utilisées dans plusieurs recherches sur le TTP et des méthodes de recherche locale utile pour ce problème.

CHAPITRE 3

Revue de littérature.

3.1 Introduction

A cause de la nécessité de trouver des solution a ce type de problèmes composé de plusieurs sous-problème combiné, de nombreuses recherches sur le TTP sont publier depuis la création de problème en 2013, dans ce chapitre Nous présentons une revue de littérature, des méthodes de résolution utilisé dans plusieurs recherches sur le TTP et aussi des méthodes de recherche local utile pour ce problème.

3.2 L'origine du problème TTP

Bonyadi et al. [1] ont introduit pour la première fois le TTP en 2013 comme problème de référence pour la résolution des problèmes multi-composants avec leur interaction. Ils ont utilisé une petite instance avec quatre villes et de cinq éléments pour montrer l'interconnexion entre les deux composantes du TTP.

L'année suivante, Polyakovskiy et al. [34] ont créé de grandes instances de l'ensemble TTP avec presque 100 000 villes et 1 000 000 objets.

3.3 Algorithmes à solution unique

3.3.1 Les méthodes heuristiques SH, RLS et (1-1) EA

Polyakovskiy et al. [34] ont été proposés plusieurs méthodes heuristiques pour résoudre le TTP. Leur approche est principalement basée sur deux étapes. La première étape consistait à générer une bonne séquence de tournées pour le composant TSP en utilisant

LKH. La deuxième étape consistait en une heuristique d'emballage sur une tournée spécifiée pour obtenir une meilleure solution.

Leur première approche pour résoudre le TTP était l'heuristique simple (SH) Algorithme 1[34]. Dans cette approche, les éléments sont choisis en fonction de la valeur du score. Ils ont également introduit un couple d'heuristiques itératives, recherche locale aléatoire (RLS) Algorithme 2 [34] et (1+1) EA Algorithme 3 [34] basée sur le retournement des objets choisis avec une certaine probabilité. Pour chaque itération, s'il y avait une amélioration, les changements sont conservés, sinon ils sont ignorés.

Algorithm 1 Simple Heuristic (SH)

1: Fill the array D with values d_{x_i}, $x_i \in \{x_2, \ldots, x_n\}$
2: Calculate the total traveling time t'.
3: **for all** items I_{x_ik}, $x_i \in \Pi$, $k \in M_{x_i}$ **do**
4: Calculate t_{x_ik} by using Equation 1
5: Set $t'_{x_ik} := t' - d_{x_i} + t_{x_ik}$
6: Set $score_{x_ik} := p_{x_ik} - R \times t_{x_ik}$
7: Set $u_{x_ik} := R \times t' + (p_{x_ik} - R \times t'_{x_ik})$
8: Create the joint set of items I and sort it in descending order score values
9: Set the current used capacity variable $Wc := 0$
10: **for all** items $I_{x_ik} \in I$ **do**
11: **if** $(Wc + w_{x_ik} < W)$ and $(u_{x_ik} > 0)$ **then**
12: Add the item I_{x_ik} to the packing plan P
13: Increase the used capacity variable $Wc := Wc + w_{x_ik}$
14: **if** $(Wc = W)$ **then**
15: Exit the loop.
16: Set the resulting objective value $Z^* := max\,(Z(\Pi, P), -R \times t')$

Algorithm 2 Random Local Search (RLS)

1: Initialize P^* such that no items are packed.
2: **repeat until no improvement for** X **iterations**
3: Create P by inverting the packing status of a uniformly at random picked item of P^*.
4: **if** $Z(\Pi, P) \geq Z(\Pi, P^*)$ and $w(P) \leq W$ **then**
5: $P^* := P$

Algorithm 3 (1+1) Evolutionary Algorithm (EA)

1: Initialize P^* such that no items are packed.
2: **repeat until no improvement for** X **iterations**
3: Create P by inverting the packing status of each item of P^* independently with probability $1/m$.
4: **if** $fZ(\Pi, P) \geq Z(\Pi, P^*)$ and $w(P) \leq W$ **then**
5: $P^* := P$

3.3.2 Les heuristique DH et CoSolver

Bonyadi et al. [35] ont proposé deux méthodes heuristiques pour résoudre le TTP. Dans la première approche, le TTP a été décomposée en sous-problèmes, ils ont donc résolu

chaque sous-problème séparément avec une certaine interaction entre eux, puis ils ont composé les solutions pour obtenir la solution globale du TTP.

Cette méthode est appelée CoSolver. Dans la deuxième approche, appelée Density-based Heuristic (DH), une tournée est d'abord générée pour le TSP, puis une solution pour la composante KP est générée en fonction de la tournée, Cette procédure est commune à toutes les heuristiques d'empaquetage de la littérature [34, 33, 36, 37, 38, 15, 39, 40, 41, 42].

La principale différence entre les heuristiques d'empaquetage est soit l'ordre dans lequel les éléments sont traités, soit le mécanisme qui inclut un élément dans le plan d'emballage. Certaines heuristiques utilisent une fonction objectif et d'autres une fonction objectif approximative.

CoSolver est un algorithme qui décompose le TTP en deux sous-problèmes. Il résoudre ces sous-problèmes et échange des informations entre eaux. A la fin, les deux solutions sont combinées pour former une solution de TTP.

La façon dont ces sous-problèmes sont résolus n'est pas explicite. Ils déclarent que le sous-problème TSP est résolu par un algorithme exact et que le sous-problème KP est relaxé et résolu par programmation dynamique. CoSolver est détaillé dans Algorithme 4 [35] .

Algorithme 4 : CoSolver

$d_k \leftarrow 0$
$W_k \leftarrow 0$
$P \leftarrow -\infty$
for $r \leftarrow 1$ **to** $MaxIter$ **do**
  $pickedItems' \leftarrow$ solve KRP with p_k, w_k, d_k and parameter W_k
  $W_k \leftarrow \sum_{i \in pickedItems'} w_i[0 < x_i \leq k]$
  $\Pi' \leftarrow$ solve TSKP with W_k, d
  $P' \leftarrow calcObj(pickedItems', \Pi')$
  if $P' > P$ **then**
    $P \leftarrow P'$
    $\Pi \leftarrow \Pi'$
    $pickedItems \leftarrow pickedItems'$
    $d_k \leftarrow d(\Pi_k, \Pi_{k+1})$
  else
    break
return $pickedItems, \Pi$

Les résultats montrent que sur la plupart des instances de test, CoSolver est plus performant que DH sur les petite instances N<25, probablement parce que CoSolver possède

une sous-routine exacte. D'après les auteurs, le fait que DH soit moins performant que CoSolver signifie que la prise en compte de l'interdépendance est bénéfique pour la résolution de problèmes à composantes multiples. Pour les grandes instances, CoSolver est plus performant que DH.

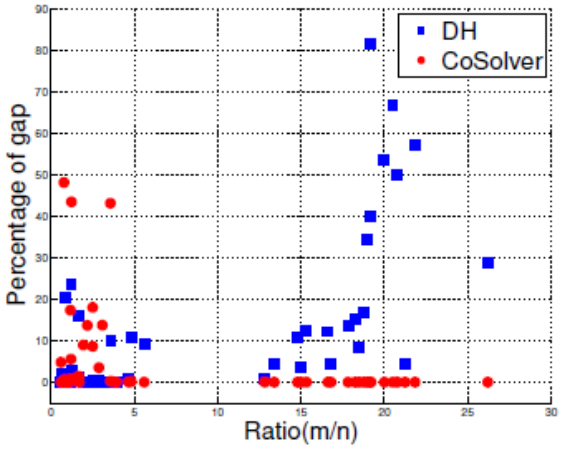

Figure 5 : Performance de DH et CoSolver dans la résolution des problèmes de référence générés par rapport au nombre d'éléments et au nombre de villes, extrait de Bonyadi et al. [35]

3.3.3 Les algorithmes itératifs de voisinage JNB et J2B

Yafrani et Ahiod [43] ont proposé deux algorithmes itératifs de voisinage basés sur la recherche locale. Le premier combine le voisinage N1 (en échangeant deux villes adjacentes) pour le TSP et un bit-flip pour la KP, appelé JNB (Joint N1-BF). La seconde est une combinaison de 2-OPT et d'un bit-flip, nommée J2B (Joint 2opt-BF).

Ils ont également expérimenté avec des tournées initiales et ont conclu que l'utilisation d'une bonne tournée de TSP comme solution initiale donne de bien meilleurs résultats que l'utilisation d'une tournée aléatoire comme initialisation. Ils ont trouvé que JNB et J2B étaient plus performants que (1+1)-EA et RLS lorsqu'ils pouvaient terminer dans le temps imparti.

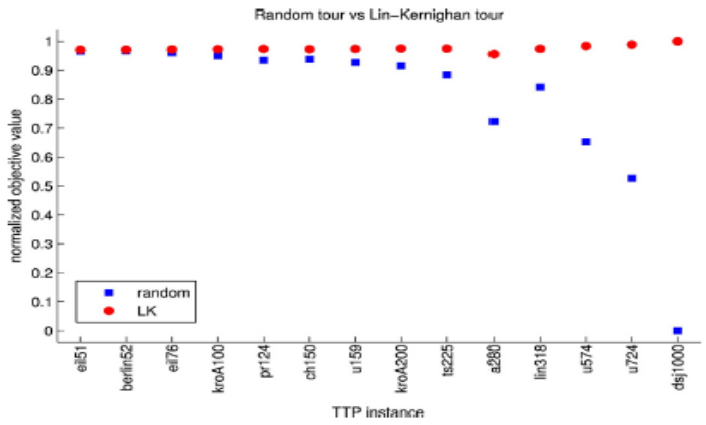

Figure 6 : Comparaison de la tournée initiale de Lin-Kernighan et de la tournée aléatoire en termes de qualité de la solution, extrait de Yafrani et Ahiod [43].

3.3.4 Les heuristiques d'emballage itérative S1-S5 et C1-C6

Faulkner et al. [33] ont fourni une série d'algorithmes tous basés sur leur heuristique d'emballage appelée PackingIterative, utilise une sous-routine Pack. Pack est une heuristique d'empaquetage avide et ajoute des éléments en fonction d'un score qui est défini de la manière suivante :

$$score2\ (I_{ik}\ , \alpha)\ =\ \frac{Z_{ik}^{\alpha}}{P_{ik}^{\alpha}\ *\ d_i}$$

où Z_{ik} et P_{ik} sont le bénéfice et le poids de l'objet I_{ik} et d_i est la longueur totale restante de la tournée dans la ville i. L'exposant α modifie l'impact des variables sur le score. L'exposant influence donc l'ordre dans lequel les objets sont ramassés. Le succès de l'heuristique dépend d'une bonne valeur pour α. Pack est décrit dans l'Algorithme 5 4 [35]:

Algorithme 5: Packing Routine Pack

```
compute score for each of the items I_m ∈ M
sort the items of M in non-decreasing order of their scores
set the frequency μ = ⌊m/τ⌋
set the current packing plan P = ∅ and W' = 0
set the best packing plan P* = ∅
set best objective value Z* = −∞
set the counter k = 1 and set k* = 1
while (W' < W) and (μ > 1) do
    if (W' + w_k ≤ W) then
        add item I_k ∈ M to the packing plan P = P ∪ {I_k}
        set W' = W' + w_k
        if (k mod μ = 0) then
            compute the objective value Z = Z (Π, P)
            if (Z < Z*) then
                restore the packing plan P = P*
                set k = k* and update W'
                set μ = ⌊μ/2⌋
            else
                update the best packing plan with P* = P
                set k* = k and Z* = Z
    set k = k + 1
return P
```

Le calcul du score de fitness G(X,Y) est coûteux en termes de calcul. C'est pour ça que Faulkner et al. ne calculent le fitness qu'après l'ajout de plusieurs éléments et reviennent en arrière si le score se dégrade.

PackingIterative produit différents plans de prélèvement pour des valeurs variables de α afin de trouver la meilleure valeur pour α, α commence à une certaine valeur et après chaque itération, l'algorithme vérifie si elle doit être augmenté ou diminué. Après quelques itérations, il renvoie le meilleur plan d'emballage trouvé. PackingIterative est décrit dans l'Algorithme 6 [39] .

Algorithme 6 :

Iterative Packing Routine PACKITERATIVE (Π,c,δ,q)

obtain P_l by PACK (Π,$c-\delta$) and compute $Z_l = Z(\Pi, P_l)$
obtain P_m by PACK (Π,c) and compute $Z_m = Z(\Pi, P_m)$
obtain P_r by PACK (Π,$c+\delta$) and compute $Z_r = Z(\Pi, P_r)$
set the best packing plan $P^* = \emptyset$
set the counter $i = 1$
while $i \leq q$ **do**
  if $(Z_l > Z_m)$ and $(Z_r > Z_m)$ **then**
    if $(Z_l > Z_r)$ **then**
      set $Z_m = Z_l$, $c = c - \delta$ and $P^* = P_l$
    else
      set $Z_m = Z_r$, $c = c + \delta$ and $P^* = P_r$
  else if $(Z_l > Z_m)$ **then**
    set $Z_m = Z_l$, $c = c - \delta$ and $P^* = P_l$
  else if $(Z_r > Z_m)$ **then**
    set $Z_m = Z_r$, $c = c + \delta$ and $P^* = P_r$
  $\delta = \delta/2$;
  obtain P_l by PACK (Π,$c-\delta$) and compute $Z_l = Z(\Pi, P_l)$
  obtain P_r by PACK (Π,$c+\delta$) and compute $Z_r = Z(\Pi, P_r)$
  set $i = i + 1$
  if $(Z_l - Z_m < \epsilon)$ and $(Z_r - Z_m < \epsilon)$ **then** break
return P^*

Dans leur article, Faulkner et al[33]. proposent 11 variantes différentes du même algorithme appelées S1-S5 et C1-C6.

Tous les algorithmes commencent par un tour X trouvé par l'heuristique Chained Lin-Kernighan. Ensuite, la tournée est maintenue fixe et PackingIterative est exécuté sur le plan d'emballage. Selon la variante de l'algorithme, plusieurs recherches locales sont effectuées ou l'algorithme est recommencé. Les recherches locales incluses dans l'article sont: BitFlip, Insertion et (1+1)-EA.

Ils proposent également un algorithme basé sur la procédure de programmation mixte en nombres entiers de Polyakovskiy et Neumann [31]. Là encore, CLK est utilisé pour obtenir une tournée après que le plan d'emballage optimal est approché par la programmation mixte en nombres entiers.

La variante S5 est la plus performante sur une grande variété d'instances et elle est utilisée comme comparaison pour de nombreux algorithmes [36, 37, 33]. Il s'agit d'un algorithme multi-start qui répète S1. S1 exécute CLK et PackingIterative sans autre recherche locale.

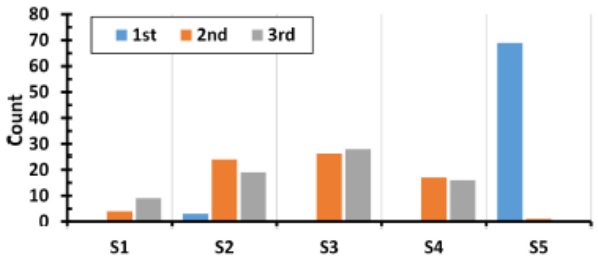

Figure 7 : Résultats de S1-S5 : classement sur les 72 instances, extrait de Faulkner et al. [33].

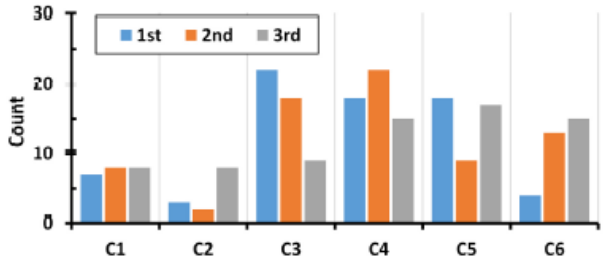

Figure 8 : Résultats de C1-C6 : classement sur les 72 instances, extrait de Faulkner et al. [33].

3.3.5 La combinaison 2-OPT et le recuit simulé (SA) CS2SA

Yafrani et Ahiod [36] ont présenté un couple d'algorithmes heuristiques. La première approche est une méthode heuristique basée sur une solution unique, appelée CS2SA, qui combine 2-OPT et le recuit simulé (SA) pour résoudre le TTP.

La deuxième méthode est un algorithme mimétique nommé MA2B qui utilise le 2-OPT et le bit-flip comme opérateur basé sur MPX. Ces deux algorithmes sont plus performants que S5 et MATLS pour les grandes instances.

Yafrani et Ahiod [16] ont introduit deux algorithmes basés sur le 2-OPT et le recuit simulé nommés CS2SA* et CS2SAR pour résoudre le TTP en améliorant la méthode CS2SA [36] précédemment mise en œuvre. CS2SA* est une implémentation directe de CS2SA dans laquelle une stratégie de réglage avancée est utilisée pour optimiser les paramètres. Dans CS2SA-R, des redémarrages aléatoires sont utilisés s'il n'y a pas d'amélioration dans l'état de retour de la meilleure solution trouvée.

3.4 Algorithmes évolutifs

3.4.1 Algorithmes basés sur la population (CC, MA et MATLS)

Mei, Li et Yao [15] ont proposé deux algorithmes basés sur la population. Un algorithme de coévolution coopérative (CC) similaire à CoSolver mais avec une population et un algorithme mimétique (MA).

CC conserve deux populations contenant des solutions partielles aux sous-problèmes, une population avec des plans d'emballage et une autre avec des tournées. Il fait progresser itérativement le plan d'empaquetage avec BitFlip, Exchange et le croisement une-point OPX.

Il fait progresser itérativement la tournée avec 2-opt et le croisement d'ordre OX. Afin d'effectuer une évaluation de l'adéquation des solutions partielles, un collaborateur de l'autre population est nécessaire.

Seuls les k meilleurs membres d'une population sont considérés comme des collaborateurs. L'évaluation de l'adéquation d'une solution partielle est effectuée avec tous les k meilleurs membres de l'autre population et la meilleure évaluation de ces k membres est utilisée. MA est un algorithme mimétique qui initialise la population de façon aléatoire. Après ce qui suit se produit pour chaque génération :

1. Deux membres de la population sont choisis au hasard.

$$P1 = (\Pi p1, Yp1) \; and \; P2 = (\Pi p2, Yp2)$$

2. Une descendance C est créée en effectuant un crossover d'ordre sur la tournée et un crossover à deux points sur le plan d'emballage.

$$C = (OX(\Pi p1, \Pi p2), OP\,X(Yp1, Yp2))$$

3. Avec une certaine probabilité, une recherche locale est effectuée sur C. La recherche locale consiste en une procédure de montée en puissance de la meilleure amélioration qui apporte consécutivement une amélioration avec un passage de 2-opt, puis de BitFlip et enfin d'Exchange.

4. Après qu'un nombre suffisant de descendants ait été créé, une sélection par troncature est effectuée.

Ils ont constaté que MA surpasse CC et affirment que cela est dû au fait que CC optimise les sous-problèmes séparément et que MA résout le problème dans son ensemble. De plus, ils affirment que cela illustre l'importance de considérer l'interdépendance des sous-problèmes.

Dans un autre article de Mei, Li et Yao [44], proposent une version efficace en termes de calcul de leur algorithme mimétique avec une recherche locale en deux étapes (MATLS).Cela rappelle en partie le S5. D'abord, une population est créée et initialisée. La tournée est initialisée avec CLK ou une heuristique d'arbre d'étendue minimale.

Le plan d'empaquetage est initialisé par une heuristique d'empaquetage gloutonne similaire à SH. La différence est l'évaluation du fitness. Dans MATLS, elle est approximée par le pire possible et par le gain attendu. Le gain le plus défavorable est l'augmentation nette d'un objet.

Où l'on suppose que les éléments sont ramassés dans le pire des cas possibles. Le pire cas est celui où tous les éléments du sac à dos sont ramassés avant la ville de l'élément actuel. Le gain attendu est l'hypothèse selon laquelle tous les articles précédemment ramassés sont ramassés tout au long de la tournée avec une distribution égale.

Dans chaque génération, OX est effectué sur les tours, après quel 2-opt est effectué avec un voisinage réduit par la triangulation de Delaunay. On utilise l'évaluation de la fitness de TSP qui prend un temps constant au lieu de $O(n + m)$.

Pour chaque nouveau membre de la population le plan d'emballage est créé par la même heuristique d'emballage (greedy packing heuristic) il compare leur algorithme à (1+1)-EA et à RLS. Ils ont trouvé qu'ils avaient de meilleurs résultats.

3.4.2 L'algorithme mimétique MA2B

Dans le même article où El Yafrani et Ahiod [36] ont présenté CS2SA, ils ont également proposé un algorithme mimétique MA2B. La population est initialisée de la manière suivante : Tout d'abord, la tournée est créée par CLK, puis un plan d'empaquetage est généré sur la base d'une heuristique de Mei, Li et Yao [44], puis une recherche locale restreinte est appliquée avec un maximum de 50 passages. Ils utilisent la même recherche locale qu'avec CS2SA.

MA2B utilise le double pont comme opérateur de mutation et MPX comme opérateur de croisement. Le raisonnement qui sous-tend l'utilisation de MPX est qu'il s'agit d'un croisement perturbateur. Ils soutiennent que cela est préférable dans les algorithmes mimétiques. Les tests préliminaires ont confirmé que MPX était plus performant qu'ERX.

Aucun croisement n'a été utilisé directement sur le plan d'emballage, le statut de la progéniture a été déduit de MPX et du plan d'emballage. Si la ville contenant l'objet est hérité du premier parent, l'état de l'élément est aussi hérité du premier parent.

Ils ont également comparé leurs deux algorithmes avec S5 et MATLS et ont conclu que les deux algorithmes présentaient des performances compétitives. Aucun algorithme n'a dominé les autres sur toutes les instances. Les algorithmes mimétiques sont plus performants sur les petites instances et S5 et CS2SA sur les grandes instances .

Ils ont également constaté que pour les grandes instances, MATLS passe la majorité du temps sur son initialisation. De plus, ils déclarent qu'il est un peu inhabituel que les algorithme gloutons soient très performants pour de nombreuses instances et peut-être qu'une étude paysagère pourrait donner un aperçu de la raison pour laquelle c'est le cas.

3.4.3 Optimisation par colonies des fourmis

Wagner [39] a proposé un algorithme qui trouve une solution TTP à l'aide d'une intelligence en essaim. Un algorithme d'optimisation de colonies de fourmis MAX-MIN appelé MMAS.

Les fourmis construisent une tournée en choisissant la ville suivante en fonction d'une probabilité qui est proportionnelle à la phéromone associée aux bords entre les villes. Après qu'une tournée a été trouvée, une recherche locale spécifique à la PST est effectuée, 2-opt, 2.5-opt ou 3-opt. Comme dans MATLS, il s'agit d'une recherche locale où l'évaluation de l'aptitude est effectuée sans tenir compte de la partie KP du TTP.

Seulement les distances entre les villes sont prises en compte. Après l'établissement d'une tournée, un plan d'empaquetage est construit en utilisant PackIterative de Faulkner et al. [33].

Comme étape optionnelle, une recherche locale est effectuée en tenant compte de la fonction de fitness totale : (1+1)-EA, une passe d'Insertion et une passe de BitFlip. Ensuite,

les pistes de phéromones sont mises à jour en fonction de la performance des fourmis. Par étapes4 cela ressemblerait à ceci :

1. Construire une tournée en utilisant des fourmis.

2. Effectuer une recherche locale spécifique à la TSP sur les tours : 2-opt, 2.5-opt ou 3-opt.

3. Pour chaque tournée, créer un plan d'emballage avec PackIterative.

4. Effectuer une recherche locale spécifique à TTP sur les tournées : (1+1)-EA, une passe d'Insertion et une passe de BitFlip.

5. Mise à jour de la piste des phéromones.

Wagner montre que son algorithme d'optimisation par colonies de fourmis choisit des tournées plus longues que les algorithmes approximatifs S1-5 et C1-6. le tours est plus longs mais avec de meilleur résultats. MMAS surpasse la plupart des approches actuelles sur des instances contenant jusqu'à 250 villes et 2000 objets [39, 45].

3.4.4 Les hyper heuristiques

Une approche intéressante pour résoudre le TTP consiste à utiliser une hyper-heuristique. El Yafrani et al [37] l'ont fait, ainsi qu'un article similaire de Martins et al [42]. Une population d'encodages est conservée où chaque individu de la population représente une combinaison de plusieurs heuristiques de bas niveau (LLH).

Les LLH sont toutes appliquées successivement après la création d'une tournée initiale via CLK et d'un plan d'emballage via PackIterative. Les LLH sont soit des recherches locales, soit des opérateurs perturbateurs.

Les différentes recherches locales sont : BitFlip, le recuit simulé [36] sur le plan d'emballage et 2-opt. Les opérateurs disruptifs sont : l'échange aléatoire de 2-opt, le double bridge move et le bit flips sur un pourcentage d'éléments.

El Yafrani et al. [37] utilisent la programmation génétique et Martins et al. [42] échantillonnent de nouvelles solutions en utilisant un algorithme d'estimation de la distribution. Martins et al. Approximent l'évaluation du fitness avec un réseau de fonctions à base radiale. Leurs algorithmes sont compétitifs avec S5, MMAS et MA2B.

3.4.5 Algorithme de colonies d'abeilles artificielles ABC

Alharbi [51] a introduit un algorithme ABC (Artificial Bee Colony) modifié pour résoudre le TTP d'une manière interdépendante. Il est efficace par rapport aux approches de l'état de l'art, principalement pour les instances TTP de petite et moyenne taille.

3.4.6 D'autres algorithmes mimétiques

Dans [41, 38, 40], des algorithmes mimétiques assez similaires ont été proposés avec comme différence notable l'opérateur de croisement: croisement basé sur l'ordre [40] et croisement partiellement mappé [41, 38].

Louren¸co, Pereira et Costa [41] ont compté le nombre d'arêtes qui sont différentes entre la tournée optimale de TSP et la tournée de la meilleure solution de TTP trouvée. Ils ont trouvé que dans tous les cas sauf un, la différence dépassait 50% des bords meilleure solution TTP trouvée.

3.5 Exact Approches

Wu et al. [46] ont proposé quelques approches exactes, l'une avec programmation dynamique, l'autre avec recherche par branch and bound et la dernière avec programmation par contraintes.

Étant donné la dureté du problème, la complexité temporelle de ces algorithmes croît de manière exponentielle avec le nombre de villes. Sur de petits échantillons ($n \leq 20$), la solution optimale peut donner un aperçu de la performance des algorithmes.

Ils ont trouvé que PackingIterative de S5 pourrait être encore un peu amélioré en utilisant une approche d'emballage exacte et ils ont trouvé que MA2B a une performance exceptionnelle sur toutes les instances avec une fiabilité élevée.

3.6 Approches pour MTTP

En plus, quelques recherches ont également été effectuées impliquant une objectivité multiple dans le TTP par Blank et al. [47], Yafrani et al.[48] et Wu et al. [50,48] et Wu et al. [50]. Chand et al. ont été les premiers à introduire le problème du voyageur voleur multiple

(MTTP) [49]. MTTP permet à plusieurs voleurs de se déplacer dans différentes villes dans le but de maximiser le profit total du groupe.

3.7 Les méthodes de recherche locale

Une recherche locale est une méthode permettant d'améliorer de manière itérative une solution en effectuant des modifications locales. Ces modifications locales sont effectuées à l'aide d'un opérateur. La plupart du temps, l'opérateur peut être appliqué de plusieurs façons. Chaque application d'un opérateur crée une solution voisine.

Le passage itératif à une meilleure solution voisine est appelé un algorithme d'ascension de colline. Si une solution n'a aucun voisin ayant un meilleur score d'évaluation, alors la solution est un optimum local par rapport à l'opérateur.

Pour le TTP, les recherches locales sont effectuées sur le plan d'empaquetage ou sur la tournée. Le plan d'empaquetage est représenté par une chaîne de bits et la tournée par une permutation.

3.7.1 La méthode 2-opt

2-opt est une procédure de recherche locale créée pour TSP mais peut être appliquée à n'importe quel type de représentation de permutation. Le 2-opt-swap inverse un segment de la permutation En termes de tour, elle supprime et crée deux arêtes .Puisque chaque segment peut être utilisé pour une application valide de 2-opt-swap, cet opérateur a un total de $O(n^2)$ solutions voisines.

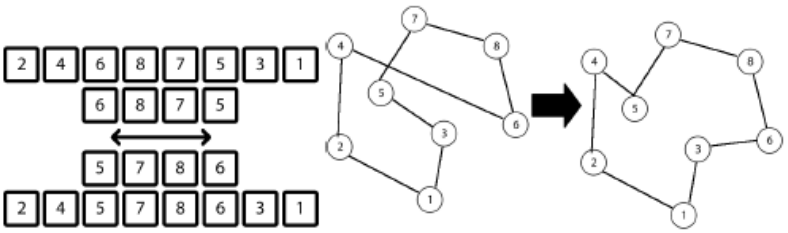

Figure 9 : 2-opt. Représentation de permutation (à gauche) et le tour correspondant (à droite) , extrait de R.H.Wuijts [80].

3.7.1.1 Amélioration pour 2-opt

La complexité de 2-Opt peut être améliorée de deux façons :

- Pour chaque ville, il est possible de pré-calculer la valeur totale et le poids total ramassé dans cette ville [44]. Cela réduit la complexité de l'évaluation du fitness de $O(n + m)$ à $O(n)$.

- Avec la réduction du voisinage, vous pouvez réduire le voisinage de $O(n^2)$ à $O(n)$.

Il en résulte une complexité totale de O(n ^{2}). Une accélération supplémentaire peut être obtenue par le fait qu'un seul échange 2-Opt n'affecte qu'une partie de la tournée.

En 2-Opt, un segment de la tournée est inversé. La partie qui précède le segment et celle qui le suit restent la même. Par conséquent, lors de l'évaluation de la forme physique, vous ne devez prendre en compte que le segment inversé. En moyenne, cela réduit le temps de moitié.

3.7.2 L'insertion

L'insertion, également appelée 2,5-opt, est étroitement liée à 2-opt. Ici, un élément de la permutation est choisi et réinséré quelque part dans la permutation.

Il y a O(n) candidats à l'insertion et O(n) endroits à insérer, soit encore O(n^2)solutions voisines.

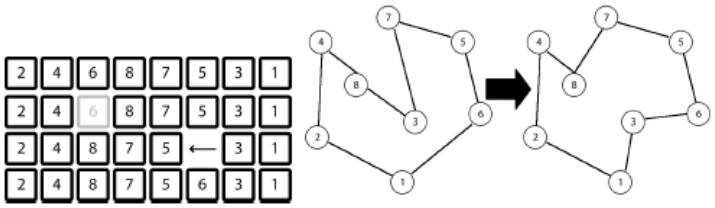

Figure 10 : Insertion. Représentation de permutation (à gauche) et tour correspondant (à droite) , extrait de R.H.Wuijts [80].

3.7.2.1 Amélioration pour Insertion

Comme pour 2-Opt, l'évaluation de la fitness d'Insertion peut aussi être réduite à O(n) en pré-calculant le poids et le bénéfice des éléments dans les villes. Mais nous pouvons

réduire encore plus la complexité de l'évaluation du fitness en utilisant une évaluation incrémentale. Ainsi, l'ensemble du voisinage est considéré en seulement O(n) temps.

La taille du voisinage d'Insertion provient du fait que chaque ville O(n) peut être insérée dans chaque position de la tournée O(n). Une implémentation naïve a une complexité temporelle de $O(n^2).O(n) = O(n^3)$. Mais nous pouvons faire beaucoup mieux. L'aptitude de chaque insertion possible peut être calculée efficacement de la manière suivante :

1. Placez la ville sélectionnée à la fin de la visite.

2. Calculer le fitness de cette permutation O(n)

3. Echanger la ville avec son prédécesseur et réévaluer la tournée O(1)

4. Répéter l'étape 3 jusqu'à ce que tous les emplacements d'insertion possibles soient évalués O(n)

L'étape 3 est O(1) puisque seulement une petite partie de la tournée change. L'échange de deux positions successives ne modifie que le coût de 3 arêtes.

$f(X') = f(X)$ - le cout d'ancien segment + le cout de nouveau segment. le résultat est un temps de complexité $O(n^2)$

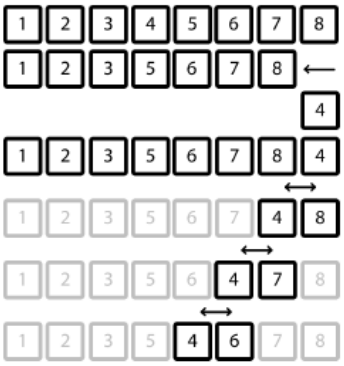

Figure 11 : Utilisation de l'évaluation incrémentale pour l'insertion. Après avoir placé la ville 4 à la fin de l'évaluation de la tournée, la tournée est O(n). Mais pour toutes les autres évaluations, O(1) temps (le segment non-gris) est nécessaire pour calculer la nouvelle valeur, extrait de R.H.Wuijts [80].

3.7.3 BitFlip

Comme son nom le suggère probablement, BitFlip est une recherche locale qui retourne un bit d'une chaîne de bits. En termes de plan d'emballage, il s'agit d'insérer ou de retirer un élément du sac à dos. Il y a O(n) bits possibles à inverser.

3.7.4 Exchange

L'échange est une recherche locale qui prend deux bits avec des valeurs différentes et qui permute leur état. En termes de plan d'empaquetage, il s'agit de retirer et d'insérer consécutivement un élément. L'insertion d'un élément. Il y a O(n 2) combinaisons possibles. Cette procédure de recherche locale est plus forte que celle de BitFlip car elle peut d'abord "visiter" une solution invalide ou pire en retirant ou en insérant un élément.

3.7.4.1 Amélioration pour exchange

Pour améliorer la complexité d'Exchange, on peut réduire le voisinage. Au lieu d'essayer O(m^2) échanges possibles, on n'essaie que des paires prometteuses. Les paires sont créées de la manière suivante :

- Pour chaque objet i qui n'est pas inclus, créer un ensemble S_i contenant les objets j qui sont inclus dans le plan d'emballage et dont le poids est égal ou supérieur à i.

$$Si = \{j | yj = 1 \wedge Pj \geq Pi\}$$

-A partir de S_i, trouver l'élément j ayant la perte minimale (ou le gain maximal) de fitness lorsque l'élément n'est plus inclus. i et j forment une paire et sont considérés pour l'échange.

Puisqu'il n'y a que m items, seulement O(m) échanges sont essayés.

3.7.5 La complexité des méthodes de recherche locale

la complexité avant et après la réduction des méthodes de recherche local: 2-opt, Insertion, BitFlip et exchange est représenté dans le tableau suivant:

	Taille de voisinage	Complexité	Voisinage réduite	Complexité réduite
2-opt	$O(n^2)$	$O(n^2(n+m))$	$O(n)$	$O(n^2)$
Insertion	$O(n^2)$	$O(n^2(n+m))$	Pas de réduction	$O(n^2)$
BitFlip	$O(m)$	$O(m(n+m))$	Pas de réduction	Pas de réduction

Exchange	$O(m^2)$	$O(m^2(n+m))$	$O(m)$	$O(m(n+m))$

Tableau 1 : la complexité avant et après la réduction des méthodes de recherche local

3.8 Conclusion

Dans ce chapitre Nous avons présenté une revue de littérature, des méthodes de résolution utilisé dans des recherches sur le TTP et aussi des méthodes de recherche local utile pour ce problème, pour passé dans le prochain chapitre a notre proposition pour la résolution de TTP: ALNS et SA.

CHAPITRE 4 Méthodes proposées.

4.1 Introduction

Après la description du problème de voyageur voleur (TTP) dans le chapitre 2, et le survol effectué sur les différentes méthodes proposées depuis les premiers travaux réalisés. Nous nous intéressons dans ce chapitre sur la présentation de notre approche de résolutions choisie pour la résolution de problème.

4.2 Méthode de recuit simulé (SA)

L'algorithme de recuit simulé (Simulated Annealing - SA) est une méthode métaheuristique itérative recommandée pour la résolution de problèmes d'optimisation non linéaires [53]. La méthode SA consiste essentiellement à simule les mouvements atomiques des solides dans le processus de recuit.

Le recuit est un processus thermique utilisé pour atteindre de faibles niveaux d'énergie dans une substance. Dans ce processus, la température est d'abord élevée à la température à laquelle un solide se fond, puis le matériau se refroidi.

Si la vitesse de refroidissement dans le processus de recuit est suffisante pour permettre aux atomes de se déplacer, ceux-ci tendent à atteindre les niveaux d'énergie interne les plus bas et les niveaux d'énergie interne les plus élevés.

Par le biais des mouvements aléatoires [54]. Grâce à ces fluctuations aléatoires, le matériau sera autorisé à s'échapper du minimum d'énergie locale et sera reconstruit et refroidi

par une partie de l'énergie résiduelle qui caractérise la contrainte thermique dans le matériau [55].

Metropolis et al. [56] ont simulé mathématiquement le processus de recuit en utilisant les mécanismes de probabilité de Monte Carlo pour trouver l'état d'énergie la plus basse requise pour les solutions optimales [54].

Alors que l'algorithme de Monte Carlo standard ne traite que les mouvements vers un état d'énergie, la procédure de Metropolis a rendu possible le traitement d'un état d'énergie plus élevé [57]. Kirkpatrick et al. [58] et Cerny [59] ont transformé cette idée avec une méthode heuristique pour traiter les problèmes d'optimisation combinatoire.

Par analogie, la température physique est exprimée par un paramètre de température traité par l'algorithme, la fonction objective est l'énergie interne de matériau, et la solution est une configuration atomique [60]. L'algorithme SA peut être expliqué en six étapes suivantes :

Étape 1 - Initialement, la valeur d'itération i et la température initiale T0 sont identifiées. L'algorithme génère un point de départ aléatoire, et la valeur Xi est calculée comme la fonction objective qui correspond au point où i= 0.

Même si la solution obtenue initialement est défavorable avec une valeur de T0, il y a une possibilité d'une solution plus favorable dans le futur. Par conséquent, la sélection de la valeur T0 est un paramètre important qui affecte la performance de l'algorithme SA.

Si T0 est choisi trop faible au début, le SA ne peut pas effectuer la recherche de manière efficace et il peut être impossible d'aller a une solution optimale locale [54].

Étape 2 - Le critère de fin de cycle est déterminé, critère d'arrêt est vérifié à chaque itération et la valeur de température est fixée à la plus haute valeur trouvée.

En utilisant un temps de fin fixe [61], détermination d'un certain nombre d'itérations [62], l'arrêt à une certaine température minimale [63], et l'arrêt lorsque la valeur d'acceptation totale tombe sous un certain niveau [64]. Sont des méthodes utilisées dans la littérature comme critère d'arrêt de l'algorithme.

Étape 3 - Une nouvelle solution pour les valeurs de voisinage générées de manière aléatoire est trouvée conformément à l’équation.

$$X_{i+1} = X_i + VxR$$

Ici, R est une valeur aléatoire dans la plage de (-1,1) et V est une matrice diagonale qui définit la variation maximale autorisée pour chaque variable de contrôle.

Étape 4 - on détermine si la solution nouvellement trouvée est meilleure que la solution précédente. Dans cette étape, la règle de Metropolis est utilisée pour déterminer si une solution nécessitant plus de coûts sera considérée comme une solution existante.

La déclaration $X_{i+1} \leq X_i$ indique que la solution voisine est meilleure que la solution actuelle pour résoudre les problèmes de minimisation. Dans ce cas, la solution voisine est acceptée comme Xnew, et la nouvelle solution existante comme X_{i+1}.

Cependant, dans le cas où $X_{i+1} > X_i$, la solution voisine est supposée être pire que la solution actuelle. un mécanisme de probabilité est utilisé pour déterminer si cette solution voisine sera considérée comme la nouvelle solution existante. La possibilité d'accepter la nouvelle valeur de solution Xnew comme la valeur de solution X_{i+1} par rapport à celle de X_i est calculée comme suit [57].

$$X_{i+1} = \begin{cases} X_{new} & Si\ exp\left(-\frac{\Delta f}{T}\right) > r \\ Xi & sinon \end{cases}$$

Il représente une valeur aléatoire uniforme dans l'intervalle où $\Delta f = f(X_{i+1}) - f(X_i)$ et r [0, 1]. Les actes d'acceptation ou de rejet permettent à SA d'échapper à l'optimum local.

Étape 5 - La valeur de la température est mise à jour. En accord avec les lois de la physique , pendant le recuit physique, la vitesse de refroidissement diminue à mesure que la température diminue. Par conséquent, la méthode de refroidissement est nécessaire pour refroidir le système beaucoup plus rapidement au début [54].

Cela signifie que la baisse de température n'est pas constante à chaque étape, mais progressera peu à peu à chaque étape du programme. Dans la littérature, il existe des méthodes sur le changement de température présentées en utilisant les méthodes géométrique [65], logarithmique [66], Lundy-Mees et variantes [67], quadratique [68] et arithmétique [69]. La règle la plus courante de réduction de la température est définie comme suit

$$T_{k+1} = \alpha T_k$$

Où α (coefficient de température) est un paramètre de taux de réduction défini par l'utilisateur. α est une constante positive, généralement admise dans l'intervalle de 0,800-0,999 [57].

Étape 6 - L'algorithme est relancé avec une nouvelle valeur de température.

L'algorithme 7 [78], exécute le processus de recuit simulé.

Algorithme 7 : Méthode de recuit simulé (SA)

Input: an INITIAL SOLUTION X_0, CONTROL PARAMETERS: T_{start}, T_{end} and cooling factor α
Output: The Best Solution X^* found during search

```
Calculate initial FITNESS FUNCTION for X0 → f(X0) ;
Set best solution X* ← X0 ;
i = 0;
T = Tstart;
while STOPPING CRITERION is not met do
    calculate a solution Xi+1 → f(Xi+1) in the neighborhood of Xi
    according to EXPLORATION CRITERION;
    if Xi+1 meets ACCEPTANCE CRITERION then
        To MAXIMIZE FITNESS Δf = f(Xi+1) - f(Xi) ;
        If Δf ≥ 0 then
            Set best solution X* ← Xi+1 ;
        else METROPOLIS CONDITION
            If exp(-Δf / T) ≥ Uniform(0,1) then
                Set best solution X* ← Xi+1 ;
            end
        end
    end
    Decrease the temperature according to COOLING SCHEME:
    T = Tstart x α ;
    i=i+1;
end
return X* ;
```

En général, le recuit simulé utilise des techniques probabilistes pour trouver la valeur optimale globale de la fonction objective, qui comprend les paramètres du modèle [55].

4.2.1 Le voisinage d'une solution TTP

Comme nous l'avons dit précédemment, a chaque itération de SA un voisin est choisi aléatoirement, ce voisin est une permutation entre deux ou plusieurs villes dans le chemin de voleur (ou/et) un changement dans objets de sac à dos choisies.

D’autres méthodes pour le choix de voisins plus adaptés au TTP ont été envisagées autres que la sélection de voisin aléatoire. Dans chaque itération, une sélection aléatoire de voisins appropriés est choisie à base d’un choix équiprobable.

4.2.1.1 Le voisinage aléatoire

Un voisin est une permutation entre 2 villes choisie aléatoirement ou/et un changement dans le choix des objets ramassés dans le sac a dos, soit l'ajout d'un ou plusieurs nouveaux objets si la contrainte de capacité est respecté, soit l'enlèvement d'un ou plusieurs objets du sac.

4.2.1.2 Autres voisinages utilisés

Deux méthodes décrites dans le chapitre 3 ont été utilisées : le voisinage **2-opt** et le mouvement **BitFlip** (algorithme 9 [79]**),** avec des pondérations faibles par apport à la méthode de voisinage aléatoire, à cause de :

- La complexité temporelle, ces méthodes prennent autant de temps d’exécution.
- Le risque de s’attraper dans des optimums locaux.

Une méthode similaire à la méthode de recherche locale 2-opt, notée 2-opt2 a été proposée, cette dernière prend en considération la contrainte de sac a dos, l’équation suivante permet la classification des villes à choisir pour une permutation possible.

$$SV/(SP + D)$$

Où

SV: la somme des valeurs des objets du sac dans la ville.

SP: la somme des poids des objets du sac dans la ville.

D: la distance de la ville considérée jusqu’au la dernière ville dans le chemin.

Algorithme 8: 2-opt Algorithme

```
    Input: n targets and its xy locations
 1: evaluate the distance matrix
 2: define a tour T initialised randomly
 3: for i=1 to n-2 do
 4:     for j=i+2 to n do
 5:         evaluate d1 = total length of 2 edges.
 6:         evaluate d2 = total length of the edges
            when the targets are swapped.
 7:         if d1 > d2 then
 8:             swap indices of targets in tour T.
 9:         end if
10:     end for
11: end for
```

Algorithme 9 : Bit-flip (x, z^*)

```
Set G** = G*(x, z*)
z** = z*
for k = 1 to m do
    r = a random number between 1 to m
    if r ∉ z* then
        add the item r, z* = z** ∪ r
    else
        remove the item r, z* = z** \ r
    end if
    compute the objective value G*(x, z*)
    if G* > G** then
        set G** = G*
        set z** = z*
    end if
end for
return z**
```

Algorithme 10: 2-opt2 Algorithme

```
    Input: n targets, its distance matrix and the
    packing plan
 1: for i=1 to n-2 do
 2:     for j=i+2 to n do
 3:         evaluate b1 = SV/(SP+D) of 2edges.
 4:         evaluate d2 = SV/(SP+D) of the edges
 5:         when the targets are swapped.
 6:         if b2 > b1 then
                swap indices of targets in tour T.
 7:         end if
 8:     end for
 9: end for
```

4.3 La recherche à voisinage large

La deuxième méta-heuristique proposée est basée principalement sur la méthode de recherche à voisinage large (large neighborhood search - LNS) introduite par Shaw [70]. L'heuristique LNS a été appliquée en premier pour le VRPTW, [70], [71] et Bent et Van Hentenryck [72].

Récemment, l'heuristique a également été appliquée au PDPTW (Bent et Van Hentenryck [73]). L'heuristique LNS est similaire à l'heuristique "ruin and recreat" proposée par Schrimpf et al [74].

Le pseudo-code d'une heuristique LNS traitant un cas de minimisation est présenté dans l'Algorithme 10 [75]. Le pseudo-code suppose qu'une solution initiale s a déjà été trouvée, par exemple par une heuristique de construction simple.

Le deuxième paramètre (q) détermine l'étendue de la recherche. Les lignes 5 et 6 de l'algorithme constituent la partie intéressante de l'heuristique. Dans la ligne 5 un certain nombre de demandes sont retirées de la solution actuelle s', cette dernière est reconstruite à base d'une procédure de réparation (ligne 6).

Algorithme 11 : LNS Heuristic

```
1  Function LNS(s ∈ {solutions}, q ∈ ℕ )
2    solution s_best=s;
3    repeat
4      s' = s;
5      remove q requests from s'
6      reinsert removed requests into s';
7      if (f(s') < f(s_best)) then
8        s_best = s';
9      if accept(s', s) then
10       s = s';
11   until stop-criterion met
12 return s_best;
```

4.4 La méthode de recherche à voisinage large adapté pour le TTP

La méthode de recherche à voisinage large et adapté (Adaptatitve Large neighborhood search -ALNS) est une heuristique récemment proposée par Ropke et al. [75]. Plusieurs opérateurs de destructions et de réparations dans le même processus de recherche sont utilisés.

A chaque itération, l'heuristique détruit une partie de la solution courante et la répare d'une manière différente pour réaliser une nouvelle solution, cette même approche est utilisée dans [76] pour résoudre le PDPT, le principe fondamental de l'ALNS est de détruire et réparer une solution de manière itérative afin de l'améliorer.

Après une initialisation, l'heuristique commence à construire une solution en choisissant deux méthodes de destruction et de réparation dans la liste des méthodes disponibles en utilisant la méthode de sélection par roulette. Ce choix se fait en fonction de la valeur de performance pondérée de chaque méthode de destruction et de réparation.

Deux modifications ont été apportées à l'algorithme LNS pour avoir un algorithme ALNS, Tout d'abord, la mise à jour de la valeur du score de chaque méthode de destruction et de réparation, étape importante pour définir laquelle des méthodes seras utilisées dans l'étape prochaine, et après, le critère d'acceptation de recuit simulé utilisé pour permettre aux mauvaises solutions d'être accepter pour s'échapper des optimums locaux. l'algorithme 11 montre les différentes étapes de la procédure.

Algorithme 12: ALNS heuristic

```
1  Input : control parameters of SA: T_start ,T_end and cooling factor α
2  output: best
3  S← Create initial solution
4  T= T_start
5  Repeat
6          Select D_i R_i using Roulette selection and p_i^D , p_i^R
7          s'← Repair(D_i , Destroy (R_i ,s))
8          Δ = G(s') - G(s)
9          if Δ ≥ 0 then
10                 s← s'
11         else
12                 if exp(Δ/T) ≥ Uniform(0,1) then
13                         s← s'
14                 end
15         end
16         If G(s')>G(best) then
17                 best← s'
18         end
           Update p_i^D , p_i^R
   until stop criterion met
```

4.4.1 Les méthode de destructions

Trois méthodes sont utilisées pour la destruction, deux méthodes pour la suppression des villes et une pour la destruction des objets:

- la méthode ***randomRemovalCities*** qui supprime une ville aléatoirement.

- la méthode ***worstRemovalCities*** qui supprime la ville la plus loin de ces voisins.

- la méthode ***randomRemovalItems*** qui supprime un objet aléatoirement.

4.4.2 Les méthode de réparation :

Trois méthodes sont utilisées pour réparer les solutions détruites, deux méthodes pour insérer les villes et une pour les objets:

- la méthode ***randomInsertCities*** capable d'insérer une ville aléatoirement.

- la méthode ***worstInsertCities*** permettant d'insérer une ville entre les plus proches voisins.

- la méthode ***randomInsertItems*** capable d'insérer un objet aléatoirement.

4.5 Conclusion

Dans ce chapitre nous avons présenté notre méthodes proposé pour la résolution de problème de voyageur voleur : (Simulated Annealing - SA) et (Adaptative large neighborhood search -ALNS).

Dans le prochain chapitre nous présentons l'application développée, les instances de référence utilisées pour les testes d'algorithmes ainsi les résultats obtenus.

CHAPITRE 5 Implémentation.

5.1 Introduction:

Ce chapitre, est consacré à la présentation de l'implémentation d'algorithmes proposés dans la partie conception, le langage de programmation choisi est le **Java** sous comme un l'environnement de développement **Netbeans.** Des instances de référence propre au problème étudié ont été testés, dont les résultats obtenus sont montrés dans les tableaux à la fin de ce chapitre.

5.2 Benchmark:

Pour tester les algorithmes proposés à la résolution de problème TTP, une série d'instances de référence a été utilisée. Proposé en premier par Polyakovskiy et al. [34], La des benchmarks est construite sur des instances de la bibliothèque des données de voyageur de commerce, TSPLIB [11].

En effet, une instance de voyageur de commerce de la bibliothèque TSPLIB est représentée à l'aide d'un graphe composé d'un ensemble de villes interconnectées par des distances euclidiennes. La transformation d'une instance TSP en une instance TTP s'effectuée à travers l'affectation des objets eux villes, la capacité du sac à dos et le taux de location devront être définie.

5.2.1 Composante de sac à dos:

La corrélation entre le poids et la valeur des éléments peut décider de la complexité d'un problème de sac à dos [28]. Polyakovskiy et al. [34], avait choisis d'avoir trois types de

KP différents : non corrélé, non corrélé avec des poids similaires et fortement corrélé. Le fait que les rapports entre les poids et les profits sont proches les uns des autres pour les objets fortement corrélés rend la résolution difficile.

Cependant, ce qui est difficile à résoudre dans le sac à dos ne l'est pas nécessairement difficile pour le TTP. Dans le TTP, ce rapport de pondération des profits a moins d'importance, car la contribution dans la fonction objectif ne dépend pas uniquement du profit mais aussi de la distance entre les villes.

Nous pouvons trier les objets en fonction d'un rapport, qui dépend du profit, du poids et de la distance. Ceci dans toutes les heuristiques traitantes le TTP sous la forme d'une heuristique d'empaquetage glouton.

Il serait intéressant de disposer également d'instances où les éléments sont générés pour avoir une forte corrélation entre les profits et les poids. Ces instances seront beaucoup plus difficiles à résoudre car l'ordre contient moins d'informations que dans le cas des instances fortement corrélées des problèmes standards de sac à dos.

5.2.2 Distribution des objets et contrainte de capacité:

Pour chaque ville d'une instance de problème est associée un nombre constant d'objets en fonction d'un facteur défini. Chaque instance a un facteur d'objet Fi $\in \{1, 3, 5, 10\}$. Il s'agit d'un choix intéressant fait par les auteurs car on pourrait imaginer que la distribution non-uniforme des objets sur les villes conduirait à des exemples intéressants.

Faire simplement varier la quantité d'objets qu'une ville reçoit pourrait ne pas conduire à quelque chose intéressant, en dehors de variables de décision supplémentaires. En fait, cela pourrait même conduire à des problèmes moins intéressants puisqu'un pourcentage plus élevé de villes ont des objets avec un bon rapport bénéfice/poids. En d'autres termes, s'il n'y a pas de villes qui méritent d'être prioritaires, l'ordre dans lequel les villes sont classées peut être modifié.

La capacité est définie comme une fraction du poids total, où la fraction est choisie dans l'ensemble $\{\frac{1}{11}, \frac{2}{11}, ..., \frac{10}{11}\}$. Il en résulte 10 catégories de capacité différentes

5.2.3 Tarif de location R:

La valeur du tarif de location R est cruciale pour l'interdépendance du problème. Elle doit être choisie de manière à ce que le profit retourné par le choix d'un objet et le temps de trajet contribuent de manière égale à la fonction objective. Afin de s'assurer que l'un ne domine pas l'autre. Cet équilibre est obtenu en fixant le tarif de location comme suit :

$$R = \frac{valeur\ optimale\ KP}{valeur\ optimale\ approximative\ TSP}$$

Où la valeur optimale de la tournée TSP est approximée par une solution trouvée via l'heuristique Lin-Kernighan. Le tarif de location est un aspect essentiel de la difficulté du problème. Wu, Polyakovskiy,et Neumann [52] ont étudié l'effet du tarif de location sur le problème de l'emballage sans contrainte pendant le trajet.

Ils ont examiné les plages de tarif de location dans lesquelles la décision de ramasser un objet est non-triviale. Signifie ici qu'il n'est pas toujours rentable de ramasser un certain objet (ou pas). En général, le tarif de location décide de l'influence que le profit des objets ou la distance parcourue sont sur la fonction objective. Polyakovskiy et al. [34] choisissent de fixer R à une valeur dépendante de la tournée CLK et du plan d'emballage optimal.

5.2.4 exemple d'une instance:

la figure suivante représente un exemple d'une instance de référence de TTP, elle est constituer de 3 partie, dans la première on trouve des information importantes sur l'instance: le nom, le type de sac a dos, la dimension ou bien le nombre de villes, le nombre des objets, la capacité de sac, la vitesse minimal et maximal du voleur et le tarif de location de cette instance, dans la deuxième partie on trouve les cordonnés de chaque ville et dans la dernière la valeur, le poid et l'emplacement de chaque objet.

```
PROBLEM NAME:   kroA100-TTP
KNAPSACK DATA TYPE: uncorrelated
DIMENSION:      100
NUMBER OF ITEMS:        99
CAPACITY OF KNAPSACK:   4794
MIN SPEED:      0.1
MAX SPEED:      1
RENTING RATIO:  0.38
EDGE_WEIGHT_TYPE:       CEIL_2D
NODE_COORD_SECTION      (INDEX, X, Y):
1       1380    939
2       2848    96
        .
        .
        .
ITEMS SECTION   (INDEX, PROFIT, WEIGHT, ASSIGNED NODE NUMBER):
1       119     1       2
2       187     896     3
        .
        .
        .
```

Figure12 : exemple d'une instance de référence de TTP

5.3 Application

5.3.1 L'objectif de l'application

Nous avons développé cette application pour le but de pouvoir charger, afficher des instances de problème de voyageur voleur, lancer une recherche en utilisent les 2 approches proposé dans le chapitre 3: ALNS, SA et finalement afficher le résultat de la recherche.

5.3.2 Exécution de l'application:

Une fois l'application exécutée, la fenêtre principale apparaît. Elle contient le suivant:

- un menu qui contient trois fonctionnalités principale: charger une instance (et l'afficher aussi), lancer une recherche en utilisent l'ALNS, lancer une recherche en utilisent le SA.

- une zone pour afficher le graphe qui représente les villes d'une instance chargé et pour aussi afficher le chemin de voleur après la fin de la recherche.

- un zone de texte pour afficher les propriété d'une instance chargé et le détaille de ces objets.

- un ensemble des boutons pour contrôler l'affichage des villes.

- une petite partie de la fenêtre est réservé pour afficher le résultat de la fonction objective après une recherche.

Après le chargement de l'instance et le lancement d'un algorithme SA ou ALNS, a la fin de l'exécution de l'algorithme, le résultat s'affiche comme montré dans la **Figure 13**.

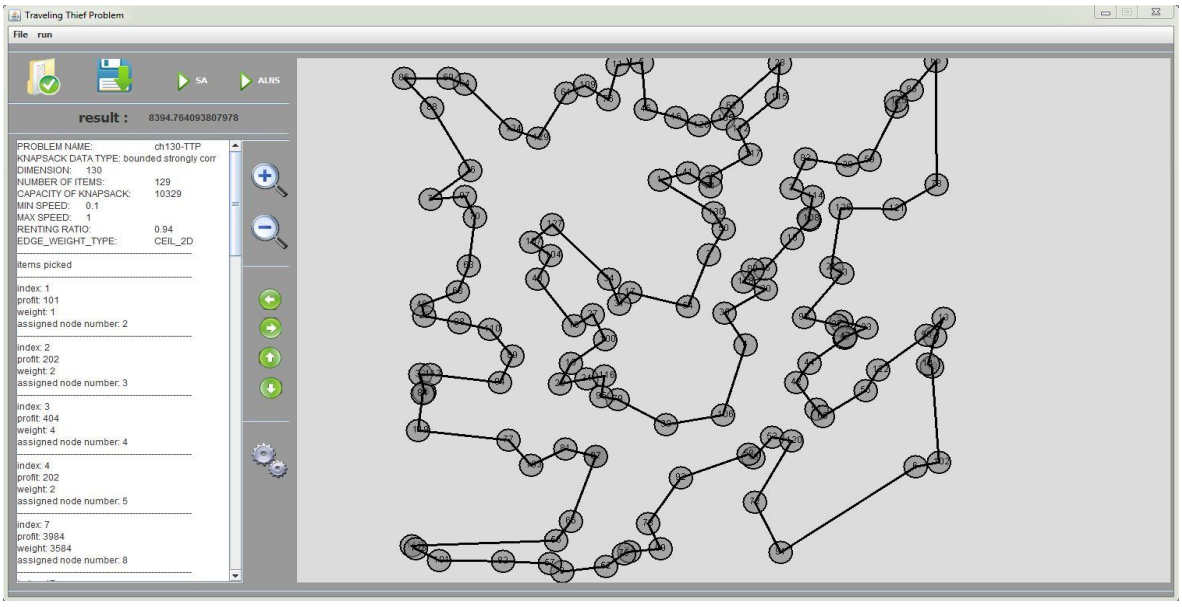

Figure13 :l'affichage du résultat.

5.3.3 Le paramétrage des méthodes:

On a utilisé les paramètres suivante pour les deux méthodes proposés dans nos testes:

SA: - la température initial = 1000

- le taux de refroidissement = 0.9999

- la température final = 0.001

ALNS:- la température initial = 1000

- le taux de refroidissement = 0.99995

- la température final = 0.001

5.3 Résultats:

Les tableaux Tab. 2, Tab. 3 et Tab. 4 donnerons un résumé des résultats sur les deux méta-heuristiques proposés. Les tests sont effectués sur sept instances différentes de problème TTP. Dix exécution ont été réalisés pour chaque instance, les résultats présentés montrent que la méthode basés sur le recuit simulé donnera des meilleurs résultats (figure 14).

- **ALNS Algorithme**

instance	resultants										moy
berlin52_n51_bounded-strongly-corr_01	4446,02	2989,58	4193,50	3440,33	3330,37	4124,08	4181,12	4112,39	4361,50	3810,01	**3898,89**
eil76_n75_bounded-strongly-corr_01	3379,98	3692,82	3336,80	3766,75	3618,23	3750,34	3359,19	3502,87	3509,21	3776,10	**3569,22**
kroA100_n99_bounded-strongly-corr_01	3586,19	2571,16	2490,54	3287,51	2669,92	1965,12	3858,82	2556,26	3810,08	2009,15	**2880,47**
pr124_n123_bounded-strongly-corr_01	3183,76	5029,33	2758,48	3780,20	2877,56	2394,04	3034,07	4150,25	3497,34	3615,57	**3432,06**
ch130_n129_bounded-strongly-corr_01	5060,57	6133,02	6111,82	6568,42	5371,70	6351,55	7391,22	7071,43	6129,95	6482,20	**6267,18**
u159_n158_bounded-strongly-corr_01	5131,35	4565,63	5661,16	2410,09	6380,22	4713,90	4026,62	3989,10	5066,09	4887,90	**4683,20**
a280_n279_bounded-strongly-corr_01	7949,13	9787,86	9140,22	11063,80	9891,00	9876,55	9366,67	8257,99	8572,77	11679,45	**9558,54**

Tableau 2: résultats obtenus par ALNS algorithme

- **SA Algorithme**

instance	resultants										moy
berlin52_n51_bounded-strongly-corr_01	4083,23	3538,94	3714,36	3702,60	3725,14	3456,22	3884,12	4247,66	4383,38	2823,46	**3755,91**
eil76_n75_bounded-strongly-corr_01	3981,47	3488,78	3247,79	3923,12	3655,61	3625,97	3901,99	3510,71	4464,17	3206,14	**3700,57**
kroA100_n99_bounded-strongly-corr_01	3352,59	3993,26	5007,29	3355,01	4125,27	3952,11	4254,78	4149,18	3945,31	3179,43	**3931,42**
pr124_n123_bounded-strongly-corr_01	5485,32	5960,07	5968,07	5652,29	7044,51	5899,35	5996,33	7367,91	5767,54	6954,54	**6209,59**
ch130_n129_bounded-strongly-corr_01	7782,41	7970,23	8011,10	8586,79	7470,41	8464,09	7731,20	7927,29	8352,74	8265,71	**8056,19**
u159_n158_bounded-strongly-corr_01	7498,72	7367,38	7455,08	7949,88	7561,29	7778,69	7900,70	7615,73	7564,82	8007,89	**7670,01**
a280_n279_bounded-strongly-corr_01	14548,49	14911,25	14682,14	16574,12	14701,77	14926,74	17057,60	14929,35	15109,86	16402,18	**15384,35**

Tableau 3: résultats obtenus par algorithme SA

- **ALNS vs. SA Algorithme**

Instances	ALNS	SA
a280_n279_bounded-strongly-corr_01	**9558,54**	**15384,35**
berlin52_n51_bounded-strongly-corr_01	**3898,89**	**3755,91**
ch130_n129_bounded-strongly-corr_01	**6267,18**	**8056,19**
kroA100_n99_bounded-strongly-corr_01	**2880,47**	**3931,42**
pr124_n123_bounded-strongly-corr_01	**3432,06**	**6209,59**
u159_n158_bounded-strongly-corr_01	**4683,20**	**7670,01**
eil76_n75_bounded-strongly-corr_01	**3569,22**	**3700,57**

Tableau 4: ALNS vs. SA pour quelque instances testés.

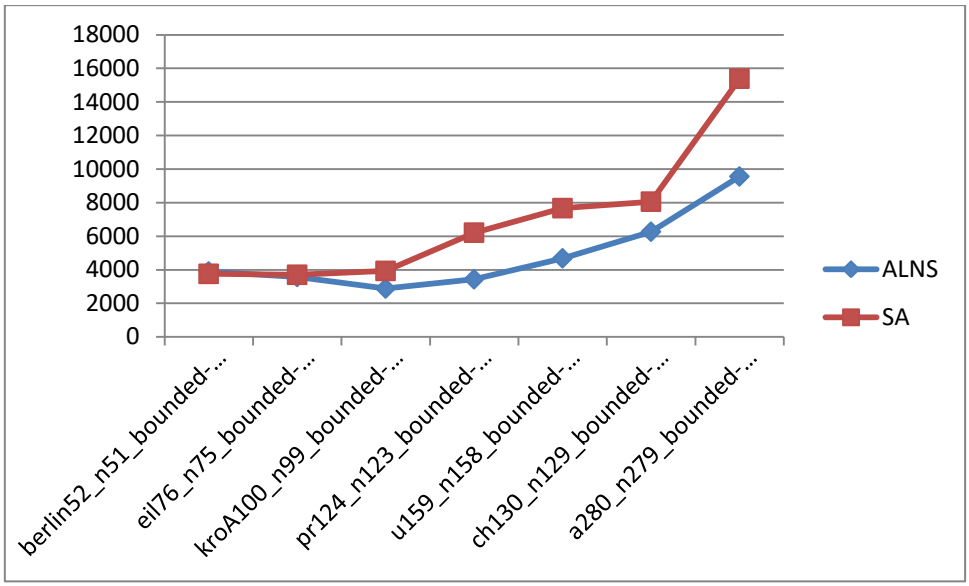

Figure 14 : Diagramme d'exécution ALNS vs. SA pour les instances testés

5.4 Conclusion:

Dans ce chapitre, nous avons présenté notre application java pour l'implémentation des 2 méthodes proposé: SA et ALNS, on a détaillé les instances de référence de problème et afficher les résultats de notre application sur quelque instances.

CHAPITRE 6

Conclusion générale

Les problèmes d'optimisation combinatoire à composantes multiples jouent un rôle majeur dans de nombreuses applications du monde réel. Le problème de voyageur voleur est un problème d'optimisation combinatoire à deux composantes: le problème du voyageur de commerce (TSP) et le problème du sac à dos 0-1 (KP).

Le TTP a été introduit pour représenter le problème du monde réel, l'interdépendance entre les deux sous-problèmes est la cause de la complexité de ce problème. Plusieurs approches ont été introduites pour le résoudre.

Dans ce mémoire, Nous avons présentés quelques travaux liés au problème avec la proposition de méta-heuristique ALNS pour le problème comme une méthode de résolution, nous avons utilisé le SA comme étape intermédiaire avant de passer à l'ALNS pour bien comprendre le problème et alors bien adapter l'ALNS.

Les résultats obtenus par le biais des algorithmes proposés : SA et ALNS, montrent qu'un tel choix de résolution peut conduire à des résultats compétitive par apport aux travaux montrés dans la littérature. Même avec une telle complexité effrayante, les résultats réalisés pour les instances de problème pouvant être amélioré à travers l'amélioration des heuristiques de voisinage afin de s'adapter avec la nature bi-objective du problème.

En perspective, nous espérons que ce travail donnera un premier aperçu pour les futurs étudiants cherchons à travailler sur ce type de problème.

Bibliographie

[1] Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1037–1044. IEEE (2013)

[2] M. Baghel, S. Agrawal, S. Silakari, Survey of metaheuristic algorithms for combinatorial optimization, Int. J. Comput. Appl. (2012) 58.

[3] Edmonds, J. (2008). Definition of Optimization Problems. In How to Think About Algorithms (pp. 171-172). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511808241.015

[4] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," Oper. Res., vol. 21, pp. 498-516, 1973.

[5] H. Pirkul, "A heuristic solution procedure for the multiconstraint zero one knapsack problem," Na. Res. Log. (NRL), vol. 34, pp. 161-172, 1987.

[6] David L Applegate et al. The traveling salesman problem: a computational study. Princeton university press, 2011.

[7] William Cook. In pursuit of the traveling salesman: mathematics at the limits of computation. Princeton University Press, 2012.

[8] G. Gutin and A. P. Punnen, The traveling salesman problem and its variations. Springer Science & Business Media, 2006, vol. 12.

[9] Michael Held and Richard M Karp. \A dynamic programming approach to sequencing problems". In: Journal of the Society for Industrial and Applied Mathematics 10.1 (1962), pp. 196{210.

[10] Michael Hahsler and Kurt Hornik. \TSP-Infrastructure for the traveling salesperson problem". In: Journal of Statistical Software 23.2 (2007), pp. 1{21.

[11] Gerhard Reinelt. \TSPLIB|A traveling salesman problem library". In: ORSA journal on computing 3.4 (1991), pp. 376{384.

[12] Gregory Gutin, Anders Yeo, and Alexey Zverovich. \Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP". In: Discrete Applied Mathematics 117.1 (2002), pp. 81{86.

[13] Nicos Christo_des. Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[14] David S Johnson and Lyle A McGeoch. \The traveling salesman problem: A case study in local optimization". In: Local search in combinatorial optimization 1 (1997), pp. 215{310.

[15]] Yi Mei, Xiaodong Li, and Xin Yao. "On investigation of interdependence between sub-problems of the travelling thief problem". In: Soft Computing 20.1 (2016), pp. 157–172.

[16] M. El Yafrani, B. Ahiod, Efficiently solving the traveling thief problem using hill climbing and simulated annealing, Inform. Sci. 432 (2018) 231–244.

[17] Shen Lin and BrianWKernighan. \An efective heuristic algorithm for the traveling-salesman problem". In: Operations research 21.2 (1973), pp. 498-516.

[18] Keld Helsgaun. \An e_ective implementation of the Lin-Kernighan travelingsalesman heuristic". In: European Journal of Operational Research 126.1 (2000), pp. 106{130

[19] Sergey Polyakovskiy and Frank Neumann. \The packing while travelingproblem". In: European Journal of Operational Research 258.2 (2017),pp. 424{439.

[20] Silvano Martello, David Pisinger, and Paolo Toth. \New trends in exact algorithms for the 0{1 knapsack problem". In: European Journal of Oper-ational Research 123.2 (2000), pp. 325{332.

[21] David Pisinger. \A minimal algorithm for the 0-1 knapsack problem". In:Operations Research 45.5 (1997), pp. 758{767.

[22] David Pisinger. \An expanding-core algorithm for the exact 0{1 knapsack problem". In: European Journal of Operational Research 87.1 (1995),pp. 175{187.

[23] Pedro Larranaga et al. "Genetic algorithms for the travelling salesman problem: A review of representations and operators". In: Artificial Intelligence Review 13.2 (1999), pp. 129–170.

[24] Yuichi Nagata and Shigenobu Kobayashi. "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem". In: INFORMS Journal on Computing 25.2 (2013), pp. 346–363.

[25] Danilo Sanches, Darrell Whitley, and Renato Tin´os. "Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover". In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM. 2017, pp. 329–336.

[26] IM Oliver, DJd Smith, and John RC Holland. "Study of permutation crossover operators on the traveling salesman problem". In: Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates, 1987. 1987.

[27] Timothy Starkweather et al. "A Comparison of Genetic Sequencing Operators." In: ICGA. 1991, pp. 69–76

[28] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem.Manage. Sci., 45:414–424, 1999.]

[29] Bernard Manderick. "The genetic algorithm and the structure of the fitness landscape". In: Proc. 4th International Conference on Genetic Algorithms. 1991, pp. 143–1

[30] David Pisinger. "Where are the hard knapsack problems?" In: Computers & Operations Research 32.9 (2005), pp. 2271–2284.

[31] Sergey Polyakovskiy and Frank Neumann. "Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems". In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems. Springer. 2015, pp. 332– 346.

[32] Frank Neumann et al. "A Fully Polynomial Time Approximation Scheme for Packing While Traveling". In: arXiv preprint arXiv:1702.05217 (2017).

[33] Hayden Faulkner et al. "Approximate approaches to the traveling thief problem". In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM. 2015, pp. 385–392.

[34] S. Polyakovskiy, M.R. Bonyadi, M. Wagner, Z. Michalewicz, F. Neumann, A comprehensive benchmark set and heuristics for the traveling thief problem, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 477–484

[35] M. Weiszer, J. Chen, S. Ravizza, J. Atkin, P. Stewart, A heuristic approach to greener airport ground movement, in: Evolutionary Computation (CEC), 2014 IEEE Congress on, IEEE, 2014, pp. 3280–3286.

[36] Mohamed El Yafrani and Belaid Ahiod. "Population-based vs. singlesolution heuristics for the travelling thief problem". In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. ACM. 2016, pp. 317–324.

[37] Mohamed El Yafrani et al. "A hyperheuristic approach based on low-level heuristics for the travelling thief problem". In: Genetic Programming and Evolvable Machines (), pp. 1–30.

[38] Mahdi Moeini, Daniel Schermer, and Oliver Wendt. "A Hybrid Evolutionary Approach for Solving the Traveling Thief Problem". In: International Conference on Computational Science and Its Applications. Springer. 2017, pp. 652–668.

[39] Markus Wagner. "Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem". In: International Conference on Swarm Intelligence. Springer. 2016, pp. 273–281.

[40] Daniel KS Vieira et al. "A Genetic Algorithm for Multi-component Optimization Problems: The Case of the Travelling Thief Problem". In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer. 2017, pp. 18–29

[41] Nuno Louren¸co, Francisco B Pereira, and Ernesto Costa. "An evolutionary approach to the full optimization of the traveling thief problem". In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer. 2016, pp. 34–45.

[42] Marcella SR Martins et al. "HSEDA: A Heuristic Selection Approach Based on Estimation of Distribution Algorithm for the Travelling Thief Problem". In: Proceedings of the Genetic and Evolutionary Computation Conference. 2017, p. 8.

[43] M. El Yafrani, B. Ahiod, A local search based approach for solving the travelling thief problem: the pros and cons, Appl. Soft Comput. 52 (2017) 795–804.

[44] Yi Mei, Xiaodong Li, and Xin Yao. "Improving efficiency of heuristics for the large scale traveling thief problem". In: Asia-Pacific Conference on Simulated Evolution and Learning. Springer. 2014, pp. 631–643.

[45] Markus Wagner et al. "A case study of algorithm selection for the traveling thief problem". In: Journal of Heuristics (2017), pp. 1–26.

[46] Junhua Wu et al. "Exact Approaches for the Travelling Thief Problem". In: Asia-Pacific Conference on Simulated Evolution and Learning. Springer. 2017, pp. 110–121.

[47] J. Blank, K. Deb, S. Mostaghim, Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2017, pp. 46–60.

[48] M.E. Yafrani, S. Chand, A. Neumann, B. Ahiod, M. Wagner, Multiobjectiveness in the single-objective traveling thief problem, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, 2017, pp. 107–108.

[49] S. Chand, M. Wagner, Fast heuristics for the multiple traveling thieves problem, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, 2016, pp. 293–300.

[50] J. Wu, S. Polyakovskiy, M. Wagner, F. Neumann, Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem, 2018, ArXiv preprint, arXiv:1802.02434.

[51] S.T. Alharbi, The design and development of a modified artificial bee colony approach for the traveling thief problem, Int. J. Appl. Evol. Comput. 9 (3) (2018) 32–47.

[52] Junhua Wu, Sergey Polyakovskiy, and Frank Neumann. "On the impact of the renting rate for the unconstrained nonlinear knapsack problem". In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. ACM. 2016, pp. 413–419.

[53] W. Zhang, A. Maleki, M.A. Rosen, J. Liu, Optimization with a simulated annealing algorithm of a hybrid systemfor renewable energy including battery and hydrogen storage, Energy 163 (2018) 191e207

[54] H.L. Shieh, C.C. Kuo, C.M. Chiang, Modified particle swarm optimization algorithm with simulated annealing behavior and its numerical verification, Appl. Math. Comput. 218 (2011) 4365e4383.

[55] G. Sodeifian, S.A. Sajadian, N.S. Ardestani, Experimental optimization and mathematical modeling of the supercritical fluid extraction of essential oil from Eryngium billardieri: application of simulated annealing (SA) algorithm, J. Supercrit. Fluids 127 (2017) 146e157.

[56] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (1953) 1087e1092.

[57] F. Javidrad, M. Nazari, A new hybrid particle swarm and simulated annealing stochasticoptimization method, Appl. Soft Comput. 60 (2017) 634e654.

[58] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671e680.

[59] V. Cerny, A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, J. Optim. Theor. Appl. 45 (1) (1985) 41e51.

[60] M.C. Aguitoni, L.V. Pavao, M.A. Silva Sa Ravagnani, Heat exchanger network synthesis combining simulated annealing and differential evolution, Energy 181 (2019) 654e664

[61] M.S. Hussin, T. Stützle, Tabu search vs. simulated annealing for solving large quadratic assignment instances, Comput. Oper. Res. 43 (2014) 286e291.

[62] D.T. Connolly, An improved annealing scheme for the qap, Eur. J. Oper. Res. 46 (1) (1990) 93e100.

[63] I.H. Osman, C.N. Potts, Simulated annealing for permutation flow-shop scheduling, Omega 17 (6) (1989) 551e557.

[64] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation: part II, graph coloring and number partitioning, Operational Research 39 (3) (1991) 378e406

[65] S. Kirkpatrick, Optimization by simulated annealing: quantitative studies, J. Stat. Phys. 34 (5e6) (1984) 975e986.

[66] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. 6 (6) (1984) 721e741.

[67] M. Lundy, A. Mees, Convergence of an annealing algorithm, Math. Program. 34 (1) (1986) 111e124, 1986.

[68] K. Andersen, R.V.V. Vidal, V.B. Iversen, Design of a teleprocessing communication network using simulated annealing, in: R.V.V. Vidal (Ed.), Applied Simulated Annealing, Springer, 1993, pp. 201e215.

[69] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, J. Comput. Phys. 90 (1) (1990) 161e175.

[70] P. Shaw, A new local search algorithm providing high quality solutions to vehicle routing problems, Technical report, Department of Computer Science, University of Strathclyde, Scotland, 1997.

[71] P. Shaw, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), 1998.

[72] R. Bent, P. Van Hentenryck, A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows, To appear in Transportation Science.

[73] R. Bent, P. Van Hentenryck, A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows, Proceedings of the International Conference on Constraint Programming (CP-2003), Lecture Notes in Computer Science 2833, 123-137

[74] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck, Record Breaking Optimization Results Using the Ruin and Recreate Principle, Journal of Computational Physics 159 (2000), 139-171.

[75] S.Ropke and D.Pisinger, An adaptative large neihborhood search heuristic for the pickup and delivery problem with time windows, Transportation sciences, 2006, vol, 40, no 4, p. 455-472

[76] R.Masson. F.Lehuédé. O.piton, An adaptative large neihborhood search heuristic for the pickup and delivery problem with transfers, Transportation science, 2013, vol, 47, no 3, p. 344-355

[77]L.M.Masmoudi, M.Hosny, K.Brackers and A.Dammak, three effective metaheuristics to solve the muli-depot multi-trip heterogeneous dial-a-ride problem, Transportation Research Part E: Logistics and Transportation Review, 2016, vol, 96, p. 60-80.

[78] Gürcan Çetin, Ali Keçebas¸ Optimization of thermodynamic performance with simulated annealing algorithm: A geothermal power plant, Renewable Energy 172 (2021) 968e982

[79] Alenrex Maity, Swagatam Das, Efficient hybrid local search heuristics for solving the travelling thief problem, Applied Soft Computing Journal 93 (2020) 106284

[80] Rogier Hans Wuijts, D. Thierens, Investigation of the Traveling Thief Problem, Utrecht University, June 11,2018