Application d’un algorithme de recuit simulé pour résoudre le problème de routage des véhicules de collecte des déchets

Par

Amine ABOULQUASSEM

RAPPORT DE PROJET PRÉSENTÉ À L’ÉCOLE DE TECHNOLOGIE SUPÉRIEURE COMME EXIGENCE PARTIELLEÀ L’OBTENTION DE LA MAITRISE EN GÉNIE DE LA PRODUCTION AUTOMATISÉE

MONTREAL, LE 27 JUILLET 2022

ECOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Mustapha Ouhimmou, directeur de projet

Département génie des systèmes à l'École de technologie supérieure

M. Julio Cesar Montecinos, codirecteur de projet

Département génie des systèmes à l'École de technologie supérieure

REMERCIEMENTS

Mes remerciements vont tout d'abord à mes chers professeur, Monsieur Mustapha Ouhimmou et Monsieur Julio César Montecinos, qui m'ont permis d'effectuer ce travail de maitrise, qu'ont cru en ma capacité pour l'accomplir, et qui m'ont accompagné tout le long de mon projet. Leur disponibilité à mon égard et leur excellents support académique furent fondamentaux pour ma réussite.

J'exprime beaucoup de reconnaissance envers mes parents et ma famille pour leur support et leurs encouragements continus.

Enfin, je remercie mes amis pour leur soutien durant les moments difficiles

Application d'un algorithme de recuit simulé pour résoudre le problème de routage des véhicules de collecte des déchets

Amine ABOULQUASSEM

RÉSUMÉ

La logistique dans l'industrie de nos jours est l'un des domaines les plus sensibles et stratégiques. Dans le but de réduire ses dépenses, chaque entreprise cherche à minimiser son nombre de véhicules utilisés mais aussi les distances à effectuer de ces derniers, permettant des économies d'énergies. Le problème de routage de véhicules (VRP) se présente alors comme le type d'optimisation le plus répandu (ou le plus utilisé) dans ce contexte car il offre un éventail de techniques pour apporter des solutions optimales dans plusieurs cas.

Il est important de noter que l'algorithme développé dans les problèmes de tournées de véhicules avec contraintes de capacité (CVRP), qui est un type important de VRP, donne des bons résultats. Dans cette étude, qu'il s'agit d'un modèle de transport optimisé adapté à la collecte des déchets, qui vise à trouver la distance la plus courte en respectant certaines contraintes, nous commençons par la présentation d'une étude bibliographique qui nous permettra de bien comprendre et réussir notre projet.

Tout d'abord nous présenterons notre revue de littérature ce qu'est un problème d'optimisation de transport en recherche opérationnelle de manière générale, après on va montrer les différences et les particularités de chacun de ses variantes, et nous expliquerons aussi les méthodes utilisées dans la résolution de ces problèmes. Ensuite, dans la partie de méthodologie, nous commencerons par la définition du problème plus en détail ainsi que la méthodologie suivie afin de construire notre modèle, nous décrirons enfin la méthode utilisée pour la résolution de celui-ci ainsi que les outils que nous utiliserons pour notre résolution.

À la fin du rapport, nous allons présenter l'algorithme recuit simule qui va être implanté sur PYTHON. Et nous expliquerons les résultats obtenus d'une manière précis.

Application of a simulated annealing algorithm to solve the waste collection vehicle routing problem

Amine ABOULQUASSEM

ABSTRACT

Logistics in industry today is one of the most sensitive and strategic areas. In order to reduce its expenses, each company tries to minimize the number of vehicles used and the distances they have to cover, thus saving energy. The vehicle routing problem (VRP) is the most common type of optimization in this context because it offers a range of techniques to provide optimal solutions in several cases.

It is important to note that the algorithm developed in capacity constrained vehicle touring problems (CCTV), which is an important type of VRP, performs well. In this study, that it is an optimized transport model adapted to the waste collection, which aims to find the shortest distance respecting some constraints, we start with the presentation of a bibliographic study that will allow us to understand and succeed in our project.

First of all we will present our literature review what is a problem of optimization of transport in operational research in a general way, then we will show the differences and the particularities of each of its variants, and we will also explain the methods used in the resolution of these problems. Then, in the methodology part, we will start with the definition of the problem in more detail as well as the methodology followed in order to build our model, and finally we will describe the method used for the resolution of this one as well as the tools that we will use for our resolution.

At the end of the report, we will present the simulated annealing algorithm that will be implemented on PYTHON. And we will explain the results obtained in a precise way.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

VRP : Vehicule Routing Problems (Problèmes de routage de véhicule)

CVRP : Capacitated Vehicule Routing Problems (Problèmes de routage de véhicule capacitifs)

VRPTW : Vehicule Routing Problems with Time Windows (Problèmes de routage de véhicule avec fenêtres de temps)

VRPB : Vehicle Routing Problem with Backhauls (Problème d'acheminement de véhicules avec backhauls)

PDP : Pick-up and Delivery Problems (Problèmes de cueillette et livraison)

GA : Genetic Algorithm (Algorythme génétique)

TS : Tabu Search (recherche Tabou)

VRP : Vehicule Routing Problems (Problèmes de routage de véhicule)

ACO : Ant Colony Optimization (Optimisation des colonies de fourmis)

INTRODUCTION

La gestion des déchets est l'une des problématiques majeures dans le monde en raison de son impact direct sur la faune et les êtres humains. L'urbanisation rapide et les activités humaines quotidiennes génèrent une grande quantité de déchets venant des zones résidentielles, commerciales ou industrielles. La collecte des déchets oblige les entreprises à utiliser des véhicules partant du dépôt et fais un circuit sur des itinéraires fixes pour collecter les déchets en visitant tous les points de collecte, ce qui implique un coûte et un budget énorme pour les entreprises. Pour une collecte efficace des déchets, l'itinéraire de collecte doit être optimisé et bien calculer de manière à prendre en compte tous les scénarios possibles.

Le cas étudié ici est un type particulier de transport, celui du transport adapté pour la collection des déchets. Le problème peut être traduit comme la recherche d'un itinéraire parfait pour un ensemble de camions, qui ont une capacité limitée et doivent desservir un ensemble des villes une seule fois dans son trajet. La particularité de celui-ci par rapport aux autres méthodes de résolution réside dans le fait d'avoir une bonne qualité de solution et dans un temps raisonnable.

Dans notre rapport on va générer des scénarios pour notre problématique. On va commencer par une présentation de revue de littérature, dans laquelle nous expliquons notre problème, débutant par les problèmes de routages de véhicules ainsi que leurs domaines d'application et après on va analyser les variantes de ce problème. À la fin de ce chapitre, on va présenter les différentes méthodes de résolution.

Dans le chapitre qui suit, on va donner une présentation avec plus de détails sur le problème en général ainsi que la méthodologie de résolution.

Dans le troisième et dernier chapitre, nous présentons notre modèle proposé et nous essaierons de valider le modèle afin de générer des scénarios. Nous discuterons à la fin de ce chapitre de ce que nous avons réalisé comme améliorations.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Introduction

Dans ce chapitre, on va explorer quelques articles qui traite le problème de l'optimisation des tournées de véhicules, analysant les revues de littérature effectuer par les chercheurs dans ce domaine, par la suite, nous allons décortiquer les différentes variantes de VRP, étudiant les modèles proposés par les articles d'une façon précise. Enfin, nous décrirons les principales méthodes utilisées pour la résolution de ces types de problèmes.

1.2 Les problèmes de routage de véhicule (VRP)

Le problème que nous allons résoudre durant notre projet est un problème de routage de véhicule (VRP). Ce genre de problème génère une grande préoccupation en termes d'impacts économiques, environnementaux ces dernières années. L'article de Dantzig et Ramser (Dantzig et Ramser, 1959) ont proposé la première formulation de programmation mathématique et la première approche algorithmique pour le problème de l'acheminement des véhicules, ils ont considéré le VRP comme un problème NP-difficile. Le but de VRP est de concevoir des itinéraires pour les véhicules situés dans un ou plusieurs entrepôts afin de collecter les déchets dans les plus brefs délais et au moindre coût. L'objectif de VRP est généralement de minimiser le coût du routage. De plus, la fonction objective peut varier pour des contraintes et des applications spécifiques. Il est conçu pour minimiser les couts associés aux véhicules, la distance parcourue, le nombre de véhicules transportés et a équilibré le système en cas de dépassement du temps de transport ou de déplacement et de la charge du véhicule (Sitek ve Wikarek, 2019). l'article de Caceres et al. (Caceres et al., 2014) ont exploré largement sur les différentes variantes de VRP en notant que chacun de ces types se diffère selon la complexité du problème. La figure ci-dessus illustre un exemple de problème de VRP avec 7 clients en utilisant 3 véhicules.

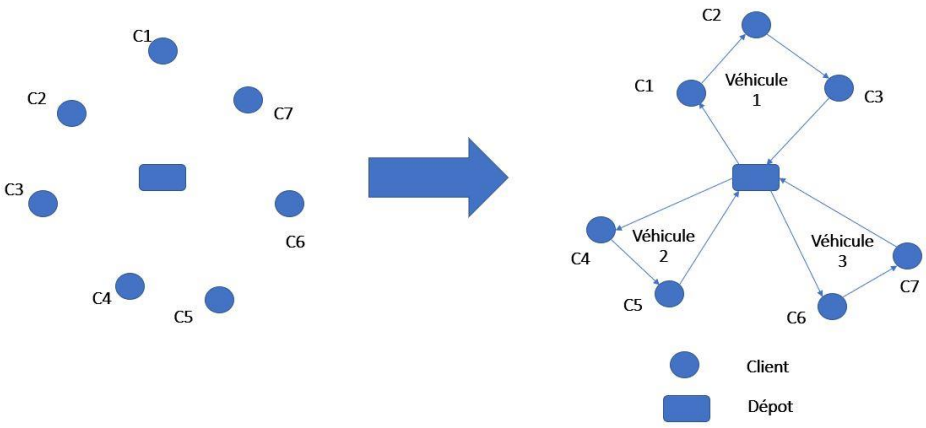

Figure 1 Exemple d'un problème de type VRP

1.3 Les variantes des problèmes (VRP)

On peut trouver plusieurs variantes de problème VRP. En plus, ces variantes changent selon la complexité de problème et à travers des applications dans le domaine du transport. Dans notre projet, nous traitons le problème d'élaboration de tournées de véhicule nommé CVRP, par conséquent, nous allons décrire les principales variantes du VRP.

1.3.1 Le problème de routage de véhicules avec contraintes de capacité (CVRP)

Ce genre de problème, consiste à identifier un ensemble de routes, commençant et se terminant au site v0, couvrant un ensemble de point sortant (ville, client …). Chaque point a caractérisé par une demande et un seul véhicule est passé à la fois pour chaque itinéraire. Chaque véhicule a une capacité donnée et transporte un type de produit. Ainsi, aucun véhicule ne peut desservir (collecte ou livrer) plus qu'un client (que sa capacité ne le permet). L'objectif est de minimiser la distance totale parcourue. Ainsi, le CVRP est connu pour simplifier à travers un graphe dans le but que chaque circuit correspond à un itinéraire véhicule avec un coût minimum. Ce problème a de nombreuses applications dans la vie réelle telle que la collecte et la livraison de marchandises, l'acheminement des autobus scolaires, le nettoyage des rues, la collecte des déchets, les systèmes d'appel à distance, le transport des personnes handicapées et l'acheminement des vendeurs. Je vais citer quelque article qui traite ce type de problème. Commençant par article, l'article Benrahou (F Benrahou et A Tairi, 2019) ont optimisé un process de collecte d'une société de commercialisation et de distribution de pétrole en Algérie. D'abord, ils ont étudié le problème (CVRP), ensuite, ils ont introduit comme méthode de résolution un algorithme heuristique appelé l'algorithme d'insertion le plus proche à un problème réel de collecte d'huiles usées. L'objectif de cette étude est limité la pollution en

diminuant la consommation de carburant, de réduire les émissions de gaz à effet de serre dans le but de protéger l'environnement. Le Blanc et al. (Le Blanc et al. 2004) présentent le concept d'inventaire en tant que contrepartie de la logistique inverse qui est gérée par le collecteur (CMI). Résolvent la problématique par une combinaison de génération de route afin de trouver le meilleur ensemble d'itinéraires, ils introduisent les deux types de planification proactive, une basée sur les ordres obligatoires et l'autre sur les ordres de canette. Ils ont trouvé comme résultats, le nombre total de kilomètres parcourus peut être réduit de 26,3 % par rapport à la méthodologie de planification réactive classique.

1.3.2 Le problème de routage de véhicules avec fenêtres de temps (VRPTW) :

Un des problèmes le plus développer du VRP en ajoutant une contrainte de temps, ça veut dire que le service chez un client commence dans un intervalle de temps, prenant en considération qu'il y a un nombre précis de véhicules et chaque client doit être servi une seule fois. Lau, al., 2003 ont introduit des notions de pénalité pour chaque retard dans le but d'assouplir les plages horaires d'une manière exacte, utilisant une contrainte par une flotte de véhicules limitée (m-VRPTW). L'approche de recherche tabou a été appliquée comme méthode de résolution, par conséquent, ils ont obtenu des résultats des bonnes qualités dans un temps de calcul raisonnable. Schneider al., 2014 ont découvert un problème de tournées de véhicules électriques pour les livraisons du dernier kilomètre en introduisant des fenêtres de temps pour servir chaque client, tenant compte la variation de taux de chargement de batterie afin de trouver le chemin le plus optimal. Agra, al. (Agra, al., 2013) ont traité du problème de routage de véhicules robuste avec des fenêtres de temps dans le domaine de transport maritime, analysant les retards les plus fréquentés. Pour mieux expliquer le problème, ils ont généré deux nouvelles formulations pour le problème, la première c'est pour une optimisation changeable avec le temps, et la deuxième c'est en développant une nouvelle technique pour des problèmes d'optimisation combinatoire robuste.

1.3.3 Le problème de routage de véhicules avec backhauls (VRPB)

Il s'agit d'une version le plus développer du problème (VRP) dans lequel les livraisons et les ramassages doivent être effectués en même temps. Tous les nœuds doivent être visités de manière contigüe, la route doit commencer par rendre visite à un client après avoir quitté le dépôt et finir par rendre visite à un fournisseur juste avant de retourner au dépôt, du coup, toutes les livraisons sont chargées au dépôt et toutes les cueillettes sont déchargées au dépôt afin de trouver la meilleure route. L'article Goetschalckx (Goetschalckx, 1989) distingue le problème

en deux phases de solution, la première est basée sur des courbes de remplissage d'espace qui seront utilisées pour le regroupement et l'identification des clients les plus proches, dérivent donc une solution initiale au problème de transport de ligne. La seconde basée sur l'optimisation des sous-problèmes formulés dans un modèle mathématique, trois conclusions majeures ont été tirées, le premier algorithme de remplissage d'espace était suffisant pour produire une première solution qui sera utile pour un prédémarrage d'un algorithme d'amélioration. Deuxièmement, une amélioration supplémentaire de la première solution utilisant les heuristiques 2 Opt et 3 Opt, ce dernier a montré de meilleurs résultats par rapport à l'algorithme de remplissage d'espace. Enfin, une amélioration est effectuée à l'aide d'un algorithme glouton. Étant donné que l'étape de regroupement de cette méthode met avidement les points sur des routes successives. L'article Min (Min et Hokey, 1989) dans leur étude tentent de développer un modèle et une procédure de solution suffisamment efficace pour satisfaire à la fois le transport de ligne et le transport de retour, son approche de solution est basée sur la méthode « cluster-first route-second ». Le résultat de ses études démontre une diminution significative du rapport temps/distance. Des études ont été menées sur le système de distribution des bibliothèques publiques du comté de Franklin, dans l'Ohio, pour voir l'efficacité de la procédure utilisée. L'article Nagy (G. Nagy et S. Salhi, 2005) proposent une méthode qui traite les collectes et les approvisionnements dans une approche intégrée. Cette méthode est basée sur des heuristiques qui peuvent résoudre des problèmes de ramassage et de livraison simultanés, en plus de cela, cette méthode est utilisée pour résoudre des types plus complexes qu'il s'agit d'un problème avec plusieurs dépôts. Le test a été effectué pour un dépôt unique et multiple, avec cinq dépôts et un nombre limité de clients, le test ne prend que quelques secondes, et les résultats ont fourni une bonne qualité de solution.

1.3.4 Le problème de tournées de véhicules multi-dépôts (PTVMD) :

Ce type de problématique est une généralisation du VRP standard, dans lequel il y a plusieurs dépôts, et cela constitue sa principale structure. De plus, les clients doivent être affectés à un certain dépôt, en effet, les itinéraires peuvent commencer à n'importe quel dépôt, mais doivent retourner à la source de départ. L'article Azadeha (A. Azadeha et H. Farrokhi-Asl, 2019) ont traité une combinaison de problème de routage de véhicules multi-dépôts, et le problème de routage de véhicules mixtes fermé-ouvert en utilisant une flotte hétérogène de véhicules dont la capacité et la longueur maximale de l'itinéraire diffèrent, dans l'objective est d'avoir le cout minimum chercher par l'entreprise afin de servir les clients. Les auteurs ont proposé un nouveau modèle de programmation mixte en nombres entiers (MIP) ainsi qu'une nouvelle

métaheuristique hybride qui combine un algorithme génétique avec des outils d'aide à la décision. Après la comparaison entre les deux modèles, on peut conclure que cette méthode nous donne des résultats sous optimale mais de bonne qualité, par conséquent, dans les problèmes à grande échelle, les résultats numériques ont prouvé l'efficacité des algorithmes métaheuristiques. L'article Geetha (S.Geetha et al., 2012) ont décrit dans leur article que le problème de routage de véhicules multi-dépôts peut exercer des influences positives sur la quantité d'émissions de carbone générées en parcourant des distances moindres, dans l'objective d'avoir un compromis entre les deux.

1.4 Méthodes utilisées en résolution des problèmes (VRP)

Plusieurs méthodes peuvent être utilisées pour résoudre le problème du (VRP), chacune de ces méthodes à ses spécificités. En effet, ces méthodes donnent des résultats approximativement les mêmes, le temps de calcul est aussi un facteur déterminant qui change selon méthode de résolution appliquée. Dans le même ordre d'idées, Nous allons mettre l'accent sur les différentes méthodes de résolution existantes ainsi que leurs particularités.

1.4.1 Les méthodes exactes

Les méthodes exactes ce sont des méthodes qui nous donne la solution la plus exacte et juste, ça veut dire l'extrêmement globale, en introduisant une stratégie mathématique qui démontre qu'on va converger vers une solution optimale. On général il y a plusieurs méthodes existant, donnant exemple la méthode de la programmation dynamique et la méthode A*, par conséquent, les instances relativement grandes ne peuvent pas être traité dans ces méthodes ce qui provoque une augmentation de cout et de dépense pour l'entreprise sur le plan de mémoire et sur le plan d'espace calcule.

1.4.2 Les méthodes heuristiques

Les méthodes heuristiques est une idée algorithmique pour résoudre un problème optimisation, mais qui est spécifique à un seul type de problème. Donc il y a deux types heuristiques en générale qui sont les heuristiques basées sur la construction (par exemple les méthodes gloutonnes), qui construisent itérativement d'une solution en explorant la région admissible, il y a les heuristiques de descente qui commence par une solution initiale afin de chercher un optimum local. En effet les heuristiques sont dépendantes à la structure du problème à résoudre et principalement dans le choix du voisinage. Laporte et al. (Laporte et al. 2014) ont développer

un algorithme heuristique qui peut résoudre des types des problèmes avec des instances volumineuses d'une manière optimale en minimisant la variance des temps de calcul.

1.4.3 Métaheuristiques

Les métaheuristiques est une méthode plus générique et qui peut être appliqué a plusieurs types de problèmes afin de trouver une solution approcher, donc on peut dire que les métaheuristiques sont une forme d'algorithme d'optimisation stochastique et qui sont hybridés avec une recherche locale, alors on peut utiliser plusieurs idées heuristique pour résoudre plusieurs problèmes d'optimisations afin explorer des espaces de recherche d'une façon intelligente avec un cout minimum et dans un temps raisonnable. Il existe deux types de métaheuristiques, la première est basée sur la population des solutions donc il est inspiré a des modèles naturels comme les algorithmes génétiques et les algorithmes basés sur la recherche d'essaim ou il y a un regroupement comme la colonie de fourmis, et la deuxième est focalisé sur la solution simple qui baser sur la cherche local. La figure en bas donne un panorama des méthodes les plus utilisées. Selon (Dorigo et Stützle, 2004), les méthodes méta-heuristiques, qui sont des types d'heuristiques, sont plus adaptées aux solutions de haute qualité de l'espace de recherche.

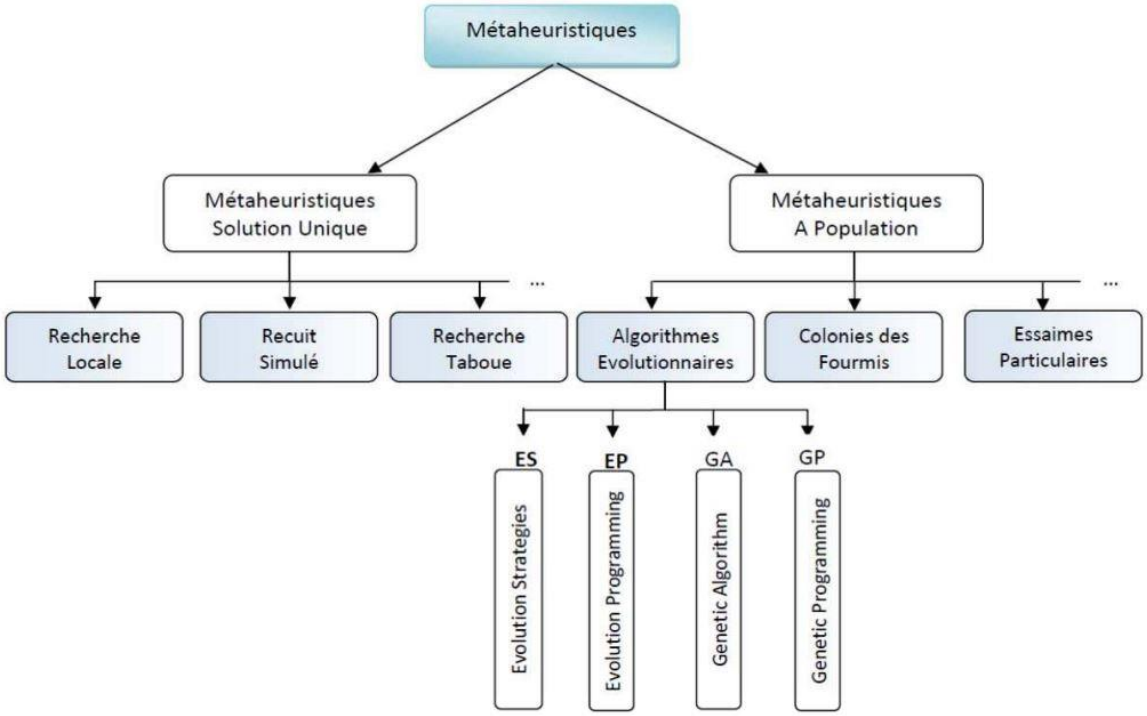

Figure 2 Classes des métaheuristiques (Dr. LEMOUARI ALI 2014)

1.4.3.1 Les algorithmes génétiques

Les algorithmes génétiques ce sont des méthodes qui nous donne des solutions approchées ou bien pseudo-optimal, l'idée de cet algorithme est empruntée d'une théorie scientifique pour l'évolution des espèces dans le monde de la nature qui s'appelle la théorie darwinienne, donc AG utilise la notion de la sélection naturelle sur une population initiale, alors qu'un ensemble d'individus qui va former une nouvelle population initiale dans le but d'atteindre la solution optimale et cette amélioration ça se fait avec la reproduction. Par conséquent, la population subit une sélection, en simulant les comportements naturels, et seuls les individus les mieux adaptés à l'environnement naturel survivent, par la suite, cette génération se fait par le croisement entre les parents pour avoir des nouveaux descendants en possédant les caractères de leurs parents, en plus de ce croisement, l'algorithme peut faire de mutation pour pouvoir diversifier la population.

Pour l'appliquer, nous nécessitons 5 éléments :

- ✓ La base de données que l'on cherche à parcourir doit être implémentée et fonctionnelle, *« La qualité du codage des données conditionne le succès des algorithmes génétiques. »* (J.-M. Alliot et N. Durand, 2005)
- ✓ Un outil de génération, afin que la sélection puisse être pertinente nous avons besoin de générer une population initiale hétérogène, l'importance ici est capitale, car cette génération ''Parent'' va déterminer la rapidité et l'efficacité du AG.
- ✓ Les AG utilisent une fonction communément appelée *fitness* ceci est un type de *fonction objective*, cette fonction permet de guider l'algorithme vers une solution souhaitée au même titre qu'une fonction objective, mais elle a aussi le rôle de tri, si la condition n'est pas validée alors la solution obtenue n'est pas envisageable.
- ✓ Des opérateurs qui serviront à diversifier la solution (croisement, mutation).
- ✓ Des paramètres de dimensionnement sont aussi utilisés tels que la taille de population ainsi que des pondérations sur les opérateurs de variations.

Le principe de fonctionnement de cet algorithme, d'abord une première génération d'individu est générée des couples de solutions sont choisis, ici nous les appellerons le couple parent P1 et P2, en fonction de leur solution par rapport à la fonction fitness, ensuite nous appliquons un opérateur de croisement. Le but ici est de croiser des différents éléments de deux solutions considérées comme bonnes par la fonction fitness afin d'obtenir une meilleure. Cette opération va donc générer une paire d'enfants C1 et C2. Cette marche à suivre est appliquée à l'ensemble de la génération d'origine. Un autre opérateur permet d'ajouter des mutations aux solutions, cela nous permet d'ajouter des éléments aléatoires à chaque génération, ainsi une mutation qui

permet à une solution de prendre un nouveau qui pourrait être une meilleure solution, et enfin on sélectionne nos solutions pour obtenir la nouvelle génération.

La figure ci-dessous tirée de l'article *« Algorithmes génétiques »* de Jean-Marc Alliot et Nicolas Durand, illustre très bien l'AG.

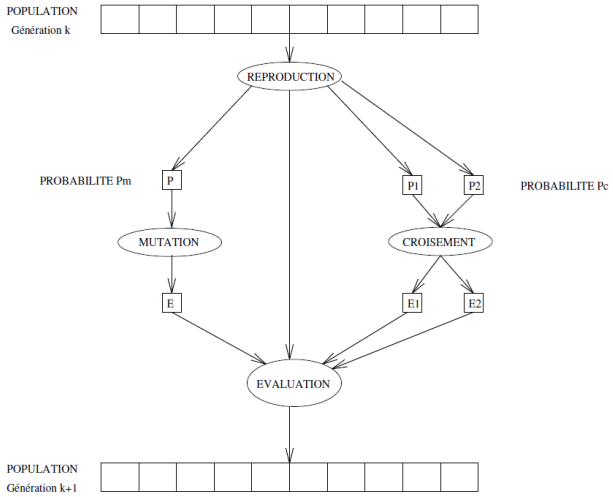

Figure 3 Fonctionnement des algorithmes génétiques (J.-M. Alliot et N. Durand, 2005)

Bien que ce système semble nous guider vers la solution optimale, la littérature scientifique nous met en garde sur l'importance de la première génération, une hétérogénéité est primordiale pour étudier un maximum de solutions optimales et non pas une partie de solution qui n'auront aucun rapport avec la réalité. En effet cet algorithme ne cherche pas toutes les solutions donc nous ne voulons pas qu'il élimine les individus qui pourraient devenir la solution optimale, car au moment de la sélection ces individus étaient moins intéressants que le reste des échantillons. D'où l'importance de la mutation, comme dans le processus de sélection naturelle, les mutations assurent le rôle d'aléatoire pendant le processus de sélection et permet à l'algorithme d'éviter de s'enfoncer dans une direction qui ne mènerait nulle part. (J.-M. Alliot et N. Durand, 2005)

1.4.3.2 L'algorithme de fourmis

L'algorithme de fourmis ou optimisation par colonies de fourmis peut être appliqué à différents problèmes de type NP et inspiré du comportement des fourmis pour trouver le chemin le plus court vers un point donné.

Les fourmis possèdent des phéromones, c'est leur moyen de communication, afin de pouvoir informer les autres fourmis du chemin qu'elle a pu emprunter, une fourmi va laisser sur son passage des phéromones qui guideront les autres fourmis, si une autre fourmi prend un autre chemin elle laissera des phéromones sur son passage. Prenons pour exemple qu'une fourmi

trouve de la nourriture elle laissera donc une trace pour aider ses congénères, qui a leur tour laisseront aussi une trace quand elles iront se déplacer, l'ACS est donc basé sur l'idée d'entraide commune qu'applique les fourmis de manière naturelle, comme pour le AG nous nous inspirons du vivant pour trouver des solutions.

Par conséquent, il n'est pas question d'utiliser de vraies fourmis, nous utiliserons des fourmis artificielles. Plusieurs modèles mathématiques existent, car on peut résoudre le problème sous la forme de la somme des chemins empruntés par la fourmi ou par les phéromones laissées sur le chemin qui relient 2 villes.

Une fourmi se situe dans une ville i elle décide d'aller vers une autre ville j, en fonction de deux paramètres, la visibilité notée η ainsi que le taux de phéromone associé à l'arc noté τ arrivé à la ville j, le choix de la prochaine ville se fait de manière aléatoire, cette condition aléatoire est définie par l'équation suivante :

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N_i^k(t)} [\tau_{il}(t)]^{\alpha} \cdot \eta_{il}^{\beta}} & si\ j \in N_i^k \\ 0 & sinon \end{cases}$$

En présentant N comme l'ensemble des villes à visiter. Les puissances α et β sont les pondérations respectives des phéromones et de la visibilité qui permettent de définir l'importance de ces paramètres, en effet si α est nul alors la fourmi devoir choisir son chemin uniquement sur la visibilité et donc ne pourra pas savoir les autres chemins possibles.

Pendant la première itération chaque fourmi va visiter toutes les villes N, elles ne peuvent pas visiter une ville plusieurs fois, elles ont donc une mémoire des villes. Elles choisiront le chemin avec le plus de phéromone, cela implique plusieurs fourmis qui sont déjà passées par ce chemin et donc c'est potentiellement le chemin le plus court depuis la ville i vers la ville j. Si la distance entre deux villes est identique, les fourmis choisiront le chemin avec le plus de phéromone.

Après plusieurs itérations l'alliance entre la distance d'une ville à une autre et le taux de phéromone présent sur un arc permet d'obtenir la solution optimale.

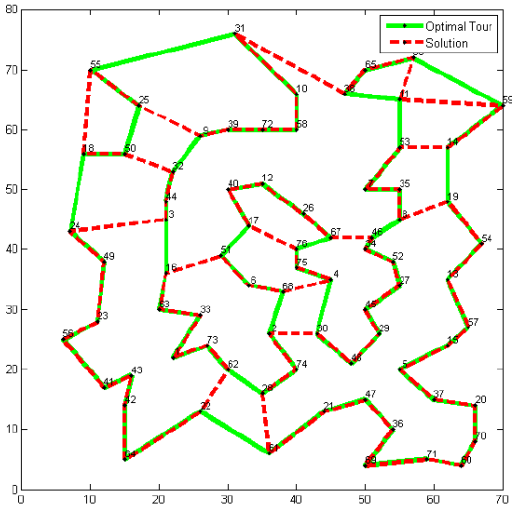

Figure 4 Solutions trouvées pour TSPlib (E. H. Hicham, Al., 2015)

Sur la figure ci-dessus nous avons la solution obtenue après 18 secondes de calcul, ici les fourmis trouvent une solution de 546 sachant que le meilleur tour est 538, nous avons donc une erreur de 1,02% ce qui est raisonnable. (E. H. Hicham, Al., 2015)

1.4.3.3 Le recuit simulé

Le recuit simulé est une méthode de recherche locale, il utilise le concept de voisinage duquel on explore l'espace de recherche point par point. Cette méthode est une métaheuristique qui commence avec une seule solution initiale comme point de départ et s'éloigne progressivement, en construisant une trajectoire dans le domaine admissible.

Le recuit simulé a été proposé par (kirkpatrick et al. 1983), s'inspire de la thermodynamique d'un système de particules alors que le recuit c'est un processus utilisé dans la métallurgie pour obtenir un acier de bonne qualité, donc lorsqu'on va augmenter la température du métal, on doit automatiquement casser la liaison entre les atomes, car ils vont être mobile puis on va les refroidir pour revenir à l'état solide. Si le refroidissement est rapide ce qui va donner un métal de mauvaise qualité et si le refroidissement est lent alors dans celui-ci on peut aboutir à une optimalisé meilleure et sans défauts. Par conséquent, le principe de la méthode applique itérativement l'algorithme de Metropolis parce que c'est le premier qui y a abordé cette idée, pour le but est d'engendrer une séquence de solution qui tend vers l'équilibre thermodynamique. En ajoutant le paramètre température, car il va nous permet d'échapper à l'optimum local dans le but d'atteindre une solution meilleure. Raúl Baños et al (Raúl Baños et al. 2013) ont appliqué une approche multi-objectifs parallèle basée sur le recuit simulé à des problèmes de routage de véhicules avec des fenêtres temporelles.

Dans l'algorithme de Metropolis, on part d'une configuration donnée, et on fait subir une modification aléatoire, si cette modification fait diminuer la fonction objective (ou énergie du système), alors on accepte cette solution, sinon, elle peut être accepté malgré qui n'améliore pas les solutions, mais a certaine probabilité, la formule de cette probabilité est comme suite :

P=e^(-Δf/T) (Avec le E= Énergie, et T= température)

Pour bien illustrer le fonctionnement de l'algorithme, on propose un organigramme de l'algorithme du recuit simulé appliqué au découpage aérien.

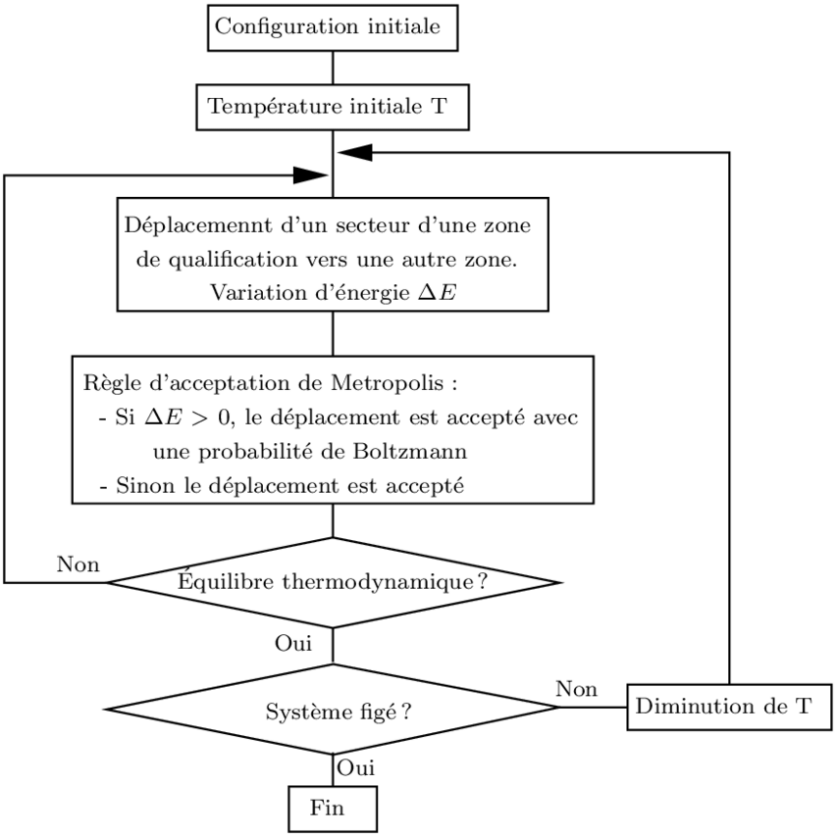

Figure 5 organigrammes de l'algorithme du recuit simulé appliqué au découpage aérien (Charles-Edmond Bichot 2004)

1.4.3.4 La recherche avec liste de tabous

Comme la méthode de recuit simulé, la recherche Tabou est aussi une méthode approximation utilisée pour résoudre d'un problème de très grande taille (NP difficile), il est basé sur la recherche locale, ça veut dire qu'à partir d'une solution initiale, on essaye d'explorer le voisinage du point (s) pour trouver la meilleure solution (s') du voisinage de (s) telle que f(s') < f(s) en cas d'une minimisation, en prenant exemple d'un problème de voyageur de commerce ou on veut minimiser la distance parcourue. Et aussi cette méthode est hybridée avec d'autres méthodes pour faire une optimisation efficace. C'est une technique qui a développé dans un

cadre particulier par Glover et Laguna (Glover & Laguna, 1998), et puis il était introduit d'une façon indépendante dans un cadre scientifique général par Hansen (Hansen, 1986).

La particularité de cette méthode, on peut dire qu'il ne doit pas être stagner sur un optimum local cependant il va continuer sa recherche pour atteindre l'optimum globale en effectuant des mouvements interdits ça veut qu'il accepte les solutions des mauvaises qualités en explorant dans une autre zone d'espace de recherche. L'idée de l'approche tabou est de rajouter un peu d'intelligence dans cette recherche pour éviter de passer deux fois au même endroit. Ainsi, ils ont proposé d'introduire une liste (Tabu list) qui stocke les solutions déjà testées pendant une certaine durée.

Le principe de la méthode est comme suit : on part d'abord d'une solution initiale et on va essayer de générer d'autre solution au fur à mesure des différentes itérations, donc lorsqu'on est dans la solution courant (s), l'algorithme explore d'autres solutions en appliquant un opérateur de voisinage, par la suite on va évaluer tous les points candidats et puis on sélectionne la meilleure parmi ces solutions de voisinages afin de les stocker temporairement dans la liste des tabous. Une fois la liste pleine, alors on va continuer l'exploration de l'espace en générant une nouvelle solution qui écrasera la solution ancienne afin d'échapper à l'optimum local. L'itération continuera jusqu'à ce que le critère d'arrêt soit satisfait, on peut préciser le nombre maximum d'itérations ou le temps de calcul maximum comme critère d'arrêt. Une fois la liste faite, on prend la valeur optimale comme la meilleure solution donnée par la méthode.

CHAPITRE 2

MÉTHODOLOGIE

Dans cette partie, nous allons présenter le problème de tournée de véhicule avec la contrainte de sa capacité. Pour commencer, on va présenter le problème sous forme d'un modèle mathématiques, expliquant toutes les contraintes. Par la suite, on effectue une analyse de la méthode de résolution adoptée, décrivant toutes les étapes. Puis on propose des situations afin de comparer ces solutions en se basant sur l'ensemble de données, la qualité de la solution, les paramètres et la durée d'exécution et aussi le taux occupation pour chaque camion. À la fin, nous discuterons l'outil utilisé pour la réalisation du projet.

2.1 Description du problème

Dans les problèmes de VRP avec contraintes de capacité, on a un seul dépôt, le fait qu'une station se trouve à un seul point, ce dernier force tous les véhicules à quitter la station et à revenir à la station (même point) une fois le tour terminé. Dans ce cas, l'entreprise doit déterminer la capacité et le nombre de véhicules pour répondre à la demande des clients. Avec une capacité insuffisante et trop peu de véhicules, la demande des clients ne peut être satisfaite et trop de véhicules entrainent des couts supplémentaires. Dans cette étude, l'objectif était de répondre adéquatement à la demande des clients avec le bon nombre de véhicules et la capacité des véhicules, tout en minimisant la distance totale.

Pour notre problème, on imaginera un scénario, où l'on a pour chaque ville, un ensemble de points de collecte de déchets, et un ensemble de points de dépôts de ces villes, on considère que dans chaque point i on a un nombre de déchets à ramasser en respectant la contrainte de la capacité. Le problème de tournées consiste à trouver un ensemble de routes réalisables de coût minimum de façon que chaque ville soit visitée une et une seule fois et collecter la quantité demandée.

je vais aussi illustrer une méthode de résolution le problème (CVRP) en utilisant un modèle mathématique, en décrivant toutes les étapes. D'abord, il y a des paramètres qui doivent être identifier, ensuite il y a la fonction objective et les contraintes que je vais expliquer dans l'exemple suivant :

- **Formulation du modèle :**

Ensembles :

G = Ensemble des nœuds

A = Ensemble des arc

K = Ensemble des camions

Indices :

i ∈ G = Nœud i appartnant a l'arc (G)

k ∈ K = Camion n appartenant à la flotte

Paramètres :

CAP : capacité de camion

q : demande de ville

Variables de décision :

x_{ijk}= variable de décision binaire égual a1 si le dechets de ville i est collectée par camion k, 0 sinon.

Fonction-objectif :

$$\text{Minimiser } Z = \sum_{k \in v} x_{ijk} d_{ij} \quad (1)$$

Contraintes :

$$\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ijk} = 1 \qquad \text{Pour i = 1, …, n (2)}$$

$$\sum_{j=1}^{n} \sum_{k=1}^{m} x_{ijk} = 1 \qquad \text{Pour j = 1, …, n (3)}$$

$$\sum_{(ij) \in A} x_{ijk} = \sum_{(ji) \in A} x_{ji}^{k} \qquad (4)$$

$$\sum_{j=1}^{n} x_{0jk} = 1 \qquad \text{Pour k = 1, …, m (5)}$$

$$\sum_{i=1}^{n} x_{i0k} = 1 \qquad \text{Pour k = 1, …, m (6)}$$

$$\sum_{(ji) \in A} x_{ij}^{k} q \leq \text{CAP} \qquad \text{Pour k = 1, …, m (7)}$$

$$x_{ijk} \in \{0, 1\} \text{ Pour chaque } 0 \leq i, j \leq n, \text{ pour k = 1, …, m (8)}$$

$$x_{ijk} = \begin{cases} 1 \text{ si } (i,j) \text{est parcouru par le véhicule K} \\ 0 \text{ sinon} \end{cases} \quad (9)$$

Dans l'équation contrainte (2) et (3), on assure que chaque demande va être servie une fois, la contrainte (4), assure la conservation des flux, l'équation (5) assure la contrainte du fait que si un chauffeur est désigné pour un véhicule, le véhicule doit commencer et se terminer à l'entreprise (6) et (7) sont les contraintes liées à la capacité qui doit être respecté (8) sont la définition des variables binaires et la dernière le (9) nous montre que chaque ville est bien affecté à un camion.

2.2 Modélisation

Dans cette partie, on a appliqué la méthode de recuit simulé pour modéliser notre problématique (CVRP). Cette méthode est avantageuse par la durée de calcul qui réalise, il permet aussi d'avoir une bonne qualité de solution avec la possibilité de traiter un problème de grande taille. Cependant, ici on va décrire les étapes clés utilisées pour la mise en œuvre de l'algorithme :

Étape 1 : D'abord, on génère une solutions initiales d'une facon aleatoire, tant que sa demande ne viole pas la contrainte de capacité du véhicule. Définissant également l'indice d'itération de k sur 0 (k = 0). Calculant la valeur d'énergie pour l'ensemble de chaque itération (fonction objective).

Étape 2 : Choisir une valeur de température initiale T0 et attribuer la valeur de température actuelle avec T = T0.

Étape 3 : Pour chaque itération, on génére une solution dans le voisinage de la solution actuelle par une transition (2-opt), et d'une manière systematique, en inversant une paire de villes, notant que la condition de la capacité doit être repercter.

Étape 4 : La solution voisin obtenue doit être optimal que la precedente, si non, on l'accepte selon le critère de Metropolis.

Étape 5 : Calculer la variation des fonctions de cout entre la solution générée S1 et les valeurs de la fonction objective de la solution S (courante) (dE = E (S1) - E(S)).

Étape 6 : Si S1 est meilleure que S (dE < 0), assignez la solution S1 à la solution S. Si S1 est pire que S, on accepte mais avec une probabilité d'acceptation (P = e (-dE /T)), donc si

e (-dE /T) > beta (beta est un nombre généré aléatoirement entre 0 et 1). Donc on a 2 choix soit en remplacer la solution S par la solution S1. Ou, on continue a cherché S dans le voisinage.

Étape 7 : Modifier la température T (progressivement réduite) selon la formule de l'équation : T = T0*exp(-t/tau).

Étape 8 : On fixe le critère d'arrêt jusqu'à la température minimale, afin de choisir la solution finale.

2.2.1 Détermination des paramètres

Pour avoir des meilleurs résultats, il faut choisir les bons paramètres, ils peuvent être définis d'une manière statique ou en fonction de la taille de problème. Notant que les paramètres sont des composants configurables et qui peuvent être modifiés pour concevoir les performances de l'algorithme. Notant que la valeur de température initiale doit être généralement assez grande et égale à la température actuelle.

2.3 Outils de résolutions

Pour la résolution des problèmes d'optimisation, plusieurs outils et solveurs sont généralement utilisés, l'un des plus utilisés est Matlab, c'est ce dernier qui a été utilisé pour la résolution de plusieurs problèmes similaires à celui-là, pour notre version, nous avons choisi de travailler avec Python, ce dernier étant le plus conseillé et simple à implanter. Il offre la possibilité de gérer plusieurs types de problèmes.

CHAPITRE 3

RÉSULTAT ET DISCUSSION

Dans ce chapitre, nous présenterons notre solution. Pour mieux interpréter les résultats obtenus, on s'est basée sur différentes études de cas, le premier cas où le programme prendra en considération un seul camion avec une capacité assez grande. Quant au deuxième cas deux camions utilisés avec une capacité identique. En ce qui concerne la troisième proposition le nombre de camions sera augmenté à quatre camions, chaque camion collecte une quantité de déchets.

3.1 Présentation des solutions obtenue

Tout d'abord, nous allons essayer de partir sur une bonne application de l'algorithme, en suivant les différentes étapes expliquées dans la méthodologie, afin d'obtenir des résultats optimaux, puisque nous n'avions pas les données d'exemples réels, nous avons dû mettre les données nous-mêmes. Finalement, nous nous sommes retrouvés avec trois situations :

- Situation N 1 : un seul camion
- Situation N 2 : deux camions
- Situation N 3 : quatre camions

Dans ce sens, un tableau est mis en place afin de montrer les résultats de chaque itération et expliquer la démarche suivie. Les paramètres d'entrée du programme sont le nombre de villes et le nombre de camions et la demande pour chaque ville (quantité de déchets) et la capacité de camion et la distance parcourue (distance euclidienne), pour avoir la solution optimale pour chacune de situation, la capacité et les nombres des camions vont subir des modifications.

On a fixé le nombre de villes à 10 pour que le temps de calcul lors de la phase de développement ne soit pas trop long.

N=10

Les paramètres de température sont exprimés :

T=250

T=T0

Tmin=35

tau=1000

La sélection des valeurs de température initiale et minimale pour déterminer la fin du recuit nécessite de multiples essais. En règle générale, la valeur de T0 doit être du même ordre de grandeur que la variation d'énergie. La valeur de (tau) doit être choisie suffisamment grande pour que la chute de température soit suffisamment lente, mais pas si grande qu'elle soit couteuse en calculs.

La valeur beta qu'on doit comparer avec la probabilité d'acceptation selon critère Métropolis, on va le fixer aléatoirement entre 0 et 1

beta= [0,1]

Itération initiale

k=0

Probabilité acceptation la mauvaise solution

P=exp(-dE/T)

La loi de refroidissement

T=T0*exp(-t/tau)

Notant, la formule qu'on appliquer pour calculer de distance euclidienne

$D = \sqrt{(x - x)2 + (y - y)2}$ (voir Annexe I).

Situation 1 :

Pour résoudre le problème de collecte de déchet plusieurs scénario ont été mis en place, comme premier scénario nous avons choisi d'employer un seul camion comme stratégie pour parvenir de telle sorte à collecter le maximum des déchets, tout en passant par un trajet soi-disant le plus optimal, commençant par le dépôt comme point de départ. L'objectif visé à travers cette proposition c'est en fait est de répondre aux besoins de toutes les villes.

Itération	Tour	Cout réaliser (la distance)	Distance courant	Nouveau distance	T	P=e (-dE/T)	Condition	Beta [0,1]	Décision
K=0	10-4-9-0-5-8-7-1-3-2-6-10	279	280	276	249.75		Respecté		acceptée
K=1	10-0-9-4-5-8-7-1-3-2-6-10	278	279	279	249.50		Respecté		acceptée
K=2	10-8-9-4-0-5-7-1-3-2-6-10	275	278	275	249.25		Respecté		acceptée
K=13	10-6-1-9-4-0-5-8-7-3-2-10	191	264	191	249		Respecté		acceptée
K=2440	10-6-2-3-1-5-9-4-0-8-7-10	269	262	269	34.96	P=0,81 beta<p Solution acceptée	Respecté	Beta=0,38	acceptée

Tableau 1 Résultat de la solution de l'algorithme recuit simulé dans la situation 1

Le tableau 1 montre les résultats obtenus dans la première situation. D'abord, on remarque que le cout augmente pour chaque itération. Cependant, l'algorithme peut accepter la solution mauvaise solution. L'application du critère de Métropolis permet de calculer la probabilité acceptation afin de la comparer avec la valeur beta choisi aléatoirement (beta= [0,1]). Dans le cas (k=2440), on va accepter cette solution puisqu'il s'agit de (beta< p). On constate aussi que la contrainte de la capacité du véhicule a été respectée dans toutes les itérations. Comme ci-mentionné dans le tableau, le critère d'arrêt a été fixé en température minimale de 35 degrés dans l'itération 2440, et on a réussi à avoir un cout rationnel qui s'élève à 191 km (voir Annexe III).

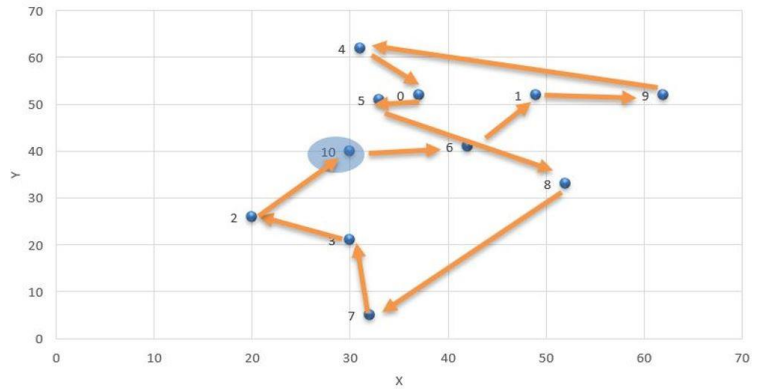

Figure 6 Itération de l'état final (situation 1)

Situation 2 :

Dans la deuxième situation, nous avons pu affecter deux camions. Pour avoir une solution optimale que les premières, nous avons mis des capacités identiques dans chacun des camions. La limite est de 1128 kg pour le premier camion, et 1128 kg pour le deuxième. Cependant, dans la première situation, la capacité a été fixée à 2255 kg.

Itération	Tour	Cout réaliser (la distance)	Distance courant	Nouveau distance	T	P=e (-dE/T)	Condition	Beta [0,1]	Décision
K=0	10-4-9-5-3-10 10-6-1-2-7-8-0-10	287	293	287	249.75		Respecté		acceptée
K=1	10-4-9-5-3-10 10-6-1-8-2-7-0-10	295	287	295	249.50	P=0,95 beta<p Solution acceptée	Respecté	Beta=0,95	acceptée
K=2	10-4-9-5-3-10 10-6-1-0-2-7-8-10	282	295	282	249.25		Respecté		acceptée
K=14	10-1-4-0-5-10 10-2-3-7-8-9-6-10	204	242	204	249		Respecté		acceptée
K=2772	10-2-5-9-0-10 10-7-4-3-6-1-8-10	324	286	324	34.96	P=0,33 p<beta Solution rejeter	Respecté	Beta=0,06	acceptée

Tableau 2 Résultat de la solution de l'algorithme recuit simulé dans la situation 2

Le tableau ci-dessus montre qu'il y a deux trajets possibles à réaliser, avant de retourner au dépôt principal, et chaque camion doit faire son propre trajet, tout en respectant la contrainte de la capacité. Comme dans la première situation, l'algorithme peut accepter la solution mauvaise mais avec certaines probabilités, dans le but d'atteindre à l'optimum global. On remarque aussi que le meilleur cout total de tout le circuit est de 204 km (voir Annexe III).

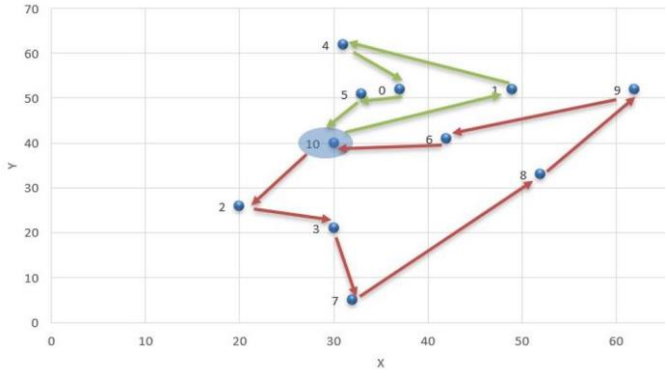

Figure 7 Itération de l'état final (situation 2)

Situation 3 :

Dans la troisième situation, pour mieux visualiser la problématique, d'autres camions ont été ajoutés. La capacité de chaque camion est fixée comme suit : le premier camion 564 kg, le deuxième camion 564 kg, le troisième camion 564 kg, et le dernier avec une capacité de 564 kg. Donc, on a utilisé quatre camions avec la même capacité.

Itération	Tour	Cout réaliser (la distance)	Distance courant	Nouveau distance	T	P=e (-dE/T)	Condition	Beta [0,1]	Décision
K=0	10-0-7-10 10-2-9-10 10-4-8-6-10 10-3-1-5-10	360	380	360	249.75		Respecté		acceptée
K=1	10-0-7-10 10-2-3-10 10-4-8-6-10 10-9-1-5-10	299	360	299	249.50		Respecté		acceptée
K=2	10-0-7-10 10-2-3-10 10-8-4-6-10 10-9-1-5-10	312	299	312	249.25	P=0,86 beta<p Solution acceptée	Respecté	Beta=0,79	acceptée
K=9	10-0-7-10 10-2-3-10 10-6-1-4-10 10-8-9-5-10	295	303	295	249		Respecté		acceptée
K=1548	10-7-0-10 10-8-4-10 10-2-3-6-10 10-9-5-1-10	341	383	341	34.96		Respecté		acceptée

Tableau 3 Résultat de la solution de l'algorithme recuit simulé dans la situation 3

Après le résultat du tableau 3, on constate que les quatre camions sont utilisés, et chacun d'eux a pris un trajet qui semble le plus optimal, il est ainsi primordial que la capacité soit respectée. À cet égard, notant que le résultat final obtenu, il est de 295 km (voir Annexe III).

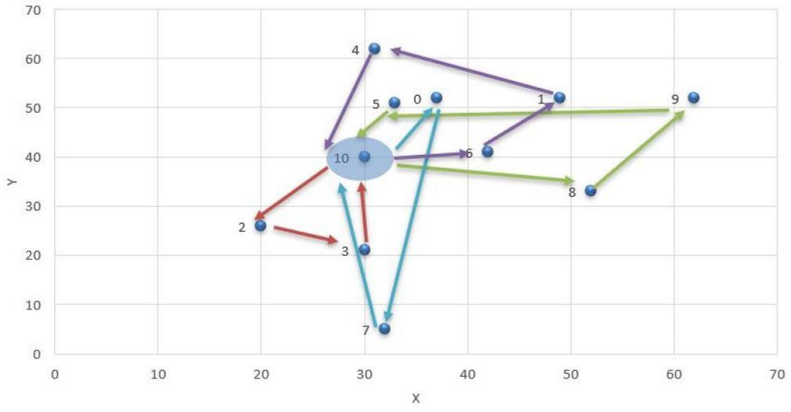

Figure 8 Itération de l'état final (situation 3)

3.2 Discussion

Dans les trois cas que nous proposons, le gestionnaire choisit automatiquement la meilleure option pour l'itinéraire qu'il doit prévoir pour chaque camion disponible dans la flotte. L'avantage d'utiliser une telle application (l'algorithme recuit simulé) est non seulement qu'elle permet de gagner du temps et de planifier la bonne direction, mais réduire le cout de transport pour améliorer le profit de l'entreprise et aussi de minimiser cout d'émission de CO2 pour éviter les impacts environnementaux, et la durée d'exécution.

Si on analyse bien la solution obtenue dans les trois situations, on trouve que le véhicule choisit à chaque itération, le trajet qui coute le moins (voir Annexe I). Si on prend par exemple les tableaux 1 et 2. Alors, quand la charge du véhicule est pleine, le prochain nœud que le véhicule ira visiter est le dépôt dans le but de respecter la condition de sa capacité.

Les résultats récapitulatifs sont répertoriés dans le tableau. Par conséquent, on peut conclure que la distance des véhicules varie en fonction du nombre de véhicules, de la capacité des véhicules. Plus le nombre de véhicules est élevé, plus la distance parcourue est longue. De même, on a observé que les distances diminuent à mesure que la capacité des véhicules augmente et que le nombre de véhicules diminue. Donc, après les résultats obtenus dans les quatre situations, on trouve que si on utilise un seul véhicule, cela nous donne des bonnes solutions comparant aux autres situations.

Situation	Nb de ville	Cap du véhicule	Nb de véhicule	Distance
1	10	[2255]	1	191
2	10	[1128, 1128]	2	204
3	10	[564, 564, 564, 564]	4	295

Tableau 5 les résultats récapitulatifs de l'algorithme

Après avoir examiné la distance parcourue, ici, on va calculer les taux d'occupation de chacun des véhicules, les résultats suivants ont été obtenus en considérant les problèmes 1 et 2 et 3.

Problème 1 Résultats de la solution :

1.Route du véhicule :

[10-6-1-9-4-0-5-8-7-3-2-10]

Nb de véhicule	Tour	Cap du véhicule	Demande sur véhicule	Taux occupation %
1	10-6-1-9-4-0-5-8-7-3-2-10	2255	2020	90

Tableau 6 Résultat de la solution du problème 1

Problème 2 Résultats de la solution :

1.Route du véhicule :

[10-1-4-0-5-10]

2.Route du véhicule :

[10-2-3-7-8-9-6-10]

Nb de véhicule	Tour	Cap du véhicule	Demande sur véhicule	Taux occupation %
1	10-1-4-0-5-10	1128	1073	95
2	10-2-3-7-8-9-6-10	1128	947	84

Tableau 7 Résultat de la solution du problème 2

Problème 3 Résultats de la solution :

1.Route du véhicule :

[10-0-7-10]

2.Route du véhicule :

[10-2-3-10]

3.Route du véhicule :

[10-6-1-4-10]

4.Route du véhicule :

[10-8-9-5-10]

Nb de véhicule	Tour	Cap du véhicule	Demande sur véhicule	Taux occupation %
1	10-0-7-10	564	503	89
2	10-2-3-10	564	554	99
3	10-6-1-4-10	564	514	91
4	10-8-9-5-10	564	449	80

Tableau 8 Résultat de la solution du problème 3

On en déduit donc qu'en ajoutant un véhicule supplémentaire à chaque scénario, on se retrouve à une situation de perte pour l'entreprise concernant le taux d'occupation, ainsi que la durée d'exécution en temps réel. La solution la plus confortable est donc de choisir un nombre limité de camions. Avec cette stratégie, nous pouvons obtenir de bons résultats.

Situation	Nb de ville	Cap du véhicule	Nb de véhicule	Distance	Tour	Durée d'exécution	GAP %
1	10	[2255]	1	154	10-9-1-4-5-0-6-8-7-3-2-10	0.20 s	19
2	10	[1128, 1128]	2	167	10-0-4-5-10 10-9-1-6-8-7-3-2-10	0.35 s	18
3	10	[564, 564, 564, 564]	4	213	10-0-5-10 10-2-10 10-6-8-7-3-10 10-9-1-4-10	0.55 s	26

Tableau 9 les résultats récapitulatifs de LINGO en appliquant la méthode exacte

Le tableau ci-dessus montre les résultats récapitulatifs obtenus si on utilise la méthode exacte, notant qu'on a gardé les mêmes données. Lors de l'évaluation des résultats, dans LINGO, le temps de résolution était cours pour les problèmes avec un petit nombre de villes, et le programme donnait des solutions optimales mieux que l'algorithme recuit simulé. Au fur et à mesure le nombre de villes ou de véhiculé augmentait, la solution prenait trop de temps et le programme devait être arrêté pendant un certain temps afin de trouver la bonne solution (voir le code LINGO dans l'annexe IV). Pour l'algorithme recuit simulé, on peut aussi aboutir à des solutions plus proches a l'optimal dans un temps raisonnable, si on applique plusieurs exécutions, ou bien si on décide à modifier les paramètres qu'on a travaillés.

CONCLUSION

Grâce à nos recherches, nous avons pu décrire les différentes approches d'optimisation pour résoudre des problèmes liés au domaine des transports. Par la suite, nous avons pu construire un modèle, permettant en même temps de prendre en compte les contraintes de capacité. Nous pourrions apprendre à modéliser et programmer des cas particuliers de modèles d'optimisation de transport tout en respectant toutes les contraintes fondamentales, telles que la gestion des flux et capacité du véhicule. Nous avons également pu introduire la notion de capacité dans le modèle, ce qui nous a permis de limiter les trajets pour chaque véhicule. Néanmoins, le modèle en question s'est testé selon trois situations différentes. Dans le premier scénario, un seul camion est utilisé avec une capacité reconnue, jugée suffisante pour répondre aux besoins de toutes les villes. Dans le second scénario, une stratégie de deux camions est proposée pour transporter les déchets, les deux camions se caractérisent par des capacités identiques. En ce qui concerne le troisième scénario, deux camions supplémentaires ont été ajoutés à la situation précédente, sachant que chaque des camions ont la même capacité.

Pour conclure, les résultats obtenus sont tirés sur la base de plusieurs facteurs à savoir la température initiale, la valeur bêta utilisée dans la sélection de nouveaux voisins et la température minimale à spécifier (critère d'arrêt). À travers plusieurs analyses et interprétations, on a pu relever la solution optimale dans chaque situation en fonction du cout total (distance totale), pour satisfaire toutes les demandes de toutes les villes (recueillir toutes les quantités de déchets pour chaque ville) et assurer une bonne gestion de la flotte.

ANNEXE I

Tableaux de données

X	Y
30	40
37	52
49	52
20	26
30	21
31	62
33	51
42	41
32	5
52	33
62	52

Tableau 10 les coordonnées X et Y pour chaque ville

0	14	22	17	19	22	11	12	35	23	34
14	0	12	31	32	12	4	12	47	24	25
22	12	0	39	36	21	16	13	50	19	13
17	31	39	0	11	38	28	27	24	33	49
19	32	36	11	0	41	30	23	16	25	45
22	12	21	38	41	0	11	24	57	36	33
11	4	16	28	30	11	0	13	46	26	29
12	12	13	27	23	24	13	0	38	12	23
35	47	50	24	16	57	46	38	0	34	56
23	24	19	33	25	36	26	12	34	0	21
34	25	13	49	45	33	29	23	56	21	0

Tableau 11 Matrice de distance euclidienne

Ville	Demande
0	304
1	81
2	315
3	113
4	320
5	287
6	89
7	250
8	73
9	188
10	0

Tableau 12 La demande ou bien la quantité de déchets à collecter pour chaque ville

ANNEXE II

Code Python

Les données entrant (input)

```
# choisir la situation qu'il faut travailler (1 ou 2 ou 3)
situation=3
if situation==2:
    # nombre camion choisi (situation 2)
    nc= 2
    # la capacité de camion (situation 2)
    cc=[1128, 1128]
elif situation ==3:
    # nombre camion choisi (situation 3)
    nc = 4
    # la capacité de camion (situation 3)
    cc = [564, 564, 564, 564]

elif situation==1 :
    # nombre camion choisi (situation 1)
    nc = 1
    # la capacité de camion (situation 1)
    cc = [2255]

else:
    pass
# nombre de ville a visiter
nv = 10
# la demande pour chaque ville
dv = [304, 81, 315, 113, 320, 287, 89, 250, 73, 188]
# la distance parcourue pour chaque ville
distance = [
        [0, 14, 22, 17, 19, 22, 11, 12, 35, 23, 34],
        [14, 0, 12, 31, 32, 12, 4, 12, 47, 24, 25],
        [22, 12, 0, 39, 36, 21, 16, 13, 50, 19, 13],
        [17, 31, 39, 0, 11, 38, 28, 27, 24, 33, 49],
        [19, 32, 36, 11, 0, 41, 30, 23, 16, 25, 45],
        [22, 12, 21, 38, 41, 0, 11, 24, 57, 36, 33],
        [11, 4, 16, 28, 30, 11, 0, 13, 46, 26, 29],
        [12, 12, 13, 27, 23, 24, 13, 0, 38, 12, 23],
        [35, 47, 50, 24, 16, 57, 46, 38, 0, 34, 56],
        [23, 24, 19, 33, 25, 36, 26, 12, 34, 0, 21],
        [34, 25, 13, 49, 45, 33, 29, 23, 56, 21, 0],
    ]
```

Le programme

```
import random
```

```
from random import randint
import numpy as np
import input as inp
import copy
from math import *
global n_v01
global n_v02
global n_c01
global n_c02
global nnc
nnc=inp.nc
# pour incrementer les index
def add1(sol):
    global n_v01
    global n_v02
    global n_c01
    global n_c02
    n_v01 += 1
    len_c1 = len(sol[n_c01])
    if n_v01 >= len_c1:
        n_v02 += 1
        n_v01 = n_v02
    len_c2 = len(sol[n_c02])
    if n_v02 >= len_c2:
        n_c02 += 1
        n_v02 = 0
        n_v01 = 0
    if n_c02 >= nnc:
        n_c01 += 1
        n_c02 = n_c01
# verification les index
def verf(sol):
    global n_v01
    global n_v02
    global n_c01
    global n_c02
    if n_c01 >= nnc :
        n_c01 = 0
        n_c02 = 0
        n_v01 = 0
        n_v02 = 0
    len_c1 = len(sol[n_c01])
    len_c2 = len(sol[n_c02])
    finnn=False
    if n_v01 >= len_c1:
        finnn = True
    if n_v02 >= len_c2:
        finnn = True
    if n_c02 >= nnc:
```

```
        finnn = True
    if n_c02 == 0 and n_v02 == 0 and n_c01 == 0 and n_v01 == 0:
        pass
    elif n_c01==n_c02 and  n_v01 == n_v02 :
        finnn = True
    return finnn
class individu:
# pour initialiser les fonctions
    def __init__(self):
        self.sol=self.get_new_indi()
        self.fo=0
# cette fonction, il nous aide a afficher le chemin
    def random_list_sum_fix(self,m, n):
        arr = [0] * m;
        for i in range(n):
            arr[randint(0, n) % m] += 1;
        return arr
# contrainte de la capacité du camion qu'il faut respecter
    def contrent_c(self,sol):
        i=0
        for rec in sol:
            s=0
            for rec2 in rec:
                s+=inp.dv[rec2]
            if s > inp.cc[i]:
                return  True
            #   print('s= ',s)
            i+=1
        return False
# trouver la nouveau chemin
    def get_new_indi(self):
        sec_v=list(range(inp.nv))
        fin=True
        sol = []
        while fin:
            sec_c = self.random_list_sum_fix(inp.nc, inp.nv)
            random.shuffle(sec_v)
            sol = []
            k = 0
            for i in range(inp.nc):
                lis = []
                for j in range(sec_c[i]):
                    lis.append(sec_v[k])
                    k += 1
                sol.append(lis)
            fin=self.contrent_c(sol)
        return sol
# calculer la fonction objectif d'un chemin courante trouver
    def calc_fo(self):
```

```
        sol=self.sol
        ds=inp.distance
        d=0
        l=0
        for i in sol:
            if len(i)!=0:
                k = i[0] + 1
                d += ds[0][k]
                for j in range(len(i) - 1):
                    k = i[j] + 1
                    l = i[j + 1] + 1
                    d += ds[k][l]
                d += ds[l][0]
        self.fo =d
# la solution voisin trouver en swapon des point de ville de la solutio
n courante
    def get_voisin(self):
        sol=copy.deepcopy(self.sol)
        def get_new_indi(self):
            sec_v = list(range(inp.nv))
        nnc = inp.nc
        fin = True
        while fin:
            finnn = True
            while finnn:
                add1(self.sol)
                finnn = verf(self.sol)
                if finnn == False:
                    break
            # if n_c02 == 0 and n_v02 == 0 and n_c01 == 0 and n_v01 ==
0:
            #     # print("////////////////////////////")
            #     break
            sol = copy.deepcopy(self.sol)
            v1 = sol[n_c01][n_v01]
            sol[n_c01][n_v01] = sol[n_c02][n_v02]
            sol[n_c02][n_v02] = v1

            fin = self.contrent_c(sol)
            # fin =True
        self.sol = sol
        self.calc_fo()
        return self
# appliquer l'algorithme recuit simulé
def recuit_simule(ind):
    N = 10
    t = 0
    T0 = 250
    T = T0
```

```
    Tmin = 35
    tau= 1000
    k = 0
    ind_courante = copy.deepcopy(ind)
    ind_courante.calc_fo()
    d_courante = ind_courante.fo
    global n_v01
    global n_v02
    global n_c01
    global n_c02
    n_c01 = 0
    n_c02 = 0
    n_v01 = 0
    n_v02 = 0
    inv=1
    # initialisation meilleur solution
    best_sol= ind_courante.fo
    best_ind = ind_courante
    while Tmin < T:
        if inv==1:
            print('_____________________________________')
            print("cout du chemin de distance du chemin courante = ", d
_courante, "chemin hamilto courante = ",
                  ind_courante.sol)
            inv=0
        beta = random.uniform(0,1)
        print("beta = ", beta)
        ind_voisine = ind.get_voisin()
        d_voisine= ind_voisine.fo
        # mise a jour (meilleur solution)
        if d_voisine < best_sol:
            best_sol = copy.deepcopy(ind_voisine.fo)
            best_ind = copy.deepcopy(ind_voisine)
        # affichage des resultat finaux
        print('          ********        ')
        print("k=", k)
        print(n_c01, n_v01, " // ", n_c02, n_v02)
        print("cout du chemin de distance du chemin voisin = ", ind_voi
sine.fo, "chemin hamilto voisin = ", ind_voisine.sol)
        if d_voisine < d_courante:
             d_courante =copy.deepcopy( d_voisine)
             ind_courante = copy.deepcopy(ind_voisine)
             inv = 1
             k += 1
        else:
            dE =  d_courante -d_voisine
            p = np.exp(dE / T)
            print(p)
            if beta > p or dE==0:
```

```
                pass
                #  nouveau solution est rejeter
            else:
                # on accepte la nouvelle solution malgre qu'elle amélio
re pas la fonction avec la
                #  probabilité de CM
                # print(dE)
                #print(T)
                # print(dE / T)
                print("p=", p)
                d_courante = d_voisine
                ind_courante = copy.deepcopy(ind_voisine)
                inv = 1
                k += 1
            # abaisser la temperature =======> refroidissment
            t += 1
            T = T0 * np.exp(-t / tau)
            print("T=",T)
    return best_ind
    # return ind_courante
if __name__ == "__main__":
    ind=individu()
    ind=recuit_simule(ind)
    ind.calc_fo()
    print('_____________________________')
    print(ind.fo)
    print(ind.sol)
```

ANNEXE III

Solution situation 1

cout du chemin de distance du chemin courante = 280 chemin hamilto courante = [[9, 4, 0, 5, 8, 7, 1, 3, 2, 6]]

beta = 0.5284644364538561

****

k= 0

0 1 // 0 0

cout du chemin de distance du chemin voisin = 279 chemin hamilto voisin = [[4, 9, 0, 5, 8, 7, 1, 3, 2, 6]]

0.9960079893439915

p= 0.9960079893439915

---

cout du chemin de distance du chemin courante = 279 chemin hamilto courante = [[4, 9, 0, 5, 8, 7, 1, 3, 2, 6]]

beta = 0.8081962205247489

****

k= 1

0 2 // 0 0

cout du chemin de distance du chemin voisin = 279 chemin hamilto voisin = [[0, 9, 4, 5, 8, 7, 1, 3, 2, 6]]

---

cout du chemin de distance du chemin courante = 262 chemin hamilto courante = [[6, 2, 3, 1, 0, 9, 4, 5, 8, 7]]

beta = 0.38764953242083733

****

k= 2440

0 7 // 0 4

cout du chemin de distance du chemin voisin = 269 chemin hamilto voisin = [[6, 2, 3, 1, 5, 9, 4, 0, 8, 7]]

p= 0.8187492320403977

_____

Meilleure solution

Distance = 191

Chemin = [[6, 1, 9, 4, 0, 5, 8, 7, 3, 2]]

Solution situation 2

cout du chemin de distance du chemin courante = 293 chemin hamilto courante = [[4, 9, 5, 3], [6, 0, 2, 7, 8, 1]]

beta = 0.3372323354773554

****

k= 0

1 5 // 1 1

cout du chemin de distance du chemin voisin = 287 chemin hamilto voisin = [[4, 9, 5, 3], [6, 1, 2, 7, 8, 0]]

p= 0.9731973598762469

_____

cout du chemin de distance du chemin courante = 287 chemin hamilto courante = [[4, 9, 5, 3], [6, 1, 2, 7, 8, 0]]

beta = 0.997589483375625

****

k= 1

1 3 // 1 2

cout du chemin de distance du chemin voisin = 303 chemin hamilto voisin = [[4, 9, 5, 3], [6, 1, 7, 2, 8, 0]]

beta = 0.9575554858916355

****

k= 1

1 4 // 1 2

cout du chemin de distance du chemin voisin = 295 chemin hamilto voisin = [[4, 9, 5, 3], [6, 1, 8, 2, 7, 0]]

0.9909667894766221

220.403711695854

p= 0.9909667894766221

---

cout du chemin de distance du chemin courante = 295 chemin hamilto courante = [[4, 9, 5, 3], [6, 1, 8, 2, 7, 0]]

beta = 0.11159736284451338

****

k= 1

1 5 // 1 2

cout du chemin de distance du chemin voisin = 282 chemin hamilto voisin = [[4, 9, 5, 3], [6, 1, 0, 2, 7, 8]]

---

cout du chemin de distance du chemin courante = 286 chemin hamilto courante = [[2, 5, 9, 0], [7, 8, 3, 6, 1, 4]]

beta = 0.03512781621845695

****

k= 2772

1 5 // 1 1

cout du chemin de distance du chemin voisin = 324 chemin hamilto voisin = [[2, 5, 9, 0], [7, 4, 3, 6, 1, 8]]

p= 0.33770188732232437

Meilleure solution

Distance = 204

Chemin = [[1, 4, 0, 5], [2, 3, 7, 8, 9, 6]]

Solution situation 3

1 1 // 1 0

cout du chemin de distance du chemin voisin = 380 chemin hamilto voisin = [[0, 7], [4, 9], [2, 8, 6], [3, 1, 5]]

beta = 0.38458294177660546

****

k= 0

1 0 // 2 0

cout du chemin de distance du chemin voisin = 360 chemin hamilto voisin = [[0, 7], [2, 9], [4, 8, 6], [3, 1, 5]]

---

cout du chemin de distance du chemin courante = 360 chemin hamilto courante = [[0, 7], [2, 9], [4, 8, 6], [3, 1, 5]]

beta = 0.06990298659244665

****

k= 1

1 1 // 3 0

cout du chemin de distance du chemin voisin = 299 chemin hamilto voisin = [[0, 7], [2, 3], [4, 8, 6], [9, 1, 5]]

---

cout du chemin de distance du chemin courante = 299 chemin hamilto courante = [[0, 7], [2, 3], [4, 8, 6], [9, 1, 5]]

beta = 0.5661764578975965

****

k= 2

2 1 // 2 0

cout du chemin de distance du chemin voisin = 312 chemin hamilto voisin = [[0, 7], [2, 3], [8, 4, 6], [9, 1, 5]]

p= 0.949180560759943

cout du chemin de distance du chemin courante = 383 chemin hamilto courante = [[7, 0], [8, 4], [2, 3, 6], [5, 1, 9]]

beta = 0.9295547157910178

****

k= 1548

3 1 // 3 0

cout du chemin de distance du chemin voisin = 341 chemin hamilto voisin = [[7, 0], [8, 4], [2, 3, 6], [1, 5, 9]]

Meilleure solution

Distance = 295

Chemin = [[0, 7], [2, 3], [6, 1, 4], [8, 9, 5]]

ANNEXE IV

Code LINGO

```
SETS:
! on définit le nombres de ville et leurs paramètre ;
! Q(i) représente la capacité demandé par chaque ville i,
! U(i) représente la charge du camion quand il quitte la ville i;
 CITY/1..11/: Q, U;
! la fonction CXC() a pour paramètre la matrice des distances et la variable
de décision binaire X;
! ici 'CITY' représente les colonnes et les lignes ;
 CXC( CITY, CITY): DIST, X;
ENDSETS
! on définit les valeurs numériques de nos constantes ;
DATA:
! Capacité des villes;
! ici la valeur 0 représente dépôts ;
 Q  = 0 304 81 315 113 320 287 89 250 73 188;
! Matrices des distances euclidiennes des 10 villes;
 DIST = ! To City;
0 14 22 17 19 22 11 12 35 23 3
14 0 12 31 32 12 4 12 47 24 25
22 12 0 39 36 21 16 13 50 19 13
17 31 39 0 11 38 28 27 24 33 49
19 32 36 11 0 41 30 23 16 25 45
22 12 21 38 41 0 11 24 57 36 33
11 4 16 28 30 11 0 13 46 26 29
12 12 13 27 23 24 13 0 38 12 23
35 47 50 24 16 57 46 38 0 34 56
23 24 19 33 25 36 26 12 34 0 21
34 25 13 49 45 33 29 23 56 21 0;

VEHCLR = 1;

! Capacité d'un camion;
 VCAP = 2255;
ENDDATA
! Notre fonction objectif;
! Ici nous cherchons à minimiser le coût de transport;
 MIN = @SUM( CXC: DIST * X);
```

```
! Pour chaque ville sans compter le dépôt qui est 1;
! #GT# retourne true si k>1 ;

 @FOR( CITY( K)| K #GT# 1:
! un camion ne peut pas voyager dans lui-même, c'est à dire qu'un camion ne
peut pas voyager d'une ville i vers cette même ville i;
 X( K, K) = 0;
! ici on vérifie la condition qu'un camion doit rentrer dans une ville;
 @SUM( CITY( I)| I #NE# K #AND# ( I #EQ# 1 #OR#
 Q( I) + Q( K) #LE# VCAP): X( I, K)) = 1;
! ici on vérifie qu'un camion qui circule entre k et j quitter forcément la
ville ;
 @SUM( CITY( J)| J #NE# K #AND# ( J #EQ# 1 #OR#
 Q( J) + Q( K) #LE# VCAP): X( K, J)) = 1;
 ! on définit les limites de la variable U() tel que Q(i) < U(K) < VCAP;
 @BND( Q( K), U( K), VCAP);
! Si la ville k est à la suite de la ville I on s'assure que la charge du
camion dans la ville I ne peut pas dépasser la charge du camion dans la ville
K;
 @FOR( CITY( I)| I #NE# K #AND# I #NE# 1:
 U( K) >= U( I) + Q( K) - VCAP + VCAP *
 ( X( K, I) + X( I, K)) - ( Q( K) + Q( I))
 * X( K, I);
 );
! Si la ville K est le premier client alors la charge du camion en K est
égale à la demande du ville en K;
 U( K) <= VCAP - ( VCAP - Q( K)) * X( 1, K);
! Si K n'est pas le premier ville sur la route alors la charge du camion en
K doit être supérieur ou égale à la charge en K + la charge de la ville tel
que l'arc(i,k) est vrai donc X(i,k) =1;
 U( K)>= Q( K)+ @SUM( CITY( I)|
 I #GT# 1: Q( I) * X( I, K));
 );
! Nous définissons la variable X comme une fonction binaire retourne 0 ou 1;
 @FOR( CXC: @BIN( X));
! on vérifie que l'on envoie suffisamment de véhicule depuis le dépôt ;
 @SUM (CITY(J)| J #GT# 1: X (1, J)) >= VEHCLR;
END
```

BIBLIOGRAPHIE

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. Management Science, 6(1), 80-91.

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., & Juan, A. A. (2014). Rich Vehicle Routing Problem: Survey. ACM Computing Surveys, 47(2), 32 (28 pp.). doi: 10.1145/2666003. Repéré à http://dx.doi.org/10.1145/2666003

Pawel Sitek & Jarosław Wikarek, 2019. "Capacitated vehicle routing problem with pick-up and alternative delivery (CVRPPAD): model and implementation using hybrid approach," Annals of Operations Research, Springer, vol. 273(1), pages 257-277, February.

F Benrahou, A Tairi, (2019). Solution of capacitated vehicle routing problem with invasive weed and hybrid algorithms January 2021International Journal of Industrial Engineering Computations 12(4):441-456 DOI:10.5267/j.ijiec.2021.4.002

Le Blanc et al. (2004). Collector Managed Inventory, a Proactive Planning Approach to the Collection of Liquids Coming from End-of-Life Vehicles CentER Discussion Paper No. 2004-22 24 Pages Posted: 24 Jun 2004

Lau, al., (2003). Vehicle routing problem with time windows and a limited number of vehicles August 2003European Journal of Operational Research 148(3):559-569 DOI:10.1016/S0377-2217(02)00363-6 SourceRePEc

Schneider al., (2014). The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations March 2014Transportation Science 48(4):500-520 DOI:10.1287/trsc.2013.0490

Agra, al., (2013). The robust vehicle routing problem with time windows March 2013Computers & Operations Research 2(3):273-287 DOI:10.1016/j.cor.2012.10.002A

Goetschalckx, Marc & Jacobs-Blecha, Charlotte. (1989). The vehicle routing problem with backhauls. European Journal of Operational Research. 42. 39-51. 10.1016/0377-2217(89)90057-X.

Min, Hokey. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. Transportation Research Part A: General. 23. 377-386. 10.1016/0191-2607(89)90085-X

Nagy, Gabor & Salhi, Said. (2005). Heuristic algorithms for single and multiple depot Vehicle Routing Problems with Pickups and Deliveries. European Journal of Operational Research. 162. 126-141. 10.1016/j.ejor.2002.11.003.

A. Azadeha et H. Farrokhi-Asl, (2019). The close–open mixed multi depot vehicle routing problem considering internal and external fleet of vehicles. Transportation Letters Volume 11, Issue 2, 2019, Pages 78-92

S.Geetha et al., (2012). Nested particle swarm optimisation for multi-depot vehicle routing problem January 2013International Journal of Operational Research 16(3):329 - 348 DOI:10.1504/IJOR.2013.052336

Dorigo, M., Birattari, M. & Stutzle, T. (2006). Ant colony optimization, in IEEE Computational Intelligence Magazine, 1, 4, 28-39, doi: 10.1109/MCI.2006.329691

J.-M. Alliot et N. Durand, «Algorithmes génétiques,» 14 march 2005. [En ligne]. Available: https://www.researchgate.net/publication/237333460_etiques.

E. H. Hicham, S. A. Haroun, B. Said et B. Jamal, «Comparaison de l'optimisation par colonies defourmis et des algorithmes génétiques pour la résolution du TSP,» mai 2015. [En ligne]. Available: https://www.researchgate.net/publication/275966237_Comparaison_de_l%27optimisation_par_colonies_de_fourmis_et_des_Algorithmes_Genetiques_pour_la_resolution_du_probleme_du_voyageur_de_commerce.

kirkpatrick et al. (1983)· Cité par 53464 — 13 May 1983, Volume 220, Number 4598. Optimization by. Simulated Annealing. S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi. 10

Baños, R., Ortega, J., Gil, C., Fernández, A., & Toro, F. (2013). A simulated annealing-based parallel multi-objective approach to vehicle routing problems with time windows, Expert Systems with Applications 40 (5), 1696–1707

Charles-Edmond Bichot, (2004). Optimisation par fusion et fission, application auprobl`eme du d´ecoupage a´erien europeen, LOG (Laboratoire d'Optimisation Globale) CENA/ENAC7, avenue Edouard Belin 31055 Toulouse cedex, France.

Glover & Laguna, (1998). Tabu search I January 1999Informs Journal on Computing 1(3) DOI:10.1287/ijoc.1.3.190 SourceDBLP Publisher: KluwerISBN: 978-0-7923-9965-0

Hansen, (1986).Hansen, R. M., 1986. Rumen digestive capability of zebu steers in wet and dry seasons. J. Range Manage., 39 (2): 139-140, https://doi.org/10.2307/3899286

Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics, 123(1-3), 487-512. doi: 10.1016/S0166-218X(01)00351-1. Repéré à http://dx.doi.org/10.1016/S0166-218X(01)00351-1