الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي
جامعة وهران للعلوم والتكنولوجيا محمد بوضياف
كلية الرياضيات و الاعلام الالي
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur Et de la Recherche Scientifique
Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF
Faculté des Mathématiques et Informatique

Département *: Informatique*

Mémoire de fin d'études

Problèmes de transport difficiles (Cas du VRP with Multiple Starts/Ends)

Pour l'obtention du diplôme de **Master**

Domaine : **Mathématiques – Informatique**

Filière : **Informatique**

Spécialité : **……SID……**

Présenté le :
Par :

- TAÏR AMINE

Jury	Nom et Prénom	Grade	Université
Président	Hamdaoui Sid Ahmed	MCA	USTO
Encadrant	Zennaki Mahmoud	MCB	USTO
Examinateur	Nemiche Mohamed Amine	MCB	USTO

2022/2023

Abstract

In this work, we address a Multiple Start and Multiple End VRPTW Vehicle Routing Problem with Time Window for transporting customers. The objective is to develop a heuristic that minimizes the total cost. Our algorithm searches for the best solution generated by the exit of a vehicle from a depot among the k nearest neighbors KNN (K-Nearest Neighbor) to the intermediate stations. For each intermediate station, each vehicle must visit the NN (NN Nearest Neighbor) stations within the defined time. Each vehicle in the fleet must visit no more than k intermediate stations and the entire fleet of vehicles must transport all customers.

The results obtained show that our method is effective and can even achieve a quality solution. The solutions are very satisfactory for large instances.

Keywords : Optimization, Nearest Neighbor Heuristic, K-Nearest Neighbor, Vehicle Routing Problem with time window

Résumé

Dans ce travail, nous traitons un problème de tournée de véhicules VRPTW-Multiple start/end (Multiple Start and Multiple End VRPTW Vehicle Routing Problem with Time Window) de transport des clients. L'objectif est d'élaborer une heuristique qui permet de minimiser le coût total. Notre algorithme recherche la meilleur solution engendrée par la sortie d'un véhicule d'un dépôt parmi les k plus proche voisin KNN (K-Nearest Neighbor) à destination des station intermédiaires. Pour chaque station intermédiaire, chaque véhicule doit visiter les stations les plus proches NN (NN Nearest Neighbor) dans les délais définis. Chaque véhicule de la flotte ne doit pas visiter plus de k plus proches stations intermédiaires et l'ensemble de la flotte de véhicules doit transporter tous les clients.

Les résultats obtenus montrent que notre méthode est efficace et peut même atteindre une solution de qualité. Les solutions sont très satisfaisantes pour les instances de grandes tailles.

Mots clés : Optimisation,Heuristique du plus proche voisin, K-plus proche voisin, Problèmes de transport difficiles avec fenêtre de temps.

Remerciement

Je tiens à exprimer ma profonde gratitude envers mon encadrant, Mr M.Zennaki, pour son soutien et ses précieux conseils tout au long de notre travail.

Je remercie également le Mr S.A.Hamdaoui et Mr M.A.Nemmiche pour avoir accepter examiner ce travail.

Je souhaite également remercier l'ensemble du corps enseignant du département d'informatique pour leur enseignement de qualité.

Enfin, je veux remercier ma familles et mes proches pour leur amour, leur soutien et leur encouragement constants.

Liste des acronymes

Acronyme	Signification
NN	Nearest Neighbor
KNN	K-Nearest Neighbor
MDVRP	Multi-Depot Vehicle Routing Problem
VRPSTT	VRP with Stochastic Travel Times
VRP	Vehicle Routing Problem
SDVRP	Vehicle Routing Problem with Split Delivery
VRPSD	VRP with Stochastic Demands
VRPSC	VRP with Stochastic Customers
SVRP	Stochastic Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
CVRP	Capacitated Vehicle Routing Problem
CDVRP	Capacitated Distance-Constrained Vehicle Routing Problem
PVC	Problème de Voyageur de Commerce
TSP	Traveling Salesman Problem
TW	Time Window

Sommaire

Introduction Générale

L'optimisation des chaînes logistiques occupe une place importante dans les entreprises industrielles et commerciales. Pour les compagnies de transport, le recours aux méthodes d'optimisation a pour but la construction de tournées de véhicules pour desservir des clients avec un ensemble de véhicules tout en minimisant le coût totale. La construction de tournées de véhicules fait partie d'une classe de problèmes difficile à résoudre. Ces problèmes appelés aussi problèmes de routage de véhicules (Vehicle Routing Problem, VRP) est un problème d'optimisation combinatoire et de recherche opérationnelle. Le problème de routage de véhicules a été largement étudié. Les principales motivations de l'étude du VRP sont d'une part la difficulté de sa résolution et d'autre part ses nombreuses applications pratiques. Initialement développées sur des problèmes simples (nombre de clients limité, distances euclidiennes), la programmation mathématique ou la théorie des graphes, ont été proposées et sont très efficaces pour répondre à ces problèmes combinatoires. Cependant, de nombreuses variantes ont été proposées pour représenter le réalisme des instances étudiées. Mais, la majorités des méthodes ne sont pas compatibles avec l'ensemble des nombreuses variantes existantes, et perdent en efficacité avec l'augmentation de la taille des instances. Souvent, la taille des problèmes est très grande (plusieurs centaines de clients à visiter) et l'utilisation de méthodes exactes est souvent inappropriée en termes de temps de résolution. Des méthodes heuristiques et métaheuristiques ont donc été proposées pour obtenir de bonnes solutions dans des temps de résolutions acceptables.
Dans notre projet nous allons présenter une variante du problème VRP qui est le CVRPTW with multi start/end (Capacited Vehicle Routing Problem Time Window with multiple starts/ends), ce problème est une variante du VRP ou on a ajouté les contraintes de capacité, de fenêtre de temps et de multiples départs et arrivées. Pour ce problème, nous allons le définir et le modéliser afin de le résoudre avec un solveur mathématique (CPLEX) et étudierons les resultats du solveur pour des instances de petite taille. Comme le solveur ne peut traiter les instances de grande taille nous élaborerons une heuristique basée sur le plus proche voisin (Nearest Neighbor ou NN). Nous commençons par l'implémentation d'un algorithme NN naif, puis nous étendons notre modèle aux contraintes de fenêtres de temps (Time Window ou TW). Enfin nous proposons une amélioration de l'heuristique NN dite k-NN afin d'améliorer la qualité des solutions. A chaque étape les résultats des approches heuristiques développées sont comparés avec ceux obtenus par le solveur mathématique CPLEX sur des instances de petite taille et étudions les resultats de l'heuristique sur des instances de grande taille que le solveur ne peut traiter. ce manuscrit est organisé comme suit :

1. Dans le premier chapitre intitulé État de l'art, nous présenterons la defini-

tion du problème VRP et ses différentes variantes ainsi que les méthodes de résolutions existantes et particulièrement les méthodes à base d'heuristiques.

2. dans le chapitre deux nous détaillerons la démarche pour élaborer l'algorithme KNNTW.
3. pour le troisième chapitre nous le consacrons pour des expérimentations et interprétations des résultats.
4. Et nous terminons ce manuscrit par une conclusion générale.

Chapitre 1

État de l'art

1.1 Introduction

Les problèmes de transport se trouvent au centre des problématiques stratégiques de l'économie moderne. Le problème de tournées de véhicules (VRP : Vehicle Routing Problem) est une combinaison des problèmes du voyageur de commerce à prendre en charge dans le domaine de la logistique des entreprises commerciales. Il a fait l'objet de nombreux travaux de ses variantes dans la littérature. Le recours à la recherche opérationnelle et aux différentes méthodes de résolution des problèmes de tournées constitue un défi considérable étudié dès les années 1960. Des publications sont régulièrement proposées sur des variantes du problème.

Dans ce chapitre nous présentons dans la première section un rapide état de l'art sur les problèmes de tournées de véhicules suivi des méthodes de résolution. Une troisième section est consacrée aux méthodes heuristiques existantes et algorithmes de la résolution qui introduisent les notions de voisinage et plus proche voisin. Ces travaux vont nous aider dans notre démarche pour développer une heuristique.

1.2 Problèmes des tournées des vehicules : Définition et variantes

1.2.1 Définition

Le problème de tournées de véhicules (VRP : Vehicle Routing Problem) a été introduit pour la première fois en 1959 par Dantzig et Ramser [5]. Le VRP est un des problèmes qui ont sollicité l'attention de plusieurs équipes de recherches vu son importance et son utilité dans la résolution de nombreux problèmes rencontrés dans la vie quotidienne. Le VRP est un problème d'optimisation combinatoire, il

appartient à la classe des problèmes NP-Difficiles [13] où il n'existe pas d'algorithme permettant la résolution d'un problème donné en un temps polynomial.
Le principe du problème de tournées de véhicules consiste à trouver les meilleurs trajets à parcourir par une flotte de M véhicules afin de visiter N clients (sites) pour des raisons de collecte ou de livraison ou de transport de clients. Tous les véhicules commencent et terminent leurs tournées à un site central appelé dépôt. Chaque client est affecté à une tournée et il doit être visité une seule fois par un seul véhicule. La capacité de transport de ce dernier pour une tournée est limitée et elle ne doit pas être dépassée. Tous les trajets proposés doivent respecter les différentes capacités des véhicules et satisfaire les quantités demandées par les clients ou à livrer à ces derniers[8].

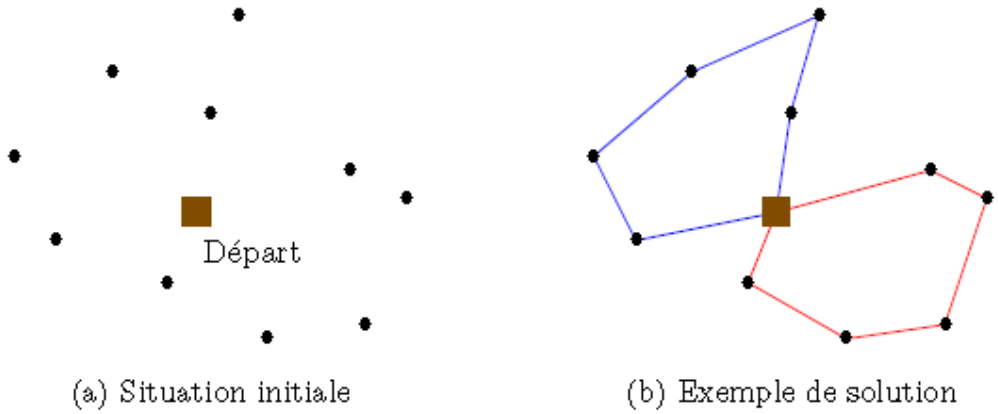

FIGURE 1.1 – Exemple de problème VRP

En général, l'objectif principal des problèmes de tournées de véhicules est de minimiser le coût (distance) total de parcours des trajets par un nombre minimum de véhicules. Outre son objectif principal, le VRP impose d'autres contraintes à respecter parmi les quelles :

1. Chaque client doit être servi une seule fois par un et un seul véhicule.
2. Le client visité doit impérativement être quitté.
3. Le nombre de véhicules est limité.
4. Les véhicules doivent tous commencer leurs tournées à partir d'un dépôt.
5. Les sous tours sont interdits, i.e. tous les véhicules reviennent au dépôt et non pas aux autres sites.

L'ajout, la modification ou l'élimination d'une ou des contraintes du VRP peut nous mener vers d'autres variantes qui sont essentiellement apparues suite aux activités des chercheurs qui travaillent de plus en plus sur les problèmes de transport et de distribution que rencontrent les sociétés [8].

1.2.2 Historique

En mathématiques et en économie, la théorie du transport est le nom donné à l'étude du transfert optimal de matière et à l'allocation optimale de ressources. Le problème a été formalisé par le mathématicien français Gaspard Monge en 1781. D'importants développements ont été réalisés dans ce domaine pendant la Seconde Guerre mondiale par le mathématicien et économiste russe Léonid Kantorovitch. Par conséquent, le problème dans sa forme actuelle est parfois baptisé problème (du transport) de Monge-Kantorovitch. Le problème de tournées de véhicules connu sous le nom de Véhicule Routing Problem (VRP)est une extension du problème du voyageur de commerce, connu également sous le nom de Travelling Salesman Problem (TSP). Nous allons donc dans un premier temps définir le problème du voyageur de commerce, puis nous verrons son extension Problème de Tournées de Véhicules.[10]

1.2.3 Problème de Voyageur de Commerce(PVC) :

Il est dit aussi Traveling Salesman Problem (TSP) ou PVC. C'est un problème classique de la recherche opérationnelle. Le voyageur du commerce désire visiter un certain nombre de villes (ou clients). Il doit débuter et finir sa tournée par la même ville de départ, en visitant chacune des autres villes, une et une seule fois, l'objectif étant de trouver la tournée qui minimise la distance totale parcourue. Le PVC permet la formulation de nombreuses situations réelles. La difficulté de ce problème est de trouver l'ordre dans lequel chacun des clients sera visite, en minimisant un certain critère (temps ou cout du parcours, ou bien longueur totale parcourue . . .).[10]

1.2.4 Les Variantes du VRP

Le problème de tournées de véhicules est une généralisation du TSP, qui s'en distingue selon les auteurs par le nombre de véhicules, et/ou la contrainte de capacité et/ou la contrainte d'autonomie. La contrainte de capacité spécifie que la somme des demandes des clients regroupées en une tournée ne doit pas excéder la capacité des véhicules, sachant que la demande de chaque client est supposée inférieure à la capacité du véhicule, et que tous les véhicules sont ici de même capacité (flotte homogène).

La contrainte d'autonomie indique que la durée maximale entre le départ d'un véhicule et son retour au dépôt est limitée par une borne maximale.
Cette divergence de points de vue entre auteurs, a fait naître plusieurs définitions contradictoires du VRP dans la littérature, qui ont conduit à identifier 4 types de VRP, correspondant à quatre des cas présentés ci-dessous, mais qui restent discutables :

1. VRP de base : le VRP est distingué du TSP uniquement par le fait que plusieurs véhicules soient disponibles. Mais alors par quoi se distingue-il du mTSP, dans lequel « m » signifie également la présence de plusieurs voyageurs.
2. Capacitated Distance-Constrained VRP ou CDVRP : une contrainte de capacité des véhicules et une contrainte d'autonomie des véhicules séparent le VRP du TSP.
3. Le Capacitated Vehicle Routing Problem (CVRP) est l'une des variantes pour lesquelles il existe des définitions contradictoires. Pour certains elle se caractérise par la présence de la contrainte de capacité et l'absence de contrainte d'autonomie. Pour d'autres c'est un problème à plusieurs véhicules, avec contraintes de capacité et d'autonomie, et une seule visite par client.
4. Distance Constrained VRP : le quatrième cas est à nouveau une définition contradictoire avec les discussions précédentes, par exemple, elle distingue le VRP du TSP uniquement par la contrainte d'autonomie.
5. Le Vehicle Routing Problem with Time Windows (VRPTW), ou VRP avec fenêtres de temps, quant à lui, spécifie que chaque client a une fenêtre de temps. Celle ci est un intervalle de temps au cours duquel son service (par exemple le chargement ou le déchargement de marchandises) doit être accompli. Un véhicule peut arriver plus tôt, mais il doit attendre jusqu'à ce que le service soit possible. S'il arrive plus tard, le service ne peut pas être rendu et le client correspondant ne sera jamais satisfait. Dans ce cas, le problème est dit "à contraintes dures" (Hard time window constraints). L'objectif du problème est alors de minimiser le nombre de véhicules et la distance totale parcourue pour servir les clients sans violer les contraintes de fenêtres de temps. Dans les modèles dits à contraintes molles (soft time windows constraints), les véhicules peuvent servir le client en dehors de sa fenêtre de temps mais au prix d'une certaine forme de pénalité. Bien que cette définition du VRP à fenêtre de temps soit la plus répandue, il est possible de rencontrer dans certaines variantes du problème des fenêtres de temps relatives aux horaires d'ouverture du dépôt ou aux horaires de travail du personnel.

6. Le Stochastic Vehicle Routing Problem (SVRP), est un VRP dans lequel au moins un élément du problème est aléatoire. Les trois cas les plus connus de SVRP (Cordeau et al., 2005) sont :
   (a) Le VRP with Stochastic Customers (VRPSC), où Pi représente la possibilité qu'un client i émette une demande ;
   (b) Le VRP with Stochastic Demands (VRPSD), où la demande d'un client est une variable aléatoire ;
   (c) Le VRP with Stochastic Travel Times (VRPSTT), où le temps de service (si) d'un client (i) et le temps de trajet (tij) d'un arc (i, j) sont des variables aléatoires.
7. Le Multi-Depot Vehicle Routing Problem (MDVRP), comporte plusieurs dépôts dans lesquels sont localisés les véhicules. Chaque tournée d'un véhicule doit commencer et finir au même dépôt. De plus, chaque client doit être visité exactement une fois par l'un des véhicules situés dans les dépôts indiqués par ce client.
8. Le Vehicle Routing Problem with Split Delivery (VRPSD ou SDVRP) remet en cause deux contraintes du problème classique :
   (a) Chaque client peut être visité plus d'une fois si cela est nécessaire. Autrement dit, la demande d'un client peut être divisée sur plusieurs tournées.
   (b) la demande de chaque client peut alors être plus grande que la capacité des véhicules.[12]

Classifications

Les critères de classification ont classé les VRP selon les caractéristiques des données si le problème est :

1. statique si toutes les données sont parfaitement connues dès le début de la résolution,
2. stochastique si au moins un élément du problème est aléatoire,
3. dynamique si l'une au moins des données nécessaires à la planification dépend du temps. Dans ce cas, toutes les données ne sont pas disponibles au début de la résolution, de nouvelles données se présentent aléatoirement en cours de résolution et doivent être intégrées dans celle-ci.[12]

1.3 Méthodes de résolution du VRP

La résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales : la classe de méthodes exactes et la classe de méthodes approchées. L'hybridation des méthodes de ces deux classes a donné naissance à une pseudo classe qui englobe des méthodes dites hybrides.
Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire. C'est la raison pour laquelle, elles sont beaucoup plus utilisées pour la résolution de problèmes de taille moyenne. La nécessité de disposer d'une solution de bonne qualité (semi optimale) avec un coût de recherche (temps de calcul et espace mémoire) raisonnable a excité les chercheurs à proposer un autre type de méthodes de résolution de problèmes, communément connues par les méthodes approchées. Ces dernières constituent une alternative aux méthodes exactes. En fait, elles permettent de fournir des solutions de qualités accéptables en un temps de calcul raisonnable. De nombreuses méthodes approchées ont été proposées. Elles sont plus pratiques pour la résolution de problèmes difficiles ou de problèmes dont on cherche des solutions en un bref délai. Ces méthodes sont souvent classées en deux catégories : des méthodes heuristiques et des méthodes métaheuristiques. Les méthodes dites heuristiques sont des méthodes spécifiques

à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures.[6]

1.3.1 Les méthodes exactes

L'intérêt des méthodes exactes réside dans le fait qu'elles assurent l'obtention de la solution optimale du problème traité. En fait, elles permettent de parcourir la totalité de l'ensemble de l'espace de recherche de manière à assurer l'obtention de toutes les solutions ayant le potentiel d'être meilleures que la solution optimale trouvée au cours de la recherche. Cependant, les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles. De ce fait, la complexité de ce type d'algorithme croit exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions «objectif»s et/ou critères.
Il existe de nombreux algorithmes exacts y compris l'algorithme du simplexe, la programmation dynamique, l'algorithme A*, les algorithmes de séparation et évaluation (Branch and Bound, Branch and Cut, Branch and Price et Branch and Cut and Price), les algorithmes de retour arrière (Backtracking).[6]

1.3.2 Les Heuristiques

Une heuristique (règle heuristique, méthode heuristique) est une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre type de dispositif qui limite considérablement la recherche de solutions dans des espaces problématiques importants. Les heuristiques ne garantissent pas des solutions optimales. En fait, elles ne garantissent pas une solution du tout. Tout ce qui peut être dit d'une heuristique utile, c'est qu'elle propose des solutions qui sont assez bonnes la plupart du temps.[2]
Une méthode heuristique (ou simplement une heuristique) est une méthode qui aide à découvrir la solution d'un problème en faisant des conjectures plausibles mais faillible de ce qui est la meilleure chose à faire.[2]
Une heuristique est une règle d'estimation, une stratégie, une méthode ou astuce utilisée pour améliorer l'efficacité d'un système qui tente de découvrir les solutions des problèmes complexes.
Les heuristiques sont des règles empiriques et des morceaux de connaissances, elles sont des méthodes approchées et intelligentes mais ne garantissent pas d'obtenir

la solution optimale. Cependant, leur faible temps d'exécution les rend très attractives dans le milieu industriel.

1.3.3 Les métaheuristiques

Dans la vie pratique, on se retrouve souvent confronté à des problèmes de différentes complexités, pour lesquelles on cherche des solutions qui satisfont deux notions antagonistes : la rapidité et la qualité. Devant le coût de recherche prohibitif des méthodes exactes (particulièrement avec des problèmes de grande taille) et la spécificité des heuristiques au problème donné, les méta-heuristiques construisent une solution moins exigeante. En fait, elles sont applicables sur une grande variété de problèmes d'optimisation de différentes complexités. En outre, elles permettent de fournir des solutions de très bonne qualité (pas nécessairement optimales) en temps de calcul raisonnable. La majorité des méta-heuristiques sont inspirées des systèmes naturels, nous pouvons citer à titre d'exemple : le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes révolutionnaires et les algorithmes génétiques qui sont inspirés des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence d'essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou et l'algorithme d'optimisation par coucou qui s'inspirent du comportement social de certaines espèces évoluant en groupe. L'ensemble des métaheuristiques proposées dans la littérature sont partagées en deux classes : des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions.

1.3.4 Les méthodes heuristiques et/ou méthodes constructives pour les problèmes de transport ou pour le VRP

Heuristique Plus proche voisin

Cette heuristique (NN Nearest Neighbor) est la plus intuitive : on commence une tournée en livrant le client le plus proche du dépôt. Ensuite, la tournée est complétée en livrant le client le plus proche de cette nouvelle position, et ainsi de suite. Lorsque la capacité du véhicule est atteinte, le véhicule retourne au dépôt et une nouvelle tournée est créée. L'algorithme se termine lorsque tous les clients ont été affectés à une tournée. Cette méthode est très facile à coder, et rapide à s'exécuter. Cependant, la solution obtenue est acceptable mais est de mauvaise qualité [9].

L'algorithme de Clarke and Wright

En 1964, Clarke et Wright [7] ont développé un algorithme pour résoudre le CVRP et est toujours sollicité dans les algorithme récents. cet algorithme commence par assigner une tournée par client, ensuite, les tournées seront fusionnées jusqu'à ce que aucune fusion réduisant le coût de la solution ne soit possible. Cette heuristique est un algorithme d'échange dans le sens où à chaque étape une série de visites est échangée pour un meilleur ensemble de visites.

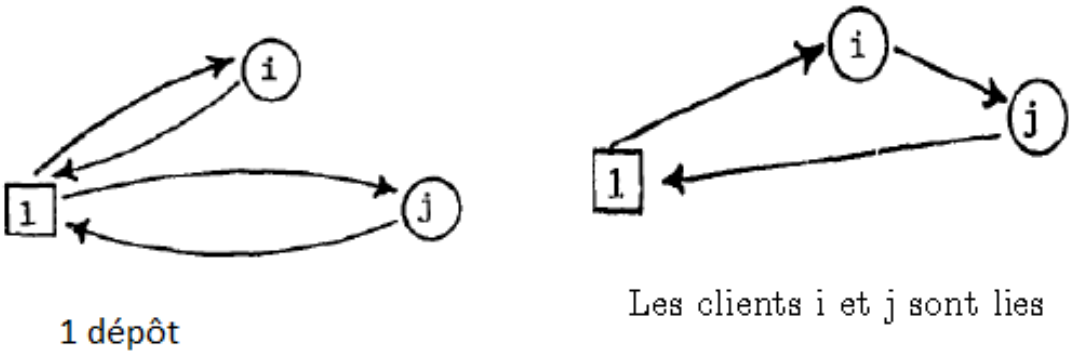

FIGURE 1.2 – Exemple de fusion de deux tournées par l'algorithme de Clarke Wright

Méthodes à recherche locale et notion de voisinage

Les solutions créées à partir d'un algorithme constructif sont obtenues rapidement, mais sont généralement de mauvaise qualité [9]. Souvent, quelques perturbations suffisent pour améliorer une solution. Les méthodes à recherche locale étudient une ou plusieurs familles de perturbations de la solution courante. L'ensemble des solutions accessibles via une perturbation est appelé voisinage. Les perturbations sont appelées mouvements (mooves), mais sont parfois nommées voisinage [3][9]. Disposer de plusieurs voisinages est donc un excellent moyen d'améliorer la qualité des solutions obtenues [9]. En contrepartie, chaque voisinage supplémentaire augmente le temps d'exécution du solveur. pour ce type de méthodes on distingue différents types de voisinages :

1. Voisinages intra-route : Ces voisinages représentent les modifications n'impactant qu'une tournée à la fois. Ils modifient donc uniquement l'ordre dans lequel les clients sont servis au sein d'une tournée.
2. Voisinages inter-routes : Les voisinages inter-routes modifient au moins deux tournées. Ils permettent en particulier d'améliorer l'affectation clients par véhicules.

la plupart des méthodes à recherche locale utilisent les mêmes voisinages. Les Voisinages fréquents adoptent principalement deux façons de modifier une solution. La plus connue consiste à retirer k arêtes d'une solution et à les remplacer par k autres arêtes. Ce mouvement est intitulé k-opt. Le temps d'exécution augmentant avec le paramètre k, le k-opt le plus utilisé est le 2-opt, très utile pour réduire le nombre de croisements, aussi bien entre tournées, qu'au sein d'une tournée [9].

Les méthodes à recherches locales ont des temps d'exécution très élevés. Ainsi, il est toujours intéressant de voir s'il est possible de réduire la taille du voisinage, afin de réduire les calculs inutiles. La manière la plus connue consiste à trier les arêtes partant du client i, et à ne conserver que les k plus courtes [9] ou celle qui est dévelloppé dans [4] où le choix du paramètre k a été étudié.

Résolution avec algorithme des plus proche voisin KNNA

Principe de l'algorithme

Dans cette étude utilisant un algorithme nommé KNNA(K-Nearest Neighbor Algorithm)[11], les solutions sont générées respectivement en plusieurs étapes. Au début, chaque véhicule part du dépôt (noeud numéro 0) et retourne au même dépôt pour terminer son itinéraire. Pour le premier noeud de tous les véhicules est donc le dépôt. Le deuxième noeud de chaque véhicule sera choisi au hasard. Les autres noeuds de l'ensemble de véhicules seront déterminés en appliquant les règles du plus proche voisin. Les règles du plus proche voisin sont les suivantes : le noeud suivant doit être le noeud le plus proche parmi tous les autres noeuds et chaque noeud est visité une seule fois (donc par un seul vehicule).

Fonction objective

La "fonction Fitness" est chargée d'évaluer et de restituer un nombre entier positif, qui reflète le degré de perfection du résultat : plus le nombre est petit, meilleur est le résultat. Les valeurs Finess sont ensuite utilisées dans le cadre d'une procédure de comparaison entre la NewSol générée et la BestSol stockée pour identifier les résultats possibles qui passeront à une autre itération et ceux qui seront ignorés. La distance de chaque itinéraire (d_{route}) est calculée en collectant la distance totale entre le dépôt (noeud central) et le dernier noeud pour chaque véhicule. Ensuite, les distances totales de tous les véhicules sont additionnées et donnent la distance de la solution concernée (d_s). Le deuxième critère d'évaluation

de la fiteness est la capacité de chaque noeud. De la même manière, les demandes en capacité totales de tous les véhicules à l'intérieur de la solution sont additionnées et donnent la valeur de la capacité de la solution concernée(courante). (q_s). Ce deux valeur interviennent dans la formule de calcul de la valeur de la fonction Fitness totale :

$$Fitness = d_s + q_s;$$

Algorithm

1. generate the InitialSolution (group of routs) with applying a nearest rule between nodes ;
2. calculate the fitness of InitialSolution ;

$$Fitness = d_s + q_s;$$

3. set best solution BestSol = InitialSolution ;
4. loop for 10,000 iterations, in each iteration :
5. generate new random solution (NewSol) with applying a nearest rule between nodes ;
6. calculate the fitness of a NewSol ;
7. if NewSol is better than saved BestSol, then BestSol = NewSol ; end of loop ;
8. checking BestSol feasibility ;
9. write out the path of each route stored inside the BestSol ;

Cet Algorithme est nommé KNNA car à chaque noeud qu'on doit relier à un de ses voisins l'algorithme assure si le plus proche voisin a été déjà visité par un véhicule (affféctée à une route) la sélèction par la génération de la route courante va séléctionner le $k - ime$ plus proche voisin non encore affféctée à une route.

1.4 Conclusion

La thèse [9] est une introduction à élaborer d'une méthode qui se basera sur l'apprentissage automatique pour la résolution de problèmes VRP. L'auteur réalise une investigation sur les types de voisinages au sens de classification. l'auteur informe clairement que malgré les nombreuses recherches sur le domaine du transport, on ne connait toujours pas clairement ce qui définit une bonne solution, notamment pour les problèmes complexes issus de données réelles. Dans le contexte où les données réeles des solutions connues pour être de qualité et où les données sont de plus en plus en augmentation, est apparue l'idée d'utiliser des méthodes

d'analyse statistique ou d'apprentissage pour formaliser ce qu'est une bonne solution pour orienter les recherches et peut être améliorer l'efficacité des heuristiques constructives. Actuellement on constate une orientation vers ce type d'études. Pour notre projet, on choisi de développer une amélioration de l'heuristique NN (plus proche voisin), pour sa simplicité à l'implémentation et qui ne nécessitent pas des mémoires de stockage ni d'investigations dans des perturbations (voisinage) ou qui alourdissent l'exécution de recherche de solutions. Notre démarche sera expliquée dans le chapitre suivant.

Chapitre 2

Une démarche vers l'heuristique KNN qui dois résoudre le problème de transport multiple Starts/Ends CVRPTW

2.1 Introduction

Dans le chapitre précédent nous avons repris la phrase de [9] qui informe que les solutions créées à partir d'un algorithme constructif sont obtenues rapidement, mais sont généralement de mauvaise qualité et pourtant souvent, quelques perturbations (changements dans les tournées) suffisent pour améliorer une solution. Dans cet ordre d'idée notre démarche consiste à implémenter l'heuristique NN qu'on testera sur le même modèle de données développé pour un solveur (CPLEX)d'une méthodes éxacte développé par[1]. On doit comparer les résultats sur les instances réalisables sur le solveur et analyser les caractéristiques des solutions optimales par rapport aux résultas de l'heuristique NN ce qui motivera l'approche finale de notre travail. Notre travail a vu une évolution par différentes implémentations suivantes :

1. NN qui assure le transport de tous les clients
2. NN qui assure le transport de tous les clients avec interval de temps (Time window)
3. NN corrigée par une KNN heuristique

2.2 Identification du problème

Le problème de transport qu'on doit résoudre est une variante du VRP qui est définie comme suit :

1. Le réseau routier est composé de plusieurs dépôts de départs, de dépôts d'arrivée et de plusieurs stations de bus intermédiaire.
2. Dans chaque dépôts de départs et stations de bus intermédiaire il y a un nombre aléatoire de clients.
3. On définie des distances aléatoires entre :
   (a) chaque dépôt de départ vers toutes les stations de bus intermédiaires.
   (b) chaque station de bus intermédiaire vers toutes les stations de bus intermédiaires.
   (c) chaque station de bus intermédiaire vers tous les dépôts d'arrivée.
4. Chaque véhicule a une capacité définie.
5. Pour chaque dépôt d'arrivée et station de bus intermédiaire est définie une fenêtre de temps de valeur aléatoire qui représente l'heur d'arrivée maximale sauf pour les dépôts de départs sa fenêtre de temps aléatoire représente l'heur minimale de départ.
6. Notre solution doit transporter tous les clients dans les délais définies par les fenêtres de temps en minimisant la distance totale de l'ensemble des tournées des véhicules.

2.3 Modèle Mathématique

Le modèle que nous proposons permet de concevoir des trajets optimaux à partir de stations de départs vers des stations d'arrivée, en passant par des stations intermédiaires. Sa résolution est directe avec un solveur mathématique qui peut résoudre des instances de moyenne taille en un temps raisonnable.
Les données suivantes sont nécessaires au modèle :
S : Ensemble de stations de départ. $s = |S|$
$\mathscr{E}$: Ensemble de stations d'arrivée. $e = |\mathscr{E}|$
$\mathscr{B}$: Ensemble de stations intermédiaires, $b = |\mathscr{B}|$
L : Ensemble de véhicules, $m = |L|$
On pose :
$I = S \cup \mathscr{B}$: Ensemble de stations où les personnes sont en attente
$J = \mathscr{B} \cup \mathscr{E}$: Ensemble des stations intermédiaires et stations d'arrivée

Les données suivantes sont aussi nécessaires au modèle :
C_{ij} : Distance (calculée à partir de coordonnées GPS) entre les différentes stations $(i \in I, j \in J \text{ and } j \neq i)$
Q_l : Capacité du véhicule l $(l \in L)$
P_i : Nombre de personnes à récupérer à la station i $(i \in I)$
m_i : Nombre de véhicules pré-affectés à la station de départ i $(i \in S)$; $(m = \sum_{i \in S} m_i)$
Nous posons : N = Nombre total de personnes à récupérer $N = \sum_{i \in I} P_i$
Le modèle utilise les variables de décision suivantes :
x_{ij}^l : Variable binaire égale à 1 si le véhicule l $(l \in L)$ va de la station i $(i \in I)$ à la station j $(j \in J \text{ and } j \neq i)$
y_i^l : Variable entière représentant le nombre de personnes récupérées à la station i $(i \in I)$ par le véhicule l $(l \in L)$
u_i^l : Variable réelle utilisée pour éliminer les sous-tours entre les stations intermédiaires.
et pour l'extention du model au cas TW(Time Window/Fenêtre de temps)
Nous ajoutons pour cela d'autres variables notées w_i^j représentants le temps de passage de véhicule l au nœud i ainsi que les données suivantes :
t_{ij} :Durée du trajet entre les différentes stations$(i \in I, j \in J \text{ and } j \neq i)$
a_i : Temps de départ minimal des start-offices$(i \in S)$
b_j : Temps d'arrivée maximal aux stations intermédiaires et d'arrivée $(j \in J)$

Le problème d'optimisation du transport de personnes est formulé comme suit :

$$min \sum_{i\in S}\sum_{j\in\mathscr{B}} c_{ij} \sum_{l\in L} x_{ij}^l + \sum_{i\in\mathscr{B}}\sum_{\substack{j\in\mathscr{B}\\ j\neq i}} c_{ij} \sum_{l\in L} x_{ij}^l + \sum_{i\in\mathscr{B}}\sum_{j\in\mathscr{E}} c_{ij} \sum_{l\in L} x_{ij}^l \tag{2.1}$$

$$\sum_{\substack{i\in I\\ i\neq j}}\sum_{l\in L} x_{ij}^l \geq 1 \quad \forall j \in \mathscr{B} \tag{2.2}$$

$$\sum_{\substack{j\in J\\ i\neq j}}\sum_{l\in L} x_{ij}^l \geq 1 \quad \forall i \in \mathscr{B} \tag{2.3}$$

$$\sum_{\substack{i\in I\\ i\neq k}} x_{ik}^l - \sum_{\substack{j\in J\\ j\neq k}} x_{kj}^l = 0 \quad \forall k \in \mathscr{B}, \quad \forall l \in L \tag{2.4}$$

$$\sum_{j\in\mathscr{B}}\sum_{j\in L} x_{ij}^l \leq m_i \quad \forall i \in S \tag{2.5}$$

$$u_i^l + x_{ij}^l - M(1 - x_{ij}) \leq u_j^l \quad \forall i \in \mathscr{B}, \quad \forall j \in \mathscr{B} \quad et \quad j \neq i, \quad \forall l \in L \tag{2.6}$$

$$\sum_{i\in S}\sum_{j\in\mathscr{B}} x_{ij}^l \leq \quad \forall l \in L \tag{2.7}$$

$$\sum_{i\in I} y_i^l \leq Q_l \quad \forall l \in L \tag{2.8}$$

$$\sum_{l\in L} y_i^l \geq P_i \quad \forall i \in I \tag{2.9}$$

$$y_i^l \leq N \sum_{j\in\mathscr{B}} x_{ij}^l \quad \forall i \in S, \quad \forall l \in L \tag{2.10}$$

$$y_k^l \leq N[\sum_{i\in I, i\neq k} x_{ik}^l + \sum_{j\in J, j\neq k} x_{kj}^l] \quad \forall k \in \mathscr{B}, \quad \forall l \in L \tag{2.11}$$

$$w_i^l \geq a_i \quad \forall i \in S, \forall l \in L \tag{2.12}$$

$$w_j^l \leq b_j \quad \forall j \in J, \forall l \in L \tag{2.13}$$

$$w_i^l + t_{ij} - w_j^l \leq M(1 - x_{ij}^l) \quad \forall i \in I, \quad \forall j \in J \quad et \quad j \neq i, \forall l \in L \tag{2.14}$$

$$x_{ij}^l \in [0, 1] \quad \forall i \in I, \quad \forall j \in J \quad et \quad j \neq i, \quad \forall l \in L \tag{2.15}$$

$$y_i^l \quad entier \qquad \forall i \in I, \quad \forall l \in L \tag{2.16}$$

$$u_i^l \geq 0 \quad \forall i \in I, \quad \forall l \in L \tag{2.17}$$

$$w_i^l \geq 0 \quad \forall i \in S, \quad \forall l \in L \tag{2.18}$$

$$w_j^l \geq 0 \quad \forall j \in J, \quad \forall l \in L \tag{2.19}$$

1. La fonction objective 2.1 représente le coût de transport à minimiser (distance, temps, ...). Elle est composée de trois termes, le premier représente le coût de transport des stations de départ vers les stations intermédiaires, le second représente le coût de transport entre les stations intermédiaires et le troisième terme le coût de transport des stations intermédiaires vers les stations d'arrivée.
2. Les contraintes 2.2 et 2.3 indiquent que chaque station intermédiaire doit être visitée par au moins un véhicule.
3. Les contraintes 2.4 et 2.5 représentent les contraintes de flux qui garantissent que chaque véhicule part d'une station de départ, puis visite au moins une station intermédiaire pour enfin terminer son trajet à une station d'arrivée.
4. Les contraintes 2.6 permettent d'éliminer les solutions contenant des sous-tours entre les stations intermédiaires.
5. Les contraintes 2.7 permettent d'éviter qu'un véhicule prenne départ plusieurs fois.
6. Les contraintes 2.8 permettent de ne pas dépasser la capacité des véhicules.
7. Les contraintes 2.9 garantissent le transport de toutes les personnes.
8. Les contraintes 2.10 et 2.11 garantissent la cohérence entre les variables de décision x et y.
9. la contrainte 2.12 représente l'heur minimale de départ depuis le dépôt.
10. la contrainte 2.13 représente l'heur maximale de passage par les station de bus intermédiaire et les dépôts d'arrivée.
11. la linéarisassions 2.14 linéarise les deux contrainte prece
12. Enfin les contraintes 2.15, 2.16, 2.17, 2.18 et 2.19 représentent le domaine de chaque variable de décision du modèle.

Exemple d'illustration :

Nous avons appliqué le solveur CPLEX sur l'instance (de petite taille) nomé EX14 généré de manière aléatoire constituée des valeurs suivante :
Stations de départ : s=3, S={1,2,3}
Stations intermédiaires : b=9, B={4,5,6,7,8,9,10,11,12}
Stations d'arrivée : e=2, E={13,14}
Nombre de véhicules : m=6
Capacité des véhicules : Q=30
Le nombre de clients dans chaque station est :{26, 17, 23, 13, 12, 12, 13, 13, 12, 12, 12, 13}
Nombre total de personnes à transporter N=178
Les temps de départ des a_i :{5, 6, 6}
Les temps de passage des b_j :{7, 7, 8, 8, 8, 9, 9, 8, 7,9, 9}
Après exécution la valeur de la fonction objective est : $Objective = 315$
avec un temps d'exécution : $time = 10.45sec.$

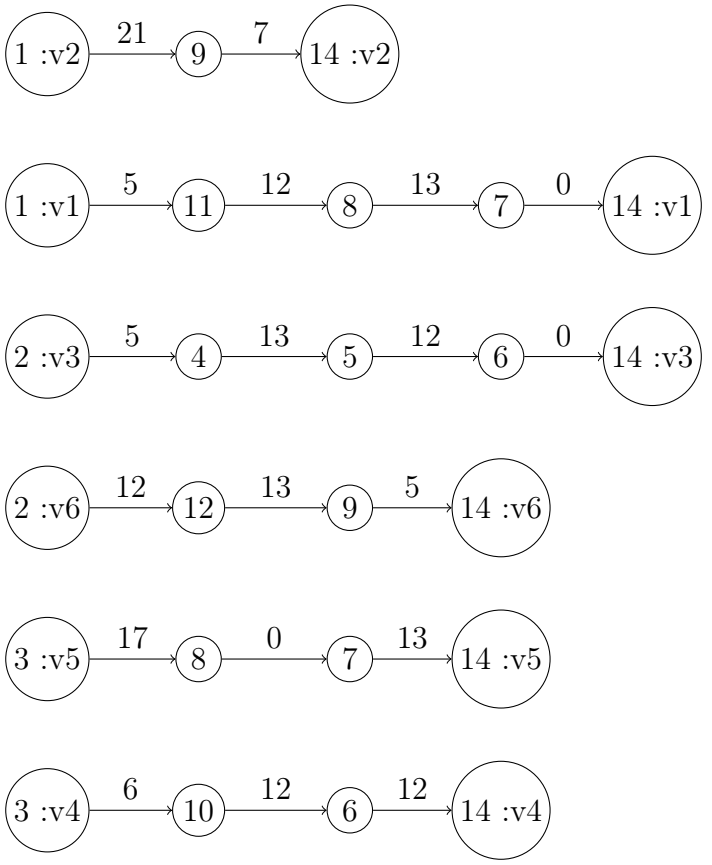

FIGURE 2.1 – Les routes générées à l'exécution du solveur CPLEX sur EX14

Avantage et inconvénient Le solveur mathématique, pour le modèle précédent assure des solutions optimales. Cependent, le temps d'exécution augmente avec la taille des instances étudiées. Et quand la taille des instances dépassent 30 à 40 noeuds, le solveur ne répond plus. c'est pourquoi nous élaborerons une heuristique pour étudier sa faisabilité avec les grandes instances.

2.4 Elaboration d'heuristiques à base du plus proche voisin

Comme nous l'avons déjà expliqué dans le chapitre précédent, nous allons implémenter une variante d'heuristique pour palier au problème posé par le solveur mathématique en suivant différentes étapes. Nous commencerons par une version la plus intuitive du plus proche voisin NN, qui donne une solution réalisable dans un temps d'exécution trés rapide ; mais qui n'assure pas la qualitée recherchée. Une

série de modifications seront appliquées sur l'algorithme de base afin d'améliorer la qualité de la solution. Nous verrons la version avec fenêtre de temps et la version des K plus proches voisins KNN.

2.4.1 L'heuristique du plus proche voisin de base

Pour l'implémentation de notre heuristique nous avons codé l'algorithme NN de la manière la plus naïve, tout en l'adaptant à notre problème. Donc, au lieu qu'une tournée commence et se termine dans le même dépôt nous avons codé des dépôts de départ et des dépôts d'arrivée. A partir de chaque dépôt de départ, chaque véhicule commence sa tournée pour desservir les clients en attente et se dirige vers une station de bus intermédiaire la plus proche, où il prend des clients et si la capacité du véhicule n'est pas atteinte par le nombre de clients dans le véhicule, ce dernier continue sa route vers une autre station de bus la plus proche de sa position. Dés que le véhicule aura atteint sa capacité il termine sa route en se dirigeant vers le dépôt d'arrivée le plus proche.

Algoritme NN

Algorithm 1 Algorithme NN

```
begin
  totD=0
  totroute = []
  for i in S do
    start_depot = i
    for l in L[i] do
      if N ≠ 0 then
        cap = Q[i][l]
        d = 0
        upcust = []
        route = []
        if P[i] < Q[i][l] then
          cap = cap − P[i]
          upcust_add(upcust, P[i])
          N = N − P[i]
          P[i] = 0
          route = solve_vrp_nn(start_depot, cap)
        else
          solve_vrp_start_end()
        end if
        for k in (len(route − 1) do
          d = d + c[route[k]][route[k + 1]]
        end for
      end if
    end for
    totroute_add(totroute, route)
    totD=totD+d
  end for
  end
```

Explication :

L'algorithme 1 est l'algorithme principal qui permet d'appliquer le principe du plus proche voisins dans le parcours de chaque véhicule de chaque dépôt de n'importe quelle instance. l'algorithme a deux cas de figure le premier consiste à appeler la fonction solve_vrp_nn (start_depot,cap) si le nombre de clients pris dans le dépôt de départ est plus petit que la capacité du véhicule, cette dernière fera parcourir le véhicule avec la méthode du plus proche voisin en passant par différentes sta-

tions de bus intermédiaire. Le deuxième cas(où la capacité du véhicule est atteinte) l'algorithme doit appeler la fonction solve_vrp_start_end() qui fera diriger le véhicule vers le dépôt d'arrivée en passant par une station de bus intermédiaire tout en appliquant le principe du plus proche voisins.
Dans l'algorithme nous retrouverons les ensemble S,I,J,L...etc. Et de nouveaux objets nécessaires à notre algorithme, qui ont pour signification :

1. start_depot :dépôt de départ intialisé à la station i.

2. cap : capacité du véhicule l dans la station i.

3. d :distance que chaque véhicule l a parcourue.

4. upcust : nombre de clients desservis par le véhicule à chaque station.

5. route :le trajet du véhicule l (les stations qu'il a traversées).

6. totroute :l'ensemble des routes traversées par l'ensemble des véhicules.

7. totD :distance totale parcourue par l'ensemble des véhicules.

Algorithm 2 Function solve_vrp_start_end

begin

$current\_station = start\_depot$
$upcust\_add(upcust, cap)$
$P[i] = P[i] - cap$
$N = N - cap$
$cap = 0$
$route\_add(route, current\_station)$
$current\_station = find\_nearest\_firsts(current\_station)$
$route\_add(route, current\_station)$
$route\_add(route, find\_nearest\_ends(current\_station))$

end

Explication :

Comme nous l'avons expliqué pour l'algorithme principal, quand un véhicule a une capacité atteinte dés le dépôt de départ, la fonction solve_vrp_start_end est appelée, donc cette fonction détermine la station de bus intermédiaire la plus proche du dépôt de départ, le bus passe sans prendre aucun client vu que sa capacité est atteinte et se dirige directement vers le dépôt d'arrivée le plus proche de cette station, cela se réalise par l'appel des fonctions suivante :

1. find_nearest_firsts
2. find_nearest_ends

cette fonction est nécessaire puisque dans cette variante du VRP on ne définie pas de liaison entre start et end. Pour cette fonction on utilise l'objet suivant :
current_station : qui represente la station ou le dépôt où se trouve le bus.

Algorithm 3 Function solve_vrp_nn($start\_depot, cap$)

begin

```
visited_station = []
current_station = start_depot
route_add(route, current_station)
while len(visited_station) < 𝒷andcap > 0 do
    nearest_station = find_nearest_station(current_station, visited_station)

    visited_station_add(visited_station, nearest_station)
    route_add(route, nearest_station)
    if P[nearest_station]<cap then
        N = N − P[nearest_station]
        cap = cap − P[nearest_station]
        upcust_add (upcust, P[nearest_station])
        P[nearest_station] = 0
    else
        upcust_add(upcust, cap)
        N = N − cap
        P[nearest_station] = P[nearest_station] − cap
        cap = 0
    end if
end while
route_add(route, find_nearest_ends(current_station))
return route
```

end

Explication :

Cette fonction est appelée par l'algorithme dans le cas ou le nombre de clients qui se trouvent dans le dépôt de départ est plus petite que la capacité du véhicule, dans ce cas la fonction cherche les stations les plus proche de la station courrante qui contient des clients, avec l'appel de la fonction find_nearest_station, et les ajoute dans l'ensemble des stations visitées du véhicule. La fonction boucle se travail jusqu'a ce que la capacité du véhicule soit atteinte(quand le bus aura plus de place pour d'autre clients) ou tous les clients sont tous desservis il se dirige vers le dépôt d'arrivée le plus proche avec l'appele de la fonction find_nearest_end, les objets ajoutés dans cette fonction sont :
visited_satation : les stations déjà visitées.
nearest_station : la station la plus proche trouvée.

Algorithm 4 Function find_nearest_station(current_station, visited_satation)

begin

$nearest\_station = find(current\_station, \mathscr{B} - visited\_satation)$

return $nearest\_station$

end

Explication :

c'est la fonction qui recherche la station la plus proche de current_station selon la distance et qui ne figure pas dans visited_satation et retourne nearest_station.

Algorithm 5 Function find_nearest_end(current_station)

begin

$nearest\_station = find(current\_station, \mathscr{E})$

return $nearest\_station$

end

Explication :

c'est la fonction qui recherche le dépôt d'arrivée la plus proche de current_station selon la distance.

Algorithm 6 Function find_nearest_first(current_station)

begin

$nearest\_station = find(current\_station, \mathscr{B})$

return $nearest\_station$

end

Explication :

c'est la fonction qui recherche la station la plus proche de current_station selon la distance.

Exemple d'illustration :

Nous avons appliqué l'algorithme NN sur l'instance précédente (EX14) celle qu'on a résolu avec CPLEX, et le résultat est :
un temps d'exécution de t=0.0049
une distance total (fonction objective) = 454 les routes sont représenté comme suit :

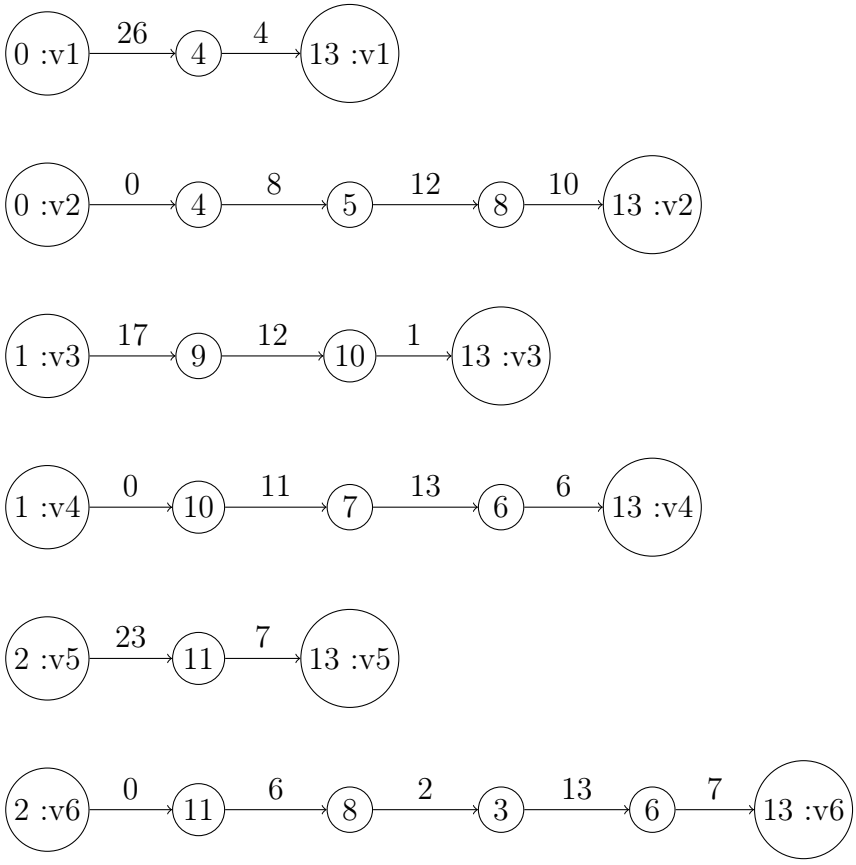

FIGURE 2.2 – Les routes générées à l'exécution du NN sur EX14

Remarques et Observations

Durant notre expérimentation de cet algorithme sur différentes instances nous citons les observations faites par comparaison des résultats du sloveur mathématique et la méthode NN :
Avantages :

1. le temps d'exécution de NN est trés satisfaisant.

2. l'algorithme donne une feuille de route pour désservir tous les clients en attente.

Inconvénients :

1. le coùt totale des trajets est moins intéressants que celui obtenue par le solveur mathématique(voir les exemples d'illustration précédents).
2. l'algorithme du plus proche voisin pousse les premiers véhicules à puiser leurs capacité en emprintant une chaine de courtes distances ce qui oblige les derniers véhicules à parcourir toutes les stations nécessaires pour desservir tous les clients restants. Alors, qu'on remarque que la longueur des routes calculée par le solveur mathématique est homogéne pour l'ensemble des véhicules même si le véhicule doit passer par une station vide. Ce qui est illustré dans la figure2.1 où le véhicule 5 passe par la station 8 sans prendre aucun client dans le but d'optimiser le coùt totale.
3. dans l'algorithme NN tous les véhicules du même dépôt entament leurs tournée en direction de la même station intermédiaire sauf si elle est vide. Ce qui n'est pas le cas pour le solveur mathématique.

Dans la prochaine étape, nous allons introduire les fenêtres de temps comme contraine supplémentaire et voir son impacte.

2.4.2 Plus proche voisin avec fenêtre de temps

Rappelons que la fenêtre de temps définie dans le modèle mathématique pour cette variante de VRP, consiste à attribuer des valeurs aléatoires à chaque station de départ(TWs). Ces dernières représentent l'heur minimale de départ des véhicules de la station. Et d'attribuer des valeurs aléatoires à chaque station de bus intermédiaire (TWb) et chaque dépôt d'arrivée (TWe). Ces dernières représentent l'heur maximale de passage des véhicules par cette station.
Pour introduire les fenêtres de temps dans notre algorithme nous avons modifié l'algorithme en ajoutant les instructions qui ont servi à faire en sorte que chaque véhicule ne passe par une station de bus intermédiaire que si le temps de trajet (variable duree) ajouté à son temps départ de la station précédente (dépôt de départ ou station de bus intermédiaire) est plus petit que le temps maximal de passage défini pour la station la plus proche à visiter. Et le même principe est adopté pour le trajet vers le dépôt d'arrivée.
les instructions ajoutées sont :
duree=(distances[current_station, nearest_station])/V
Hnearest_station=Hcurrent_station+duree
if Hnearest_station<TWb[nearest_station-num_starts] :
route.append(nearest_station)

ou :
duree : est la durée du trajet entre la station où le bus se trouve et la station suivante
V : la vitesse moyenne que peut atteindre un véhicule.
Hcurrent_station : l'heure du bus à la station courrante qui est initialisé à TWs[i].
Hnearest_station : l'heure d'arrivée à la station suivante.
TWb : est un tableau où sont définies les heures max de passage pour les stations de bus intermédiaires.
TWs : est un tableau où sont définies les heures minimales de départ.
et pour les dépôts d'arrivées c'est la même chose juste au lieux de TWb on a TWe qui est le tableau ou sont définies les heures maximales d'arrivée au dépôt d'arrivée.
ces instruction sont ajoutées dans la fonction solve_vrp_nn pour sélectionner le plus proche voisin qui vérifie la contrainte de temps.

Exemple d'illustration :

Nous avons appliqué l'algorithme NN avec TW sur l'instance précédente (EX14), et le résultat est :
Temps d'exécution = 0.0001 secondes.
Il y a 13 clients qui n'ont pas été desservi.
La distance total parccourue est :436.

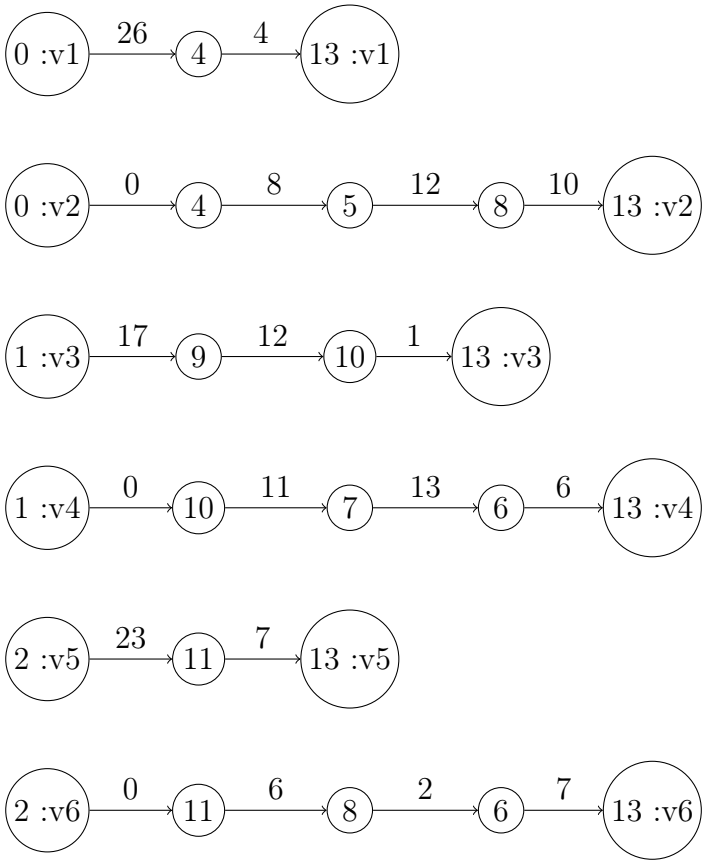

FIGURE 2.3 – Les routes générées à l'exécution du NN TW sur EX14

Explication :
Pour ne pas encombrer le graphe de la figure 2.3, nous avons préféré expliquer juste pour le véhicule 0 à partir de la station de départ 0 son heure de départ minimale est 5, la station la plus proche 4 d'heure maximal de passage est 7 la durée = 0.05(distance=3 / vitesse=60) Hnearest_station = 5+0.05=5.05 et à partir de la station 4 vers le dépôt 13 de heure maximale d'arrivée est 9 c qui va donnée l'heure d'arrivée au dépôt 13 Hnearest_station = 5.05+(64/60) = 6.1166666.

Remarques et Observations

Pour cet algorithme, avec la contrainte temps, le transport de la totalité des clients n'est pas garantie si la capacité de l'ensemble des véhicules est presque égale à la totalité des clients. Comme dans cet exemple (EX14 $N = 178 \quad Q_l = 30$ et 6 véhicules capacité totale 180) 13 clients ne seront pas desservis.
La cause de ce problème est dû au fait que le solveur NNTW attribue les mêmes chemins (si le bus précédent ne vide pas la station visité alors le prochain bus

prend le même chemin) aux bus du même dépôt de départ. Ce qui ne garantie pas l'existence d'au moins un bus qui assure la contrainte de temps défini pour toute station. Et pourtant l'algorithme a bien fonctionné pour la majorité des cas étudiés quand le nombre des clients est suffisamment inférieur à la capacité de l'ensemble des véhicules.

2.4.3 Heuristique KNN pour améliorer la qualité des solutions de NN

D'une manière intuitive, nous exploitons les remarques et les observations faites pour les algorithmes précédents(NN et NNTW), nous proposons en premier une amélioration au début du parcours des véhicules par une sélection aléatoire d'un sommet parmi les K plus proches stations du dépôt de départ comme direction que doivent entamer les véhicules. Cette modification doit donner une chance d'obtenir une autre solution réalisable et peut être de meilleur coût. une deuxième amélioration consiste à attribuer une longueur de chemins homogène à l'ensemble des véhicules, nous avons choisi une longueur K.

KNN sans itérations

Pour introduire KNN dans l'algorithme NN, nous avons défini un paramètre K. K représentera dans notre algorithme le nombre des K plus proches voisins pour le dépôt de départ, donc au lieu de choisir aveuglement la station la plus proche du dépôt de départ comme dans NN. L'algorithme choisira aléatoirement une des K stations les plus proches du dépôt de départ et ensuite il choisira la plus proche entre les stations intermédiaires comme pour NN. K représentera aussi le nombre maximal de stations que chaque bus desservira avant de se diriger vers le dépôt d'arrivée. Pour cela nous avons ajouté les instructions suivantes à l'algorithme :
K=4 nous avons fixé une valeur de K (4 pour EX14) pour tout l'algorithme.
dans la fonction find_nearest_station(current_station, visited_satation) nous avons ajouter une condition comme suit :
donc si current_station est le dépôt de départ, on tri l'ensemble des sommets voisins du dépôt de départ par rapport à leurs distances à celui ci.
puis l'algorithme choisi aléatoirement une des K station les plus proches(un des K premiers sommets dans l'ensemble trié), ce sommet sera le nearest_station à retourner.
et dans la fonction solve_vrp_nn(start_depot,cap,K), le paramêtre K sera ajouté dans la condition d'arrêt de la boucle (WHILE $len(visited\_station) < \mathscr{B}$ and $cap >$ 0) par **len(route)<K**, cette dernière va contrôler que la longueur de la route ne dépassera pas K sommets avant de passer au dépôt d'arrivée.

Exemple d'illustration :

Nous avons appliqué l'algorithme KNN sur la même l'instance (EX14), et le résultat est :
Temps d'execution 0.001977205276489258 secondes.
Distance totale est : 425

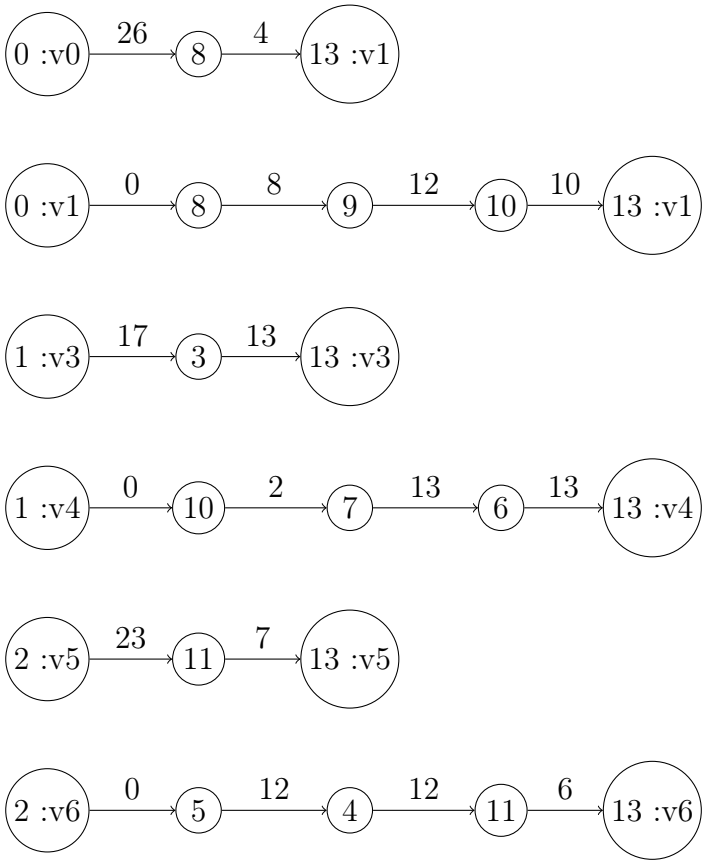

FIGURE 2.4 – Les routes générées à l'exécution du KNN 1 iteration sur EX14

Remarques et Observations

nous avons applique cet algorithme deux ou trois fois sur l'instance EX14 pour avoir ce résultat. ce qui nous a obliger à introduire un certain nombres d'itérations de cet algorithme pour sélectionner la meilleure solution réalisable (tous les clients desservis). **KNN avec des itérations**
Nous introduisons une boucle qui répète l'exécution du dernier algorithme un certain nombre (it) de fois en sauvegardant la meilleure solution.

Exemple d'illustration :

Nous avons appliqué l'algorithme KNN avec 100 itérations sur l'instance (EX14), et le résultat est :
Temps d'exécution = 0.10740995407104492 secondes.
Distance totale est :393

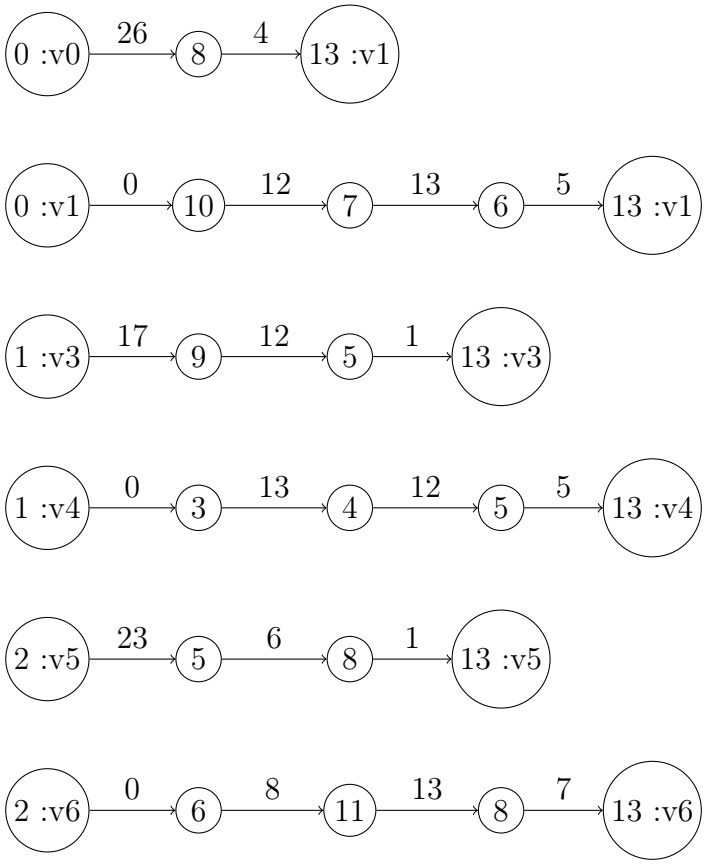

FIGURE 2.5 – Les routes générées à l'exécution du KNN avec 100 iteration sur EX14

Remarques et Observations

Puisque l'algorithme KNN a amélioré considérablement l'algorithme NN, nous avons adopté la même approche sur l'algorithme NNTW et nous avons eu l'algorithme KNNTW qui résolue le problème cité précédament sur NNTW.

2.4.4 Heuristique KNNTW pour améliorer la qualité des solutions de NNTW

De la même manière, nous réalisons une amélioration à l'algorithme NNTW par l'algorithme KNNTW avec itérations, en introduisant le paramêtre K pour la sélection aléatoire d'un sommet parmi les K plus proches stations du dépôt de départ comme direction que doivent entamer les véhicules. Cette modification doit donner une chance d'obtenir une autre solution réalisable et peut être de meilleur coût. Avec la deuxième amélioration qui consiste à attribuer une longueur de chemins homogène à l'ensemble des véhicules.

Exemple d'illustration :

Nous avons appliqué l'algorithme KNNTW avec 1000 iteration sur l'instance (EX14), et le résultat est :
Temps d'execution = 0.11127829551696777 secondes.
Distance Totale est :387

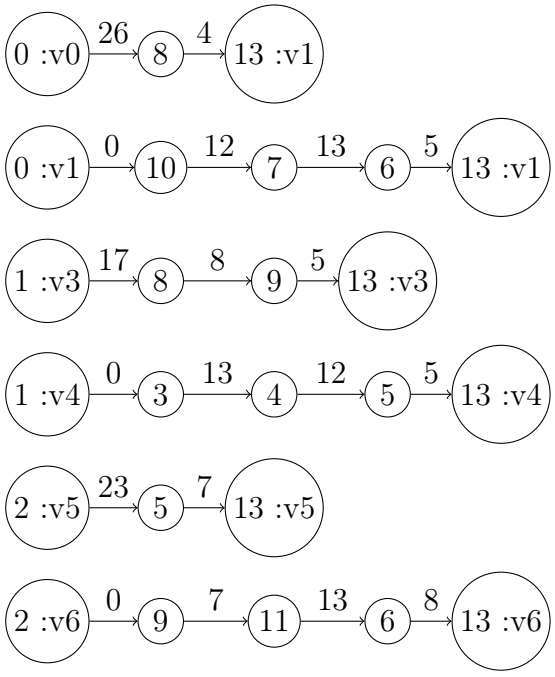

FIGURE 2.6 – Les routes générées à l'exécution du KNN TW avec 1000 iteration sur EX14

2.5 Conclusion

Dans notre démarche, pour élaborer l'algorithme KNNTW pour résoudre le problème vrp multis starts/ends avec fenêtre de temps, nous nous sommes basés sur l'heuristique du plus proche voisin NN que nous avons modifié pour prendre en compte la contrainte de fenêtre de temps comme préalable. A défaut de ne pouvoir étudier toutes les solutions possibles (comme un solveur mathématique) sur des instances de grande taille, nous nous somme intéressés aux solutions résultantes qu'offre le principe du K plus proche voisin pour résoudre les problèmes spécifiques cités dans nos différentes observations et différentes comparaisons et analyses des solutions obtenues des différentes méthodes étudiées(solveur mathématique, NN, NNTW, KNN sans et avec itérations), nous sommes arrivés à l'élaboration de l'algorithme KNNTW qui donne de meilleurs résultats à notre problème dans un temps raisonnable. Dans le chapitre suivant, nous donnerons des détails des expérimentations étudiées dans ce projet.

Chapitre 3

Etude expérimentale

3.1 Introduction

Ce chapitre est consacré à l'étude expérimentale et résultats des différentes méthodes étudiées.

3.1.1 Environement de travail

Nous avons implanté et exécuté les algorithmes que nous avons élaborés et présentés dans le chapitre précédent, et nous avons mis-à-jour les programmes de génération de données (datagen.py et modelgen.py [1]) en ajoutant les instructions pour prendre en compte les contraintes fenêtre de temps, avec le langage Python (v3.9). cette implantation est supporté sur un ordinateur PC portable DELL avec les caractéristique suivantes :

1. Processeur Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz,

2. Mémoire RAM installée 16,0 Go (15,8 Go utilisable),

3. Type du système Système d'exploitation 64 bits, processeur x64,

nos programme ont pour squelette :

1. lecture du fichier data.txt généré par dataGen.py et affectation des données d'une instance.
2. exécution du programme de l'un des algorithmes sur l'instance.
3. affichage de la solution et sauvgarde sur fichiers Excel.

Nous avons choisi le langage python. Nous avons exploité principalement l'objet numpy de python dont les méthodes sont trés utiles dans les traitements sur des

données de type tableaux et ensembles de différentes dimension. De plus python est trés adapté à extraire les données des fichiers excel pour la représentation graphique.

3.2 Expérimentation

3.2.1 NN et CPLEX

Nous avons exécuté le solveur mathématique CPLEX et l'algorithme NN sur les mêmes instances et nous avons constitué un tableau de données comme suit :

1. Graph size (la taille de l'instance),
2. cp_exe_Time(temps d'exécution du solveur CPLEX),
3. nn_exe_Time(temps d'exécution de l'algo NN),
4. cp_distance(la distance total obtenue par le solveur CPLEX),
5. nn_distance(la distance total obtenue par l'algorithme NN),
6. %gap(le pourcentage de distance gap entre les distances obtenue par CPLEX et NN).

Graph size	cp_exe_Time	nn_exe_Time	cp_distance	nn_distance	%gap
16	0,05	0,005	106	249	57,42971888
18	0,03	0,005	96	218	55,96330275
20	0,06	0,003	177	295	40
22	0,06	0,005	130	245	46,93877551
22	0,08	0,001	184	324	43,20987654
23	0,31	0,003	137	459	70,15250545
23	0,17	0,006	123	346	64,45086705
24	0,06	0,001	127	333	61,86186186
25	0,11	0,013	146	267	45,31835206
25	0,22	0,004	180	332	45,78313253
26	0,3	0,005	186	243	23,45679012
26	0,39	0,006	163	316	48,41772152
29	0,11	0,008	143	299	52,17391304
31	2,31	0,009	148	353	58,07365439
32	1,58	0,01	173	335	48,35820896
33	3,11	0,002	236	377	37,4005305
34	204,39	0,012	232	585	60,34188034
35	257,53	0,0008	240	418	42,58373206
40	496,63	0,015	164	492	66,66666667

TABLE 3.1 – Tableau des données de comparaison entre le solveur CPLEX et l'heuristique NN

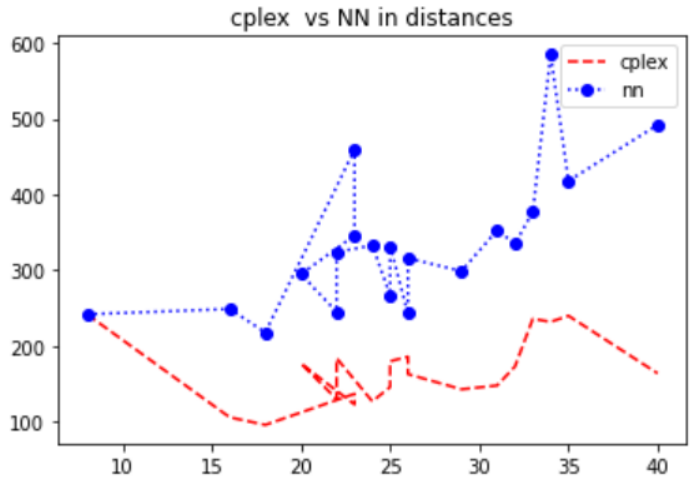

FIGURE 3.1 – Graphe de comparaison entre les distances du solveur CPLEX et l'heuristique NN, l'axe des x correspond à la taille de l'instance, et l'axe des y correspond au distances

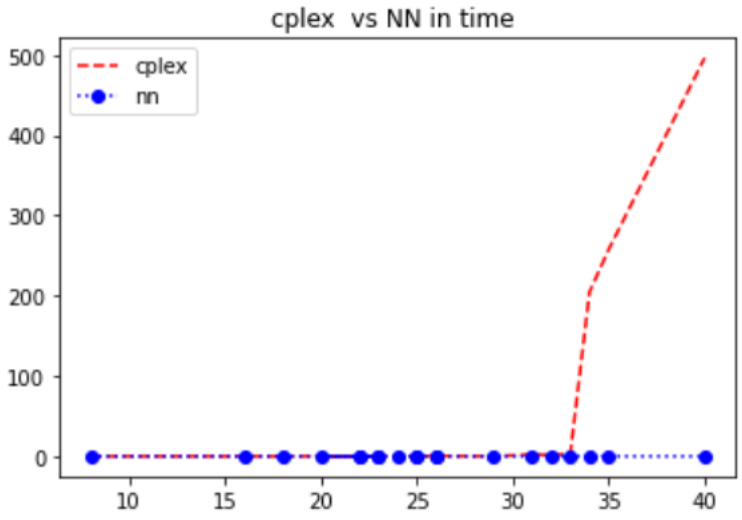

FIGURE 3.2 – Graphe de comparaison entre les temps d'execution du solveur CPLEX et l'heuristique NN, l'axe des x correspond à la taille de l'instance, et l'axe des y correspond au temps d'exécution

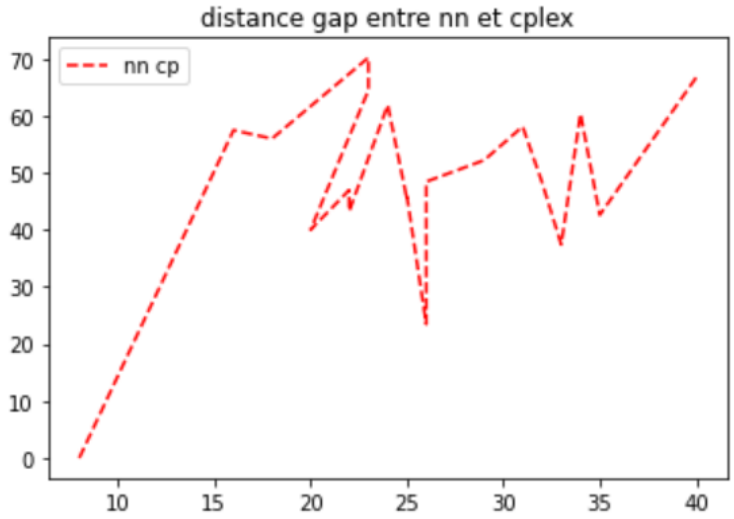

FIGURE 3.3 – Graphe de differences des distances (gap) entre le solveur CPLEX et l'heuristique NN

Interprétation : Les graphes montrent clairement que malgré que le temps d'exécution de l'algorithme NN est largement plus intéréssant mais en question de coût de distance total les resultats du solveur CPLEX sont les plus intéréssant. Et le solveur mathématique prend de plus en plus de temps d'exécution à mesure que la taille de l'instance grandi, et quand la taille de l'instance dépasse les 40 sommets le solveur mathématique ne donne plus de solutions.

3.2.2 KNN et CPLEX

Dans cette expérimentation nous présentons une comparaison entre le solveur mathématique CPLEX et l'algorithme KNN.

Graph size	knn_distance	cp_distance
6	244	244
8	206	194
10	261	198
14	345	210
20	406	422
25	389	261
30	601	599

TABLE 3.2 – Tableau des données de comparaison entre le solveur CPLEX et l'heuristique KNN

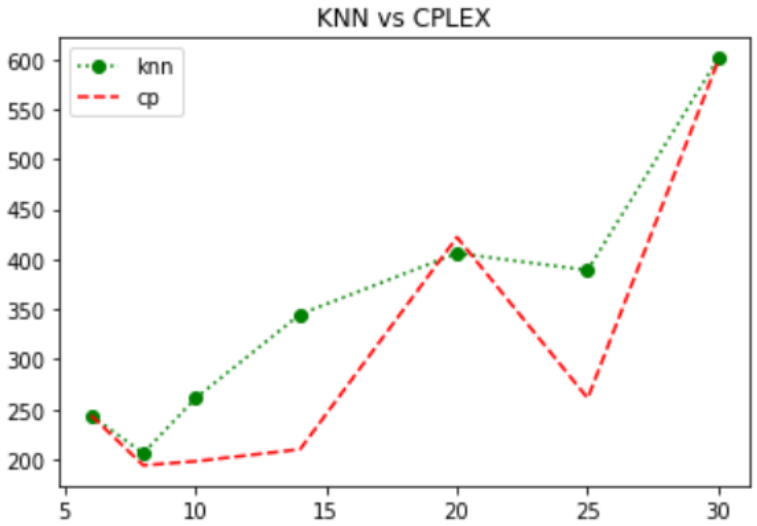

FIGURE 3.4 – Graphe de comparaison entre les distances du solveur CPLEX et l'heuristique KNN

Interprétation : Les résultats de KNN et CPLEX sont presque équivalents car les coûts de distance totale de KNN ne sont pas trop grand par rapport à ceux obtenues par le solveur CPLEX et il y a même le cas où les coûts sont égaux ou le coût obtenue par KNN est plus intéréssant que celui de CPLEX.

3.2.3 NN et KNN

Dans cette expérimentation nous présentons une comparaison entre l'algorithme NN et l'algorithme KNN.

Graph size	nn_distance	knn_distance	nn_time	knn_time	%gap
6	323	244	0,001	0,027	24,45820433
8	206	206	0,00099	0,032	0
10	278	261	0,002	0,059	6,115107914
14	506	345	0,0034	0,096	31,81818182
20	526	406	0,004	0,15	22,81368821
25	433	389	0,007	0,22	10,16166282
30	611	601	0,039	0,99	1,636661211
35	757	518	0,04	0,7	31,57199472
40	775	507	0,03	0,6	34,58064516
45	779	629	0,04	0,5	19,25545571
50	593	482	0,011	0,53	18,71838111
55	659	479	0,013	0,64	27,31411229
60	894	740	0,02	1,01	17,22595078
65	845	761	0,02	0,99	9,940828402
100	727	619	0,02	1,5	14,85557084
120	952	752	0,02	2,02	21,00840336
150	965	803	0,04	2,63	16,78756477
200	1250	843	0,07	5,11	32,56
300	1822	1486	0,13	10,5	18,44127333
500	1849	1418	0,36	26,75	23,30989724
1000	2600	2402	1,22	119,61	7,615384615

TABLE 3.3 – Tableau des données de comparaison entre l'heuristique NN et KNN

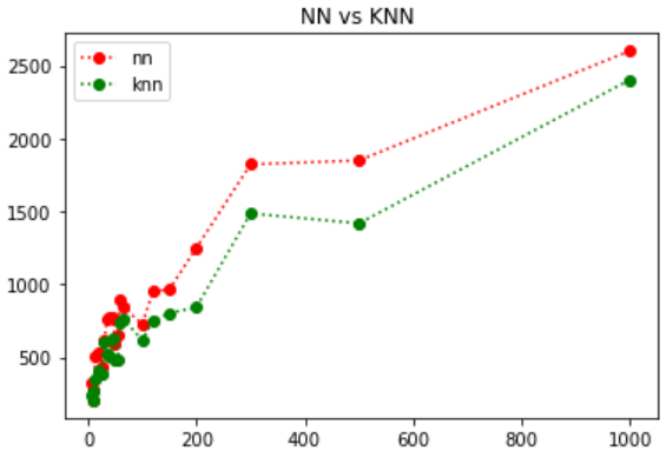

FIGURE 3.5 – Graphe de comparaison entre les distances de l'heuristique NN et KNN

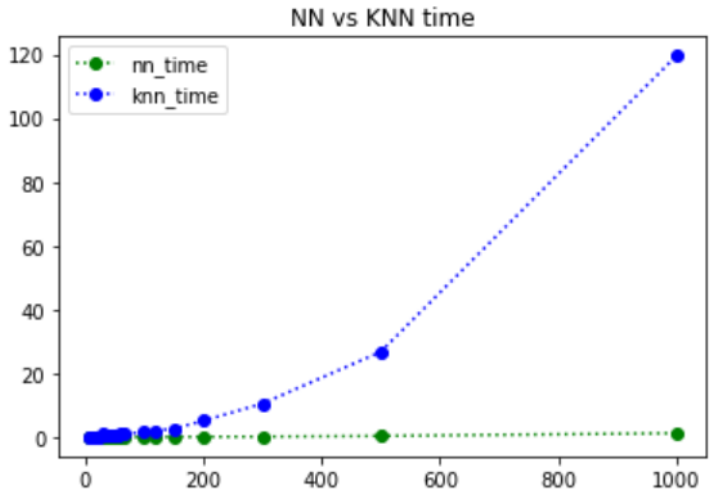

FIGURE 3.6 – Graphe de comparaison entre les temps d'execution de l'heuristique NN et KNN

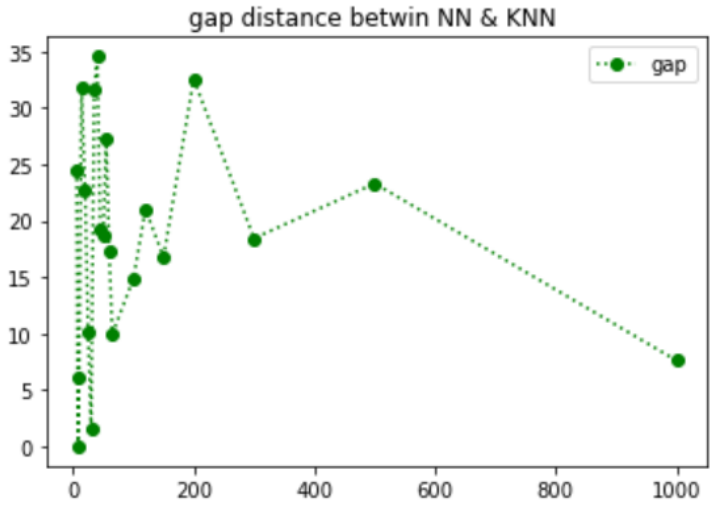

FIGURE 3.7 – Graphe de differences des distances (gap) entre l'heuristique NN et KNN

Interprétation : Ici dans la comparaison entre le NN et KNN nous remarquons que la différence entre les deux algorithme passe par deux phase. La première quand la taille de l'instance est petite, le temps d'exécution des algorithme est presque

équivalent avec le temps d'exécution de NN plus intéressant que celui du KNN, mais le coût de distance total du KNN est le plus intéressant. La deuxième phase quand la taille des instances commence a être grandes (200 sommets et plus), là le temps d'exécution du NN est bien plus intéressant que celui du KNN et de plus le coût des distances total n'est pas trop grand comparer au coût obtenue par le KNN, le coût de distance de KNN est juste un peu plus intéressant de celui obtenue par NN à mesure que la taille des instances grandisse. Nous pensons que la valeur du paramètre K doit augmenter avec la taille de l'instance ce qui implique que les solutions obtenue par le KNN et le NN se rapprochent.

3.2.4 NNTW et CPLEX

Dans cette expérimentation nous présentons une comparaison entre le solveur CPLEX et l'algorithme NNTW.

Graph size	cp_exe_Time	nntw_exe_Time	cp_distance	nntw_distance	%gap
16	0,28	0,005	280	344	18,60465116
18	0,22	0,003	147	242	39,25619835
20	0,8	0,001	129	156	17,30769231
25	0,98	0,007	165	335	50,74626866
30	0,5	0,003	264	496	46,77419355
35	61,61	0,012	174	375	53,6
40	3,55	0,015	130	356	63,48314607
45	1928,59	0,065	240	594	59,5959596

TABLE 3.4 – Tableau des données de comparaison entre le solveur CPLEX et l'heuristique NN avec TW

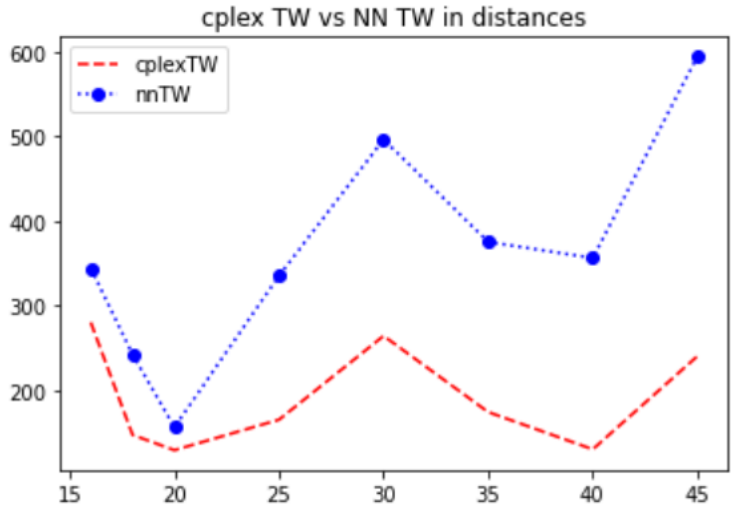

FIGURE 3.8 – Graphe de comparaison entre les distances du solveur CPLEX avec TW et l'heuristique NN avec TW

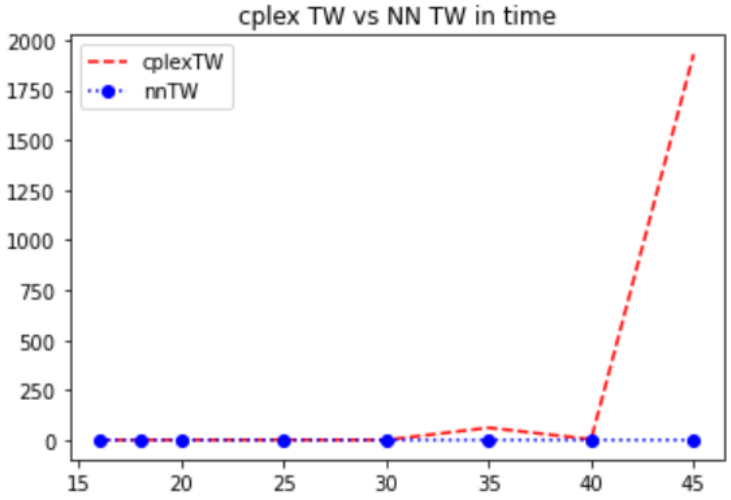

FIGURE 3.9 – Graphe de comparaison entre les temps d'execution du solveur CPLEX avec TW et l'heuristique NN avec TW

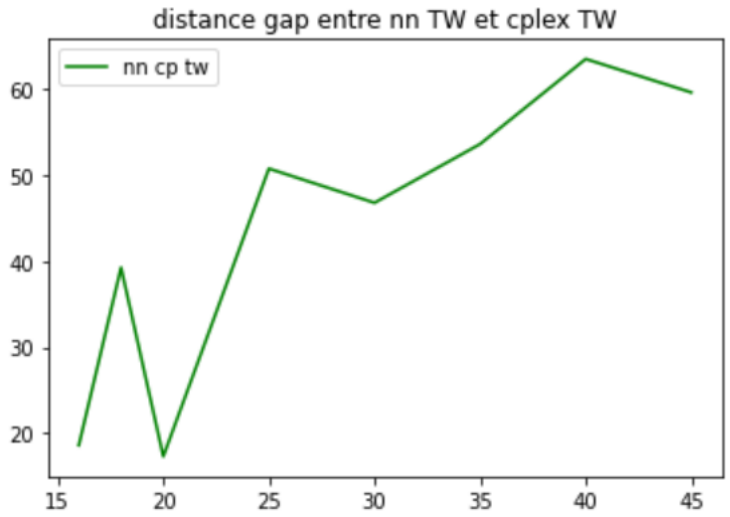

FIGURE 3.10 – Graphe de differences des distances (gap) entre le solveur CPLEX avec TW et l'heuristique NN avec TW

Interprétation : Comme pour NN et CPLEX, les graphes montrent clairement que le temps d'exécution de l'algorithme NNTW est largement plus intéressant mais les solution du CPLEX sont les plus optimales. Le solveur CPLEX ne supporte pas les instances de taille plus grandes que 50 sommets et il ne donne plus de solutions.

3.2.5 KNNTW et CPLEX

Dans cette expérimentation nous présentons une comparaison entre le solveur CPLEX et l'algorithme KNNTW.

Graph size	cp_exe_Time	knntw_exe_Time	cp_distance	knntw_distance	%gap
16	0,33	1,22	129	223	42,15246637
16	2,01	1,13	288	237	-21,51898734
18	0,19	1,21	223	279	20,07168459
18	4,45	1,27	230	346	33,52601156
20	1,41	1,13	211	216	2,314814815
25	0,06	1,43	132	165	20
25	28,23	1,49	213	207	-2,898550725
30	2,91	3,07	184	291	36,76975945
30	266,64	1,94	220	311	29,26045016
35	21,25	3,27	190	246	22,76422764
40	2,44	5,8	208	312	33,33333333
45	3,48	5,86	226	295	23,38983051
50	8,83	6,74	184	249	26,10441767
55	710,15	10,43	215	351	38,74643875

TABLE 3.5 – Tableau des données de comparaison entre le solveur CPLEX avec TW et l'heuristique KNN avec TW

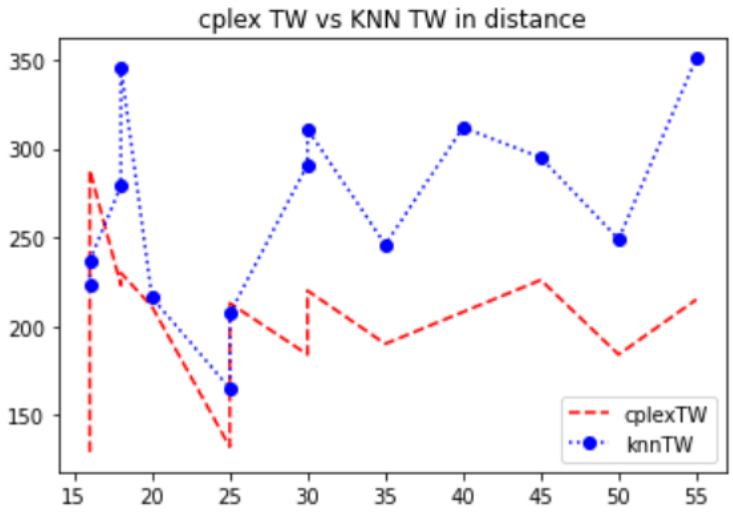

FIGURE 3.11 – Graphe de comparaison entre les distances du solveur CPLEX avec TW et l'heuristique KNN avec TW

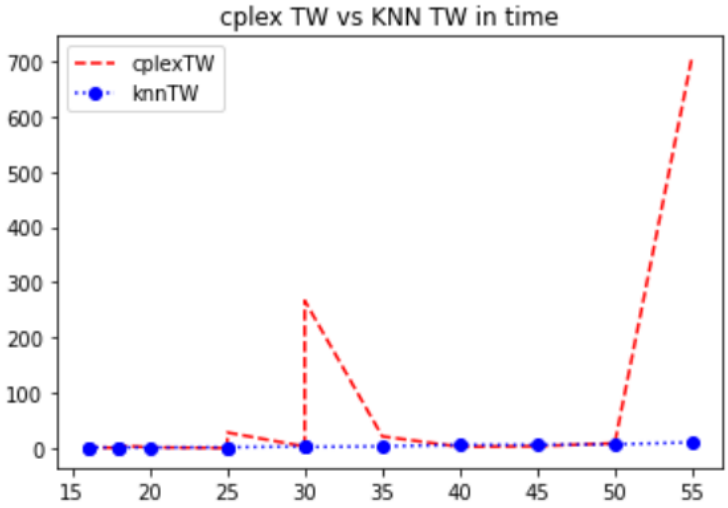

FIGURE 3.12 – Graphe de comparaison entre les temps d'execution du solveur CPLEX avec TW et l'heuristique KNN avec TW

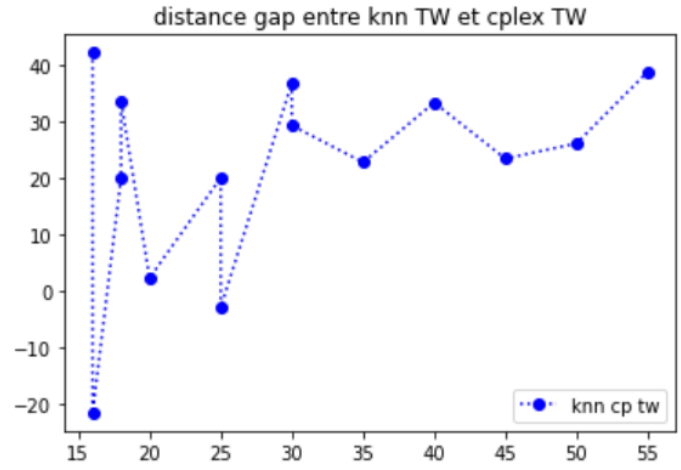

FIGURE 3.13 – Graphe de differences des distances (gap) entre le solveur CPLEX avec TW et l'heuristique KNN avec TW

Interprétation : Les résultats de KNNTW sont plus intéressants car les coûts des distances totales de KNNTW ne sont pas trop grand par rapport à ceux obtenues par le solveur CPLEX et il y a même le cas où les coûts sont égaux et le cas

où le coût obtenue par KNNTW est plus intéressant que celui de CPLEX(nous n'avons pas d'explication surtout que les instances sont aléatoire). Avec un temps d'exécution toujours inferieur pour KNNTW.

3.2.6 NNTW et KNNTW

Dans cette expérimentation nous présentons une comparaison entre l'algorithme NN et l'algorithme KNN.

Graph size	nntw_exe_Time	knntw_exe_Time	nntw_dist	knntw_dist	%gap
60	0,02	5,58	563	416	26,11012433
100	0,04	15,08	662	543	17,97583082
150	0,04	17,38	760	657	13,55263158
200	0,13	32,79	846	573	32,26950355
250	0,09	37,88	1045	923	11,67464115
300	0,1	45,91	905	832	8,066298343
400	0,13	715,95	1099	1002	8,826205641
500	0,25	716,12	1039	973	6,35226179
600	0,26	156,79	1231	1100	10,64175467
700	0,45	209,09	1393	1191	14,50107681
800	0,82	2097,2	1284	1221	4,906542056
900	1,24	2661	2012	1748	13,12127237
1000	2,17	1885,7	2458	2300	6,427990236
1500	2,87	1577,89	3506	3383	3,508271535
2000	5,93	4184,1	4057	3795	6,457973872
2500	26,15	23901,64	4554	4208	7,597716293

TABLE 3.6 – Tableau des données de comparaison entre l'heuristique NN avec TW et KNN avec TW

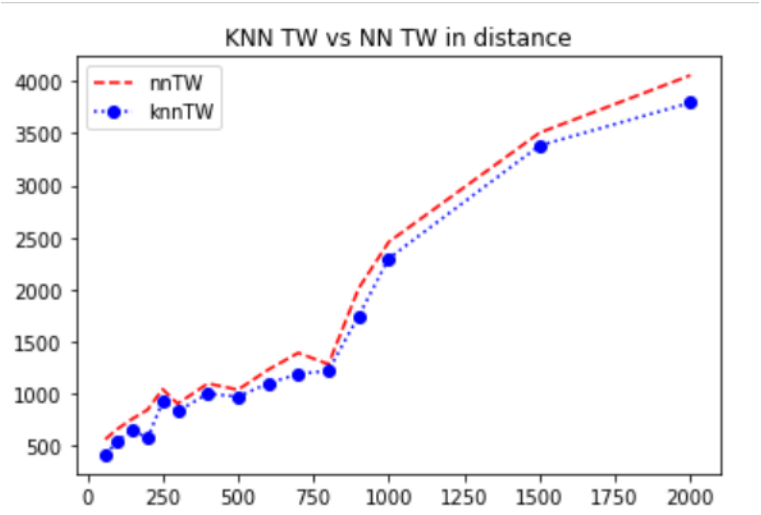

FIGURE 3.14 – Graphe de comparaison entre les distances de l'heuristique NN avec TW et KNN avec TW

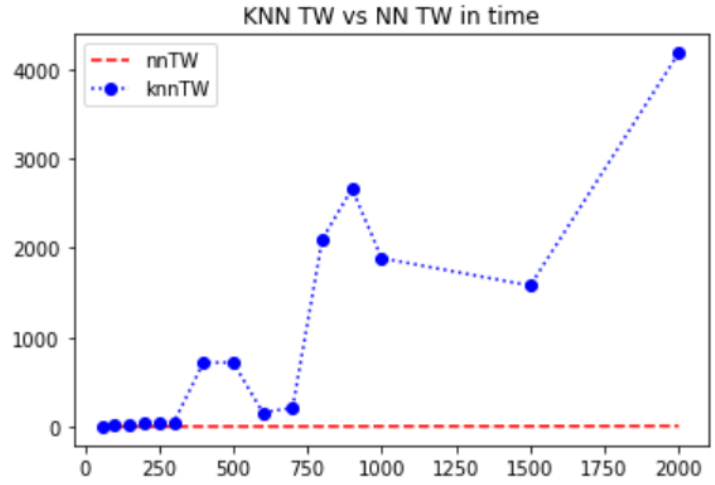

FIGURE 3.15 – Graphe de comparaison entre les temps d'execution de l'heuristique NN avec TW et KNN avec TW

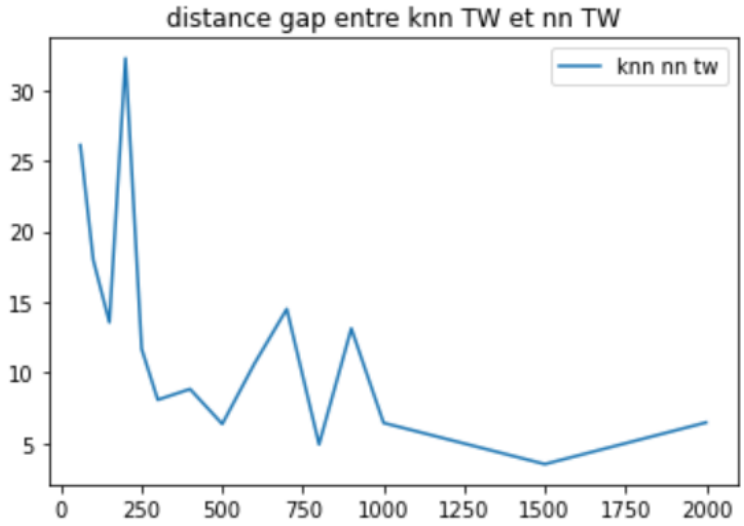

FIGURE 3.16 – Graphe de differences des distances (gap) entre l'heuristique NN avec TW et KNN avec TW

Interprétation : Les résultats de l'expérimentation montre que KNNTW ressemble à NNTW quand les instances sont assez grande.

3.3 conclusion

KNNTW présente la meilleure solution au problème VRP multiple starts/ends avec fenêtre de temps TW même si ses solutions ressemble à ceux obtenue par NNTW puisque les itérations de son algorithme donne plus de chance de trouver une solution dans le cas où la capacité de l'ensemble des véhicules de l'instance est proche du nombre de clients à desservir. Nous pouvons conclure qu'on peut utiliser NNTW dans les instances de grande taille puisque son temps d'exécution est très intéressant et si elle ne donne pas une solution qui satisfasse la totalité des clients on applique KNNTW.

Chapitre 4

Conclusion générale

Dans ce manuscrit, nous avons présenté notre travail sur le sujet des Problèmes de transport difficiles (Cas du VRP with Multiple Starts/Ends). Nous avons commencé par l'explication de ce qu'était les problèmes VRP avec ces différentes variantes. Puis nous avons cité quelques méthodes de résolutions de ces problèmes (méthode exacte, heuristique et méta-heuristique). Dans ce projet, notre tache consistait à intégrer la contrainte fenêtre de temps dans le solveur mathématique CPLEX déjà existant, élaborer les algorithmes NN, NNTW, KNN et KNNTW et puis les implémenter sous forme de solveurs par la méthode approchée du Plus Proche Voisin. Dans le but de les caractériser nous avons fait des expérimentations par comparaisons sur tous ces solveurs ce qui nous a permis de tirer les résultats suivants :

1. Le solveur mathématique donne la solution optimale mais son temps d'exécution est trop grand et qu'il ne traite pas les instances de grandes tailles.
2. Les algorithmes NN et NNTW sont les meilleures en terme de temps d'exécution mais leurs coût de distance total est plus grand que celui du solveur mathématique et des algorithme KNN et KNNTW.
3. L'algorithme NNTW n'assure pas une solution qui satisfait tous les clients de l'instance, ce qui nous a poussé à implémenter les algorithme KNN et KNNTW.
4. D'un point de vu temps d'exécution, les algorithmes KNN et KNNTW sont meilleurs que le solveur CPLEX pour les petites instances. Le faite que les algorithmes KNN KNNTW s'exécutent avec un nombre important d'itérations, quand les instances sont grandes le temps d'exécution devient de plus en plus grand et il deviens moins intéressant que celui obtenu par NN ou NNTW.
5. D'un point de vu coût de distance total pour les petites instances les solutions des algorithmes KNN et KNNTW sont presque aussi bien que ceux

du solveur et il y a même des cas où ils sont meilleurs que ceux du solveur CPLEX puisque dans ces algorithmes les itérations et le caractère aléatoire permettent de calculer différentes solutions ce qui augmente les chances de tomber sur une bonne solution. Dans le cas des grandes instances ils ont toujours de meilleur résultats que NN et NNTW même si la différence n'est pas importante mais ils nous assurent des solutions qui satisfassent tous les clients dans n'importe quel cas.

Perspective

Dans notre projet nous avons réalisé un travail expérimental et de comparaisons dans la même approche qui est l'heuristique plus proches voisins et ses variantes. Il serait intéressant de comparer la pertinence de nos algorithmes par rapport à d'autres méthodes approchées sur la même variante VRP(Problèmes de transport difficiles (Cas du VRP with Multiple Starts/Ends)).

Bibliographie

[1] Kamla Djamila. Chahid Hanane Aicha. Mémoire de master déploiement d'un système de transport intelligent pour la société de transport de la zone industrielle d'arzew. *Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF Faculté des Mathématiques et Informatique*, 2022.

[2] J.Feldman E.A.Feigenbaum. Computers and thought. *McGraw-Hill Inc, New York*, 1963.

[3] Kenneth Sörensen Florian Arnold, Michel Gendreau. Efficiently solving very large-scale routing problems. In *Computers Operations Research*, 2019.

[4] Kenneth Sörensen et Thibaut Vidal. Florian Arnold, Italo Santana. Pils : Exploring highorder neighborhoods by pattern mining and injection. In *arXiv preprint arXiv :1912.11462,*, 2019.

[5] J. H Ramser. G. B. Dantzig. *The Truck dispatching problem. Lehrstuhl fùr Wirtschaftsinformatick und Operations Research.* PhD thesis, Universitat zu Koln, Mgmt Sci., 1959.

[6] Amira Gherboudj. Méthodes de résolution de problèmes difficiles académiques. *Université de Constantine2*, 2013.

[7] G.Clarke J.W.Wright. Scheduling of vehicles from a central depot to a number of delivery points,. In *Ope- rations Research,*, 1964.

[8] R. Kammarti. *Approches Evolutionnistes Pour La Resolution Du 1-PDPTW Statique Et Dynamique.* PhD thesis, Université de Lille, France., 2006.

[9] Flavien Lucas. thèse résolution de problèmes réalistes de tournées à flotte hétérogène en milieu urbain : vers un solveur adaptatif mêlant recherche opérationnelle et apprentissage 2021 automatique. 2020.

[10] Amel Madani. *Le problème de transport des personnes dans les sociétés de taxis.* PhD thesis, FACULTE MATHEMATIQUES ET INFORMATIQUE DEPARTEMENT INFORMATIQUE-DOMAINE . . ., 2019.

[11] Raed Ibraheem Hamed Salama A. Mostafa Dheyaa Ahmed Ibrahim Humam Khaled Jameel Ahmed Hamed Alallah Mazin Abed Mohammed, Mohd Khanapi Abd Ghani. Solving vehicle routing problem by using improved k-nearest neighbor algorithm for best solution. *Journal of Computational Science*, 2017.

[12] W.Bloch Rachid Christelle Chatonnay Pascal M.Haj, Ramdane-Cherif. Classification de problèmes de tournées de véhicules. In *7eme Conférence Internationale de MOdélisation et SIMulation-MOSIM*, volume 8, 2008.

[13] M.W.P. Savelsbergh. *Local Search for Routing Problems with Time Windows.* PhD thesis, 1995.