N° d'ordre : ....

UNIVERSITÉ MOHAMMED SEDDIK BENYAHIA
JIJEL
FACULTÉ DE SCIENCES EXACTES ET D'INFORMATIQUE

MEMOIRE DE MASTER

Présenté pour l'obtention du diplôme de :

MASTER

En **INFORMATIQUE**

Option : INTELLIGENCE ARTIFICIELLE

Par :

Boudjerda Nabil
Djarit Ismail

Thème

Le problème de routage dynamique des véhicules avec fenêtres de temps

Soutenu publiquement, le 06/07/2022, devant le jury composé de :

M.	LEMOUARI Ali	Pr à l'université de Jijel	Président
M.	ALLIOUACHE Abdelaziz	Ma à l'université de Jijel	Encadreur
M.	LABANI Marouane	Dr à l'université de Jijel	Examinateur

Remerciements

Avant tout nous remercions Allah le tout puissant, le Miséricordieux, qui nous a donné le courage, la volonté, la force, la santé et la persistance pour accomplir ce modeste travail. Merci de nous avoir éclairé le chemin de la réussite.

Nous tenons à exprimer notre gratitude à :

Notre encadrant : ALLIOUCHE Abdelaziz pour sa patience, en particulier Confiance, ses observations, ses conseils, sa présence et sa gentillesse.

Tous les enseignants du département d'informatique, qui ont assisté à nos débuts en informatique, pour leurs précieux conseils.

Nos sincères remerciements aux membres du jury pour l'honneur qu'ils nous ont fait Acceptez le jugement sur ce travail et sur tous leurs commentaires pertinents.

De loin à la fin de ce travail, nos professeurs, parents et frères, Sœurs et amies, nous remercions tout particulièrement mes condisciples pour leur respect mutuel.

Enfin, nous espérons que ce travail aura la valeur souhaitée.

Résumé

Le problème de routage dynamique des véhicules avec fenêtres de temps est une variante du problème d'optimisation combinatoire bien connu de tournées des véhicules qui tient compte de la nature dynamique du problème. Cet aspect du dynamisme exige que les routes des véhicules soient mis à jour en permanence, au fur et à mesure que de nouvelles demandes de clients arrivent dans le système, et qu'ils soient intégrés dans un programme évolutif au cours de la journée de travail. En plus de la contrainte de capacité du véhicule impliquée dans le probléme de tournées de véhicules (VRP) classique, le problème de routage dynamique des véhicules avec fenêtres de temps (DVRPTW) considère en plus des fenêtres de temps, qui sont capables de mieux capturer les situations du monde réel. Un algorithme de systéme multi-colonies de fourmis combiné à une procédures de recherche locale puissantes est proposé pour résoudre le problème du routage dynamique des véhicules avec des fenêtres de temps.

Mots-clés *: Le problème de routage dynamique de véhicules avec fenêtres temps,Le probléme de tournées de véhicules , Optimisation par colonies de fourmis,MACS-VRPTW,Le problème de tournées de véhicules avec fenêtres temps.*

Abstract

The dynamic vehicle routing problem with time windows is a variant of the well-known optimization combinatorial vehicle routing problem that takes into account the dynamic nature of the problem. This aspect of dynamism requires that the vehicle routes be continuously updated as new customer requests come into the system, and that they be integrated into an evolving schedule over the course of the workday. In addition to the vehicle capacity constraint implied in the classical Vehicle Routing Problem (VRP), the Dynamic Vehicle Routing Problem with Time Windows (DVRPTW) additionally considers time windows, which are able to better capture real-world situations. A multiple ant colony algorithm combined with a powerful local search procedure is proposed to solve the dynamic vehicle routing problem with time windows.

Keywords : *The dynamic vehicle routing problem with time windows, vehicle routing problem, Ant colony optimization, MACS-VRPTW, The vehicle routing problem with time windows.*

TABLE DES MATIÈRES

Table des Matières i

Table des figures iv

Liste des tableaux vi

Liste des algorithmes vii

Liste des acronymes viii

1 L'optimisation combinatoire 3
- 1.1 Introduction . . . 3
- 1.2 Définition d'un problème d'optimisation combinatoire : . . . 3
- 1.3 Complexité . . . 4
  - 1.3.1 Complexité d'un algorithme . . . 4
  - 1.3.2 Complexité des problèmes . . . 4
- 1.4 Quelques problèmes classiques d'optimisation combinatoire . . . 5
  - 1.4.1 Problème du sac-à-dos . . . 5
  - 1.4.2 Problème d'affectation . . . 5
  - 1.4.3 Problème d'ordonnancement . . . 6
  - 1.4.4 Problème du voyageur de commerce . . . 6
- 1.5 Méthodes de résolution pour l'optimisation combinatoire . . . 7
  - 1.5.1 Les méthodes exactes . . . 7
    - 1.5.1.1 Méthode Branch and bound . . . 8
    - 1.5.1.2 Programmation Linéaire en nombres entiers (Branch And Cut) . . . 8
    - 1.5.1.3 Programmation Dynamique . . . 8
  - 1.5.2 Les méthode approchées . . . 8
    - 1.5.2.1 Les heuristiques . . . 8
    - 1.5.2.2 Les métaheuristiques . . . 9
- 1.6 Conclusion . . . 16

2 Le problème de tournée de véhicules avec fenêtre de temps dynamique 18
2.1 Introduction . . . 18
2.2 Problèmes de Tournées de véhicules . . . 19
2.2.1 Description . . . 19
2.2.2 Formulation . . . 20
2.2.3 Classification de VRP . . . 21
2.2.3.1 VRP Statique . . . 22
2.2.3.2 VRP Dynamique . . . 22
2.2.4 Complexité de problème de VRP . . . 23
2.2.5 Variantes du problème VRP . . . 23
2.3 Problème de tournée de véhicule avec fenêtre de temps . . . 25
2.3.1 Description . . . 25
2.3.2 Formulation . . . 26
2.4 Le problème d'élaboration de tournées de véhicule dynamique . . . 27
2.4.1 Description . . . 27
2.4.2 Formulation . . . 28
2.4.3 Classification du DVRP . . . 29
2.4.3.1 Problèmes faiblement dynamiques . . . 29
2.4.3.2 Problèmes modérément dynamiques . . . 30
2.4.3.3 Problèmes fortement dynamiques . . . 30
2.4.4 Degré de dynamisme . . . 30
2.4.4.1 Dynamisme sans fenêtres de temps (dod) . . . 30
2.4.4.2 Degré de dynamisme effectif (edod) . . . 31
2.4.4.3 Dynamisme avec fenêtres de temps (edod-tw) . . . 32
2.4.5 Les approches de résolution des problèmes de tournées de véhicules dynamique . . . 32
2.4.5.1 Problèmes de routage dynamiques déterministes . . . 32
2.4.5.2 Problèmes de routage dynamiques stochastiques . . . 33
2.5 Le problème de routage dynamique des véhicules avec fenêtres de temps . . . 33
2.6 Conclusion . . . 34

3 Resolution de DVRPTW par l'algorithme de système multi-colonies de fourmis (MACS) 35
3.1 Introduction . . . 35
3.2 La methode de MACS-VRPTW pour la resolution de VRPTW . . . 35
3.2.1 Le contrôleur . . . 36
3.2.2 Les colonies . . . 36
3.2.2.1 La colonie ACS-VEI . . . 36
3.2.2.2 La colonie ACS-TIME . . . 38
3.2.3 La représentation de probléme . . . 39
3.2.4 Construction de tour . . . 40
3.2.5 L'heuristique de plus proche voisin . . . 41
3.2.6 Insertion des nouveaux nœuds . . . 41
3.2.7 Recherche local . . . 42
3.2.8 Jeu de données utilisé . . . 43
3.3 Principe de résolution de DVRPTW . . . 44

3.4 La methode de MACS-DVRPTW pour la resolution de DVRPTW 45
3.4.1 Le contrôleur 45
3.4.2 Les colonies 46
3.4.3 La représentation de probléme : 46
3.4.4 Construction de tour 47
3.4.5 Insertion des nouveaux nœuds 48
3.4.6 Recherche local 49
3.4.7 Jeu de données utilisé 49
3.5 Conclusion 49

4 Implémentation **50**
4.1 Introduction 50
4.2 Description de l'environnement de travail 50
4.3 Outils de développement IntelliJ IDEA 51
4.3.0.1 Présentation 51
4.3.0.2 les langages supportées 51
4.3.1 Éditions de IntelliJ IDEA 52
4.3.1.1 IntelliJ IDEA Ultimate 52
4.3.1.2 IntelliJ IDEA Community Edition 52
4.3.1.3 IntelliJ IDEA Edu 52
4.4 Présentation de langage de programmation utilisé 52
4.5 Interface principale de l'application 53
4.6 Exemple de jeu de données 55
4.7 Expérimentation pour le problème de VRPTW 56
4.7.1 Paramètres d'exécution 58
4.7.2 Résultat obtenu pour les différentes instances 59
4.7.3 Evaluation 61
4.8 Expérimentation pour le problème de VRPTW dynamique 62
4.8.1 Exemple d'une instance de jeu de données dans le cas dynamique . 62
4.8.2 Exemples d'exécution 63
4.8.3 Résultat 64
4.8.3.1 Paramètres d'exécution 64
4.8.3.2 Discussion des résultats 65
4.9 Conclusion 65

Bibliographie **vii**

TABLE DES FIGURES

1.1 Classification des complexités de problèmes. 4
1.2 Classification des méthodes de résolution pour l'optimisation combinatoire. 7
1.3 Organigramme d'un algorithme génétique. 16

2.1 Exemple de probleme de VRP. 19
2.2 Classification de VRP selon la nature des données. 22
2.3 Exemple de fenêtre de temps sur un client. 27
2.4 Exemple de routage dynamique de véhicules. 28
2.5 Chronologie des événements pour le routage dynamique d'un seul véhicule 28
2.6 Classification du DVRP 29
2.7 Deux scénarios ayant des 'dod' équivalents. 31

3.1 Exemple pour un problème de routage de véhicules avec quatre dépôts dupliqués et quatre véhicules actifs. 40
3.2 Exemple de CROSS exchanges. Les sous-tours $X_1'Y_1$ et $X_2'Y_2$ sont échangés. Ces sous-trajets sont de longueur variable et l'un d'entre eux peut être vide. 42
3.3 Exemples de remplacements d'arêtes en 2-opt. (a) illustre un déplacement avec des arêtes provenant de différentes tournées. (b) est un exemple de déplacement dans une seule tournée. 42
3.4 Distribution des clients pour différentes catégories des problèmes du benchmark Solomon VRPTW avec 100 clients. 44

4.1 Interface de l'IDE IntelliJ Idea. 51
4.2 Interface principale de l'application. 53
4.3 Fenêtre permettent la création d'un jeu de test. 54
4.4 Exemple d'une instance de jeu de données. 55
4.5 Déroulement de l'algorithme sur l'instance c104. 56
4.6 Figure Montrant l'amélioration d'une solution 56
4.7 Déroulement de l'algorithme sur l'instance r101. 58
4.8 Exemple d'une instance de jeu de données dans le cas dynamique. 62
4.9 Exemple de simulation d'un problème dynamique 63
4.10 Resulat pour l'instance c103 avec un degré de dynamicité 10% 63
4.11 Resulat pour l'instance c103 avec un degré de dynamicité 50% 64

LISTE DES TABLEAUX

2.1 Caractéristiques du problème VRP 23

4.1 Caractéristiques des machines utilisées 50
4.2 Configuration des paramètres d'exécution 58
4.3 Résutat obtenu par notre implimentation dans les instance de type C1 . . 59
4.4 Résutat obtenu par notre implimentation dans les instance de type C2 . . 59
4.5 Résutat obtenu par notre implémentation dans les instances de type R1 . . 60
4.6 Résutat obtenu par notre implémentation dans les instances de R2 60
4.7 Résutat obtenu par notre implémentation dans les instances de type RC1 . 60
4.8 Résutat obtenu par notre implémentation dans les instances de type RC1 . 61
4.9 Présentation des moyennes de nos résultats et des résultats obtenus par l'implémentation originale 61
4.10 Configuration des paramètres d'éxecution 64
4.11 Résultats pour l'instance c106 avec diffèrent degré de dynamisme 65
4.12 Moyenne des résultats des 56 problèmes avec diffèrent degré de dynamisime 65

LISTE DES ALGORITHMES

1 Algorithme de la descente 10
2 Algorithme de Metropolis 11
3 Algorithme de recuit simulé 11
4 Algorithme de Recherche Tabou 12
5 Algorithme canonique de système de colonie de fourmis (ACO) 13
6 Controlleur 36
7 ACS-VEI(v) 38
8 ACS-TIME(v) 39
9 $ConstructTour(k, IN)$ 41
10 Contrôleur 46
11 $ConstructTour(k, IN)$ 48

LISTES DES ACRONYMES

VRP	Vehicle Routing Problem
ACO	Ant Colony Optimization
ACS	Ant Colony System
GPS	Global Positioning System
GIS	Geographic Information System
IHS	Intelligent Highway Systems
VRPTW	Vehicle Routing Problem with Time Windows
DVRPTW	Dynamic Vehicle Routing Problem with Time Windows
VRPB	Vehicle Routing Problem with Backhauls
DVRP	Dynamic Vehicle Routing Problem
SVRP	Stochastic Vehicle Routing Problem
PVRP	Periodic Vehicle Routing Problem
MDVR	Multi-Depot Vehicle Routing Problem
TSP	Traveling Salesman Problem
TRP	Traveling Repairman Problem
VRPPD	Vehicle Routing Problem with Pick-up and Delivery
OVRP	Open Vehicle Routing Problem
VRPHF	Vehicle Routing Problem with Heterogeneous Fleet
VRPSD	Vehicle Routing Problem with Split Delivery
m-VRP	Vehicle Routing Problem with Limited Number of Vehicles
MACS-VRPTW	Multiple Ant Colony System for Vehicle Routing Problem with Time Windows

INTRODUCTION GÉNÉRALE

De nos jours, les transports jouent un rôle important dans la vie des sociétés modernes. En raison de ses objectifs économiques et environnementaux, de nombreuses entreprises ont investi massivement dans la recherche scientifique, ainsi que dans la modernisation et le développement des infrastructures et de la sécurité des transports. Parmi les nombreux problèmes de transport, le transport routier est le plus important et le plus complexe. Compte tenu des contraintes spatiales et temporelles de ces problèmes, en plus de leurs ramifications économiques, leur résolution devient rapidement très difficile. C'est pourquoi de nombreux scientifiques et laboratoires de recherche se sont spécialisés dans la résolution des problèmes de transport. Ces recherches portent sur le transport de personnes, de marchandises, de fret, de matières dangereuses et de véhicules d'urgence.Elles concernent principalement la construction d'itinéraires de transport (VRP : Vehicle Routing Problem).

Le problème de routage des véhicules VRP est un problème classique qui nécessite de concevoir des itinéraires qui visitent tous les consommateurs tout en réduisant les coûts de transport en fonction de la distance, du nombre de véhicules, des besoins des clients et des capacités des véhicules.

Notre but est donc de concevoir et de construire un outil de résolution du problème de routage dynamique des véhicules (DVRP). Ce problème est l'un des problèmes les plus importants dans le domaine de la logistique d'entreprise. Les problèmes DVRP concernent ces dynamiques : l'apparence des clients, les temps de trajet, les temps de service ou la disponibilité des véhicules. L'un des aspects les plus souvent pris en compte du DVRP est l'aspect « temps » (temps de trajets et temps nécessaires pour accomplir les services). Ce travail porte sur une variante du problème de routage de véhicules avec des commandes changeant dynamiquement.

Ce mémoire est composé de quatre chapitres :
Dans le premier chapitre nous introduisons l'optimisation combinatoire et expliquons les notions de base de l'optimisation combinatoire, et quelques problèmes classiques d'optimisation combinatoire avant de passer aux méthodes et techniques (exactes et approximatives) de résolution des problèmes d'optimisation combinatoire.

Le deuxième chapitre présente le problème de routage des véhicules (VRP), sa formulation, sa classification et sa complexité ainsi que les variantes les plus importantes de ce

problème. Ensuite nous décrivons et nous donnons la formulation du problème de tournée de véhicules avec fenêtres de temps (VRPTW) qui est une variante très importante du VRP et après nous expliquons le problème de routage des véhicules dynamique. Dans la dernière partie nous décrivons le problème de routage dynamique des véhicules avec fenêtres de temps DVRPTW qui est le sujet de notre étude.

Le troisième chapitre présente ,le fonctionnement de l'algorithme MACS-VRPTW et le principe de résolution du problème de routage dynamique des véhicules avec fenêtres des temps, ainsi que les différentes mises à jour appliquées sur l'algorithme MACS-VRPTW pour traiter le problème de routage dynamique des véhicules avec fenêtre de temps .

Le dernier chapitre de ce document présente notre outil de résolution du problème de routage statique et dynamique des véhicules avec fenêtre de temps basé sur la méthode de systéme multi-colonies de fourmis (MACS-VRPTW) et les résultats obtenus dans le cas statique et dynamique.

CHAPITRE 1

L'OPTIMISATION COMBINATOIRE

1.1 Introduction

L'optimisation combinatoire définit un cadre formel pour de nombreux problèmes de l'industrie, de la finance ou de la vie quotidienne. Les problèmes d'optimisation combinatoire sont habituellement définis comme une problématique de choix d'une meilleure alternative dans un ensemble très grand mais fini d'alternatives. En raison du très grand nombre d'alternatives pour ces problèmes, l'ensemble des alternatives, dit aussi ensemble de solutions réalisables, est défini en compréhension, en d'autres termes les solutions réalisables se distinguent par un ensemble de propriétés ou de conditions, dites aussi contraintes, qu'elles doivent toutes remplir. Une évaluation est associée à toute solution réalisable à l'aide d'une fonction dite fonction objectif. Résoudre un tel problème consiste donc à trouver une solution optimale, c'est-à-dire trouver une solution réalisable qui minimise ou maximise selon le contexte la fonction objectif [1].
Dans ce chapitre, nous définissons ce qu'est un problème d'optimisation combinatoire et la théorie de la complexité. Ensuite, nous fournissons de courtes définitions des problèmes classiques d'optimisation combinatoire et les méthodes de résolution utilisées pour résoudre ce type de problème.

1.2 Définition d'un problème d'optimisation combinatoire :

Un problème d'optimisation consiste à chercher une instanciation d'un ensemble de variables soumises à des contraintes, de façon à maximiser ou minimiser un critère. Lorsque les domaines de valeurs des variables sont discrets, on parle alors de problèmes d'optimisation combinatoire [2].

1.3 Complexité

La théorie de la complexité est l'étude formelle de la difficulté des problèmes décidables. Elle désigne la quantité de ressources informatiques pour résoudre une tâche donnée [3]. Un problème est décidable s'il existe un algorithme, généralement une procédure qui se termine en un nombre fini d'étapes, qui répond par oui ou par non à la question posée par ce problème [4].

1.3.1 Complexité d'un algorithme

L'objectif de la théorie de la complexité est d'examiner les coûts des résolutions, notamment en termes de temps de calcul. Il cherche également à diviser les tâches en différents niveaux de difficulté.

En général, deux paramètres sont utilisés pour évaluer la complexité algorithmique :

Le temps alloué à l'exécution de l'algorithme est proportionnel au nombre d'instructions exécutées et à la quantité de données traitées.

L'espace mémoire requis est lié à la taille de l'instance d'un problème.

1.3.2 Complexité des problèmes

la théorie de la complexité consiste de classer les problèmes de décision en deux classes importantes P (problème polynomial) et NP (Problème Non-déterministe Polynomial).

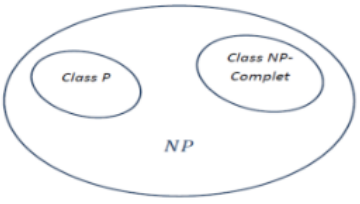

FIGURE 1.1 – Classification des complexités de problèmes.

— La classe des problèmes P est la classe de tous les problèmes qui peuvent être résolu par un algorithme déterministe en temps polynomial.

— La classe des problèmes NP est la classe de tous les problèmes qui peuvent être résolu par un algorithme non déterministe dans un temps polynomial.

Il exist d'autre classes de problèmes NP-complet et NP-difficile.

— Un problème est NP-Complet est résolu en un temps polynomial si et seulement si tous les problèmes NP-complet pouvant être résolu dans un temps polynomial [5].

— Un problème est NP-difficile s'il est plus difficile qu'un problème NP-complet, c'est à dire s'il existe un problème NP-complet se réduisant à ce problème. Les problèmes NP-difficile sont des problèmes d'optimisation [6].

1.4 Quelques problèmes classiques d'optimisation combinatoire

Quatre problèmes classiques d'optimisation combinatoire sont brièvement présentés dans cet section :

Le problème du sac-à-dos, le problème d'affectation, le problème du voyageur de commerce et le problème d'ordonnancement [2].

1.4.1 Problème du sac-à-dos

Considérons n objets, notés $i = 1, \ldots, n$ apportant chacun un bénéfice c_i mais possédant un poids a_i. On veut ranger ces objets dans un « sac » que l'on veut au maximum de poids b. Le problème de sac-à-dos (knapsack) consiste à choisir les objets à prendre parmi les n objets de manière à avoir un bénéfice maximal et respecter la contrainte du poids à ne pas dépasser. Chaque objet $i, i \in \{1, \ldots, n\}$, doit être sélectionné au moins p_i fois et au plus q_i fois.

Ce problème se rencontre bien entendu dès que l'on part en randonnée en voulant emmener le plus possible d'objets utiles (nourriture, boissons). Mais ce problème est plus fréquemment utilisé pour remplir les camions de transport, les avions ou bateaux de fret et même pour gérer la mémoire d'un microprocesseur.

La formulation PLNE du problème de sac-à-dos est très simple. On utilise pour chaque objet $i, i \in \{1, \ldots, n\}$ une variable entière x_i correspondant au nombre de fois où l'objet i est choisi. Le problème du sac-à-dos est donc équivalent au programme en nombres entiers suivant. [7].

$$\max \sum_{i=1}^{n} c_i x_i \tag{1.1}$$

$$\sum_{i=1}^{n} a_i x_i \leq b \tag{1.2}$$

$$p_i \leq x_i \leq q_i, \forall i \in \{1, \ldots, n\} \tag{1.3}$$

$$x_i \in \mathbb{N}, \forall i \in \{1, \ldots, n\} \tag{1.4}$$

La contrainte 1.2 est dite contrainte de sac-à-dos. Elle est l'unique contrainte de ce problème qui est pourtant NP-complet.

1.4.2 Problème d'affectation

Le "problème d'affectation" implique la construction de connexions entre les composants de deux ensembles séparés d'une manière à minimiser un coût et en respectant des contraintes d'unicité de lien pour chaque élément.

On considère m tâches et n agents, avec $m \leq n$. Pour tout couple (i, j) $i = 1$ à m , $j = 1$ à n, l'affectation de la tâche i à j entraîne un coût de réalisation noté $c_{i,j} (0 \leq c_{i,j})$. Chaque tâche doit être réalisée exactement une fois et chaque agent peut réaliser au plus

une tâche. Le problème consiste à affecter les tâches aux agents, de façon à minimiser le coût total de réalisation et en respectant les contraintes de réalisation des tâches et de disponibilité des agents [21].
À tout couple tâche/agent (i, j), on associe une variable d'affectation, $x_{i,j}$, binaire, qui prend la valeur 1 si la tâche i est affectée à l'agent j et 0 sinon. Le coût total de réalisation des tâches s'exprime alors par la somme :$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}$ Le nombre d'agents réalisant la tâche i est donné par :$\sum_{j=1}^{n} x_{i,j}$, pour tout $i = 1$ à m et le nombre de tâches réalisées par l'agent j est donné par :$\sum_{i=1}^{m} x_{i,j}$ pour tout $j = 1$ à n. On peut donc modéliser le problème d'affectation sous la forme [2] :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j} \tag{1.5}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \forall i = 1, \ldots, m \tag{1.6}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1, \forall j = 1, \ldots, n \tag{1.7}$$

$$x_{i,j} \in \{0, 1\}, \forall i = 1, \ldots, m, \forall j = 1, \ldots, n \tag{1.8}$$

1.4.3 Problème d'ordonnancement

Le "problème d'ordonnancement" consiste à séquencer et placer dans le temps un ensemble d'activités (entités élémentaires de travail),en respectant les contraintes temporelles (délais, contraintes d'enchaînement,) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les activité [8],donc le problème d'ordonnancement C'est un problème de satisfaction de contraintes qui trouve ses applications dans divers domaines (gestion de projets, ateliers de production, . . .) et qui fait l'objet de travaux de recherche d'un point de vue de l'aide à la décision, notamment par des approches par contraintes [9]. Dans un contexte d'optimisation, on cherche de plus à minimiser (ou maximiser) un critère, comme par exemple la durée totale de réalisation des activités [2].

1.4.4 Problème du voyageur de commerce

Étant donné un ensemble de villes et la distance à parcourir entre chaque paire de villes, le problème du voyageur de commerce consiste à trouver le chemin le plus court pour visiter toutes les villes et revenir au point de départ. Bien que le problème soit simple à énoncer, il est plus difficile à résoudre. Le problème du voyageur de commerce [10] [11] est un problème d'optimisation dont l'espace de recherche est vaste et qui est dit NP-difficile, ce qui signifie qu'il ne peut être résolu en temps polynomial.

C'est l'un des problèmes les plus fondamentaux dans le domaine de l'informatique à l'heure actuelle. Le problème du voyageur de commerce est appliqué dans de nombreux domaines de nos jours. Certaines de ses applications sont le routage de véhicules, la fabrication de puces électroniques, le routage de paquets, etc. Pour simplifier, disons que nous

avons un nombre n de villes, et que nous pouvons obtenir $(n-1)!$ itinéraires alternatifs pour couvrir toutes les n villes. Le problème du voyageur de commerce consiste à trouver la route qui présente la plus petite distance.

1.5 Méthodes de résolution pour l'optimisation combinatoire

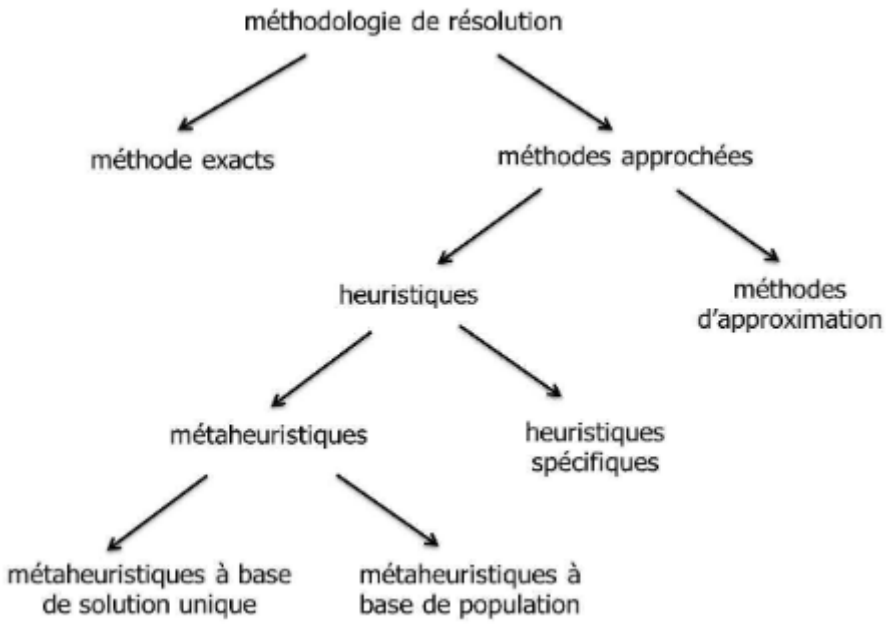

FIGURE 1.2 – Classification des méthodes de résolution pour l'optimisation combinatoire .

Les méthodes de résolution des problèmes d'optimisation combinatoire se divise en deux types : Les méthodes exactes et les méthodes approchés, cette classification est faite selon la qualité de la solution trouvée par la méthode de résolution, si la solution est optimale exacte on parle de méthode exacte, si par contre la solution est approchée on parle de méthode heuristique ou métaheuristique.

Nous présentons ici les méthodes et techniques de résolution des problèmes d'optimisation combinatoire qui sont plus utilisées pour résoudre le problème de routage des véhicules qui est notre sujet d'étude et qui sera présenté en détail dans le chapitre suivant.

1.5.1 Les méthodes exactes

Les méthodes exactes qui fournissent la meilleure solution sont limitées à de petites instances de problèmes. Dès que le nombre de variables de décision augmente, les temps de résolution avec les méthodes exactes augmentent rapidement.

Certaines approches exactes permettent cependant de réduire le nombre de solutions examinées. Il s'agit principalement des approches de type Branch and Bound, ou de type Branch and Cut et celles issues de la programmation dynamique [12].

1.5.1.1 Méthode Branch and bound

Appelé aussi (Procédure de séparation et d'évaluation) se base sur l'énumération et l'évaluation progressive de différentes solutions possibles. Le principe de cette méthode consiste à, d'une part, construire l'arbre de recherche et d'autre à couper les branches qui ne conduiront pas au résultat optimal [13]. Pour cela, une borne est calculée. Cette borne représente la valeur maximale atteignable si la branche est poursuivie. Une attention particulière doit être portée pour le calcul de la borne, en effet plus cette borne est pertinente, moins les branches sont poursuivies et donc moins il y a de calcul. Au pire des cas, la méthode va énumérer l'ensemble des solutions possibles au problème de départ.

1.5.1.2 Programmation Linéaire en nombres entiers (Branch And Cut)

Les approches précédentes ne sont plus utilisées lorsque le nombre de contraintes est trop grand. L'une des approches les plus efficaces pour résoudre précisément le VRP consiste à utiliser des stratégies de branche et de coupe. Un solveur PL (Linear Programming) est utilisé pour essayer de découvrir une solution optimale entière qui respecte les restrictions du problème PLNE (Programmation Linéaire en Nombres Entiers). Sinon, le problème doit être décomposé (i.e. Branch) en deux sous-problèmes, et la phase de découpage doit être relancée sur ces sous-problèmes [14].

1.5.1.3 Programmation Dynamique

Cette méthode est basée sur le principe de Bellman, qui stipule que toute solution au problème initial de taille N contient la solution optimale au sous-problème de taille $N-1$. En pratique, on part d'une famille de problèmes de taille 1 et on progresse jusqu'à une famille de problèmes de taille 2. Après un certain nombre d'étapes, nous arrivons au problème initial de taille N. Des états intermédiaires doivent être considérés à chaque étape et correspondent à un ensemble de problèmes à résoudre. Le nombre d'étapes et d'étapes intermédiaires doit être maintenu aussi bas que possible pour que la technique soit réalisable. Il est bien connu que la programmation dynamique ne peut résoudre qu'un petit nombre d'instances de problèmes d'optimisation combinatoire (10 à 25 clients dans notre cas) [15].

1.5.2 Les méthode approchées

Les méthodes approchées représentent une alternative pour résoudre les problèmes d'optimisation de grande taille lorsque les méthodes exactes ne sont pas capables de trouver une solution. Elles permettent d'obtenir des solutions de bonne qualité en un temps de calcul réduit [16]. Les méthodes approchées sont composées de heuristiques et de métaheuristiques.

1.5.2.1 Les heuristiques

Une heuristique est une méthode qui cherche (une stratégie) sans garantir le résultat. Destiné à un problème spécifique et ne peut pas être généralisée, le temps de calcul est raisonnable sans garantir la faisabilité ou l'optimalité [17].

— **Les méthodes constructives**
Les méthodes constructives partant d'une solution partielle initialement vide, il travaille d'une face itérative et construisent pas a pas une solution et ils cherchent a étendre a chaque étape la solution partielle de le l'étape précèdent , et ce processus se répète jusqu'à ce que l'on obtienne une solution [18]. En général, les méthodes constructives sont des algorithmes gloutons (c-à-d qu'elles ne remettent pas en question les choix exécutés lors des itérations précédentes) et ne sont pas efficaces sur les instances tests de la littérature [19]. Elles permettent cependant de construire rapidement de bonnes solution.

— **Les Méthodes à deux phases**
Le problème du VRP peut être décomposé en deux sous-problèmes : partitionner les clients en groupes (Clustering) et décider de l'ordre de parcours (Routing). Deux familles d'heuristiques sont plus couramment utilisées , les premières et les plus connues sont les méthodes "cluster first, route second" où les clients sont regroupés en clusters puis chaque cluster est transformé en tournée ; la deuxième famille d'heuristiques considère l'inverse puisque une route visitant tous les clients est construite puis partitionnée en plusieurs tournées [20].

— **Méthode amélioration**
Les méthodes d'amélioration appliquent des opérateurs heuristiques d échange de clients au sein des tournées pour améliorer la solution du VRP .il existe deux cas d'application de ces échange Dans le premier cas les échanges opérer sur des clients d'une mémé tournée dans le deuxième cas les échanges opérer sur des clients de plusieurs tournées et c'est pour sa les méthodes d'amélioration sont classer en deux classes selon le type d'amélioration des tournées (individuellement ou collectivement) [18].

1.5.2.2 Les métaheuristiques

Les métaheuristiques peuvent être considérer comme des heuristiques puissantes et évoluées dans la mesure où elles sont généralisables à plusieurs problèmes d'optimisation. Les métaheuristiques sont généralement classées en fonction du nombre de solutions qu'elles manipulent : les métaheuristiques à solution unique telles que la recherche tabou, et les métaheuristiques à population de solutions telles que que les colonies de fourmis [4]. Les méta-heuristiques sont généralement classées en fonction du nombre de solutions qu'elles traitent en deux classes.

1.5.2.2.1 Méta-heuristique avec solution unique Les méthodes itératives à solution unique sont toutes basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale, puis l'améliore pas à pas en choisissant une nouvelle solution dans son voisinage [22]. Nous présenterons ici les méthodes les plus utilisées et leur utilisation en extraction de connaissances : les méthodes de descente, le recuit simulé et la recherche tabou.

— **Méthode de la descente**
Dans le domaine de l'optimisation, la méthode la plus élémentaire et la plus intuitive est la méthode de descente. Cette méthode commence par générer aléatoirement

une solution initiale et, à chaque itération, met à jour la solution actuelle avec la meilleure solution de son voisinage jusqu'à ce qu'aucune autre amélioration ne soit possible. La sélection des solutions candidates parmi les solutions voisines se fait soit en évaluant la fonction objectif pour toutes les solutions présentes dans le voisinage et en choisissant la meilleure, soit en sélectionnant une solution aléatoire dans le voisinage et en remplaçant la solution courante par cette solution si elle améliore le critère, La procédure générale de cette méthode est présentée dans l'Algorithme 1 Le principal inconvénient de la méthode de descente est qu'elle se retrouve piégée dans le premier optimum local rencontré [23].

Algorithm 1 Algorithme de la descente

Nécessite : La fonction objectif f

Générer une solution aléatoire S
Calculer la fitness $f(S)$ associée à la solution initiale S
Initialiser la solution optimale : $S_{opt} \leftarrow S$
Tanque la condition d'arrêt n'est pas vérifiée **Faire**
  Générer la liste des solutions dans le voisinage de la solution courante
  Trouver la meilleure solution $S^{'}$ parmi les solutions voisines
  Si $f(S^{'}) < f(S)$ **Alors**
    $S \leftarrow S^{'}$
    $S_{opt} \leftarrow S$
  Fin Si
Fin Tanque
Retourner : La solution optimale S_{opt}

— **Recuit simulé**

Le recuit simulé est une méthode empirique inspirée d'un processus métallurgique appelé recuit, dans lequel un solide est chauffé à des températures élevées, puis laissé refroidir lentement pour atteindre des états de faible énergie du solid. Kirkpatrick et al [24] ont proposé l'algorithme de recuit simulé (et, indépendamment, Cerny [25]). Le recuit simulé est traditionnellement décrit comme une technique probabiliste dans laquelle un point évolue dans l'espace de recherche. Le recuit simulé est basé sur l'algorithme de Metropolis de l'algorithme 2 [26], qui permet de décrire l'évolution d'un système en thermodynamique. Lorsque la température T est élevée, cette approche permet à l'algorithme de quitter les minima locaux avec une forte probabilité et de conserver les états les plus probables lorsque la température T est basse. Dans ce cas, la technique de Metropolis (ou toute autre méthode d'échantillonnage) prend la place de la diversification, qui est contrôlée par la décroissance de la température. L'algorithme de recuit simulé est résumé en Algorithme 3 la méthode du recuit simulé a l'avantage d'être souple vis-à-vis des évolutions du problème et facile à implémenter. Elle a donné d'excellents résultats pour un certain nombre de problèmes, le plus souvent de grande taille [27].

Algorithm 2 Algorithme de Metropolis

Initialiser un point de départ x0 et une température T
Pour i=0 à n **Faire**
  Tanque x_i n'est pas accepte **Faire**
    Si $f(x_i) \leq f(x_{i-1})$ **Alors**
      accepter x_i
    Fin Si
    Si $f(x_i) > f(x_{i-1})$ **Alors**
      accepter x_i avec la probabilité $e^{\frac{f(x_i)-f(x_{i-1})}{T}}$
    Fin Si
  Fin Tanque
Fin Pour

Algorithm 3 Algorithme de recuit simulé

Déterminer une configuration aléeatoir S
Choix des mécanismes de perturbation d'une configuration
Initialiser la température T
Tanque Critère d'arrêt n'est pas satisfait **Faire**
  Tanque l'equilibre n'est pas atteint **Faire**
    Tirer une nouvelle configuration $S^{'}$
    Appliquer la régle de Metropolis
    Si $f(S^{'} < f(S))$ **Alors**
      $S_{min} \leftarrow S^{'}$
      $f_{min} \leftarrow f(S^{'})$
    Fin Si
  Fin Tanque
  Décroitre la température
Fin Tanque

— **Méthode tabou**
La méthode Tabou (tabu search l'algorithme 4) a été inventée par Glover dans les années 1980 et est devenue très conventionnelle dans l'optimisation globale . Le principe de cette méthode est à chaque itération le voisinage de la solution courante est examiné. L'algorithme enregistre la meilleur solution parmi les voisins, même si elle est moins bonne que la solution courante. L'acceptation des solutions moins performant que la solution courante permet d'éviter de tomber dans un optimum local. Pour échapper de tourner dans un cercle entre plusieurs solutions, l'algorithme interdit le passage par des solutions récemment visitées. En pratique la méthode stocke dans une liste taboue T les attributs des dernières solutions visitées. Dans l'itération suivante, la meilleure nouvelle solution voisine enlève la solution la plus ancienne dans la liste (algorithme suivant). Dans d'autres cas, la méthode mémorise les mouvements réalisés plutôt que les solutions. Ensuite, on interdit les mouvements inverses. Cette technique est rapide et consomme peu de mémoire [28].

Algorithm 4 Algorithme de Recherche Tabou

Initialiser la liste T (T vide)
S_0 point de départ
N : nombre d'itération maximal
$I \leftarrow 0$
$S_c \leftarrow S_0$ (solution courante)
Tanque i<N **Faire**
  $E \leftarrow$ voisinage $(S_c)/T$ (les voisins de S_c sans les élément de T)
  soit $y \in E$ et y meilleur solution dans E
  $T \leftarrow T + \{y\}$
  $S_c \leftarrow y$
  $I \leftarrow I + 1$
Fin Tanque

1.5.2.2.2 Méta-heuristique avec une population de solutions on Rappel par exemple l'algorithme de la colonie de fourmis et l'algorithme génétique

— **Optimisation par colonies de fourmis**
Le premier algorithme d'optimisation par colonies de fourmis (Ant colony optimization, ACO l'algorithme 5) a été propose par Dorigo [29] [30] dans les années 90s a été développé spécialement pour résoudre le problème du voyageur de commerce .La technique d'optimisation des colonies de fourmis s'inspire du comportement naturel des fourmis, qui interagissent entre elles en sécrétant des phéromones sur le sol, les fourmis peuvent trouver le meilleur chemin (le plus court) entre la source de nourriture et leur base en suivant le trajet des phéromones, les fourmis commencent par se déplacer au hasard. Puis Lorsqu'elles découvrent de la nourriture, elles retournent à leur colonie en laissant une trace de phéromones derrière elles. lorsqu'elles découvrent de la nourriture, elles retournent à leur colonie en laissant une trace de phéromones derrière elles .les autres fourmis cesseront probablement leurs mouvements aléatoires et rejoindront le chemin marqué s'il mène à la nourriture, renforçant le marquage à leur retour ,en conséquence, le chemin le plus court sera utilisé plus fréquemment, ce qui le rendra plus renforcé et plus attrayant, La quantité de phéromone déposée sur le chemin le plus long diminue avec le temps et finit par disparaître ,les fourmis empruntent alors le chemin le plus court [31].

1. **Algorithme de system de colonie de fourmis canonique ACO :**
Voyons maintenant les grandes lignes de l'algorithme d'optimisation ACO, qui s'inspire des fourmis réelles. Le pseudo-code d'un algorithme ACO canonique se trouve dans l'algorithme 5.

Algorithm 5 Algorithme canonique de système de colonie de fourmis (ACO)

initialisation des phéromones
Répéter
  Pour k = 1 au nombre de fourmis m **Faire**
    commencer la tournée de la fourmis k
    Pour i = 1 au nombre de nœuds atteignables n **Faire**
      choisir le prochain nœud de la tournée
    Fin Pour
    méthode de recherche locale (optionnel)
    mise à jour des phéromones locales (optionnel)
  Fin Pour
  évaluer les tours
  mise à jour de la meilleure solution trouvée jusqu'à présent
  mise à jour globale des phéromones
Jusqu'à condition d'arrêt
Retourner meilleure solution trouvée

Pendant l'initialisation, tous les niveaux de phéromone sont fixés à une valeur initiale. Après l'initialisation, la boucle principale de l'algorithme commence.chacune des m fourmis construit une route qui résout le problème. La fourmi est positionnée à un nœud initial, qui peut être choisi aléatoirement mais qui, pour le problème VRP, devrait être le dépôt. Ensuite, elle choisit de manière probabiliste le prochain nœud à incorporer dans sa route jusqu'à ce que tous les nœuds soient visités. Pour la fourmi k positionnée au nœud v_i, la probabilité $p_j^k(v_i)$ de choisir v_j comme prochain nœud est donnée par la règle de transition de l'équation 1.9.

$$p_j^k(v_i) = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{m \in N_i^k} [\tau_{im}]^\alpha \cdot [\eta_{im}]^\beta} & \text{if j } \in N_i^k \\ 0 & \text{if j } \notin N_i^k \end{cases} \tag{1.9}$$

avec :
τ_{ij} : niveau de phéromone sur l'arc (i, j)
η_{ij} : désirabilité heuristique du l'arc (i, j)
α : influence de τ sur la valeur probabiliste
β : influence de η sur la valeur probabiliste
N_i^k : ensemble de nœuds qui peuvent être visités par la fourmi k positionnée au nœud v_i
et τ_{ij},η_{ij},α,$\beta \geq 0$
Pendant la construction d'une tournée, l'ensemble N_i^k contient tous les nœuds accessibles à partir du nœud actuel et qui ne sont pas encore visités. Les nœuds qui ne sont pas disponibles ou qui ne peuvent pas être atteints en raison de contraintes ne font pas partie de N_i^k. Cela garantit qu'aucun nœud n'est visité deux fois et que seules des solutions réalisables sont créées. Le site désirabilité heuristique η_{ij} est un moyen d'incorporer des connaissances spécifiques au problème dans les probabilités. L'un des paramètres les plus couramment utilisés

pour le TSP est $\eta_{ij} = 1/d_{ij}$ où d_{ij} est la longueur de l'arête (i, j). Cette heuristique favorise les arêtes les plus courtes car elles obtiendront une valeur plus élevée pour η et donc une valeur plus élevée de p_j^k . De nombreuses heuristiques ont été utilisées pour différents problèmes.
Une fois qu'une fourmi a construit une route, la plupart des approches appliquent un algorithme de recherche locale pour améliorer la solution générée avec certaines connaissances spécifiques au problème. Une méthode de recherche locale bien connue pour le TSP est 2-opt (figure 3.3) [32]. Cette méthode optimise les tournées en éliminant les arêtes de croisement.
Lorsque toutes les fourmis ont construit une tournée, ces tournées peuvent être évaluées, après cette etape les pistes de phéromones sont ajustées. Toutes les pistes de phéromones sont d'abord diminuées d'un facteur $(1 - \rho)$ pour simuler l'évaporation des pistes existantes.Ensuite, la phéromone est ajoutée aux arcs qui font partie des tours qui ont été trouvés dans la dernière itération de l'algorithme en utilisant les équations 1.10 et 1.11.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{1.10}$$

$$\Delta\tau_{ij}^k = \begin{cases} 1/L_k & \text{if (i,j)} \in T_k \\ 0 & otherwise \end{cases} \tag{1.11}$$

avec :
ρ : paramètre d'évaporation de la phéromone
m : nombre de fourmis
$\Delta\tau_{ij}^k$: quantité de phéromones déposées par la fourmi k sur l'arête (i, j)
T_k : le tour trouver par la fourmi k
L_k : la longueur de la tournée T_k
et $0 \leq \rho \leq 1$

2. **System de colonie de fourmis ACS** nous sommes intéressés par L'ACS [33] c'est une variant de l'ACO utilise une mise à jour locale de la phéromone pour diminuer les niveaux de phéromone sur les arêtes qui sont traversées par les fourmis. Chaque fois qu'une fourmi a traversé une arête (i, j), elle applique l'équation 1.12. En diminuant les phéromones sur les arêtes déjà parcourues, il y a plus de chances que les autres fourmis utilisent d'autres arêtes. Cela augmente l'exploration et devrait éviter une stagnation trop précoce de la recherche.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{1.12}$$

avec :
τ_0 : la valeur initial de phéromone
la règle de mise à jour globale des phéromones donnée dans les équations 1.10 et 1.11 est également légèrement modifiée. Pour augmenter l'exploitation, les phéromones ne sont évaporées et déposées que sur les arêtes qui appartiennent à la meilleure solution trouvée jusqu'à présent et Δ_{ij} est multiplié par le paramètre de décroissance des phéromones ρ. Cela change les équations 1.10 et

1.11 en équations 1.13 et 1.14.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\sum_{k=1}^{m}\Delta\tau_{ij}^{k}, \forall(i,j)\in T^{*} \tag{1.13}$$

$$\Delta\tau_{ij}^{k} = 1/L^{*} \tag{1.14}$$

avec :
T^{*} : meilleur tour trouvé jusqu'à présent
L^{*} : longueur de T^{*}
La règle de transition utilisée pour déterminer le prochain nœud de la tournée est elle aussi modifiée. La règle de transition ACO donnée dans l'équation 1.9, n'est appliquée qu'avec une certaine probabilité q0. La nouvelle fonction est donnée par l'équation 1.15.

$$p_{j}^{k}(v_i) = \begin{cases} argmax_{j\in N_i}\left\{[\tau_{ij}]^{\alpha}\cdot[\eta_{ij}]^{\beta}\right\} & \text{if } q \leq q_0 \\ \text{Equation 1.9} & otherwise \end{cases} \tag{1.15}$$

avec :
q : un nombre aléatoire uniforme sur $[0,1]$
q_0 : paramètre qui détermine l'équilibre entre l'exploitation et l'exploration, et $0 \leq q_0 \leq 1$

— **Les algorithme génétique**
Sont des méthode bio-inspirée introduit par Holland 1975 [34] dans le cadre d'une analogie avec la sélection naturelle des espèces. Elle a été formalisée ensuite par Goldberg 1989 [35] pour être appliquée à la résolution de problèmes d'optimisation.
Les algorithmes génétiques ont été crées pour imiter des processus d'évolution naturelle. Ces méthodes se fondent sur le procèdes de reproduction et de sélection génétiques. En partant d'un ensemble de solutions faisables population) de cardinalité n (popsize), on affecte à chaque solution une certaine probabilité d'être sélectionee comme père, cette probabilité dépendant du coût de la solution. On procède ensuite au croisement entre ces solutions pour obtenir un ou plusieurs enfants par paire de solutions. Les pères choisis pour être croisés le sont en fonction des probabilités calculées précédemment. On sélectionne alors les n enfants en fonction du coût de chaque solution, et on recommence. A la manière d'un phénomène naturel, la méthode repose sur l'idée qu'au fur et à mesure que les générations se renouvellent, la valeur moyenne des solutions faisables s'améliore. Un algorithme génétique a donc la forme suivante :

1. Créer la population initiale : popsize solutions
2. Tant que le critère d'arrêt n'est pas vérifié faire
   (a) Évaluation
      i. Évaluer chaque individu i de la population
   (b) Sélection

i. Calculer, d'après l'ensemble de ces évaluations, la justesse proportionnelle (fitness) f(i) de chacun de ces solutions et ainsi la probabilité p(i) de chacune d'entre elles d'être choisie comme "reproductice".

ii. Constituer un ensemble de couples d'individus compte tenu de cette probabilité : (c) Reproduction Application des opérateurs génétiques pour sélectionner des couples.

(c) Reproduction

i. Application des opérateurs génétiques pour sélectionner des couples.

ii. Combiner chaque couple afin de produire une ou plusieurs) nouvelle solution issue des deux parents.

iii. Si le nombre des solutions produits est supérieur à pop size alors garder les popsize meilleurs "enfants" : (une nouvelle génération de taille pop size est donc constitué).

A cet algorithme de base peut être ajouté une "régénération" aléatoire de la population. Celle-ci peut être obtenue par un opérateur génétique appelé mutation selon une certaine fréquence, paramètre du programme, créer des enfants dont une partie des gènes sont aléatoires (mutants) et donc indépendants de ceux des deux parents, le reste de la solution étant construit classiquement par combinaison des gènes des parents [36].

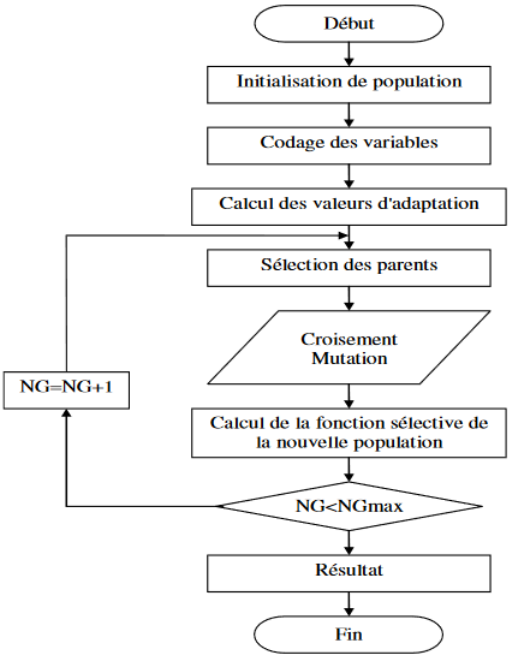

FIGURE 1.3 – Organigramme d'un algorithme génétique [37].

1.6 Conclusion

Les problèmes d'optimisation combinatoire sont des problèmes qui consistent à trouver la meilleure solution parmi un nombre fini, mais souvent très grand de solutions

réalisables. Ils font partie ainsi des problèmes classés NP-complets, qui requièrent souvent à des méthodes de résolutions approchées surtout pour les problèmes à grandes instances.

Dans ce chapitre nous avons présenté ce genre de problèmes, leur complexité, et leurs méthodes de résolution. Nous avons aussi présenté quelques exemples de problèmes souvent étudiés dans ce domaine, à savoir le problème du voyageur de commerce (TSP), qui est étroitement lié à notre sujet.

Dans le chapitre suivant nous présentons le problème de tournées de véhicules avec fenêtres de temps dynamiques, qui est une généralisation du problème cité précédemment (le TSP).

CHAPITRE 2

LE PROBLÈME DE TOURNÉE DE VÉHICULES AVEC FENÊTRE DE TEMPS DYNAMIQUE

2.1 Introduction

Les progrès des technologies de l'information et de la communication ont rendu le traitement des données en temps réel possible aujourd'hui. Les systèmes de positionnement global (GPS), les systèmes d'information géographique (GIS), les systèmes d'autoroutes intelligentes (IHS) et d'autres systèmes intégrés, en particulier, ont suscité un intérêt pour les méthodes et les stratégies permettant de résoudre le problème du routage des véhicules dans un cadre "en ligne", où les données peuvent être échangées en temps réel avec un outil d'aide à la décision.
En effet, dans de nombreuses circonstances du monde réel, les données du problème de VRP sont incomplètes lorsqu'il doit être résolu. Ces informations peuvent concerner, par exemple, les clients qui seront servis pendant la journée. Seul un petit pourcentage de ces clients peut être connu au moment de la planification. Cela signifie que de nouveaux clients peuvent demander des services au cours de la journée, une fois la planification terminée et alors que les véhicules sont déjà en route pour servir les clients précédemment programmés. Ces nouveaux clients doivent alors être intégrés dans la solution existante (c'est-à-dire le plane existant).
Dans ce chapitre nous avons commencé dans la première section par la description du VRP et sa formulation classique et sa classification selon la nature des données et leur complexité et les variantes les plus importantes de ce problème , dans la deuxième section nous avons décrit et formulé VRPTW qui est une variante très importante du VRP aprés nous avons expliquer le probleme de routage des véhicules dynamique ,et dans la dernière section nous avons décrit DVRPTW qui est le sujet de notre étude.

2.2 Problèmes de Tournées de véhicules

2.2.1 Description

Dantzig and Ramser (1959) ont été les premiers à proposer le "Truck Dispatching Problm", qui modélisait la manière dont une flotte de camions homogènes pouvait répondre à la demande en pétrole d'un certain nombre de stations-service à partir d'une plate-forme centrale tout en parcourant la plus courte distance possible. Clarke and Wright (1964), cinq ans plus tard, ont généralisé ce problème à un problème d'optimisation linéaire couramment rencontré dans le domaine de la logistique et du transport : comment servir un ensemble de clients géographiquement dispersés autour d'un dépôt central, en utilisant une flotte de camions.
Dans le problème de tournées de véhicules classique dit avec capacité il y a ,une flotte des véhicules de capacité fini basée dans un dépôt doit assurer des tournées entre plusieurs clients (villes) ayant demandé chacun une certaine quantité de marchandises. L'ensemble des clients visités par un véhicule désigne la tournée de celui-ci.

Condition de VRP [38] :

— La charge de commandes dans le chemin de distribution est inférieure à la charge maximale du véhicule.

— Chaque point de demande doit être satisfait et un seul véhicule doit être utilisé.

— Les voies doivent inclure tous les points de la demande, et chaque voie ne doit avoir qu'une seule voiture.

— Retour de chaque véhicule au centre de distribution lorsque la charge est épuisée.

Un exemple est donné en figure 2.1. Il décrit l'instance d'un VRP servant 9 clients sur 3 tournées.

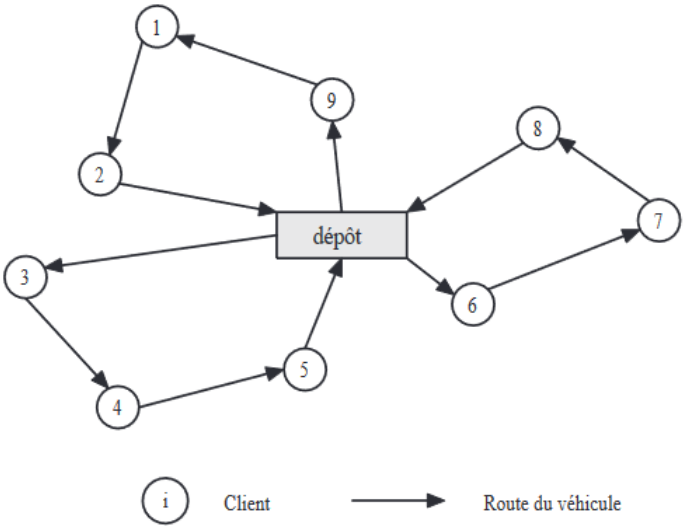

Figure 2.1 – Exemple de probleme de VRP [12].

2.2.2 Formulation

La Formulation de la version standard du VRP considère une contrainte de capacité [39] (CVRP, Capacited Vehicle Routing Problem). Il peut être représenté sous la forme d'un graphe orienté et value $G = (N, A)$. Ou :

— N représente les positions des clients et du dépôt
— A représente les arcs entre deux clients $i, j \in N$

Plus spécifiquement, nous avons un ensemble $C = \{1,\ldots,n_c\}$ de clients qui doivent obtenir une livraison de marchandise provenant du dépôt. L'ensemble des positions de ces clients ou nœuds est défini par l'ensemble $N = C \cup \{0, n_c + 1\}$ où 0 et $n_c + 1$ représente le dépôt (aller et retour). Une demande positive de produit d_i est associée à chaque client i appartenant à C.

Une flotte de véhicules $V = \{1, \ldots, n_v\}$ est disponible au dépôt et chaque véhicule possède la même capacité (flotte homogène) Q telle que $\max d_i \leq Q, \forall\ i \in N$. Pour tous les clients i et j, $\forall\ i,\ j \in N$, nous connaissons le coût de transport direct c_{ij} entre i et j (proportionnel à la distance à parcourir). Pour trouver l'ordre de visite des clients, nous définissons les variables de décisions comme suit :

$$x_{ij}^v = \begin{cases} 1 & \text{si le véhicule } v \in V \text{ visite le client j après le client i,} \\ 0 & \textit{sinon.} \end{cases} \tag{2.1}$$

En définissant y_i comme étant la charge résiduelle du véhicule après avoir desservi le client $i \in C$. Il nous est possible d'écrire formellement le modèle de VRP. Il s'agit d'optimiser la fonction suivante :

$$\min \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}^v \tag{2.2}$$

Avec les contraintes :

$$\sum_{v \in V} \sum_{j \in N} x_{ij}^v = 1, \forall i \in C \tag{2.3}$$

$$\sum_{i \in N} x_{ih}^v - \sum_{j \in N} x_{hj}^v = 0, \forall h \in C, v \in V \tag{2.4}$$

$$\sum_{j \in N} x_{0j}^v = 1, \forall v \in V \tag{2.5}$$

$$\sum_{j \in N} x_{j(n+1)}^v = 1, \forall v \in V \tag{2.6}$$

$$x_{ij}^v = 1 \implies y_i - d_j = y_j, \forall i, j \in N, v \in V \tag{2.7}$$

$$y_0 = Q, 0 \leq y_i, \forall i \in C \tag{2.8}$$

$$x_{ij}^v \in \{0, 1\}, \forall i, j \in N, v \in V \tag{2.9}$$

La fonction de coût de la solution X=(x_{ij}^v),$\forall i, j \in N, \forall v \in V$ est définie par :

$$\text{coût(x)} = \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}^v \tag{2.10}$$

Le nombre de véhicules utilisés par la solution X, est défini par :

$$\text{Nb vehicules(X)} = \sum_{v \in V} \sum_{j \in C} x_{0j}^v \tag{2.11}$$

La fonction objective (équation 2.2) représente le nombre de véhicules utilisés pour les trajets effectués et la somme des coûts s'y rapportant. La formulation du problème nécessite de satisfaire certaines contraintes :

— L'équation 2.3 assure qu'on part une et une seule fois de chaque client, avec un seul véhicule.

— L'équation 2.4 assure que le véhicule qui arrive chez un client est le même que celui qui part de ce client.

— L'équation 2.5 assure que chaque véhicule ne sort qu'une seule fois du dépôt.

— L'équations 2.6 assure le retour unique au dépôt pour chaque véhicule (ou tournée). Il n'ya pas de restriction sur le nombre de véhicule, mais un cout c_v est affecté à chaque véhicule utilisé. On impose une valeur de c_v su- samment grande pour réduire principalement le nombre de véhicule et pour réduire au minimum dans un deuxième temps les couts de transport.

— Les équations 2.7 et 2.9 définissent les contraintes de capacité et d'intégrité.

— Les équations 2.10 et 2.11 sont des fonctions de mesure qui permettent respectivement de quantifier la solution selon la distance totale parcourue, ainsi que le nombre de véhicules utilisés [12].

2.2.3 Classification de VRP

Dans la littérature, les critères de classification diffèrent d'un auteur à l'autre. (Ghiani et al. , 2003) [40] ont classé les VRP selon les caractéristiques des données (voir figure2.2). Si toutes les données nécessaires à la planification ne dépendent pas du temps, le problème est considéré comme statique , dans le cas contraire, on dit qu'il est dynamique.Le fait que toutes les données soient connues avec précision (sans incertitude) se traduit par la nature déterministe du VRP. Le VRP statique peut être déterministe ou stochastique. Le VRP dynamique, également connu sous le nom du VRP en temps réel, peut également être déterministe ou stochastique [40].

— Dans le VRP déterministe statique, également connu sous le nom de VRP classique, toutes les données sont connues avec certitude à l'avance et ne dépendent pas du temps.

— Les données sont connues avec certitude dans les problèmes déterministes dynamiques et seuls certains éléments sont dépendants du temps.

— La quantité de la demande, le temps du service au client ou la durée du trajet sont autant de sources possibles d'incertitude dans les problèmes statiques et sto-

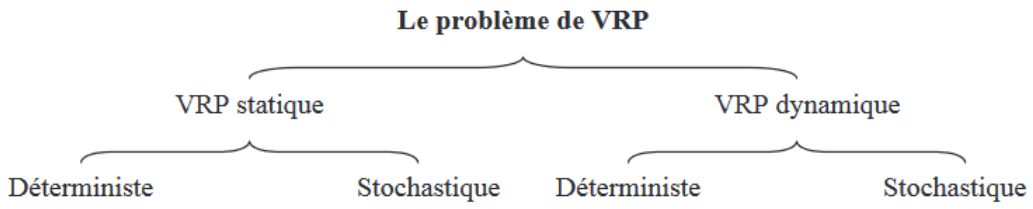

FIGURE 2.2 – Classification de VRP selon la nature des données [41].

chastiques. Le temps n'est pas pris en compte explicitement dans les données du problème.

— Dans les problèmes stochastiques et dynamiques, les données incertaines sont représentées par des variables aléatoires, de plus, des éléments dépendent du temps. Par exemple, dans, une partie des clients arrivent dans la journée, et le temps d'arrivée des clients suit une certaine loi de distribution.

2.2.3.1 VRP Statique

Un problème statique peut être soit déterministe, soit stochastique. Dans les VRP déterministes statiques (comme le VRP capacitif classique), toutes les données sont connues à l'avance et le temps n'est pas pris en compte explicitement. Dans les VRP stochastiques statiques, les itinéraires des véhicules sont conçus au début de l'horizon de planification, avant que les données incertaines ne soient connues. L'incertitude peut affecter les demandes de service présentes, les demandes des utilisateurs, les temps de service des utilisateurs ou les temps de déplacement. Si les données d'entrée sont incertaines, il est généralement impossible de satisfaire les contraintes pour toutes les réalisations des variables aléatoires. Si l'incertitude affecte les contraintes mais que la fonction objectif est déterministe, il peut être exigé que les contraintes soient satisfaites avec une probabilité donnée [40].

2.2.3.2 VRP Dynamique

Il existe deux types de problèmes dynamiques, soit déterministes soit stochastiques. Dans les problèmes dynamiques déterministes, les données confirmées sont connues à l'avance, avec seulement quelques informations dépendant du temps [42]. Les données incertaines sont représentées par des variables aléatoires dans les problèmes dynamiques stochastiques [43], connu sous le nom de problème VRP en temps réel. Les commandes des clients, par exemple, peuvent suivre une distribution de type Poisson. Parmi les événements qui conduisent à la modification de la planification, nous citons : 1. Réception d'une nouvelle candidature. 2. Dysfonctionnements du véhicule. 3. Perte de temps due aux embouteillages. Chaque événement doit être traité conformément aux plans du planificateur de flotte de véhicules.

2.2.4 Complexité de problème de VRP

Le problème d'élaboration de tournées de véhicules est un problème NP-difficile, c'est-à-dire qu'il n'existe pas à ce jour un algorithme déterministe pouvant résoudre ce problème en temps polynomial [44]. Pour des problèmes comportant un grand nombre de clients (> 100 clients), des méthodes approchées sont nécessaires pour les résoudre..

2.2.5 Variantes du problème VRP

Le tableau 2.1 [12] présente les caractéristiques des extensions du problème de tournées de véhicules qui permettent de décrire de nombreuses situations réelles

Caractéristiques	**Options possibles**
Nombre de véhicules disponibles	— un — plusieurs
Type de véhicule	— homogène — hétérogène
Capacité de véhicule	— finie — infinie
Dépôts	— un — plusieurs
Demandes des clients	— statiques (connues en avance). — dynamiques (apparaissent au cours de temps). — stochastiques (les demandes suivent des lois aléatoires). — fenêtre de temps.
Service proposé	— ramassage ou livraison — ramassage et livraison — ramassage avant livraison
Période considérée	— jour — semaine — périodique

Table 2.1 – Caractéristiques du problème VRP .

Nous présentons quelques variantes de VRP principle qui apparaissent dans la littérature :

— **VRPTW (Vehicle Routing Problem with Time Windows)** : Le problème de tournées de véhicule avec fenêtre de temps c'est un problème étudié depuis longtemps . ce problème est le même problème de VRP classique avec une contraint sur le début du service client qui doit être se situer dans un intervalle de temps précise pour chaque client [45].

— **VRPB (Vehicle Routing Problem with Backhauls)** : C'est une extension de VRP classique dont nous avons des produits à liver sur les clients et des marchandises

devons transporter du fournisseur au dépôt tell qu a chaque tournée le ramassage des marchandises doit être après la livraison des produits [46].

— **DVRP (Dynamic Vehicle Routing Problem)** : c'est une autre extension de VRP classique qui prend en compte beaucoup plus la vie pratique dont on a un ou plusieurs caractère dynamique tell qu'e l'apparition d'un nouveau client en cours de la journée où il faut change la planification des véhicules pour rependre aux nouvelles demandes [47]).

— **SVRP (Stochastic Vehicle Routing Problem)** : Lorsqu'au moins une composante d'un problème VRP est aléatoire, le problème est considéré comme aléatoire [48]. Autrement dit, un élément de la situation ne peut être connu avec certitude. Il peut s'agir des demandes des clients (la quantité à livrer ou à ramasser), du temps ou des dépenses de transport, ou de tous les clients à visiter. Le problème des demandes aléatoires a reçu le plus d'attention. La demande est alors supposée suivre une loi de distribution connue (généralement une distribution normale).

— **PVRP (Periodic Vehicle Routing Problem)** :Le problème PTVMP implique la livraison de la quantité requise d'un ou plusieurs produits à un groupe de clients dans un horizon de temps spécifié [49]. Dans ce cas, chaque consommateur dispose d'un stock de biens qui consomme une quantité déterminée de chaque jour. La principale responsabilité est de planifier les jours de livraison pour chaque client dans un délai spécifié, puis de coordonner les tournées de la flotte de véhicules pour effectuer les livraisons requises. Cette planification doit prendre en compte les niveaux de stocks qu'il faut maintenir chez les clients tout en minimisant le coût total de livraison sur tout l'horizon. Le problème présenté ici est une extension du problème traditionnel de routage des véhicules, avec l'ajout de la notion de temps et la nécessité d'intégrer des fonctionnalités de gestion des stocks dans la planification des transports.

— **MDVRP (Multi-Depot Vehicle Routing Problem)** : dans cette extension de VRP il y a plusieurs dépôts distribués géographiquement et chacun des véhicule doit commencer et retourner à son dépôt initial à la fin de la tournée [50].

— **TSP (Traveling Salesman Problem)** : TSP ou problème du voyageur de commerce peut être considérée comme une instance particulière du CVRP avec un seul véhicule et tous les clients ayant une demande nulle. le véhicule commence du dépôt visiter tours les clients et reviens au dépôt [51].

— **TRP(Traveling Repairman Problem)** : Le problème de tournée de dépanneur (TRP) est une version du problème de routage de véhicules VRP.L'objectif est de réduire le temps de réponse, qui est défini comme le total pondéré des temps de service client [52].

— **VRPPD (Vehicle Routing Problem with Pick-up and Delivery)** : Le problème de réception et de livraison (VRPPD) partage bon nombre des mêmes caractéristiques avec le VRP, car chaque client doit fournir deux emplacements géographiques distincts : un pour recevoir le produit et un pour livrer le produit.Ce type de problème introduit une contrainte de priorité : pour chaque course, le processus de ramassage du client doit venir en premier, suivi du processus de livraison [53].

— **OVRP (Open Vehicle Routing Problem)** : La principale différence entre ce problème et le VRP normal est que les véhicules ne sont pas tenus de retourner

au dépôt, ou s'ils le doivent, ils retournent aux clients qui leur sont assignés dans l'ordre inverse, de sorte que les itinéraires des véhicules ne sont pas fermés mais restent ouverts [54].

— **VRPHF (Vehicle Routing Problem with Heterogeneous Fleet)** :Ce type de problème vise à atteindre les objectifs du VRP en utilisant une flotte de véhicules de différents types. En fait, nous pouvons caractériser ces véhicules en fonction de leur capacité, de leur vitesse ou des frais de déplacement, entre autres facteurs [55].

— **VRPSD (Vehicle Routing Problem with Split Delivery)** : Si nécessaire, chaque consommateur peut être visité plusieurs fois. En d'autres termes, la commande d'un client peut être partagée entre plusieurs tournées. Contrairement à ce que l'on croit généralement dans le dilemme classique (VRP), la demande de chaque client peut être supérieure à la capacité du véhicule [56].

— **m-VRP (Vehicle Routing Problem with Limited Number of Vehicles)** : Nous essayons de résoudre le problème VRP avec un petit nombre de véhicules dans ce défi [57]. Sachant que dans un VRP de base, ce nombre devrait être illimité (ou très énorme), auquel cas nous voulons le maintenir aussi bas que possible.

2.3 Problème de tournée de véhicule avec fenêtre de temps

2.3.1 Description

Le problème de tournée de véhicules avec fenêtre temps(VRPTW (Vehicle Routing Problem with Time Windows - VRPTW [58], [59])) est une variation plus large du problème classique de tournée de véhicules dans le (VRPTW) une contrainte sur le début et la fin du service des clients est établie, c'est-à-dire que chaque client fixe une heure de début et de fin et doit être servi dans cet intervalle de temps . En outre, le dépôt central dispose d'un horizon de service, souvent connu sous le nom d'heure d'ouverture de la journée. Sa tâche consiste à déterminer une plage horaire durant laquelle les véhicules doivent effectuer leurs tournées. En raison des contraintes de temps, plusieurs véhicules seront nécessaires pour répondre aux besoins de tous les clients à l'horizon de service. Il est possible de limiter le nombre de véhicules déployés, mais cela peut avoir pour conséquence que certains clients ne reçoivent aucun service. Malgré son apparente simplicité, ce problème est difficile à résoudre, En fait, il a été prouvé que le problème VRP traditionnel est NP-difficile, et ce résultat peut être étendu à VRPTW, Ainsi, alors que la détermination d'une solution optimale pour les petites instances est relativement simple, elle devient rapidement impossible pour les instances moyennes ou grandes, La plupart des problèmes de la vie réelle, quant à eux, entrent dans cette deuxième catégorie, Malgré le fait que sa formulation soit plutôt restrictive, le problème VRPTW a un grand pouvoir descriptif, Il permet la simulation d'un très grand nombre d'applications du monde réel [59].

2.3.2 Formulation

Nous formulons le problème VRPTW en utilisant la formulation VRP classique déjà exprimée et en ajoutant les variables, constantes et équations suivantes [39] :
Variable à déterminer :
a_i = instant d'arrivée chez le client $i \in C$.
b_i = instant de début de service chez le client $i \in C$.
b_0^v = instant auquel le véhicule v quitte le dépôt.
$b_{n_c+1}^v$ = instant auquel le véhicule v retourne au dépôt.
w_i = temps d'attente chez le client $i \in C$.
Constantes connues.
e_i = borne inférieure de la fenêtre de temps du client $i \in C$.
l_i = borne supérienre de la fenêtre de temps du client i $\in C$.
c_{ij} = coùt du déplacement de i à $j : i, j \in C$.
t_{ij} = le temps de parcours entre les deux clients i et $j, i, j \in C$.
s_i = temps de service chez le client $i \in C$.

L'attente est permise lorsqu'un véhicule arrive trop tôt chez le client j après que le service soit n_i chez le client $i \in C$, autrement dit, avant e_j . Le temps de début de service chez le client $j \in C$ se définit comme étant $b_j = \{e_j, a_j\}$ où $a_j = \{b_i + s_i + t_{ij}\}$ et le temps d'attente chez le client j comme étant $w_j = b_j - a_j$. Il est alors possible d'écrire les contraintes supplémentaires de la formulation de VRP pour formuler le problème VRPTW :

$$x_{ij}^v = 1 \rightarrow b_i + s_i + t_{ij} \leq b_j, \forall i, j \in C, v \in V \tag{2.12}$$

$$x_{0j}^v = 1 \rightarrow b_0^v + t_{0j}^v \leq b_j, \forall j \in C, v \in V \tag{2.13}$$

$$x_{i(n+1)} = 1 \rightarrow b_i + s_i + t_{i(n+1)} \leq b_{n+1}^v, \forall i \in C, v \in V \tag{2.14}$$

$$e_i \leq b_i \leq l_i, \forall i \in C \tag{2.15}$$

$$e_0 \leq b_0^v \leq l_0, \forall v \in V \tag{2.16}$$

$$e_{n+1} \leq b_{n+1}^v \leq l_{n+1}, \forall v \in V \tag{2.17}$$

Le temps d'utilisation réel des véhicules dans la solution (X) est :

$$\text{temps horaire}(X) = \sum_{v \in V} \left(b_{n_c+1}^v - b_0^v \right) \tag{2.18}$$

Les contraintes (2.12 - 2.17) définissent les contraintes temporelles. La mesure (2.18) permet d'identfier le temps total d'utilisation des véhicules.

Dans le VRPTW il exist deux sous-class :

— VRPTW rigide que nous étudierons ici où le service doit impérativement être effectué dans la fenêtre de temps .
— VRPTW relâché ou le retard ou l'avance engendre uniquement une pénalité [60]

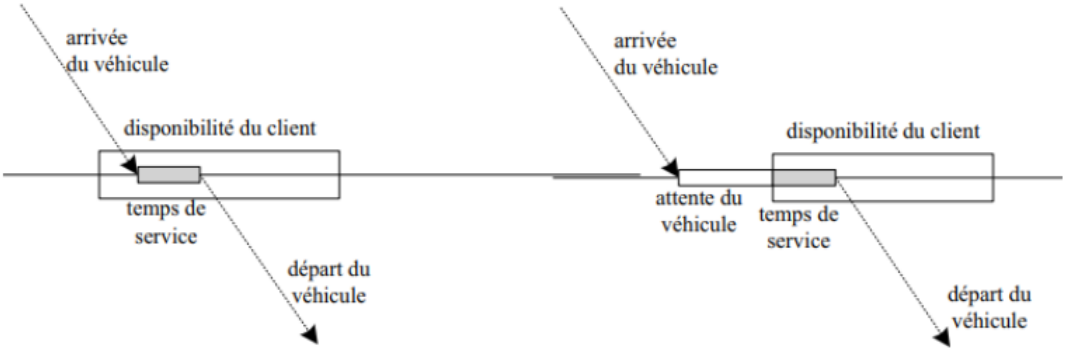

FIGURE 2.3 – Exemple de fenêtre de temps sur un client.

2.4 Le problème d'élaboration de tournées de véhicule dynamique

2.4.1 Description

Dans le domaine du transport, les systèmes de positionnement globaux (GPS), les systèmes d'information géographiques (GIS), les systèmes intelligents d'autoroutes et autres systèmes embarqués, ont augmenté l'intérêt des méthodes et des stratégies pour la résolution du problème d'élaboration de tournées de véhicules dans un contexte « en ligne » où l'échange d'information peut se faire en temps réel avec un outil d'aide à la décision. Par conséquent, les problèmes dynamiques de tournée et d'affectation de véhicules sont apparus comme un intense domaine de recherche dans la communauté de recherche opérationnelle [70] [71] [72].
La source la plus courante de dynamisme dans le routage des véhicules est l'arrivée en ligne des demandes des clients pendant l'opération. Plus précisément, les demandes peuvent être une demande de biens ou de services . Le temps de trajet et une composante dynamique de la plupart des applications du monde réel, a été pris en compte alors que le temps de service n'a pas été explicitement étudié (mais peut être ajouté au temps de trajet). Enfin, certains travaux considèrent des demandes révélées dynamiquement pour un ensemble de clients connus et la disponibilité des véhicules, auquel cas la source de dynamisme est la panne éventuelle des véhicules. [62].

La figure 2.4 illustre l'exécution de la route d'un D-VRP(le préfixe "D-" pour étiqueter les problèmes dans lesquels de nouvelles demandes apparaissent de manière dynamique) [62] à un seul véhicule pour mieux saisir ce que nous entendons par dynamique.

Un plan de route initial est créé avant que le véhicule ne quitte le dépôt (temps t_0) pour visiter les demandes actuellement connues (A, B, C, D, E). Au temps t_1, deux nouvelles demandes (X et Y) apparaissent pendant que le véhicule exécute son parcours, et l'itinéraire initial est modifié pour les accueillir.
Finalement, l'itinéraire exécuté au temps t_f est (A, B, C, D, Y, E, X).
Cet exemple montre comment le routage dynamique modifie constamment les itinéraires,

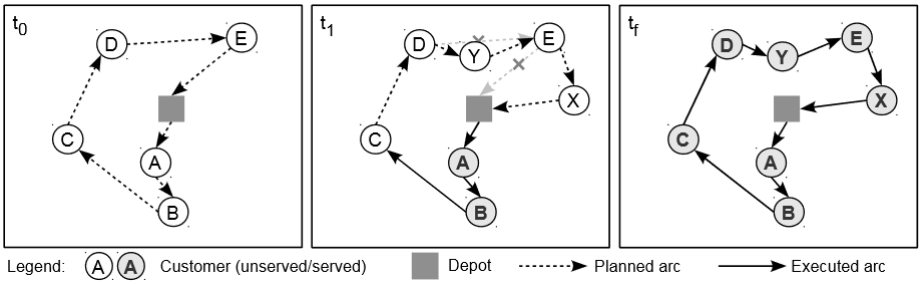

FIGURE 2.4 – Exemple de routage dynamique de véhicules [62]

ce qui nécessite une communication en temps réel entre les camions et les centres de répartition. La figure (2.5) illustre ce schéma de communication en temps réel où l'environnement fait référence au monde réel et où le dispatcheur est l'agent qui donne des instructions au véhicule. une fois que le véhicule est prêt (première flèche en pointillés), le dispatcheur prend une décision et donne l'ordre au véhicule de répondre à la demande A (première flèche à deux têtes). lorsque le véhicule commence (deuxième flèche en pointillés) et termine (troisième flèche en pointillés) son service à la demande A, il en informe le dispatcheur, qui met à jour les informations disponibles et communique au véhicule sa prochaine demande (deuxième flèche à deux têtes).

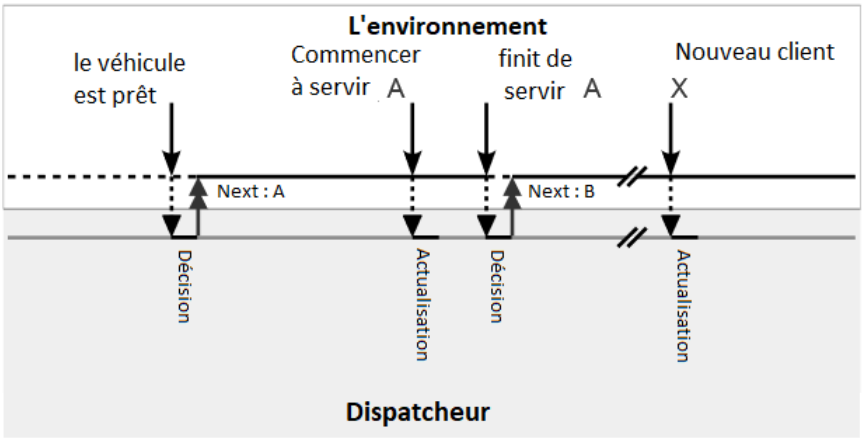

FIGURE 2.5 – Chronologie des événements pour le routage dynamique d'un seul véhicule [62] .

2.4.2 Formulation

Dans les sections précédentes, nous avons présenté la formulation du problème VRP et du VRPTW proposée par [Solomon 1987], [Antes et al. 1995], [Solomon et al. 1988].

Nous allons prolonger cette formulation pour que nous puissions spécifier simplement le problème dynamique. En fait, nous avons à formuler seulement l'évènement "réception d'une nouvelle demande". Variable de décision, à déterminer [60] :
tr_i = l'instant de réception de la demande du client
Pour étendre la formulation du VRPTW vers le DVRPTW, nous ajoutons les contraintes suivantes :

1. Contrainte 1 : $0 \leq tr_i < T$, $\forall i \in C$, T est l'instant de fermeture du dépôt.
2. Contrainte 2 : $tr_i = \alpha.e_i$, $\alpha \in [0, \ldots, 1]$, $\forall i \in C$.

Ces deux contraintes décrivent que l'instant de réception d'une demande est limité par le temps d'ouverture et de fermeture du dépôt, et que cet instant tri est toujours inférieur ou égal à la borne inférieure de la fenêtre de temps du client i.

2.4.3 Classification du DVRP

Nous allons utiliser la classification du DVRP présentée par [42]. Cette classification est faite en employant trois échelons de dynamisme. On distingue les problèmes faiblement, modérément, et fortement dynamiques (voir la figure 2.6).

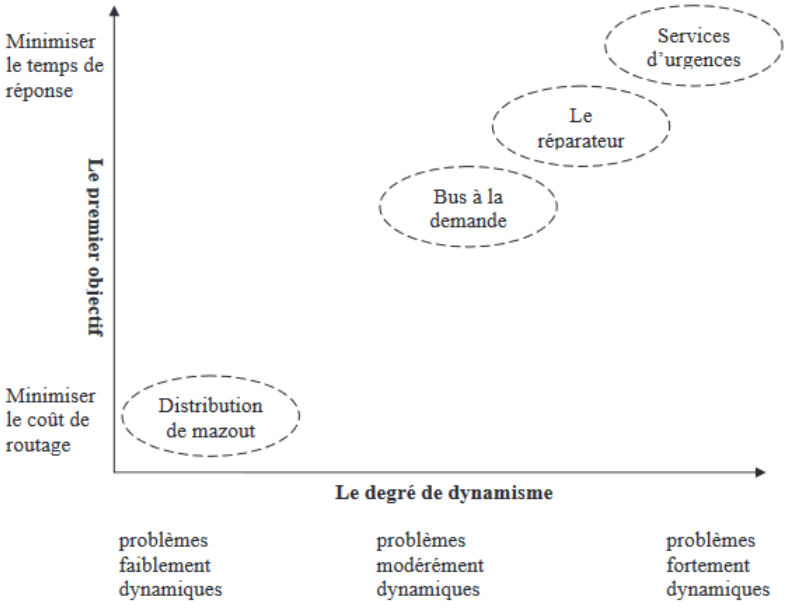

FIGURE 2.6 – Classification du DVRP

Cette figure montre que pour chaque catégorie de problème, l'objectif principal à atteindre peut varier de la minimisation du coût de routage qui correspond à la distance totale parcourue, à la minimisation du temps de réponse [60].

2.4.3.1 Problèmes faiblement dynamiques

Les problèmes de VRP avec un degré faible de dynamisme incluent la distribution de mazout ou de gaz liquide chez les particuliers. Dans cet exemple, la plupart des clients (plus de 80 %) sont connus à l'heure de la planification des routes. D'autres exemples

incluent des services d'installations chez les particuliers, tels que la télévision par câble et le téléphone. Le transport de personnes âgées ou handicapées peut être également classé comme un problème faiblement dynamique. En effet, l'objectif à atteindre est de minimiser un coût de routage tel que la distance totale parcourue par les véhicules. L'approche traditionnelle pour résoudre de tels problèmes est basée sur l'adaptation des procédures statiques. Un problème statique d'élaboration de tournées de véhicules est résolu chaque fois qu'un évènement se produit et change l'environnement du problème [60].

2.4.3.2 Problèmes modérément dynamiques

En comparant ces problèmes avec ceux ayant un degré de dynamisme faible, le nombre de demandes immédiates forment une partie substantielle du nombre total de clients. Des exemples pratiques de ces problèmes incluent les services de courrier ou les services de dépannage chez les particuliers [60].

2.4.3.3 Problèmes fortement dynamiques

Les services de secours, tels que la police, les pompiers et les ambulances montrent un comportement fortement dynamique [73]. Un autre exemple est celui du service des taxis dans lequel seul un nombre négligeable de clients ont commandé leur course à l'avance. En général, le but principal à satisfaire dans de tels problèmes est de minimiser le temps de réponse pour satisfaire la demande [60].

2.4.4 Degré de dynamisme

Contrairement à un problème statique d'élaboration de tournées de véhicules nous supposons que la performance dépend non seulement du nombre de clients et de la distribution spatiale de ces derniers, mais également du nombre d'évènements dynamiques et des instants où ces évènements ont lieu réellement. Par conséquent, une mesure pour décrire le dynamisme du système serait d'une grande utilité quand nous voulons examiner la performance d'un algorithme spécifique dans des conditions variables.

Le but de cette section est de présenter les mesures existantes données par [42], pour décrire le dynamisme d'un problème DVRP avec et sans fenêtres de temps.

2.4.4.1 Dynamisme sans fenêtres de temps (dod)

Nous examinons les mesures décrivant le dynamisme d'un problème DVRP sans fenêtres de temps. Dans un tel problème, trois paramètres seulement sont appropriés.

1. Le nombre de clients statiques.
2. Le nombre de clients dynamiques (appelés aussi demandes immédiates).
3. Les instants d'arrivée des clients dynamiques

Dans ce cas le degré de dynamisme (dod : Degree Of Dynamism) est défini comme ci-dessous [74] :

$$dod = \frac{\text{le nombre de clients dynamiques}}{\text{le nombre total de clients}} \tag{2.1}$$

Cependant, cette mesure ne tient pas compte des temps d'arrivée des demandes immédiates. Ceci signifie que deux systèmes différents, un dans lequel les demandes immédiates sont reçues au début de l'horizon de planification et un autre dans lequel elles se produisent tout au long de la journée, seront perçus comme étant équivalents (voir la figure 2.7)

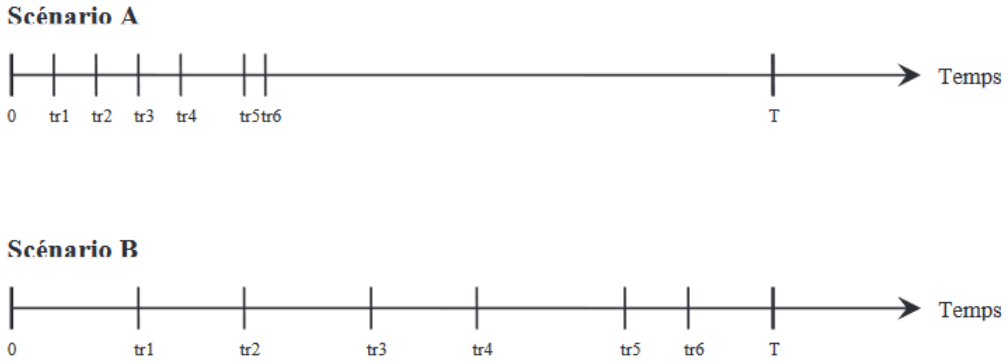

FIGURE 2.7 – Deux scénarios ayant des 'dod' équivalents.

Naturellement, dans la pratique ces deux scénarios auront des comportements très différents. Cette figure (2.7) illustre deux scénarios de DVRP dans lesquels les instants de réception des demandes immédiates diffèrent considérablement. Nous remarquons que dans le scénario A, chacune des six demandes immédiates est reçue relativement tôt pendant l'horizon de planification. Par contre, dans le scénario B, les demandes sont uniformément distribuées sur tout l'horizon de planification.
Du point de vue des performances, il est clair qu'avoir un nombre élevé de demandes en file d'attente améliore la qualité de la solution produite en ce qui concerne l'objectif de minimiser la distance totale parcourue. Par conséquent, la distance parcourue prévue serait éventuellement plus courte dans le scénario A que dans le scénario B, étant donné que le planificateur connaît à l'instant tr_6 toute l'information sur les positions des clients.

2.4.4.2 Degré de dynamisme effectif (edod)

Maintenant on considère un scénario dans lequel l'horizon de planification commence au temps 0 et finit au temps T . Les demandes statiques sont reçues avant le commencement de l'horizon de planification. Le temps de réception de la demande numéro i est noté tr_i : $0 < tr_i \leq T$. Le nombre de demandes immédiates reçues pendant l'horizon de planification est noté n_{imm} et le nombre de demandes statiques est noté n_{adv}. Le nombre total des demandes, n_{tot} est égal à $n_{tot} = n_{adv} + n_{imm}$. Le degré de dynamisme effectif, noté *edod*, est défini par [42] :

$$edod = \frac{\sum_{i=1}^{n_{imm}} \left(\frac{tr_i}{T}\right)}{n_{tot}} \tag{2.2}$$

Le degré de dynamisme effectif représente alors un pourcentage moyen du retard de chaque demande reçue par rapport au retard maximal admissible défini par T .
On peut facilement voir que : $0 \leq edod \leq 1$
Où quand $edod = 1$, le problème est complètement dynamique
et lorsque $edod = 0$, le problème est purement statique.

2.4.4.3 Dynamisme avec fenêtres de temps (edod-tw)

Finalement, nous avons déjà vu que pour le problème VRPTW, chaque client doit fournir deux limites de temps :

— e_i : est la borne inférieure de la fenêtre de temps du client i.

— l_i : est la borne supérieure de la fenêtre de temps du client i.

Le degré de dynamisme effectif peut alors être étendu pour intégrer cette notion et montrer l'urgence de la demande. Pour chaque demande i, le terme $l_i - tr_i$ représente l'horizon de décision pour traiter la demande. Plus celui-ci sera petit plus la décision sera urgente à prendre. Une valeur moyenne du caractère d'urgence peut alors être définie par :

$$\text{edod-tw} = \frac{1}{n_{tot}} \sum_{i=1}^{n_{tot}} \left(\frac{T - (l_i - tr_i)}{T} \right) \tag{2.3}$$

Aussi quand edod-tw = 1, le problème est complètement dynamique et lorsque edod-tw = 0, le problème est purement statique.

2.4.5 Les approches de résolution des problèmes de tournées de véhicules dynamique

2.4.5.1 Problèmes de routage dynamiques déterministes

En l'absence d'informations stochastiques, cette section couvre les méthodologies qui ont été efficacement appliquées au routage dynamique. Dans ce contexte, les informations critiques sont révélées au fil du temps, ce qui signifie que l'instance entière n'est connue qu'à la fin de l'horizon de planification. Par conséquent, les approches exactes ne fournissent que la meilleure solution pour la situation actuelle , elles ne peuvent garantir que la solution restera optimale lorsque de nouvelles données seront disponibles. Par conséquent, la majorité des techniques dynamiques reposent sur des heuristiques pour calculer rapidement une solution à l'état actuel du problème. Il existe deux types d'approches pour les problèmes de routage de véhicules dynamiques déterministes : celles qui utilisent la ré-optimisation périodique et celles qui utilisent la ré-optimisation continue [62].

2.4.5.1.1 Les approches de ré-optimisation périodique Les approches de ré-optimisation périodique commencent au début de la journée par une première optimisation qui produit un ensemble initial de routes. Ensuite, une procédure d'optimisation résout périodiquement un problème statique correspondant à l'état actuel, soit à chaque fois que les données disponibles changent, soit à intervalles de temps fixes - appelés époques de décision [61] ou tranches de temps [47]. L'avantage de la ré-optimisation périodique est qu'elle peut être basée sur des algorithmes développés pour le routage statique, pour lesquels des recherches approfondies ont été menées. Le principal inconvénient est que toute l'optimisation doit être effectuée avant la mise à jour du plan de routage, ce qui augmente les délais pour le répartiteur.

2.4.5.1.2 Les approches de ré-optimisation continue Les approches de ré-optimisation continue effectuent l'optimisation tout au long de la journée et conservent les informations

sur les bonnes solutions dans une mémoire adaptative [63]. Chaque fois que les données disponibles changent, une procédure de décision agrège les informations de la mémoire pour mettre à jour le routage actuel. L'avantage est que la capacité de calcul est maximisée, éventuellement au prix d'une mise en œuvre plus complexe. Il convient de noter que, le routage actuel étant susceptible d'être modifié à tout moment, les véhicules ne connaissent pas leur prochaine destination tant qu'ils n'ont pas terminé de répondre à une demande.

2.4.5.2 Problèmes de routage dynamiques stochastiques

Les problèmes de routage dynamique stochastique sont des extensions de leurs contreparties déterministes, dans lesquelles l'entrée révélée dynamiquement contient des connaissances supplémentaires (stochastiques). Il existe deux types d'approches pour ce type de problème : les approches basées sur l'échantillonnage (sampling) et les approches basées sur la modélisation stochastique (stochastic modeling) [62].

2.4.5.2.1 L'échantillonnage (sampling) Les stratégies d'échantillonnage incorporent des connaissances stochastiques en générant des scénarios basés sur des réalisations tirées de distributions de variables aléatoires. Chaque scénario est ensuite optimisé en résolvant le problème statique et déterministe qu'il définit. L'avantage de l'échantillonnage est sa relative simplicité et sa flexibilité quant aux hypothèses de distribution, tandis que son inconvénient est la génération massive de scénarios pour refléter fidèlement la réalité [62].

2.4.5.2.2 La modélisation stochastique (stochastic modeling) Les approches basées sur la modélisation stochastique intègrent les connaissances stochastiques de manière analytique. les stratégies de modélisation stochastique capturent formellement la nature stochastique du problème, mais leur formulation est très technique et nécessite de calculer efficacement des valeurs attendues éventuellement complexes [62].

2.5 Le problème de routage dynamique des véhicules avec fenêtres de temps

Le problème de routage dynamique des véhicules avec fenêtres de temps(DVRPTW), c'est le sujet principal de notre mémoire. Il est également connu sous le nom (DCVRPTW) de problème de routage dynamique de véhicules avec capacité et fenêtres temporelles, car des contraintes de capacité et de fenêtre de temps sont également appliquées [69]. un problème est dynamique lorsqu'une partie de l'entrée est révélée au solveur pendant l'optimisation(Psaraftis [72]). Cela signifie que nous ne pouvons pas construire une solution fixe, nous devons ajuster la solution pendant que le problème change. Nous nous concentrons sur l'ajout de nœuds à un problème initial.
Le problème a priori (initial) ne contient qu'une partie de tous les nœuds d'un problème. Pendant une journée de travail simulé de longueur T_{wd}, les autres nœuds deviennent visibles pour l'algorithme. Pendant la journée de travail, un calendrier provisoire est conservé. Les problèmes statiques gardent également la trace de la meilleure solution

globale trouvée par l'algorithme. La différence est qu'une solution provisoire sera très probablement irréalisable après un certain temps parce que de nouveaux nœuds sont devenus disponibles. De plus, une partie de la solution provisoire ne peut plus être modifiée car les nœuds ont déjà été desservis à l'instant t de la simulation [69].

2.6 Conclusion

Dans ce chapitre nous avons présenté le problème de routage dynamique des véhicules avec fenêtres de temps (VRPTW), qui est une généralisation du problème de tournées de véhicules (VRP). C'est l'un des problèmes d'optimisation combinatoire les plus étudiés dans le domaine du transport. Il consiste à visiter des clients à partir d'un ou plusieurs dépôts et au moyen d'une flotte de véhicules, avec un coût minimal. Nous avons aussi présenté les différentes classes et variantes de ce problème ainsi que les méthodes de résolution utilisées. Dans le chapitre suivant nous présentons l'approche adoptée dans ce travail pour résoudre ce problème.

CHAPITRE 3

RESOLUTION DE DVRPTW PAR L'ALGORITHME DE SYSTÈME MULTI-COLONIES DE FOURMIS (MACS)

3.1 Introduction

MACS-VRPTW est une approche d'optimisation basée sur les colonies de fourmis créée principalement pour résoudre le problème de tournée de véhicules avec fenêtre temps. MACS-VRPTW a été organisé avec deux colonies de fourmis artificielles pour améliorer la fonction multi-objectifs, la première colonie réduisant le nombre de véhicules et la seconde colonie réduisant la distance parcourue. La collaboration entre les colonies se fait par échange d'informations avec mise à jour des phéromones.
Dans ce chapitre nous verrons le fonctionnement de cet algorithme et le principe de résolution du problème de tournées des véhicules avec fenêtre de temps et les différentes mises à jour appliquées sur l'algorithme de MACS-VRPTW pour résoudre le problème de routage dynamique des véhicules avec fenêtres de temps.

3.2 La methode de MACS-VRPTW pour la resolution de VRPTW

Dans l'algorithme de MACS-VRPTW [64] deux fonction objective sont optimiséés simultanément par la coordination des activités des deux colonies ACS basique , le but de la première ACS-VEI, est de diminuer le nombre de véhicules utiliséés, la deuxième colonie, ACS-TIME permet l'amélioration de la solution trouver par ACS-VEI, les deux colonies utilise deux matrice de phéromone séparées et collaborer par le partage d'une variable global nomme T^* .
L'algrithme de MACS-VRPTW se compose de trois partie basique : Le contrôleur, ACS-TIME et ACS-VEI. Ces parties sont expliquées ci-dessous en détail avec leur fonctions principales.

3.2.1 Le contrôleur

Le contrôleur représente la partie principale de l'algorithme il lit le fichier de jeu de données du test et crée une solution initial en utilisant l'heuristique de plus proche voisin avec un nombre de véhicules illimité est donc si possible que la solution initial soit générée avec un nombre de véhicules plus grand que le nombre maximal de véhicules et ensuit le contrôleur calculer la valeur initial de phéromone dans l'étape suivant les deux colonies de fourmis sont actives telle que la colonie ACS-VEI est lancéé en donnant comme paramètre le nombre de véhicules dans la solution trouvée par l'heuristique de plus proche voisin moins 1 et la colonie ACS-TIME est lancéé avec le même nombre de véhicules dans la solution trouvée par l'heuristique de plus proche voisin. Les deux colonies cherchent à trouver une solution qui améliore la meilleur solution globale courant T^*. Si une solution avec moins de véhicules est trouvée, le contrôleur redémarre les deux colonies.

Algorithm 6 Controlleur

Étant donné : n est le nombre de nœuds
$T^* \leftarrow NearestNeighbor(n)$
$\tau_0 \leftarrow 1/(\text{ n . longueur } (T^*))$
Répéter
  Démarrer ACS-TIME(véhicules dans T^*) dans un nouveau thread
  Démarrer ACS-VEI(véhicules dans T^* - 1) dans un nouveau thread
  Tanque les colonies sont actives **Faire**
    Attendez qu'une solution T est trouvé
    Si véhicules dans $T <$ véhicules dans T^* **Alors**
      Arrêter les threads
    Fin Si
    $T^* \leftarrow T$
  Fin Tanque
Jusqu'à condition d'arrêt
Retourner T^*

3.2.2 Les colonies

Chacune des colonies tente d'améliorer un objectif différent du problème. La colonie ACS-VEI cherche une solution qui utilise moins de véhicules que T^*. La colonie ACS-TIME recherche une solution avec une distance de déplacement plus petite que T^* tout en utilisant au maximum autant de véhicules. Les deux objectifs ont une priorité fixe : une solution avec moins de véhicules est toujours préférée à une solution avec une plus petite distance.

3.2.2.1 La colonie ACS-VEI

La colonie ACS-VEI (l'algorithme 7) recherche une solution réalisable en maximisant le nombre de clients visités. ACS-VEI commence son calcul en utilisant $v - 1$ véhicules, c'est-à-dire un véhicule de moins que le plus petit nombre de véhicules pour lequel une solution réalisable a été calculée (c'est-à-dire le nombre de véhicules dans T^*). Pendant

cette recherche, la colonie produit des solutions infaisables dans lesquelles certains clients ne sont pas visités. Dans ACS-VEI, la solution calculée depuis le début de l'essai avec le plus grand nombre de clients visités est stockée dans la variable T^{VEI} . Une solution est meilleure que T^{VEI} lorsque le nombre de clients visités augmente. Par conséquent, ACS-VEI est différent de ACS traditionnel appliqué au TSP. Dans ACS-VEI, la meilleure solution actuelle T^{VEI} est la solution (généralement non réalisable) avec le plus grand nombre de clients visités, alors que dans ACS appliquée à la TSP la meilleure solution actuelle est la plus courte.

La colonie ACS-VEI a comme entréé le nombre maximal de véhicules que la solution produit par ACS-VEI ne doit pas dépasser, il est donné aussi à l'algorithme de la colonie de ACS-VEI la valeur initial de la matrice de phéromone τ_0,une variable T^{VEI} et utiliséé pour garder la meilleur solution trouver par la colonie .
La colonie ACS-VEI fonctionne comme suit :

1. Initialisation la matrice de phéromone par la valeur initial τ_0.
2. Création d'un tableaux nommé IN de taille égale a le nombre de nœuds (client) et initialisation des valeur de tableau par zéro .
3. Génération d'une solution initiale nommé T^{VEI} par l'heuristique de plus proche voisin ou $nearsetNeighbor$ en donnant le nombre maximal de véhicules comme paramètre cette solution peut ne pas contenir tous les nœuds (client) mais elle accepte la contrainte la plus important ici ,qui est de ne pas dépasser le nombre de véhicules maximal.
4. Création d'un nombre m de fourmis
5. Pour chaque fourmi construire un tour et stocker dans la variable T^k tell que k indique le numéro de la fourmi
6. Pour chaque nœud i qui n'existe pas dans le tour T^k incrémenter la valeur dans tableau IN correspondante .
7. Mise à jour local de phéromone sur les arcs de tour T^k en utilisant l'équation 1.12
8. Insertion des nœuds manquant dans solution T^k.
9. Trouver la fourmi l qui a le plus grand de noeuds (client) visiter.
10. Si le nombre de nœuds dans T^l et supérieure a le nombre de nœuds dans T^{VEI} alors mise à jour la valeur de T^{VEI} par T^l et réinitialiser les valeur de IN a 0 si de plus cette solution est faisable c'est à dire contenir tous les nouds alors envoyer la solution au thread Controlleur.
11. mise à jour global de phéromone avec la meilleure solution globale trouvée jusqu'à maintenant T^* par l'algorithme MACS-VRPTW en utilisant l'équation 1.13.
12. mise à jour globale de phéromone avec T^{VEI} en utilisant l'équation 1.13.

Répéter les étape 4,5,6,7,8,9,10,11,12 jusqu'a un signal de stop arrive du Controlleur.

Algorithm 7 ACS-VEI(v)

Entrée : v est le nombre maximal de véhicules à utiliser
Étant donné : τ_0 est le niveau initial de phéromone
Initialiser les phéromones à τ_0
Initialiser IN à 0
$T^{VEI} \leftarrow NearestNeighbor(v)$
Répéter
  Pour Tout Fourmi k **Faire**
    $T^k \leftarrow ConstructTour(k, IN)$
    Pour (Tout) noeud $i \notin T^k$ **Faire**
      $IN_i \leftarrow IN_i + 1$
    Fin Pour
    Mise à jour des phéromones sur les arcs de T^k en utilisant l'équation 1.12
    $T^k \leftarrow InsertMissingNodes(k)$
  Fin Pour
  Trouver la fourmi l avec les noeuds les plus visités
  Si noeuds dans $T^l >$ noeuds dans T^{VEI}. **Alors**
    $T^{VEI} \leftarrow T^l$
    Remettre IN à 0.
    Si T^{VEI} est réalisable **Alors**
      Retourner T^{VEI} au contrôleur
    Fin Si
  Fin Si
  Mise à jour globale des phéromones avec T^* et équation 1.13
  Mise à jour globale des phéromones avec T^{VEI} et équation 1.13
Jusqu'à le contrôleur envoie un signal d'arrêt

3.2.2.2 La colonie ACS-TIME

La colonie ACS-TIME (l'algorithme 8) est une colonie traditionnelle basée sur ACS dont le but est de calculer une tournée aussi courte que possible. Dans ACS-TIME m fourmis artificielles sont activées pour construire des solutions de problèmes $T^1, \ldots, T^m$. Chaque solution est construite en appelant la procédure ConstructTour, une procédure constructive expliquée en détail en plus tard qui est similaire à la procédure constructive ACS conçue pour le TSP. Lorsque $T^1, \ldots, T^m$ ont été calculées, elles sont comparées à T^* et si une solution est meilleure, elle est envoyée au Contrôleur. Le contrôleur utilise cette solution pour mettre à jour T^* . Après la génération des solutions, les mises à jour globales sont effectuées en utilisant l'équation 1.13 et T^*.
La colonie ACS-TIME a comme entrée le nombre maximal de véhicules que la solution produit par ACS-TIME ne doit pas dépasser, il est donné aussi à l'algorithme de la colonie de ACS-TIME la valeur initiale de la matrice de phéromone τ_0.
La colonie ACS-TIME fonction comme suit :

1. Initialisation la matrice de phéromone par la valeur initial τ_0.
2. Création d'un nombre m de fourmis

3. Pour chaque fourmi construire un tour et stocker dans la varible T^k tellque k indique le numéro de la fourmi
4. Mise à jour local de phéromone sur les arc de tour T^k en utilisant l'équation 1.12
5. Insertion des nœuds manquant dans solution T^k.
6. Si T^k est faisable alors appliquer un recherche local sur T^k.
7. Trouver la fourmi l qui a la solution faisable avec la distance minimale ou temps minimal. envoyer cette solution au Controlleur.
8. Mise a jour global de phéromone avec T^* en utilisant l'équation 1.13.

Répéter les étape 3,4,5,6,7,8 jusqu'a un signal de stop arrive du Controlleur.

Il existe quelques différences entre les deux colonies. ACS-VEI garde la trace de la meilleure solution trouvée par la colonie T^{VEI}, qui n'intègre pas nécessairement tous les nœuds. Comme T^{VEI} contribue également aux phéromones, cela aide ACS-VEI à trouver une solution qui couvre tous les nœuds avec moins de véhicules. ACS-TIME ne fonctionne pas avec les solutions infaisables et n'a pas de solution de type "colonie-best". Contrairement à ACS-VEI, il effectue une recherche locale. Le nombre maximum de véhicules qui peuvent être utilisés est donné comme argument à chaque colonie. Pendant la construction d'une tournée, ce nombre ne doit pas être dépassé. Cela peut conduire à des solutions infaisables qui n'intègrent pas tous les nœuds. Si une solution n'est pas faisable, elle ne peut jamais être envoyée au contrôleur.

Algorithm 8 ACS-TIME(v)

Entrée : v est le nombre maximal de véhicules à utiliser
Étant donné : τ_0 est le niveau initial de phéromone
Initialiser les phéromones à τ_0
Répéter
  Pour Tout fourmi k **Faire**
    $T^k \leftarrow ConstructTour(k, 0)$
    Mise à jour locale des phéromones sur les bords de T^k en utilisant l'équation 1.12
    $T^k \leftarrow InsertMissingNodes(k)$
    Si T^k est un tour réalisable **Alors**
      $T^k \leftarrow LocalSearch(k)$
    Fin Si
  Fin Pour
  Trouver la fourmi réalisable l avec la plus petite longueur de tournée
  Si Longueur de $T^l <$ Longueur de T^*. **Alors**
    Retourner T^l au contrôleur
  Fin Si
  Mise à jour globale des phéromones avec T^* et équation 1.13
Jusqu'à le contrôleur envoie un signal d'arrêt

3.2.3 La représentation de probléme

Une partie importante de l'algorithme est la manière dont une tournée est représentée. Chaque tournée doit visiter le dépôt plus d'une fois, ce qui représente le départ d'un

nouveau véhicule. Pour ce faire, le nœud de dépôt est copié un certain nombre de fois et toutes les copies peuvent être visitées individuellement (voir figure 3.1). Le nombre de copies est le même que le nombre maximum de véhicules autorisés, qui est différent pour ACS-TIME et ACS-VEI. En copiant les dépôts, nous permettons aux fourmis de déposer différents niveaux de phéromones dans différents dépôts, ce qui leur permet de trouver de meilleures solutions. La façon dont nous utilisons cette représentation du problème peut être vue dans la méthode ConstructTour décrite ci-dessous.

Les distances entre les copies du dépôt sont fixées à zéro. Cette approche rend le problème de tourné des véhicules plus proche du problème traditionnel du voyageur de commerce. Une solution réalisable est un chemin qui visite tous les nœuds exactement une fois. La figure 3.1 montre la solution d'un problème d'acheminement de véhicules représenté sous la forme d'une seule tournée. Les dépôts dupliqués $d_0, \ldots, d_3$ sont des points noirs et les clients des points blancs. Tous les dépôts dupliqués ont les mêmes coordonnées mais ils ont été divisés pour clarifier l'image.

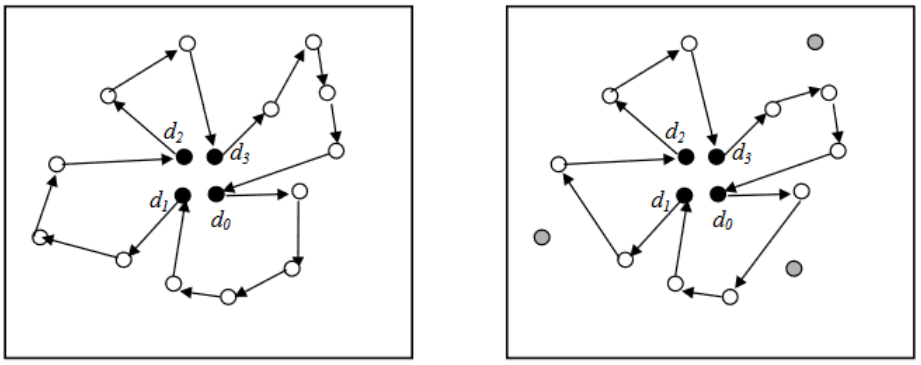

FIGURE 3.1 – Exemple pour un problème de routage de véhicules avec quatre dépôts dupliqués et quatre véhicules actifs.

3.2.4 Construction de tour

ACS-VEI et ACS-TIME utilisent la même procédure constructive ConstructTour qui est présentée en détail dans l'algorithme 9. Cette procédure constructive est similaire à la procédure constructive d'ACS conçue pour le TSP : Chaque fourmi artificielle part d'une copie du dépôt choisie aléatoirement et, à chaque étape, se déplace vers un nœud non encore visité qui ne viole pas les contraintes de fenêtre temps et les capacités des véhicules. L'ensemble des nœuds disponibles, dans le cas où la fourmi ne se trouve pas dans un dépôt dupliqué, comprend également les dépôts dupliqués non encore visités. Une fourmi positionnée dans le nœud i choisit de manière probabiliste le prochain nœud j à visiter en utilisant des mécanismes d'exploration et d'exploitation. Le site attractivité (η_{ij}) est calculée en prenant en compte le temps de déplacement t_{ij} entre les nœuds i et j, la fenêtre temporelle $[e_j, l_j]$ associée au nœud j et le nombre de fois où le nœud IN_j n'a pas été inséré dans la solution d'un problème. Lorsque la méthode ConstructTour est appelé par ACS-TIME, les variables IN ne sont pas utilisées et le paramètre correspondant est fixé à zéro.

Algorithm 9 $ConstructTour(k, IN)$

Entrée : k est la fourmi pour laquelle nous construisons une tournée
Entrée : IN est un tableau contenant le nombre de fois où les nœuds n'ont pas été incorporés dans les circuits
Étant donné : N_i^k est un ensemble de nœud et de doublons de dépôts qui sont atteignables par ant k dans le noeud i
Sélectionnez un dépôt aléatoire en double i
$T^k \leftarrow < i >$
$currentTime_k \leftarrow 0$
$load_k \leftarrow 0$
Répéter
  Pour each$j \in N_i^k$ **Faire**
    $deliveryTime_j \leftarrow max(currentTime_k + t_{ij}, e_j)$
    $deltaTime_{ij} \leftarrow deliveryTime_j - currentTime_k$
    $distance_{ij} \leftarrow deltaTime_{ij} * (l_j - currentTime_k)$
    $distance_{ij} \leftarrow max(1.0, (distance_{ij} - IN_j))$
    $\eta_{ij} \leftarrow 1.0/distance_{ij}$
  Fin Pour
  Choisissez le nœud j en utilisant l'équation 1.15
  $T^k \leftarrow T^k + < j >$
  $currentTime_k \leftarrow deliveryTime_j + serviceTime_j$
  $load_k \leftarrow load_k + q_j$
  Si j est une copie de dépôt **Alors**
    $currentTime_k \leftarrow 0$
    $load_k \leftarrow 0$
  Fin Si
  $i \leftarrow j$
Jusqu'à $N_i^k = \{\}$
Retourner T^k

3.2.5 L'heuristique de plus proche voisin

L'heuristique du plus proche voisin nommé NearestNeighbor [65]. est utilisée pour trouver les solutions initiales de l'algorithme entier et de la colonie ACS-VEI. Mais elle a été ajustée de deux manières. Tout d'abord, les contraintes sur les fenêtres de temps et la capacité sont vérifiées pour s'assurer qu'aucune tournée infaisable n'est créée. En outre, une limite sur le nombre de véhicules est transmise à la fonction. En raison de ces limitations, il n'est pas toujours possible de renvoyer une tournée intégrante tous les nœuds. Dans ce cas, une tournée avec moins de nœuds est retournée.

3.2.6 Insertion des nouveaux nœuds

Lorsqu'une tournée a été créée pour chaque fourmi, une heuristique apppelé Insert-MissingNodes est appliquée pour insérer les nœuds manquants dans ces tournées. Tous les nœuds manquants sont classés par quantités de livraison en ordre décroissant. Ensuite,

pour chaque nœud, le meilleur endroit d'insertion est trouvé, en se basant sur la distance de déplacement et en considérant qu'aucune contrainte n'est violée.

3.2.7 Recherche local

ACS-TIME met en œuvre une procédure de recherche locale pour améliorer la qualité des solutions réalisables. La recherche locale utilise des mouvements similaires aux CROSS exchanges (Taillard et al., 1997) [66] cette méthode et utiliser aussi par (gambardella et al) . Cette procédure est basée sur l'échange de deux sous-chaînes de clients. L'une de ces sous-chaînes peut éventuellement être vide, mettant en œuvre une insertion de clients plus traditionnelle et en applique aussi un operateur de recherche locale nommé 2opt. Cette méthode est appliquer seulement sur les tour qui contient tous les nœuds.

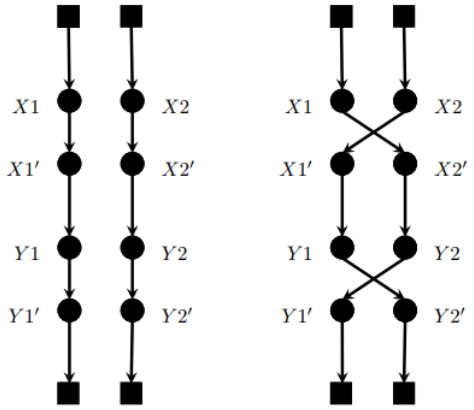

FIGURE 3.2 – Exemple de CROSS exchanges. Les sous-tours $X_1'Y_1$ et $X_2'Y_2$ sont échangés. Ces sous-trajets sont de longueur variable et l'un d'entre eux peut être vide.

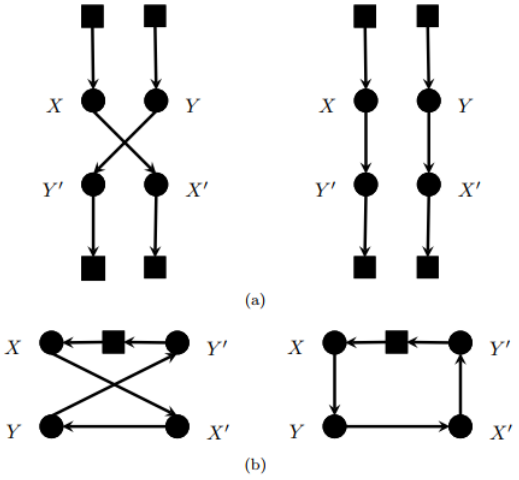

FIGURE 3.3 – Exemples de remplacements d'arêtes en 2-opt. (a) illustre un déplacement avec des arêtes provenant de différentes tournées. (b) est un exemple de déplacement dans une seule tournée.

3.2.8 Jeu de données utilisé

le VRPTW a été étudié de manière approfondie dans la littérature et des instances de référence ont été proposées afin de rendre les performances des différentes méthodes de résolution.
Dans cette section, nous décrivons les problèmes de référence les plus importants proposés pour VRPTW, car ils sont utilisés pour les performances des méthodes de résolution présentées dans ce travail et servent également de base pour générer les instances de test pour les problèmes de routage abordés. Les instances VRPTW les plus connues proviennent de Solomon (1987).
Dans le travail original, des instances avec différents nombres de clients ont été proposées (25, 50, 75, 100 clients), mais ces dernières années, seules les 56 plus grandes instances avec 100 clients ont retenu l'attention. Dans la suite de ce travail, elles sont simplement appelées instances Solomon.
Les instances de test proposées ont été générées pour représenter différentes caractéristiques du problème. Elles sont divisées en six groupes : C1, C2, R1, R2, RC1, RC2. chacun d'eux contenant entre 8 et 12 problèmes de test. Les groupes sont basés sur quatre distributions géographiques différentes des emplacements des clients : deux distributions en grappe groupées (C1 et C2), une distribution uniforme (R) et une combinaison de distributions groupées et uniforme (RC), comme le montre la figure 3.4, les groupes d'instances sont diffèrent en ce qui concerne l'horizon de planification du dépôt $[e_0 - l_0]$ et les capacités des véhicules.
Les groupes C1, R1 et RC1 ont un horizon de planification relativement court et des véhicules de faible capacité, ce qui se traduit généralement par des solutions comportant un nombre élevé de trajets courts contenant quelques clients avec un nombre élevé des véhicules de faible capacité.

Les groupes C2, R2, RC2 ont un horizon de planification considérablement plus long et des véhicules de grande capacité, ce qui donne des solutions avec un nombre relativement faible de véhicules effectuant des plus longs trajets et contenant un plus grand nombre de clients [67].

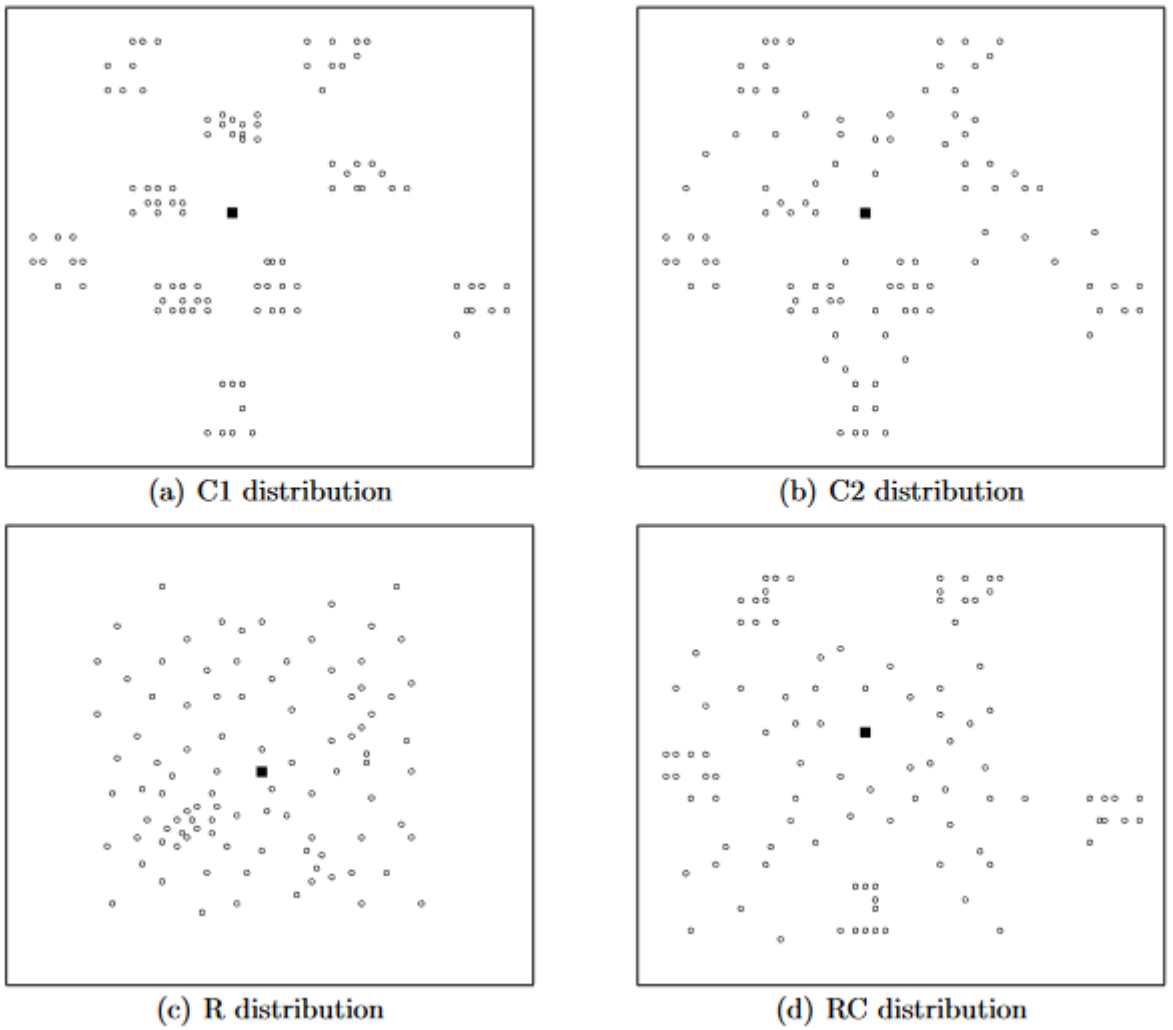

FIGURE 3.4 – Distribution des clients pour différentes catégories des problèmes du benchmark Solomon (VRPTW) avec 100 clients.

3.3 Principe de résolution de DVRPTW

Nous devons d'abord imiter une forme de dynamicité avant de pouvoir résoudre un problème dynamique.Kilby, Prosser et Shaw [47] ont décrit une méthode pour ce faire, qui est également utilisée par Montemanni et al [68]. On introduit la notion d'une journée de travail de T_{wd} secondes, qui sera simulée par l'algorithme. Tous les nœuds ne sont pas disponibles pour l'algorithme au début. On attribue à un sous-ensemble de tous les nœuds un temps de disponibilité à partir duquel ils deviendront disponibles. Ce pourcentage détermine le degré de dynamicité du problème.

Une tournée provisoire est construite au début de la journée, englobant tous les nœuds connus à l'avance. Ce tour provisoire est mis à jour tout au long de la simulation afin qu'il y ait toujours une solution réalisable au problème actuel. La journée de travail est divisée en n_{ts} tranches de temps de longueur T_{wd}/n_{ts}, qui sont également désignées par t_{ts}. Cela nous permet de décomposer le problème dynamique en une série de n_{ts} problèmes statiques qui peuvent être résolus dans l'ordre. Une autre option consiste à redémarrer

l'algorithme chaque fois qu'un nouveau nœud devient disponible. Comme l'algorithme pourrait être arrêté avant qu'une solution satisfaisante soit trouvée, cela pourrait avoir un impact important. Nous pouvons permettre à l'algorithme de travailler sans interruption pendant une durée déterminée en divisant la journée en tranches de temps. Au début de chaque tranche de temps, l'algorithme recherche les nœuds qui sont devenus disponibles au cours de la tranche de temps précédente. Ceux-ci sont alors ajoutés à la solution proposée. L'algorithme s'engage alors sur des portions de la solution tentative qui se produiront dans la tranche de temps suivante. Nous ne nous engageons sur un nœud à l'instant t que si le véhicule doit partir avant $t + t_{ts}$ secondes [69].

3.4 La methode de MACS-DVRPTW pour la resolution de DVRPTW

La structure de base de l'algorithme MACS-VRPTW reste inchangée, elle consiste en un contrôleur et deux colonies. Toutes les parties ont leurs propres extensions pour gérer la dynamicité du problème. Les objectifs de MACS-DVRPTW sont les mêmes que ceux de MACS-VRPTW. Le premier est de minimiser le nombre de véhicules, le second est de minimiser la distance parcourue.

3.4.1 Le contrôleur

Le contrôleur lit les données du jeu de données et initialise les structures de données. Ensuite, il construit une solution initiale en utilisant les nœuds disponibles à $t = 0$ et est prêt à démarrer la première tranche de temps. À ce moment-là un calculateur de temps est lancée qui garde la trace de t, le temps CPU utilisé en secondes. Au début de chaque tranche de temps, le contrôleur vérifie si des nœuds sont devenus disponibles au cours de la dernière tranche de temps. Si c'est le cas, ces nœuds sont insérés à l'aide de la méthode InsertMissingNodes pour rendre T^* à nouveau réalisable. Ensuite, tous les nœuds nécessaires sont engagés. Si v_i est le dernier nœud engagé d'un véhicule dans la solution indicative et v_j est le nœud suivant, v_j est engagé si $e_j - t_{ij} < t + t_{ts}$. Les colonies ne sont redémarrées que lorsque de nouveaux nœuds sont devenus disponibles ou lorsqu'une partie de T^* sera définie. Sinon, le contrôleur laisse les colonies fonctionner et attend la fin du prochain pas de temps.

Algorithm 10 Contrôleur

Fixer le temps t = 0
Définir les nœuds disponibles n
$T^* \leftarrow NearestNeighbor(n)$
$\tau_0 \leftarrow 1/($ n . longueur de $T^*)$
Début de la mesure du temps CPU t
Démarrer ACS-TIME(véhicules dans T^*) dans un nouveau thread
Démarrer ACS-VEI(véhicules dans T^* - 1) dans un nouveau thread
Répéter
Tanque les colonies sont actives et le pas de temps n'est pas terminé **Faire**
Attendre qu'une solution T soit trouvée
Si vehicles in T $<$ vehicles in T^* **Alors**
Arrêter les threads
Fin Si
$T^* \leftarrow T$
Fin Tanque
Si le pas de temps est terminé **Alors**
Si de nouveaux nœuds sont disponibles ou une nouvelle partie de T^* seront définis **Alors**
Arrêter les threads
Mise à jour des nœuds disponibles n
Insérer de nouveaux nœuds dans T^*
S'engager auprès des nœuds dans T^*
Fin Si
Fin Si
Si les colonies ont été arrêtées **Alors**
Démarrer ACS-TIME(véhicules dans T^*) dans un nouveau thread
Démarrer ACS-VEI(véhicules dans T^* - 1) dans un nouveau thread
Fin Si
Jusqu'à $t \geq T_{wd}$
Retourner T^*

3.4.2 Les colonies

En dehors du fait que les colonies ne fonctionnent que sur les nœuds disponibles, rien ne change. Les deux optimisent leur propre objectif bien que la colonie ACS-VEI soit limitée dans son fonctionnement car lorsqu'un véhicule est engagé, il ne peut pas être optimisé hors de la solution. Par conséquent, la colonie ACS-VEI n'est lancée que si au moins un véhicule dans T^* avec des nœuds indéfinis. Les pseudo-codes pour ACS-VEI et ACS-TIME peuvent être trouvés dans (les algorithmes 7 et 8) respectivement .

3.4.3 La représentation de probléme :

La représentation du problème ne change pas par rapport à la description Précédent. Il y a cependant un attribut supplémentaire à chaque nœud de la T^* qui détermine si un

nœud est engagé ou non. Une fois qu'un noeud est engagé, il ne peut plus être annulé et cette partie de la tournée ne peut plus être modifiée.

3.4.4 Construction de tour

La methode ConstructTour est considérablement modifié en raison du problème de la dynamique. Un changement majeur est que N_j^k dépend de la disponibilité des nœuds et du fait qu'ils soient engagés ou non. De plus, comme des parties de T^* sont engagées, elles doivent être incorporées dans chaque tournée créée par l'algorithme. Un pseudo-code ajusté de pseudo-code ajusté pour la méthode ConstructTour se trouve dans l'Algorithme11.

Algorithm 11 $ConstructTour(k, IN)$

Input : k est la fourmi pour laquelle nous construisons une tournée
Input : IN est un tableau contenant le nombre de fois où les nœuds n'ont pas été incorporés dans les circuits
Given : N_i^k est un ensemble de nœuds et de doublons de dépôt qui sont atteignables par la fourmi k dans le node i.
Véhicule actuel $x \leftarrow 0$
Sélectionnez un dépôt aléatoire i
$T^k \leftarrow < i >$
$currentTime_k \leftarrow 0$
$load_k \leftarrow 0$
Pour (Tout) noeud engagé v_i du x^{th} véhicule de T^*. **Faire**
  $T^k \leftarrow < i >$
  $currentTime_k \leftarrow deliveryTime_i + serviceTime_i$
  $load_k \leftarrow load_k + q_i$
Fin Pour
Répéter
  Pour each$j \in N_i^k$ **Faire**
    $deliveryTime_j \leftarrow max(currentTime_k + t_{ij}, e_j)$
    $deltaTime_{ij} \leftarrow deliveryTime_j - currentTime_k$
    $distance_{ij} \leftarrow deltaTime_{ij} * (l_j - currentTime_k)$
    $distance_{ij} \leftarrow max(1.0, (distance_{ij} - IN_j))$
    $\eta_{ij} \leftarrow 1.0/distance_{ij}$
  Fin Pour
  Choisissez le noeud j en utilisant l'équation 1.15
  $T^k \leftarrow T^k + < j >$
  $currentTime_k \leftarrow deliveryTime_j + serviceTime_j$
  $load_k \leftarrow load_k + q_j$
  Si j is a depot copy **Alors**
    $currentTime_k \leftarrow 0$
    $load_k \leftarrow 0$
    Pour (each) noeud engagé v_i du x^{th} véhicule de T^*. **Faire**
      $T^k \leftarrow < i >$
      $currentTime_k \leftarrow deliveryTime_i + serviceTime_i$
      $load_k \leftarrow load_k + q_i$
    Fin Pour
  Fin Si
  $i \leftarrow j$
Jusqu'à $N_i^k = \{\}$
Retourner T^k

3.4.5 Insertion des nouveaux nœuds

La méthode avec laquelle les nœuds manquants sont insérés est maintenant également appelée par le contrôleur pour insérer de nouveaux nœuds. L'idée de base est toujours la

même, mais lorsque le contrôleur appelle cette fonction, le nœud doit toujours être ajouté au problème, même si cela entraîne l'ajout d'un véhicule supplémentaire. C'est la façon de garantir qu'il existe toujours une solution provisoire réalisable. Lorsque cette méthode est appelée par les colonies, il n'est pas possible de créer un véhicule supplémentaire, seules les insertions réelles sont autorisées. De plus, les nœuds ne peuvent pas être insérés entre des nœuds qui sont déjà engagés.

3.4.6 Recherche local

La méthode de recherche locale ne change pas vraiment. La seule différence est que les parties de la tournée qui sont engagées ne peuvent plus être modifiées. Cela signifie que les sous-tours qui sont échangés ne peuvent contenir nœuds engagés et que les sous-tours ne peuvent pas être insérés entre les nœuds engagés.

3.4.7 Jeu de données utilisé

Les jeux de données utilisés pour cet algorithme sont basés sur les mêmes problèmes de Salomon sur lesquels MACS-VRPTW a été testé. Du temps supplémentaire a été accordé à chaque nœud (L'heure à partir de laquelle le client est disponible). Pour plus d'informations sur la création des jeux de données, vous pouvez vous référer à [69] .

Les problèmes de référence ont été créés pour différents degrés de dynamisme allant de 0 à 50 %.Les problèmes de référence à 0 % sont utilisés pour la comparaison avec les problèmes statiques. Pour un problème avec 10 % de dynamicité cela signifie que chaque nœud a un changement de 10 % pour obtenir un temps disponible non nul. Parce que Les temps disponibles étant générés sur l'intervalle $[0, \bar{e_i}]$ ($\bar{e_i} = \min\{e_i, t_{i-1}\}$,$t_{i-1}$ est le temps de départ du prédécesseur de v_i dans la meilleure solution connue) cela ne signifie pas nécessairement que le temps généré sera différent de zéro, car la fenêtre de temps d'un nœud peut commencer à zéro. Ainsi, un problème de référence avec une dynamique de 10 % signifie qu'au plus 10 % de tous les nœuds ont un temps disponible non nul.

3.5 Conclusion

Dans ce chapitre nous avons vu un algorithme de système multi colonies de fourmis MACS-VRPTW qui est basé sur le système de colonies de fourmis (ACO) et qui permet de résoudre le problème multi-objectif de tournée de véhicules avec fenêtre temps. Après cela, nous avons vu le principe de la dynamicité du problème de routage dynamique des véhicules avec fenêtres de temps et comment adapter l'algorithme MACS-VRPTW pour gérer la dynamicité et résoudre le problème de DVRPTW.
Dans le chapitre suivant, nous verrons l'implémentation de l'algorithme MACS-VRPTW et la simulation de différentes dynamiques et résultats obtenus dans les différentes situations de problème statiques et dynamiques.

CHAPITRE 4

IMPLÉMENTATION

4.1 Introduction

La méthode du système de multi colonies de fourmis et une méthode de résolution approximative basée sur l'optimisation par colonies de fourmis. Dans ce chapitre nous allons voire la mise en œuvre de la méthode de multi colonies de fourmis (MACS-VRPTW) et nous allons tester l'efficacité de cette méthode pour résoudre le problème de routage des véhicules dans le cas statique et dynamique pour cela nous avons créé une application qui permet de résoudre le problème de routage des véhicules avec fenêtre de temps et de simuler la résolution du problème de routage dynamique des véhicules avec fenêtre de temps et nous allons présenterons et évaluerons les résultats obtenus dans différentes situations.

4.2 Description de l'environnement de travail

Notre application a été développée dans un environnement Windows, sous le système d'exploitation windows 10. Cette version a été édité par le géant Microsoft et elle est très utilisée pour sa stabilité, sa rapidité et son efficacité.Nous avons fait implimentation et les test sur deux machine .

TABLE 4.1 – Caractéristiques des machines utilisées

	Machine1	**Machine2**
Marque	TOSHIBA	DELL
Processeur	Intel(R) Core(TM) i3-5005U CPU 2.00GHz 2.00GHz	Intel(R) Core(TM) i5-4310U CPU 2.00GHz 2.60GHz
RAM	8GB	8GB

4.3 Outils de développement IntelliJ IDEA

4.3.0.1 Présentation

IntelliJ IDEA est un environnement de développement intégré (IDE) pour les langages JVM (les langages qui s'exécutent sur la machine virtuelle Java)conçu pour maximiser la productivité des développeurs. Il effectue les tâches routinières et répétitives à votre place en fournissant une complétion de code intelligente, une analyse de code statique et des améliorations, et vous permet de vous concentrer sur le côté positif du développement logiciel, le rendant non seulement productif mais aussi une expérience agréable.
IntelliJ IDEA est un IDE multiplateforme qui offre une expérience cohérente sur Windows, macOS et Linux.

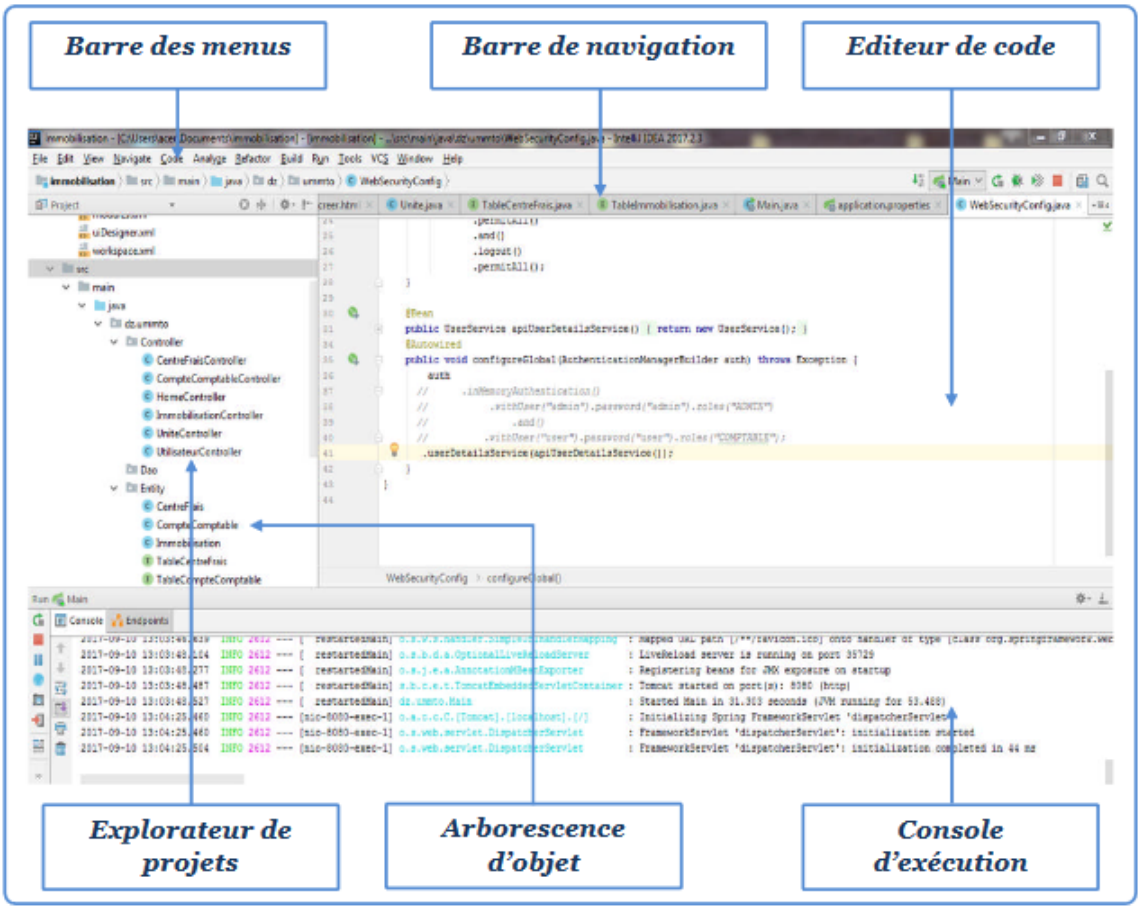

FIGURE 4.1 – Interface de l'IDE IntelliJ Idea

4.3.0.2 les langages supportées

Le développement d'applications modernes implique l'utilisation de plusieurs langages, outils, frameworks et technologies. IntelliJ IDEA est conçu comme un IDE pour les langages JVM mais de nombreux plugins peuvent l'étendre pour offrir une expérience polyglotte.Permet les langages supportées : Java, Kotlin, Scala, Groovy,Python ,Ruby ,PHP ,SQL ,Go ,JavaScript

4.3.1 Éditions de IntelliJ IDEA

4.3.1.1 IntelliJ IDEA Ultimate

l'édition commerciale pour le développement JVM, web et entreprise. Elle comprend toutes les fonctionnalités de l'édition Community, plus la prise en charge des langages sur lesquels les autres IDE basés sur la plate-forme IntelliJ se concentrent, ainsi que la prise en charge d'une variété de frameworks côté serveur et frontaux, de serveurs d'applications, l'intégration avec des outils de base de données et de profilage, et plus encore.

4.3.1.2 IntelliJ IDEA Community Edition

l'édition gratuite basée sur le code source libre pour le développement JVM et Android.

4.3.1.3 IntelliJ IDEA Edu

l'édition gratuite avec des leçons intégrées pour l'apprentissage de Java, Kotlin et Scala, des tâches de programmation interactives et des devoirs, ainsi que des fonctionnalités spéciales pour les enseignants afin de créer leurs propres cours et gérer le processus éducatif .

4.4 Présentation de langage de programmation utilisé

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'UNIX, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modifications. C'est la plate-forme qui garantit la portabilité des applications développées en Java. Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tel que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.)

4.5 Interface principale de l'application

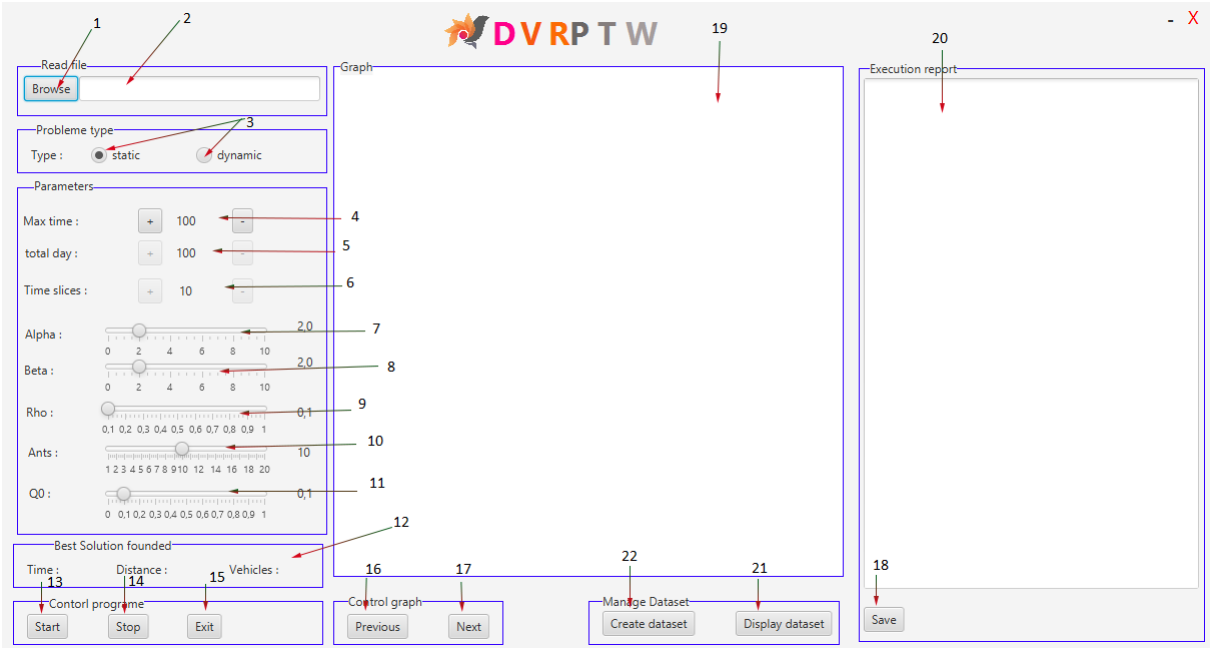

FIGURE 4.2 – Interface principale de l'application

— (1) le button Browse : permet de lire un fichier de jeu de donnée

— (2) Un champe de texte permet de visualiser le chemin de fichier sélectionné

— (3) Deux buttons permet de sélection le type de problème a traiter

— (4) paramètre pour définir le temps (En secondes) maximal d'exécution de programme pour le cas statique

— (5) permet de définir le temps (En secondes) de jour de travail pour le cas dynamique (l'intervalle des valeur de ce paramètre et entre 100 secondes et le temps de la fin de la fenêtre de temps de dépôt)

— (6) permet de définir le nombre de tranche de temps pour le cas dynamique (l'intervalle des valeur entre 10 et 1000)

— (7) paramètre (alfa) α de l'algorithme (qui détermine l'influence de τ sur la valeur probabiliste) on a spécifier l'interval des valeurs possible pour ce paramètre entre $[0, 10]$

— (8) paramètre (beta) β de l'algorithme (qui détermine influence de η sur la valeur probabiliste) on a spécifier l'intérval des valeurs possible pour ce paramètre entre $[0, 10]$

— (9) paramètre (rho) ρ (coefficient d'évaporation de phéromone) l'interval des valeurs possible pour ce paramètre et entre $[0.1, 1]$

— (10) paramètre pour definir le nombre de fourmis à utiliser

— (11) paramètre (Q0) q_0 de l'algorithme (détermine l'équilibre entre l'exploitation et l'exploration) les valeurs possible pour ce paramètre et entre $[0, 1]$

— (12) une zone permet de visualiser les informations de la meilleure solution trouver

— (13) Le button Start pour lancer l'algorithme

— (14) Le button Stop pour stopper l'exécution

— (15) Le button Exit pour quitter proprement l'application

— (16)et (17) deux buttons spécialement pour le cas statique permet de dessiner et visualiser les différentes solutions pour le joue de donne choisi

— (18) Le button Save permet de sauvegarder le rapport d'exécution de programme et le résultat obtenu dans un fichier text.

— (19) Zone permet de dessiner et visualiser le graphe obtenu .

— (20) Zone de texte permet d'afficher les différents messages encours d'exécution

— (21) Un button permet a l'utilisateur de tester son propre jeu de données.

— (22) Un button permet a l'utilisateur d'afficher le jeu de donnée choisi pour le test.

Pour plus de détail sur les paramètres (7-8-9-11) voir le chapiter 1 la parité de l'optimisation par colonie des fourmis .

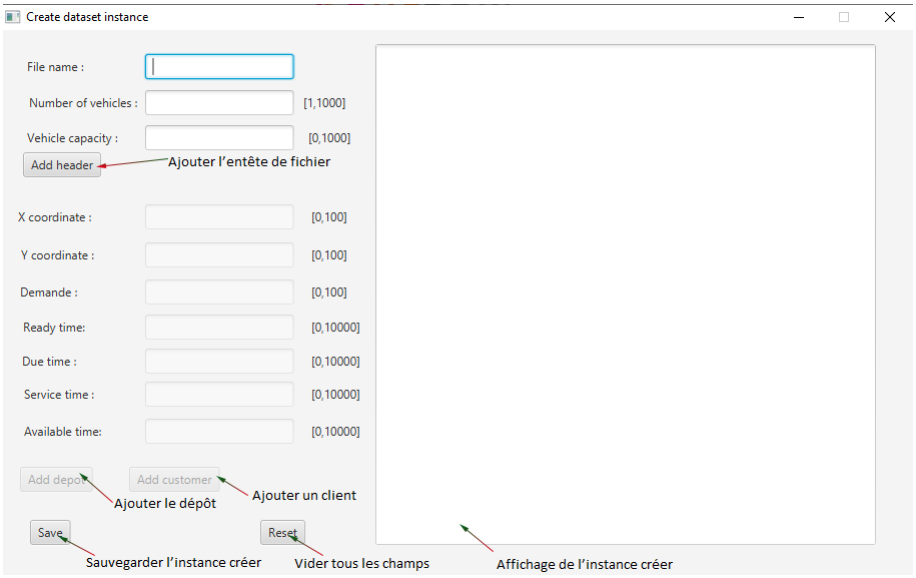

FIGURE 4.3 – Fenêtre permettent la création d'un jeu de test.

La Fenêtre de la figure 4.3 permet à l'utilisateur de créer son propre joue de données et sauvegarder sous forme d'un fichier texte.

4.6 Exemple de jeu de données

c101

VEHICLE

NUMBER	CAPACITY
25	200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME	AVAIL. TIME
0	40	50	0	0	1236	0	0
1	45	68	10	912	967	90	0
2	45	70	30	825	870	90	0
3	42	66	10	65	146	90	0
4	42	68	10	727	782	90	0
5	42	65	10	15	67	90	0
6	40	69	20	621	702	90	0
7	40	66	20	170	225	90	0
8	38	68	20	255	324	90	0
9	38	70	10	534	605	90	0
10	35	66	10	357	410	90	0
11	35	69	10	448	505	90	0
12	25	85	20	652	721	90	0
13	22	75	30	30	92	90	0
14	22	85	10	567	620	90	0
15	20	80	40	384	429	90	0
16	20	85	40	475	528	90	0
17	18	75	20	99	148	90	0
18	15	75	20	179	254	90	0
19	15	80	10	278	345	90	0
20	30	50	10	10	73	90	0
21	30	52	20	914	965	90	0
22	28	52	20	812	883	90	0
23	28	55	10	732	777	90	0
24	25	50	10	65	144	90	0
25	25	52	40	169	224	90	0

FIGURE 4.4 – Exemple d'une instance de jeu de données.

La figure 4.4 représent un exemple de jeu de donnéés telque :

— Le nom de jeu de donnée est C101.

— Le nombre des véhicules est 25 .

— La capacité des véhicules est 200.

— (XCOORD, YCOORD) Représentes les coordonnées des clients dans le repère cartésien.

— (DEMAND) Représente les demandes des clients.

— (READY TIME) Représente le début de la fenêtre de temps de chaque client.

— (DUE DATE) Représente la fin de la fenêtre de temps de chaque client.

— (SERVICE TIME) Représente le temps de service de chaque client.

— (AVAIL. TIME) Représente le temps dans laquelle le client devient disponibilité (dans le cas statique tous les clients sont disponibles au temps 0 et dans le cas dynamique il existe un certain nombre des clients ont un temps de disponibilité supérieur a zéro).

— (CUST NO.) Représente le numéro de client le dépôt et toujours possède le chiffre zéro.

— Dans ce jeu de données les distance entre les positions des clients sont calculées par la méthode euclidienne, ainsi le temps de déplacement entre les clients est égal à la distance entre les deux clients.

4.7 Expérimentation pour le problème de VRPTW

La figure 4.5 Illustrer un exemple de l'application de l'algorithme sur l'instance c104 de jeu de données

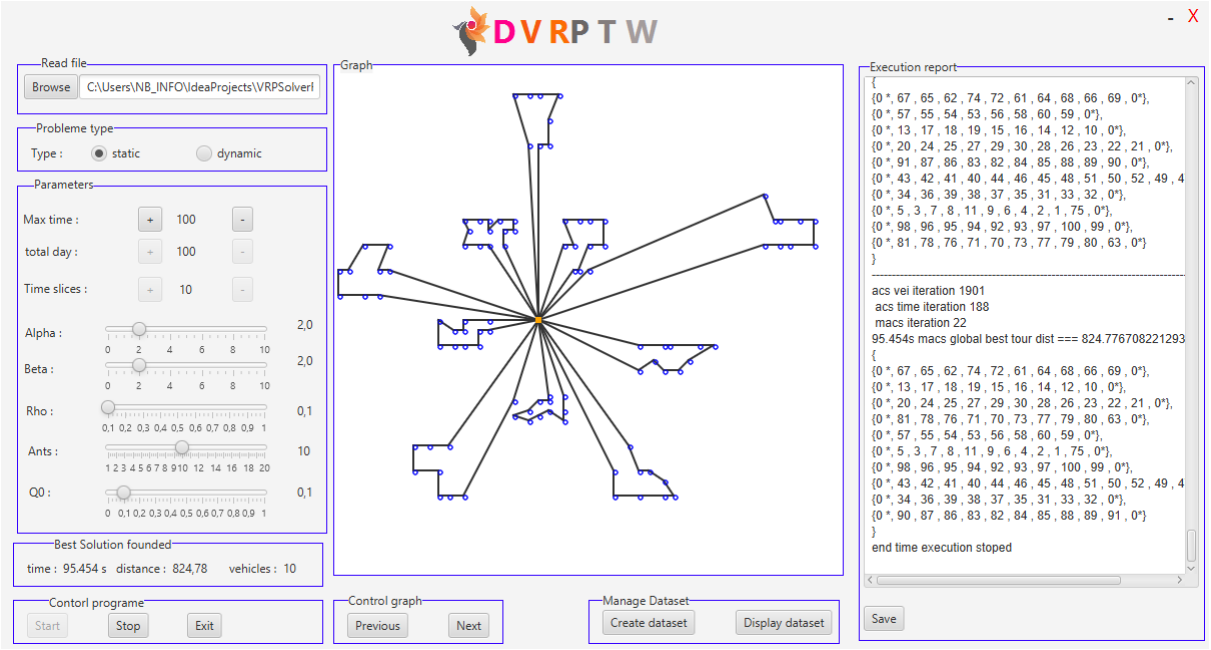

FIGURE 4.5 – Déroulement de l'algorithme sur l'instance c104.

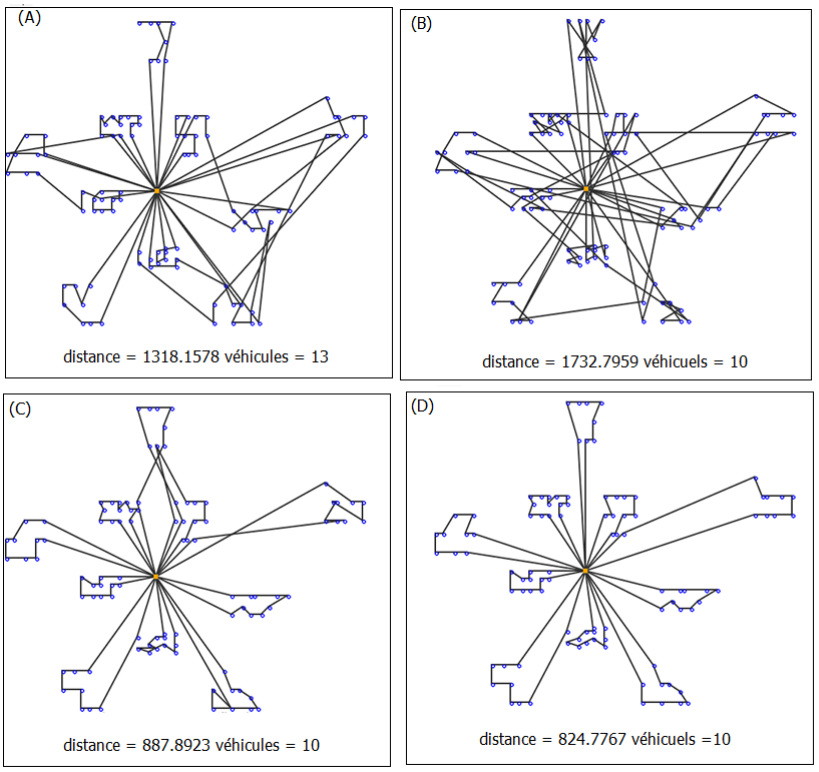

FIGURE 4.6 – .

La figure 4.6 montre certaines améliorations des solutions trouvée pour l'instance c104, notamment entre les étapes (A) et (B), où l'on constate une amélioration du nombre de véhicules, et dans les étapes (B),(C),(D) où l'on constate une amélioration de la distance. Les chemins des solutions sont :

— (A) 0,20,21,22,23,26,28,27,24,0
0,67,65,63,62,66,69,68,64,0
0,5,3,4,6,7,75,0
0, 25,29,30,34,36,39,37,35,31,32,0
0,43,42,41,40,44,45,46,48,50,51,52,49,0
0,61,72,74,90,89,88,85,86,83,82,0
0,8,9,2,1,98,96,95,0
0,10,11,100,99,71,70,73,0 0,91,87,76,77,79,80,0
0,13,15,17,18,19,16,14,12,0
0,33,38,47,0
0,55,53,58,60,56,54,57,59,0
0,92,84,94,93,97,81,78,0

— (B) 0,67,65,25,27,30,38,34,23,26,88,89,0
0,13,18,17,19,15,12,16,99,90,0
0,21,22,28,8,11,9,10,5,7,3,75,1,0
0,60,55,59,53,54,56,57,4,6,0
0,91,87,86,44,46,45,43,50,49,42,40,41,0
0,20,29,24,2,92,95,96,94,97,93,0
0,66,64,61,62,74,84,77,79,73,70,80,0
0,52,47,51,48,68,58,100,98,69,71,0
0,83,82,76,78,81,85,36,39,37,35,31,0
0,33,32,63,72,14,0

— (C) 0,67,65,62,74,72,61,64,57,69,0
0,40,55,54,53,56,58,60,59,68,66,0
0,63,76,78,81,77,79,71,70,73,80,0
0,13,17,18,19,15,16,14,12,10,0
0,20,24,25,27,29,30,28,26,23,22,21,0
0,98,96,95,94,92,93,100,99,97,0
0,5,3,7,8,6,1,19,4,2,1,75,0
0,90,87,86,83,82,84,85,88,89,91,0
0,31,35,37,38,39,36,34,33,32,0
0,43,41,42,44,46,45,48,51,50,52,49,47,0

— (D) 0,67,65,62,74,72,61,64,68,66,69,0
0,13,17,18,19,15,16,14,12,10,0
0,20,24,25,27,29,30,28,26,23,22,21,0
0,81,78,76,71,70,73,77,79,80,63,0
0,57,55,54,53,56,58,60,59,0
0,5,3,7,8,11,9,6,4,2,1,75,0
0,98,96,95,94,92,93,97,100,99,0
0,43,42,41,40,44,46,45,48,51,50,52,49,47,0
0,34,36,39,38,37,35,31,33,32,0

0,90,87,86,83,82,84,85,88,89,91,0

La figure 4.7 Illustrer un autre exemple de l'application de l'algorithme sur l'instance r101 de jeu de données

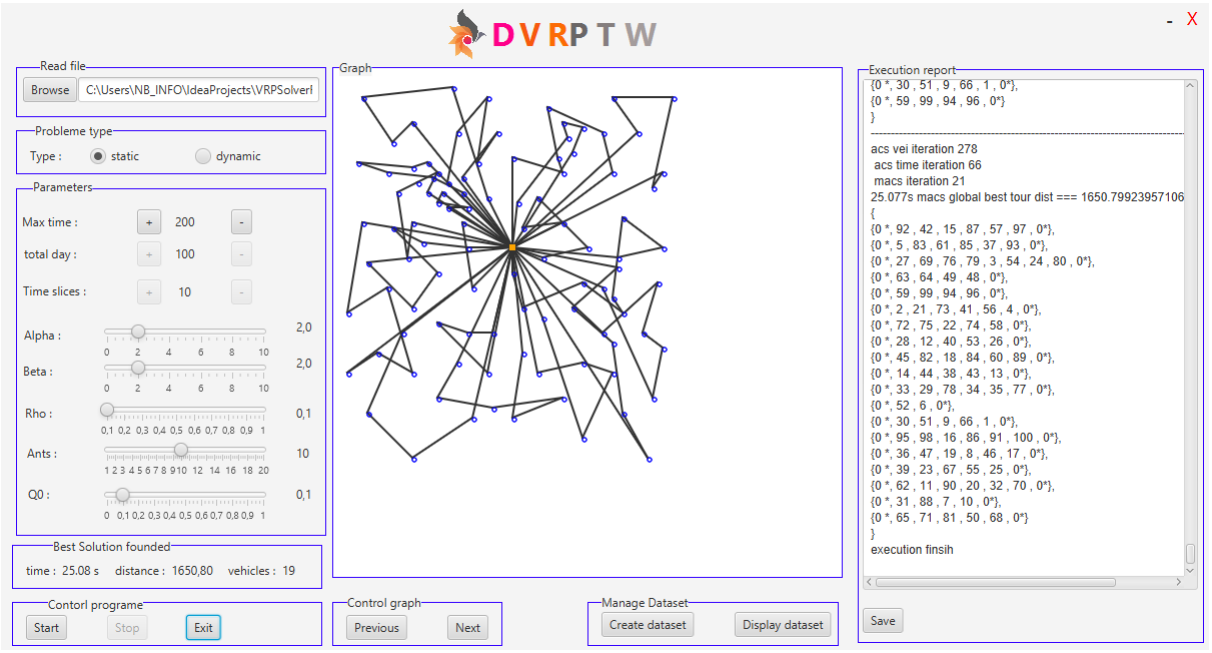

FIGURE 4.7 – Déroulement de l'algorithme sur l'instance r101.

4.7.1 Paramètres d'exécution

Par l'exécution de l'algorithme plusieurs fois, nous constatons que les paramètres du tableau 4.2 qui donnent de bons résultats dans la plupart des cas sont :

	Configuration 1	**Configuration 2**
nombre de fourmis	10	10
alpha	1	2
beta	1	2
q0	0.9	0.1
rho	0.1	0.1

TABLE 4.2 – Configuration des paramètres d'exécution

4.7.2 Résultat obtenu pour les différentes instances

	Distance	**nombre de véhicules**
c101	828.9368	10
c102	828.9368	10
c103	828.0648	10
c104	824.77670	10
c105	828.9368	10
c106	828.9368	10
c107	828.9368	10
c108	828.9368	10
c109	828.9368	10

TABLE 4.3 – Résutat obtenu par notre implimentation dans les instance de type C1

	Distance	**nombre de véhicules**
c201	591.5565	3
c202	591.5565	3
c203	591.1734	3
c204	590.5987	3
c205	588.8759	3
c206	588.4928	3
c207	588.2863	3
c208	588.3238	3

TABLE 4.4 – Résutat obtenu par notre implimentation dans les instance de type C2

	Distance	**nombre de véhicules**
r101	1650.7992	19
r102	1489.2695	17
r103	1297.2462	13
r104	981.2323	10
r105	1400.2825	14
r106	1258.9340	12
r107	1077.2983	11
r108	971.0731	10
r109	1154.5513	12
r110	1089.1317	11
r111	1066.9329	11
r112	962.5226	10

TABLE 4.5 – Résutat obtenu par notre implémentation dans les instances de type R1

	Distance	**nombre de véhicules**
r201	1253.2339	4
r202	1191.7029	3
r203	950.7313	3
r204	758.9752	3
r205	994.4276	3
r206	932.3697	3
r207	842.0923	3
r208	733.5153	2
r209	930.9056	3
r210	944.8380	3
r211	794.0240	3

TABLE 4.6 – Résutat obtenu par notre implémentation dans les instances de R2

	Distance	**nombre de véhicules**
rc101	1641.2041	15
rc102	1489.2695	13
rc103	1277.9869	12
rc104	1160.5385	11
rc105	1583.1274	14
rc106	1289.2293	12
rc107	1244.1765	11
rc108	1133.2543	11

TABLE 4.7 – Résutat obtenu par notre implémentation dans les instances de type RC1

	Distance	**nombre de véhicules**
rc201	1413.5179	4
rc202	1161.7897	4
rc203	1062.9801	3
rc204	819.0953	3
rc205	1300.2548	4
rc206	1183.0004	3
rc207	1099.6486	3
rc208	879.6777	3

TABLE 4.8 – Résutat obtenu par notre implémentation dans les instances de type RC1

4.7.3 Evaluation

Nous présentons ici une comparaison entre les résultats obtenus par l'implémentation originale [64] et notre implémentation de l'algorithme. Nous constatons que les résultats sont très proches et nous trouvons les mêmes résultats pour les problèmes de type c1 et c2 ce qui montre que notre implémentation de l'algorithme est valide les differents resésultat sont présenter dans le tableau 4.9.

		Notre Implémentation	**Implémentation original**
C1	Distance	828.38	828.38
	Véhicules	10	10
C2	Distance	589.86	589.86
	Véhicules	3	3
R1	Distance	**1199.94**	1217.73
	Véhicules	12.5	**12.00**
R2	Distance	**938.80**	967.75
	Véhicules	3	**2.73**
RC1	Distance	**1352.35**	1382.42
	Véhicules	12.375	**11.63**
RC2	Distance	**1115.0**	1129.19
	Véhicules	3.375	**3.25**

TABLE 4.9 – Présentation des moyennes de nos résultats et des résultats obtenus par l'implémentation originale

4.8 Expérimentation pour le problème de VRPTW dynamique

4.8.1 Exemple d'une instance de jeu de données dans le cas dynamique

c101

VEHICLE

NUMBER	CAPACITY
25	200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME	AVAIL. TIME
0	40	50	0	0	1236	0	0
1	45	68	10	912	967	90	0
2	45	70	30	825	870	90	334
3	42	66	10	65	146	90	0
4	42	68	10	727	782	90	537
5	42	65	10	15	67	90	0
6	40	69	20	621	702	90	0
7	40	66	20	170	225	90	102
8	38	68	20	255	324	90	0
9	38	70	10	534	605	90	0
10	35	66	10	357	410	90	0
11	35	69	10	448	505	90	0
12	25	85	20	652	721	90	275
13	22	75	30	30	92	90	0
14	22	85	10	567	620	90	0
15	20	80	40	384	429	90	0
16	20	85	40	475	528	90	0
17	18	75	20	99	148	90	43
18	15	75	20	179	254	90	0
19	15	80	10	278	345	90	60
20	30	50	10	10	73	90	0
21	30	52	20	914	965	90	0
22	28	52	20	812	883	90	565
23	28	55	10	732	777	90	214
24	25	50	10	65	144	90	27
25	25	52	40	169	224	90	89

FIGURE 4.8 – Exemple d'une instance de jeu de données dans le cas dynamique.

Dans la figure 4.8 en remarque que n'est pas tous les client sont disponible au début de jour de travail par exemple les clients (2,4,7) ne sont pas disponible au début de jour de travail.

4.8.2 Exemples d'exécution

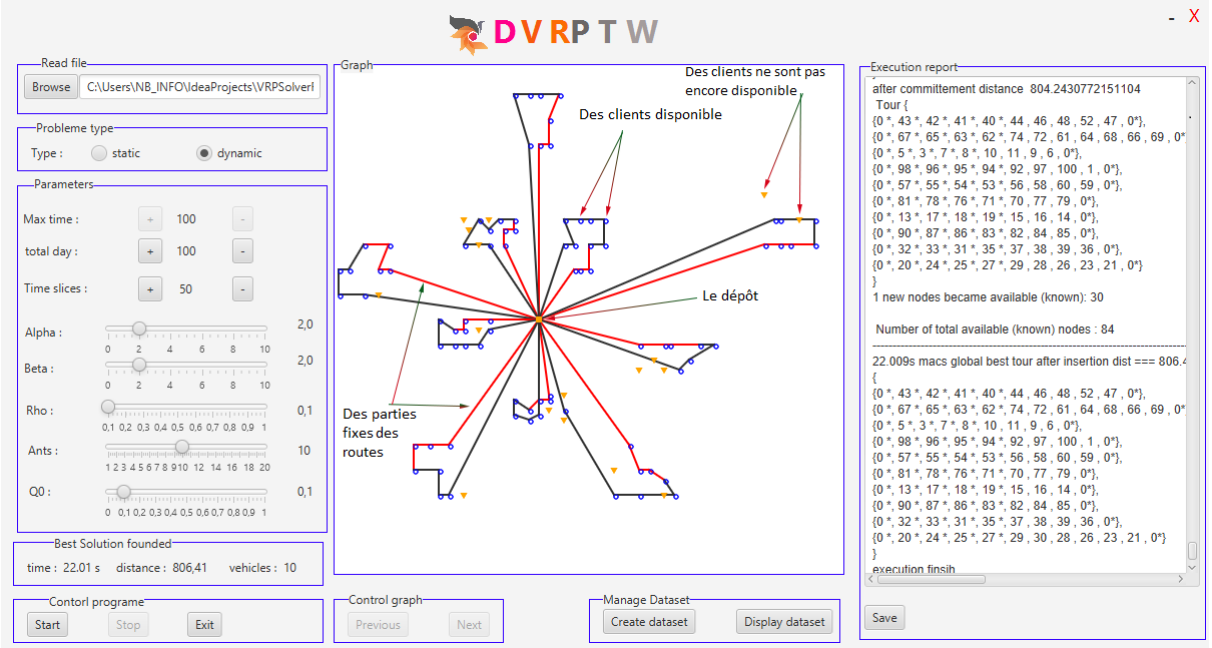

FIGURE 4.9 – Exemple de simulation d'un problème dynamique.

la figure 4.9 monter un exemple de la simulation d'un problème de routage dynamiques des véhicules avec fenêtre de temps.

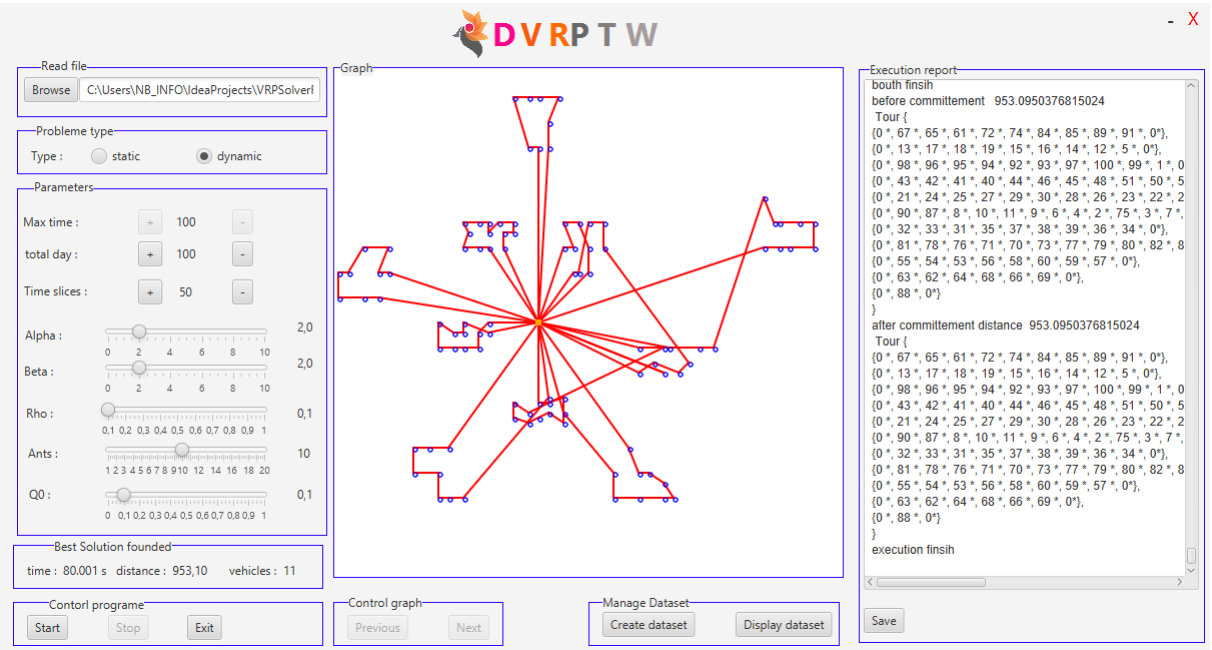

FIGURE 4.10 – Resulat pour l'instance c103 avec un degré de dynamicité 10%.

La figure4.10 monter le résultat final de la résolution de l'instance c103 avec un degré de dynamicité égale a 10% .

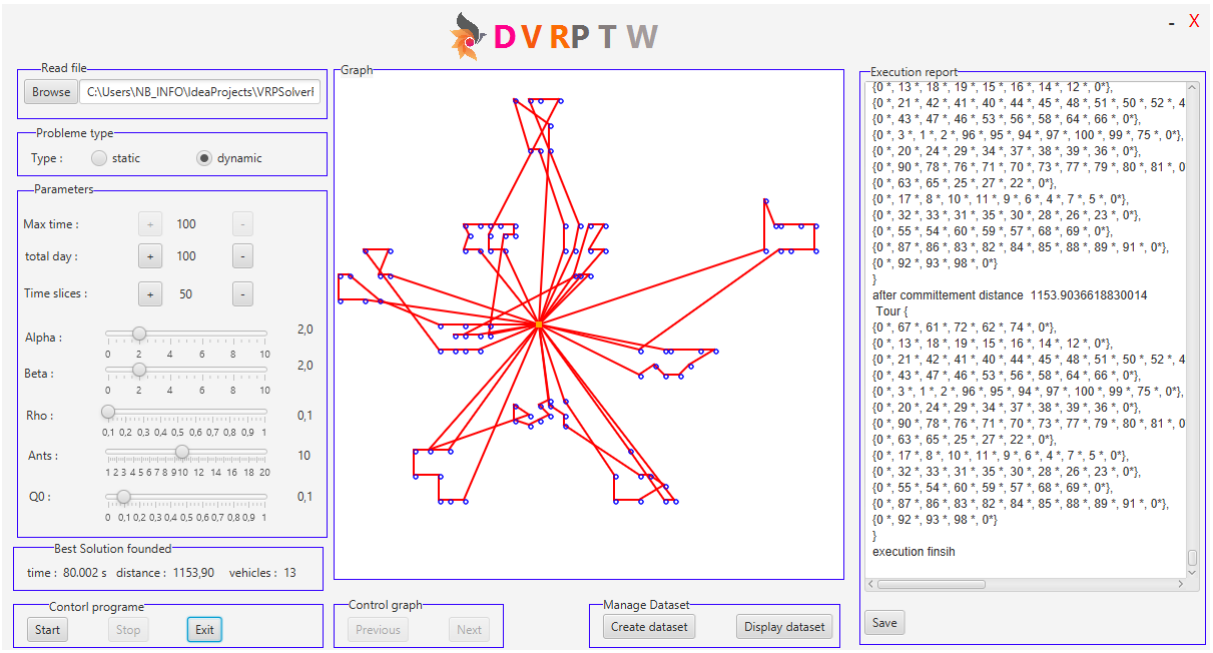

FIGURE 4.11 – Resulat pour l'instance c103 avec un degré de dynamicité 50%.

La figure 4.11 monter le résultat final de la résolution de l'instance c103 avec un degré de dynamicité égale a 50% .

4.8.3 Résultat

4.8.3.1 Paramètres d'exécution

Nous exécutant l'algorithme,avec les paramètres du tableau.

	Configuration
nombre de fourmis	10
alpha	2
beta	2
q0	0.1
rho	0.1
temps jour de travail (T_{wd})	100
nombre des tranches de temps (n_{ts})	50

TABLE 4.10 – Configuration des paramètres d'éxecution

	Distance	**nombre de véhicules**
c106-statique	828.9368	10
c106 0 %	828.9368	10
c106 10 %	828.9368	10
c106 20 %	828.9368	10
c106 30 %	918.0812	11
c106 40 %	1074.5814	11
c106 50 %	1125.2109	11

TABLE 4.11 – Résultats pour l'instance c106 avec diffèrent degré de dynamisme

Le tableau 4.11 présente les las moyenne des Résultat de la résolution des 56 problèmes en diffèrent degré de dynamicité.

	Distance	**nombre de véhicules**
DVRPTW-statique	1014.80	7.51
DVRPTW 0 %	1060.55	9.0
DVRPTW 10 %	1100.88	9.04
DVRPTW 20 %	1141.42	9.25
DVRPTW 30 %	1156.20	9.49
DVRPTW 40 %	1197.37	9.84
DVRPTW 50 %	1230.56	9.93

TABLE 4.12 – Moyenne des résultats des 56 problèmes avec diffèrent degré de dynamisime

4.8.3.2 Discussion des résultats

Le tableau 4.12 montre les performances de MACS-DVRPTW sur des problèmes avec différents degrés de dynamicité. En trouve que dans le cas génerale les résultat obtenu sont augmenter en terme de distance et du nombre de vehicule avec le niveau de dynamicité et aussi les résultat en cas statique sont meilleur que les résultat trouve dans les diffèrent niveau de dynamicité dans le cas dynamique. Ceci peut être expliqué par le fait qu'une partie de la tournée est fixée pendant l'exécution de l'algorithme et ne peut donc plus être modifiée. De plus, comme les colonies sont redémarrées plus souvent (éventuellement à la fin de chaque pas de temps et chaque fois qu'une solution avec moins de véhicules est trouvée), elles ne peuvent pas passer un long temps consécutif à résoudre le problème. Lorsque la colonie est redémarrée, ses phéromones sont réinitialisées et le tableau IN de ACS-VEI est réinitialisé. Cela peut aussi influencer les résultats de façon négative.

4.9 Conclusion

Dans ce chapitre nous avons vu l'implimentation de l'algorithme de MACS-VRPTW et les différents résultats obtenus dans différents états statiques et dynamiques et nous avons vu l'efficacité de la méthode MACS-VRPTW dans la résolution du problème de routage

des véhicules avec fenêtre de temps car elle permet de trouver des bonnes solutions en termes de distance et de nombre de véhicules en peu de temps, et nous avons aussi vu que La méthode MACS-VRPTW peut résoudre le problème de routage dynamique du véhicule avec fenêtre de temps DVRPTW et avec des bons résultats, et que la résolution DVRPTW est une tâche difficile et elle devient encore plus difficile lorsque le degré de dynamicité augmente.

CONCLUSION GÉNÉRALE

Les problèmes de tournées des véhicules font partie intégrante du quotidien des décideurs et planificateurs. Malgré leur apparente simplicité, les problèmes de transport véhiculaire sont des problèmes NP difficiles, et leur étude est utile pour comprendre et modéliser de plusieurs problèmes de véhicules. L'objectif de ce travail était d'implémenter et améliorer une métaheuristique base sur l'optimisation par colonie de fourmis (MACS-VRPTW) pour résoudre un problème d'optimisation multi-objectif en environnement dynamique. En raison de son importance pratique, nous avons traité le problème de routage dynamique des tournées de véhicules avec fenêtres de temps (DVRPTW).

Avant d'aborder la résolution de ce problème, nous avons présenté les différents problèmes de d'élaboration de tournées de véhicules. Nous avons introduit d'abord l'optimisation combinatoire et nous avons expliqué les bases de l'optimisation combinatoire et des différentes techniques et méthodes de résolution de ces problèmes. Puis nous avons détaillé le VRP (Vehicle Routing Problem) qui constitue le problème originel et aussi nous avons présentées ces différentes variantes. Et dont fait partie le VRPTW statique et dynamique, Nous avons ainsi présenté en détailler de ces deux problèmes ainsi que leurs formulations mathématiques respectives.

Pour évaluer notre approche statique et dynamique, nous avons utilisé des benchmark Solomon déjà conçus pour le problème de routage des véhicules avec fenêtre de temps et qui a été adapté pour les problèmes dynamiques. Notre outil peut lire les fichiers de jeu de données de ces benchmarks selon un seul format : un format texte.

Les résultats obtenus dans le cas statique et dynamique du temps de parcours total sont très encourageants. Qu'il s'agisse du nombre de voitures ou de la distance parcourue (temps total de parcours) ses résultats sont généralement acceptables.

La méthode de (MACS-VRPTW) baser sur l'optimisation par les colonies de fourmis a prouvé son efficacité en :

— La minimisation de nombre des véhicules.

— La minimisation du (distance) temps de parcours total de transport.

Enfin, dans une perspective d'amélioration des performances de notre approche, nous proposons :

— D'améliorer l'insertion des clients arrivés à la fin de chaque tranche de temps.

— Combiner avec d'autre méta-heuristique pour avoir meilleur résultat.

— L'utilisation de matériel performants.

— Ajuster le nombre de tranches de temps selon la période de travail.

BIBLIOGRAPHIE

[1] Lyes Belhoul. Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée. *Université Paris Dauphine - Paris IX*, 2014.

[2] Catherine Mancel. Modelisation et resolution de problemes d'optimisation combinatoire issus d'applications spatiales. *INSA de Toulouse*, 2004.

[3] Arora S, , and Barak B. Computational complexity : a modern approach. *Cambridge University Press*, 2009.

[4] El-Ghazali Talbi. Metaheuristics : From design to implementation. *John Wiley*, 2009.

[5] LEMOUARI ALI. Introduction aux métaheuristiques. *Université de Jijel*, 2014.

[6] Sidi Mohamed Douiri, Souad Elbernoussi, and Halima Lakhbab. Cours des méthodes de résolution exactes heuristiques et métaheuristiques. *Université Mohammed V Faculté des Sciences de Rabat*, 2009.

[7] Pierre Fouilhoux. Optimisation combinatoire : Programmation linéaire et algorithmes. *Université Pierre et Marie Curie*, 2015.

[8] P. Esquirol and P. Lopez. L'ordonnancement. *Economica Paris*, 1999.

[9] P. Lopez. Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation. *Institut National Polytechnique de Toulouse*, 2003.

[10] Gerard Reinelt. The traveling salesman : Computational solutions for tsp applications. *Springer-Verlag*, 2003.

[11] D. B. Fogel. An evolutionary approach to the traveling salesman problem. *Biol. Cybern*, 1988.

[12] ZHAO Xin. Une méthode génétique pour la résolution du problème dynamique de routage de véhicules avec temps de parcours variables. *these Doctorat Université d'Artois*, 2008.

[13] P. Toth and D. Vigo. Vehicle routing : problems and methods and applications. *Society for Industrial and Applied Mathematics*, 2014.

[14] P. Toth and D. Vigo. The vehicle routing problem. *European Journal of Operational Research*, 2002.

[15] C. Rego and C. Roucairol. Problème de tournées de véhicules : Etude et résolution approchée. *technical Report, INRIA,*, 1994.

[16] Sara MAQROT. Méthodes d'optimisation combinatoire en programmation mathématique. application à la conception des systèmes de verger-maraîcher. *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*, 2019.

[17] SARIKLIS, Dimitrios, and Susan Powell. A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 2000.

[18] C. Dhaenens, M.L. Espinouse, and Bernard Penz. Problèmes combinatoires classiques. recherche opérationnelle et réseaux : méthodes d'analyse spatiale. *Hermès Science Publications*, 2002.

[19] T.G. Crainic and F. Semet. Recherche opérationnelle et transport de marchandises. optimisation combinatoire. *Hermès Science :Lavoisier*, 2006.

[20] Tayeb OULAD KOUIDER. Optimisation de la planification des tournnees de véhicules électriques. *these Ecole doctorale IAEM Lorraine*, 2020.

[21] G.L. Nemhauser and L.A. Wolsey. Integer and combinatorial optimization. *Wiley New-York*, 1999.

[22] LAITITIA J. Métaheuristiques pour l'extraction connaissances application à la génomique, thèse doctorat en informatique. *Université des sciences technologique de LILLEU.F.R.*, 2003.

[23] Ahmed Nasreddine Benaichouche. Conception de métaheuristiques d'optimisation pour la segmentation d'images : application aux images irm du cerveau et aux images de tomographie par émission de positons. analyse numérique [cs.na]. *Université Paris-Est*, 2014.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, page 671–680, 1983.

[25] V. Cerny. Thermodynamical approach to the travelling salesman problem : An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, page 41–51, 1985.

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, page 1087–1092, 1953.

[27] Abbas El Dor. Thermodynamical approach to the travelling salesman problem : An ef-ficient simulation algorithm. *Autre [cs.OH] Université Paris-Est*, page 41–51, 1985.

[28] Kamel ZIDI. Système interactif d'aide au déplacement multimodal (siadm). *these Ecole Centrale de Lille*, 2006.

[29] M. Dorigo. Optimization, learning and natural algorithms. ph.d. *Thesis Politecnico di Milano Italy*, 1992.

[30] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system : optimizationby a colony of cooperating agents. *IEEE*, 1996.

[31] Hanaâ HACHIMI. Hybridations d'algorithmes mÉtaheuristiques en optimisation globale et leurs applications. *these École Mohammadia d'ingénieurs (Rabat, Maroc)*, 2013.

[32] G.A. Croes. A method for solving traveling-salesman problems. *Operations Research*, page 1087–1092, 1958.

[33] M. Dorigo and L.M. Gambardella. Ant colony system : a cooperative learning approach to the traveling salesman problem. evolutionary computation. *IEEE Transactions on*, 1997.

[34] J.H. Holland. Adaptation in natural and artifcial systems. *Ann Arbor MI : University of Michigan Press*, 1992.

[35] D. Goldberg. Genetic algorithms in search optimization and machine learning. *Addison Wesley*, 1989.

[36] C. Rego and C. Roucairol. Le probleme de tournees de vehicules : etude et resolution approchee. *Rapport de recherche*, 1994.

[37] Salhi Ahmed and Tarek Bouktir. Contribution a l'optimisation de l'ecoulement de puissance en utilisant la logique floue associee aux reseaux de neurones (neuro-flou). *Thése De Doctorat, Université De Biskra*, 2015.

[38] Wang Geng-sheng and Yu Yun-xin. An improved ant colony algorithm for vrp problem. *IEEE*, 2010.

[39] LeBouthillier. Modélisation uml pour une architecture coopérative appliquée au problème de tournées de véhicules avec fenêtres de temps. *Université de Montréal*, 2000.

[40] Gianpaolo Ghiani, Francesca Guerriero, Gilbert Laporte, and Roberto Musmanno. Real-time vehicle routing : Solution concepts algorithms and parallel computing strategies. *European Journal of Operational Research*, 2003.

[41] Mais Haj Rachid, Wahiba Ramdane Cherif-Khettaf, Christelle Bloch, and Pascal Chatonnay. Classification de problèmes de tournées de véhicules. *researchgate*, 2008.

[42] Allan Larsen. The dynamic vehicle routing problem. *Thèse, Informatics and Mathematical Modelling Technical University of Denmark IMM-PHD-2000-73 Lyngby*, 2000.

[43] R. Bent and P. Van Hentenryck. Dynamic vehicle routing with stochastic requests. *International Joint Conference on Artificial Intelligence*, pages 1362–1363, 2003.

[44] K.Q. Zhu. Heuristic methods for vehicle routing problem with time windows. *PhD thesis National University of Singapore Departement of Electrical Engineering*, 2001.

[45] J.F. Cordeau, G. Desaulniers, J. Desrosiers, M. Solomon, and F. Soumis. The vrp with time windows. *SIAM Philadelphia*, 2000.

[46] C. Jacobs-Blecha and M. Goetschalckx. the vehicle routing problem with backhauls : An optimization based approach. *Proceedings of the 2nd Industrial Engineering Research Conference*, 1993.

[47] P. Kilby, P. Prosser, and P. Shaw. Dynamic vrps : A study of scenarios. *Technical Report APES-06-1998 University of Strathclyde*, 1998.

[48] Michel Gendreau, Francois Guertin, Jean-Yves Potvin, and Eric Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, pages 381–390, 1999.

[49] J.-F. Cordeau, G. Laporte, and A. Mercier. A unfied tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, pages 928–936, 2001.

[50] M. Fischetti, A. Lodi, and P. Toth. Branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Dipartimento di Elittronica e Informatica, Università di Padova, Italy*, 1999.

[51] C. Rego and C. Roucairol. Problème de tournées de véhicules : Etude et résolution approchée. *Technical Report, INRIA*, 1994.

[52] S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295 :279–294, 2003.

[53] R. Mechti. Tournées de véhicules à la demande : problèmes et méthodes. *Technical Report, Laboratoire PRiSM, Université de Versailles-St. Quentin, France,*, 1(1) :1–19, 1995.

[54] R.W. Eglese, Z. Fu, , and L. Li. A tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society*, pages 267–274, 2005.

[55] E. Taillard. A heuristic column generation method for the heterogeneous fleet vrp. *RAIRO-Operations Research*, pages 1–14, 1999.

[56] C. Archetti, A. Hertz, and M. G. Speranza. A tabu search algorithm for the split delivery vehicle routing problem. *Transportation science*, pages 64–73, 2006.

[57] H.C. Lau, M. Sim, and K.M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, pages 559–569, 2003.

[58] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, pages 254–265, 1987.

[59] C. Duhamel. Un cadre formel pour les méthodes par amélioration itérative - application à deux problèmes d'optimisation dans les réseaux -. *PhD thesis Université Blaise Pascal - Clermont-Ferrand*, pages 254–265, 2001.

[60] HOUSROUM Haiyan. Une approche génétique pour la résolution du problème vrptw-dynamique. *Thèse Doctorat en informatique, Université d'Artois*, 2005.

[61] Zhi-Long and ChenHang Xu. Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science*, 2006.

[62] V. Pillac, M. Gendreau, C. Gueret, and A. Medaglia. A review of ´dynamic vehicle routing problems. *European Journal of Operational Research*, pages 1–11, 2013.

[63] Eric Taillard, Luca M Gambardella, Michel Gendreau, and Jean-Yves Potvin. Adaptive memory programming : A unified view of metaheuristics. *European Journal of Operational Research*, 2001.

[64] .M. Gambardella, E. Taillard, and G. Agazzi. Macs-vrptw : A multiple ant colony system for vehicle routing problems with time windows. *In New Ideas in Optimization McGraw-Hill London UK*, 1999.

[65] M.M. Flood. The traveling-salesman problem. *Operations Research*, 1956.

[66] P. Badeau M. Gendreau E. Taillard, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 1997.

[67] Michael David Schneider. New challenges in time-definite vehicle routing. *From the Department of Economics of the Technical University of Kaiserslautern for the award of the academic degree*, 2012.

[68] R. Montemanni, L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 2005.

[69] Barry van Veen. Solving the dynamic vehicle routing problem with time windows using ant colony optimization. *Universiteit Leiden Opleiding Informatica*, 2013.

[70] G. H. Haines and J. Wolff. Alternative approaches to demand responsive scheduling algorithms. *Transportation Research Part A*, 1982.

[71] W.B. Powell, P. Jaillet, and A. Odoni. Operations research and management science vol 8 network routing chapitre stochastic and dynamic networks and routing. *North-Holland : Amsterdam*, 1995.

[72] Harilaos N. Psaraftis. Dynamic vehicle routing : status and prospects. *Annals of Operations Research*, 1995.

[73] L. Brotcorne G. Laporte and F. Semet. Ambulance location and relocation models. *European Journal of OR*, 2003.

[74] K. Lund, O. Madsen, and J. M. Rygaard. Vehicle routing problems with varying degrees of dynamism. *Technical Report IMM The Department of Mathematical Modelling Technical University of Denmark*, 1996.