UNIVERSITÉ KASDI MERBAH OUARGLA
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique et des Technologies de l'Information

Mémoire de Projet de fin d'études

pour l'obtention du diplôme de Master en Informatique
Option informatique industrielle

Présenté par :

HARKAT Mensour
AMIAR Ali

Adaptation d'un nouvel algorithme métaheuristique appelé TLBO pour la résolution du problème de tournées de véhicules

Soutenu publiquement le : 30/ 09/ 2020

Devant le jury composé de :

Dr.	MAHDJOUB Mohamed Elbachir	MAA	Président	UKM Ouargla
Dr.	BEN ALI, Abdelkamel	MCA	Encadreur	UNV El Oued
Dr.	ZGA Adel	MCA	Co-Encadreur	UKM Ouargla
Dr.	BELKEBIR Djalila	MCB	Examinateur	UKM Ouargla

Année Universitaire 2019/2020

Résumé

L'objectif de ce travail de mémoire de master est de proposer puis implémenter un nouvel **algorithme de TLBO** (pour l'anglais 'Teaching-Learning-based optimization algorithm') pour la résolution du **problème de tournées de véhicules** avec contraintes de capacité (CVRP pour 'Capacitated Vehicle Routing Problem'). Comme l'algorithme de **TLBO** a été proposé originalement pour l'optimisation à variables continues, pour CVRP, qui est un problème combinatoire de permutation, ce nouvel algorithme doit travailler directement dans l'espace de recherche de permutations. Pour ce faire, un opérateur de recombinaison entre permutations est développé pour réaliser la combinaison entre solutions dans l'espace combinatoire de permutations. Une procédure de recherche locale est appliquée au schéma d'optimisation **TLBO** pour améliorer la performance. Une étude expérimentale et une comparaison avec les résultats de la littérature ont été réalisées.

Mots clés : Optimisation basée sur l'enseignement-apprentissage, Problème de Tournées de Véhicules avec contrainte de Capacité, Recherche Locale, méta-heuristique mémétique.

Abstract

The Aim of this work of thesis is to propose and implement a novel algorithm of TLBO (Teaching Learning Based Optimization algorithm) to resolve the Capacitated Vehicle Routing Problem (CVRP). The TLBO algorithm is originally proposed for continuous optimization problems, for CVRP known as a permutation-based combinatorial optimization problem, this novel algorithm should work directly on the permutation-based search space. To do so, a recombination operator is established to realize the combination between solutions in search space of permutations, In addition, a local search procedure is applied in the TLBO optimization scheme to increase the performance. An experimental study and a comparison with the results of the literature have been performed.

Key words: Teaching-Learning-based Optimization, Capacitated Vehicle Routing Problem, Local Search, Memetic Metaheuristic.

ملخـــص

الهدف من عمل مذكرة الماستر هذه هو اقتراح ثم تنفيذ خوارزمية جديدة TLBO(Teaching Learning Based Optimization algorithm)(خوارزمية التحسين القائمة على التعلم والتعليم) لحل مشكلة توجيه السيارة مع قيود السعة (‘Capacitated Vehicle Routing Problem’) CVRP لعمل ذالك تم اقتراح خوارزمية TLBO جديدة لتحسين المتغير المستمر بالنسبة لـ CVRP وهي مشكلة تبديل اندماجي تعمل هذه الخوارزمية مباشرة في مجال البحث التقليبي.

للقيام بذلك سيتم تطوير عامل إعادة التركيب بين التباديل لأداء الجمع بين الحلول في الفضاء التوافقي للتباديل. ستتم إضافة إجراء بحث محلي إلى مخطط TLBO لتحسين الأداء. من خلال إجراء دراسة تجريبية ومقارنة مع نتائج الدراسات والاطروحات المنجزة .

الكلمات المفتاحية: خوارزمية التحسين القائمة على التعلم والتعليم مشكلة توجيه السيارة مع قيود السعة، البحث المحلي ,الممتيك الفوقية للكشف عن مجريات الامور.

Remerciements

Nos remerciements sincères à notre encadreur le docteur AbdElkamel BEN ALI et notre co-encadreur le docteur Adel ZGA pour leurs bonnes volontés d'accepter de nous encadrer, pour tout le temps qu'ils nous ont octroyé et pour tous les conseils qu'ils nous ont prodigués.

Un grand merci à tous les membres de jury pour l'honneur qu'ils nous ont fait en acceptant de juger notre travail.

Nous remercions chaleureusement nos collègues du département informatique, pour leur soutien moral.

Enfin, merci à tous nos amis et nos familles qui nous ont poussé à dépasser tous les problèmes et aller jusqu'au bout et terminé ce projet.

Dédicaces

Je dédie ce travail :

A mes chers parents

A ma petite et Grande famille

A Tous mes Collègues

AMIAR Ali

Dédicaces

Je dédie ce travail :

A mes chers parents

A ma petite et Grande famille

HARKAT Mensour

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

TLBO	Teaching Learning Based Optimization
VRP	Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem
WD	Water Drop
IWD	Intelligent Water Drop
CS	Cuckoo Search
LS	Local search
classe P	Polynomial time
classe NP	Non-deterministic Polynomial time
KP	Knapsack Problem
TSP	Traveling Salesman Problem
MTSP	Multiple Traveling Salesmen Problem
VRPTW	Vehicle Routing Problem with Time Windows
DVRP	Distance Constrained Vehicle routing problem
PRP	Routing Problem with Profits
PTS	Problème de Tournées Sélectives
PTQ	Problèmes de Tournées avec Quotas
RT	Recherche Tabou
ILS	Iterated Local Search
VNS	variable neighborhoods search
GA	Genetic Algorithme
POP	Population
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization (algorithme de colonies de fourmis)
BNR	L'hybridation relais de bas niveau
LRH	Low-level Relay Hybrid method
BNC	L'hybridation co-évolutionnaire de bas niveau
LTH	Low-level Teamwork Hybrid methode
HNR	L'hybridation relais de haut niveau
HRH	High-level Relay Hybrid
HCH	L'hybridation co-évolutionnaire de haut niveau
GBS	Global Best Solution
CLPSO	Comprehensive Learning Particle Swarm Optimization
Pc	Learning Probability
TE	Teaching Exemplar
LE	Learning Exemplar
BS	Best solution
IIWD	Improved Intelligent Water Drops

Introduction générale

Beaucoup de problèmes se modélisent comme des problèmes d'optimisation. En effet, nous faisons appel aux concept d'optimisation dans notre vie quotidienne sans forcement en être conscient, en cherchant le trajet le plus court en temps ou en distance pour arriver à une destination donnée, en cherchant à augmenter sa productivité au travail, en essayant de bien gérer son salaire pour qu'il tienne jusqu'à la fin du mois, beaucoup de situations font appel au concept d'optimisation : meilleur trajet, meilleure organisation, meilleure gestion,. . . du fait que toute ressource (temps, espace, énergie, argent,. . .) est limitée, le concept d'optimisation s'impose naturellement.

Un problème d'optimisation consiste donc à chercher la meilleure configuration d'un ensemble de variables pour réaliser certains buts. Les problèmes d'optimisation se divisent en deux classes selon la nature des variables utilisées, on parle alors d'optimisation continu si les variables sont réelles et d'optimisation discrète si les variables sont discrètes, parmi les problèmes d'optimisation discrète on peut distinguer les problèmes d'optimisation combinatoire.
Le problème de tournées de véhicules VRP est un problème d'optimisation combinatoire qui consiste à optimisé les itinéraires d'un ensemble de véhicules afin de desservir un ensemble de clients, c'est un problème qui est très étudié depuis les années cinquante du dernier siècle, d'une part parce que ses applications sont nombreuses et d'autre part par sa grande difficulté. Il existe beaucoup de méthodes et approches pour résoudre ce problème.

L'objectif de ce mémoire est de proposer puis implémenter un nouvel algorithme de TLBO (pour l'anglais 'Teaching-Learning-based optimization algorithm') pour la résolution du problème de tournées de véhicules avec contraintes de capacité (CVRP). ce nouvel algorithme doit travailler directement dans l'espace de recherche de permutations. Pour ce faire, un opérateur de recombinaison entre permutations sera développé pour réaliser la combinaison entre solutions dans l'espace combinatoire de permutations. Une technique appelée l'apprentissage compréhensif (en anglais, comprehensive learning) utilisé originalement dans l'algorithme PSO, est ajoutée au

TLBO pour améliorer le niveau de connaissances dans les différentes phases de l'algorithme proposé. Par ailleurs, Une procédure de recherche locale est appliquée au schéma d'optimisation TLBO pour améliorer la performance. Une étude expérimentale et une comparaison avec les résultats de la littérature ont été réalisées.

Organisation du mémoire :

Ce mémoire est organisé en trois chapitres :

Dans le premier chapitre, nous présentons les différentes notions liées la théorie de complexité et les problèmes d'optimisation combinatoire, ainsi qu'une introduction au problème de tournée de véhicules.

Le deuxième chapitre décrit en détail les algorithmes métaheuristiques et ses différents types, on parlera des métaheuristiques à solution unique et à base d'une population de solution, et on se terminera par l'introduction des métaheuristiques hybrides ainsi qu'une description détaillé de l'algorithme TLBO originale.

Dans le troisième chapitre on décrit quelques travaux récents pour la résolution de CVRP, le détail de l'algorithme TLBO proposé, une étude expérimentale et une comparaison avec les résultats de la littérature sont réalisées.

Enfin, ce mémoire s'achève par une conclusion générale, où nous récapitulons nos contributions et proposons des perspectives, sur la base du travail effectué.

CHAPITRE I

L'optimisation combinatoire et Le problème de tournées de véhicules

1 Introduction

Nous présenterons dans ce premier chapitre les notions et les outils nécessaires à la bonne compréhension de notre mémoire de master. Nous introduisons d'abord les concepts de base en théorie de la complexité des algorithmes et des problèmes. Puis, nous donnerons les principales définitions et notions liées à l'optimisation combinatoire. Un troisième volet sera consacré à l'étude du problème de tournées de véhicules.

2 Théorie de la complexité

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté des problèmes en informatique. L'objectif de calculer la complexité d'un algorithme, est d'obtenir un ordre de grandeur du nombre d'opérations élémentaires nécessaires pour que l'algorithme fournisse la solution du problème à l'utilisateur. Ceci permet de comparer la performance des algorithmes indépendamment des caractéristiques de la machine ou du langage utilisé. Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution. Les classes de complexité ont été introduites pour les problèmes décisionnels, c'est-à-dire les problèmes posant une question dont la réponse est *oui* ou *non*. Un problème décisionnel peut appartenir à deux classes :

La classe *P* (Polynomial time) : contient l'ensemble des problèmes pouvant être résolus, de manière exacte, par un algorithme de complexité polynomiale.

La classe *NP* (Non-deterministic Polynomial time): contient l'ensemble des problèmes dont on peut vérifier qu'une proposition donnée est bien une solution du problème avec un algorithme de complexité polynomiale.

La résolution d'un problème *NP* peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas mais l'examen de chaque cas doit se faire en temps polynomial.

Les problèmes les plus difficiles de *NP* définissent la classe des problèmes *NP* -Complet. La *NP*-complétude s'appuie sur la notion de réduction polynomiale. Un problème est *NP* -Complet si n'importe quel autre problème de *NP* peut être transformé en ce problème par une procédure polynomiale. Ces problèmes constituent le noyau dur des problèmes *NP*, si bien que si l'on trouvait un algorithme polynomial permettant de résoudre un seul problème *NP* -Complet, on pourrait en déduire un autre pour tout problème *NP*.

Nous distinguons également dans la classe *NP*, la classe des problèmes *NP* -Difficile. Ce sont les problèmes d'optimisation combinatoire dont le problème de décision associé est *NP* -Complet.

2.1 Complexité des algorithmes

La complexité algorithmique est le prix à payer en termes de ressources pour exécuter un algorithme. Ces ressources peuvent être le nombre d'instructions à exécuter, la quantité de mémoire utilisée, l'utilisation du réseau, etc.

On exprime la complexité d'un algorithme sous forme d'une fonction de taille de ses instances.

La complexité temporelle est la complexité associée au temps d'exécution de l'algorithme, elle est directement liée au nombre d'instructions à exécuter par l'algorithme. On ne s'intéressera ici qu'à ce type de complexité car c'est la ressource temporelle qui est généralement la plus cruciale en pratique. La complexité temporelle est définie en fonction de la taille du problème à résoudre, dans le cas du VRP, on peut définir cette taille par le nombre de clients qui doivent être desservis.

2.2 Complexité des problèmes

La complexité d'un problème correspond à la complexité de l'algorithme le plus efficace permettant de résoudre ce problème. On utilise la notation $O(\,)$ pour représenter les classes de complexité. On utilisera par exemple $O(n^2)$ pour l'ensemble des problèmes ayant une complexité quadratique en la taille *n* des données d'entrée.

Un algorithme efficace est un algorithme dont la complexité est polynomiale.

Le VRP, le TSP et le problème de bin packing, font partie des problèmes dits *NP*-complets. Les problèmes *NP* sont des problèmes pour lesquels on peut vérifier efficacement si une solution est valide et dont les solutions possibles peuvent être générées efficacement de manière non déterministe. Les problèmes *NP*-complets sont les problèmes les plus difficiles de la classe *NP*, en effet, si un algorithme efficace était connu pour un quelconque problème *NP*-complet, alors tous les autres problèmes *NP* pourraient être résolus efficacement.

Les algorithmes connus pour résoudre le VRP ont une complexité temporelle exponentielle dans le pire cas. Ceci signifie qu'il existe des instances pour lesquelles l'algorithme prendra un temps exponentiel par rapport à la taille de l'instance. Pour illustrer l'impact sur les temps d'exécution,

comparons quelques complexités polynomiales avec une complexité exponentielle. *n* est la taille des données d'entrée, l'unité de temps n'est pas précisée car elle n'a pas d'importance.

$O(\cdot)$ \ n	1	10	100	1000
$O(n)$	1	10	100	1000
$O(n^2)$	1	100	10 000	1 000 000
$O(n^3)$	1	1 000	1 000 000	1 000 000 000
$O(2^n)$	2	1 024	≈ 1,26e + 30	≈1,07e + 301

Tableau 1.1 – Comparaison de temps d'exécution pour différentes complexités temporelles.

Comme on peut le voir, pour de grandes valeurs de la taille de l'instance, les temps d'exécution deviennent immenses.

3 Optimisation combinatoire

3.1 Problème d'optimisation combinatoire

Dans cette section, nous donnerons la définition formelle d'un problème d'optimisation combinatoire, en introduisant plusieurs concepts et notions : l'espace de recherche, la fonction objectif (ou fitness), l'optimisation combinatoire, l'optimum global ou la solution optimale, un optimum local ou une solution sous-optimale.

L'espace des solutions est l'ensemble des solutions possibles pour le problème. Pour le TSP, ceci correspond à l'ensemble des cycles différents qui passent par toutes les villes.

La fonction objectif d'un problème d'optimisation est l'équation décrivant le critère à optimiser.

L'optimisation combinatoire est une branche de l'optimisation en mathématiques appliquées et en informatique, également liée à la recherche opérationnelle, l'algorithmique et la théorie de la complexité. On parle également d'**optimisation discrète**.

Une solution optimale (optimum global) appelé $\mathbf{s}_{opt}$ est une solution admissible $\boldsymbol{s_{opt}} \in S$ telle qu'il n'existe pas d'autre solution admissible $s \in S$ de coût inférieur (en minimisation) à celui de$\boldsymbol{s_{opt}}$.

$\boldsymbol{s_{opt}}$ est dite solution optimale de(S, f). L'ensemble $\boldsymbol{s_{opt}}$ est l'ensemble de toutes les solutions optimales.

Une solution sous optimale (optimum local) est une solution $s*$ telle que $\forall s \in V(s*), f(s*) \leq f(s)$. Un problème d'optimisation consiste à trouver le meilleur élément (appelé optimum) parmi un ensemble d'éléments autorisés, en fonction d'un critère de comparaison [1]. Il est défini

par une fonction mathématique (appelée fonction objectif) et un espace de recherche représentant l'ensemble des solutions. On parle d'un problème de maximisation quand la qualité est donnée par une fonction objectif à maximiser et d'un problème de minimisation quand la qualité est donnée par une fonction objectif à minimiser.

Un problème d'optimisation combinatoire est un problème d'optimisation dans lequel l'espace de recherche (noté Ω) est dénombrable [1]. Soit $f : \Omega \rightarrow R$ la fonction objective qui assigne à chaque solution discrète $s \in \Omega$ le nombre réel$f(s)$. Le but est de trouver $s*$ tel que: $s* = argmax_{s \in \Omega} f(s)$. Une telle solution $s*$ s'appelle une solution optimale.

Dans ce contexte, nous définissons les termes d'optimum global et d'optimum local associés à un problème d'optimisation. Pour un problème de minimisation (resp., maximisation), un optimum global est une solution $s* \in \Omega$ telle que : $\forall s \in \Omega, f(s*) \leq f(s)$ (resp. $f(s*) \geq f(s)$). Si l'espace de recherche est muni d'une relation de voisinage V, un optimum local est alors défini comme une solution $s* \in \Omega$ telle que : $\forall s \in V(s), f(s*) \leq f(s)$ (resp.,$f(s*) \geq f(s)$).

3.2 Quelques problèmes classiques d'optimisation combinatoire

3.2.1 Problème du sac-à-dos

Le problème du sac-à-dos, noté également KP (pour l'anglais, *KnapsackProblem*) est l'un des 21 premiers problèmes *NP*-complets présentés dans l'article de Richard Karp [1]. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec un ensemble d'objets ayant chacun un poids et un profit. Les objets mis dans le sac-à-dos doivent maximiser le profit total, sans dépasser le poids maximum.

Le problème du sac-à-dos fait partie des problèmes d'optimisation combinatoire les plus étudiés ces cinquante dernières années, en raison de ces nombreuses applications dans le monde réel. En effet, ce problème intervient souvent comme sous-problème à résoudre dans plusieurs domaines : la logistique comme le chargement d'avions ou de bateaux, l'économie comme la gestion de portefeuille ou dans l'industrie comme la découpe de matériaux.

Ce problème en variables 0-1, dont l'énoncé est assez simple, fait partie des problèmes mathématiques *NP*-complets. Cela explique que le nombre d'ouvrages qui lui sont consacrés est important.

De nos jours le problème du sac-à-dos se résout de manière assez efficace. Les travaux actuels portent sur différentes variantes du problème du sac-à-dos qui sont beaucoup plus difficiles à résoudre. On peut en citer :

- Le problème du sac-à-dos multidimensionnel (MKP) : on notera les travaux de Freville et Plateau [2], Hanafi et al. [3] et Boyer et al.[4].
- Le problème du partage équitable (KSP) : on notera les travaux de Hifi et al. [5] Belgacem et Hifi [6], Boyer et al. [7].
- Le problème du sac-à-dos multiple (MKP) : on notera les travaux de Hung et Fisk [8], Martello et Toth [9].
- Le problème du sac-à-dos disjonctif (DCKP) : on notera les travaux de Yamada et Kataoka [10], Hifi et Michrafy [11].
- Le problème de bin packing: on notera les travaux de Bekrar et al.[12] et Hifi et al. [13].

3.2.2 Problème d'affectation

C'est un problème relevant de la classe *P* bien évidement parce que l'algorithme de Khun connu sous le nom de la méthode Hongroise [76] représente un algorithme polynomial pour sa résolution.
Le problème d'affectation consiste à affecter *n* personnes à *n* tâches. Dans le cas mono- objectif, un coût est associé à l'affectation d'une personne à une tâche et l'idée est de déterminer la permutation des *n* objets ayant le coût total minimal. Soit *i* l'indice des personnes, *j* l'indice des objets et $c_{i,j}$ le coût de l'affectation de *i* à *j*. Ce problème se formule mathématiquement comme suit :

$$(AP)\begin{cases} \text{Min } Z(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} cij\ xij \\ \sum_{i=1}^{n} xi\ j = 1 \quad j = \{1,\dots,n\} \\ \sum_{j=1}^{n} xi\ j = 1 \quad i = \{1,\dots,n\} \\ x_{ij} \in \{0,1\} \quad i, j = \{1,\dots,n\} \end{cases} \qquad (1.1)$$

Où $c_{i,j}$sont des entiers positifs et $x = (x_{1,1}, \dots, x_{n,m})$ est la matrice des variables de décision telle que $x_{i,j} = 1$ si la personne i est affecté à l'objet j et est égale à 0 sinon.

3.2.3 Problème d'ordonnancement

Il existe un grand nombre d'approches au problème d'ordonnancement. Elles se distinguent les unes des autres par un certain nombre de critères de classification dont on peut citer :

Ordonnancement préemptif ou non préemptif: Un ordonnancement est dit *préemptif* s'il est possible d'interrompre l'exécution d'une tâche avant qu'elle ne s'achève.

Ordonnancement avec ou sans divisibilité : la divisibilité est l'hypothèse selon laquelle une même tâche peut être parallélisée sur plusieurs processeurs. On la retrouve notamment dans l'ordonnancement des applications sur les architectures parallèles.

Ordonnancement statique ou dynamique : Les algorithmes d'ordonnancement sont de caractère statique ou dynamique **[**14**].** Les approches *statiques* prédéterminent un ordonnancement du système, en utilisant des informations préalables sur l'exécution du système . Ils calculent au moment de la compilation sur quel processeur chaque tâche devra s'exécuter et à quel moment elle devra être initiée. Par contre, les approches *dynamiques* déterminent un ordonnancement au cours de l'exécution du système, ce qui permet plus de flexibilité.

Ordonnancement avec ou sans recouvrement des communications :

Cette hypothèse permet de considérer qu'un processeur peut simultanément envoyer des données, en recevoir et exécuter des tâches. Pratiquement, cela signifie que l'on suppose que le processeur est équipé d'un co-processeur dédié aux communications qui a un accès indépendant à la mémoire **[**15**].**

Ordonnancement local ou global : Ordonnancement des instructions dans un *bloc de base* est appelé ordonnancement des instructions *local*, tandis que d'ordonnancement de plusieurs *blocs de base* à la fois est appelé ordonnancement des instructions *globale*.

Les Contraintes d'un ordonnancement

L'ordonnancement du code est une forme d'optimisation de programme qui est appliquée au code machine. L'ordonnanceur de code a trois types de contraintes:

Contraintes des dépendances de contrôle : Toutes les opérations exécutées dans le programme original doivent être exécutées dans le code optimisé.

Contraintes de dépendances de données : Les opérations dans le programme optimisé doivent générer le même résultat que les opérations dans le code original.

Contraintes de ressources : L'ordonnancement ne peut pas déborder sur les ressources de la machine.

3.2.4 Problème du voyageur de commerce

Le problème du voyageur de commerce (TSP) a été étudié au 18 ème siècle par un mathématicien d'Irlande nommé William Rowam Hamilton et par le mathématicien britannique nommé Thomas Penyngton Kirkman. Discussion détaillée sur le travail de Hamilton et Kirkman peut être vu à partir du livre intitulé Graph Theory [16]. C'est un problème relevant de la classe NP, il est même NP-complet. Sous sa forme la plus classique, son énoncé est le suivant : " Un voyageur de commerce doit visiter une et une seule fois un nombre fini de ville et revenir à son point d'origine. Trouvez l'ordre de visite des villes qui minimise la distance totale parcourue par le voyageur ". Le problème TSP se formule mathématiquement comme suit :

$$(TSP)\quad \begin{cases} \text{Min } Z(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} cij\ xij \\ \sum_{i=1}^{n} xi\ j = 1 \quad \forall\ j \in N \\ \sum_{j=1}^{n} xi\ j = 1 \quad \forall\ i \in N \\ \sum_{i\in T}^{n} \sum_{j\in 1}^{n} xi\ j \geq 1\ \forall\ \mathrm{T} \subset N, T = N \setminus T \\ x_{ij} \in \{0,1\} \quad \forall\ i,j \in N \end{cases} \qquad (1.2)$$

Où $N = \{1,2,\dots,n\}$ l'ensemble de villes à visiter.

$$x_{ij} = \begin{cases} 1 & \text{on effectue le trajet allant de i à j,} \\ 0 & \text{sinon} \end{cases} \qquad (1.3)$$

3.3 Résolution des problèmes combinatoires difficiles

De nombreuses méthodes sont proposées pour la résolution de ce type de problème. On trouvera une synthèse de ces méthodes dans les articles de [Laporte et al. 2000]. Ces méthodes sont principalement divisées en deux groupes, les méthodes exactes et les méthodes approchées. Ces dernières sont aussi divisées en deux sous groupes (voir la figure 1.1) :

- Les méthodes exactes
- Les méthodes approchées
  - Les heuristiques
  - Les métaheuristiques

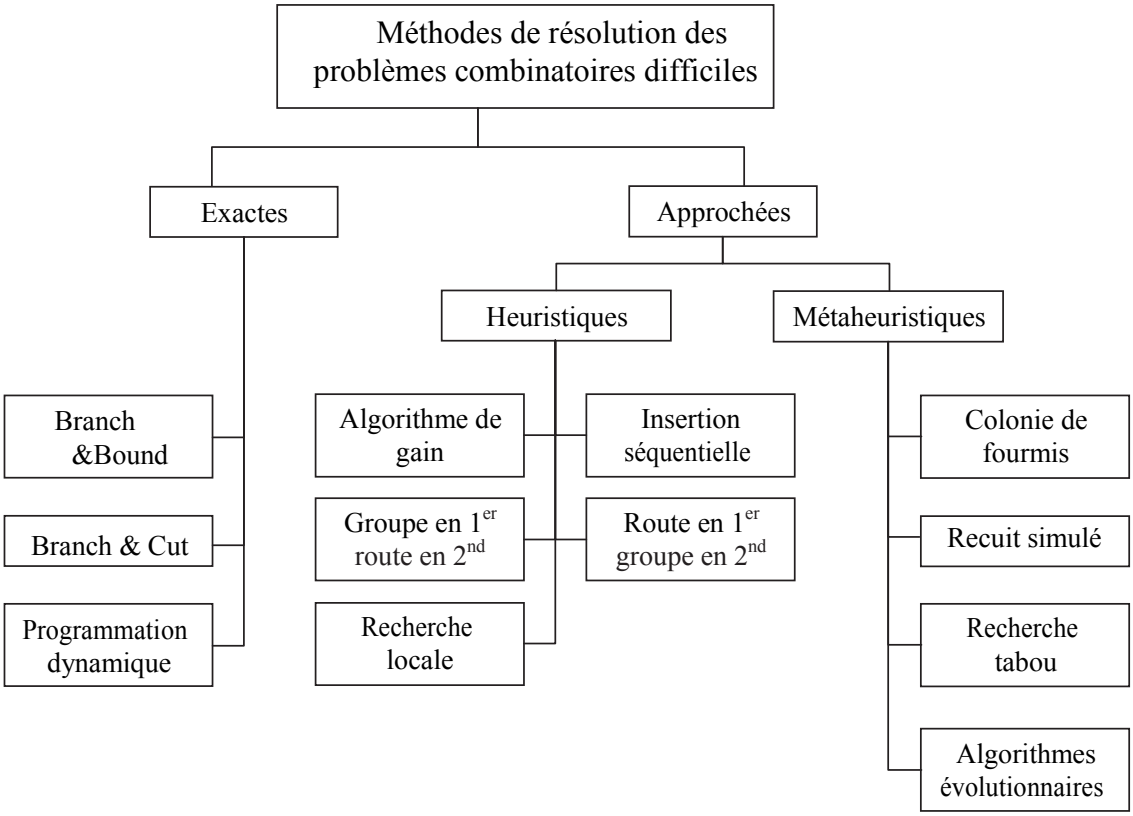


Fig 1.1 – Les Méthodes de résolution des problèmes combinatoires difficiles

4 Le problème de tournées de véhicules

Le Problème le plus simple, le plus ancien et le plus étudié des problèmes de tournées est celui du Voyageur de Commerce (*Traveling Salesman Problem* - TSP [17]). Il s'agit d'un représentant habitant une ville donnée et qui doit visiter un ensemble de clients lors de sa tournée journalière. Il désire bien évidemment d'optimiser certains critères, en particulier minimiser la longueur du trajet total effectué. Le TSP est connu comme *NP* -Difficile par la réduction au HCP [18]. Le premier programme linéaire pour le TSP est proposé par Dantzig et al. [19]. Cette formulation sert toujours comme base des méthodes de résolution exactes pour le TSP. En particulier, les algorithmes récents basés sur le *branch-and-cut* peuvent résoudre des instances de grande taille du problème.

Si plusieurs représentants d'une même agence doivent se partager un ensemble de clients, le problème devient alors un multi-voyageurs de commerce (*multiple traveling salesmen problem* -

MTSP [20]). Il est possible de ramener la résolution du MTSP à la résolution d'un TSP avec une multiplication de villes de départ à un facteur *m*.

Le problème de routage de véhicules est une extension de TSP (Dhaenens et al) [21]. Il a été introduit pour la première fois par Dantzig et al. [22] sous le nom de *Truck Dispatching Problem* et a depuis fait l'objet d'études intensives pour le modéliser et le résoudre.

Dans sa version la plus basique dite *Capacitated* VRP (CVRP) ou VRP avec contraintes de capacité, une flotte de véhicules de capacité finie (nombre de véhicules est noté m), basée dans un dépôt, doit assurer des tournées entre plusieurs clients (ou villes) (nombre de clients est noté n) ayant demandé chacun une certaine quantité de marchandises. L'ensemble des clients visités par un véhicule désigne la tournée de celui-ci. Chaque client doit être desservi une et une seule fois et chaque tournée commence et se termine au dépôt.

L'objectif du CVRP est de minimiser le coût total, c-à-d la somme des distances ou des temps de parcours des tournées, tout en respectant la contrainte de capacité des véhicules : la quantité de marchandises livrées sur une tournée ne doit pas dépasser la capacité du véhicule qui l'assure.

La figure 1.2 représente un exemple de problème de CVRP avec 20 clients, résolu avec 4 véhicules.

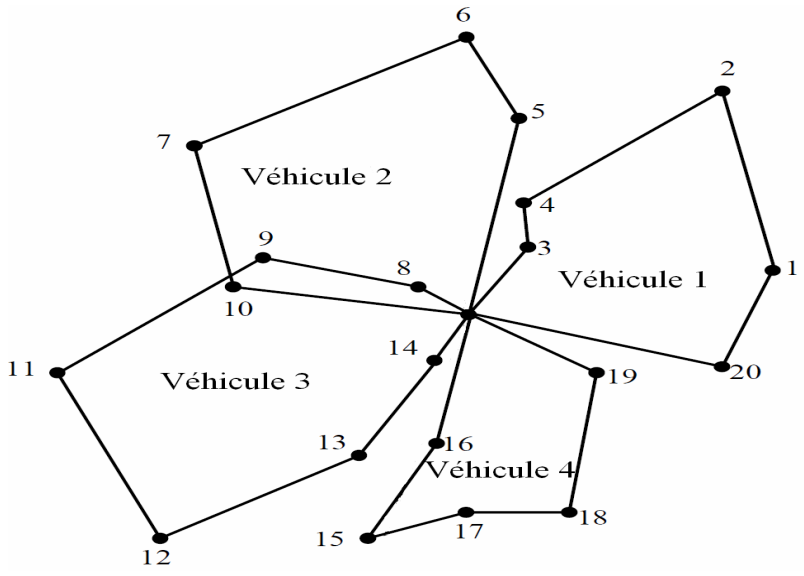


Fig 1.2 Un exemple de problème de CVRP à $n = 20$ clients résolu avec $m = 4$ véhicules.

4.1 Formulation du problème de tournées de véhicules

La formulation du VRP que nous présentons ici correspond à la formulation mathématique utilisée en programmation linéaire en nombres entiers. Elle traduit la modélisation naturelle du problème

par la définition d'une variable binaire $X_{i,j,k}$ égale à 1 si le véhicule k parcourt l'arc(v_i, v_j), noté plus simplement(i, j). Cette formulation est la plus utilisée dans la littérature, et a ainsi été adoptée par Laporte [23], Rego and Roucairol [24], Toth and Vigo [25], Crainic and Semet [26], etc.

Nous reprenons les mêmes notations définies dans la section précédente en ajoutant à l'ensemble des sommets du graphe $V = \{v_1, \dots, v_n\}$ un autre sommetv_0, correspondant au dépôt. Comme c'est le cas généralement, nous supposons que le graphe $G = (V, E)$est complet, c-à-d que tous les sommets sont reliés entre eux. Cela signifie qu'une ville peut être visitée à partir de toute autre ville.

Les autres constantes du problème sont les suivantes :

n nombre de clients (ou sommets),

m nombre de véhicules

Q capacité des véhicules,

q_i demande du client i,

$C_{i,j}$ Le coût de l'arête entre les sommets i et j (distance ou temps de parcours),

Les variables de décision du problème sont les $X_{i,j,k}$évoquées plus haut :

$$X_{i,j,k} = \begin{cases} 1 & \text{Si le trajet de } i \text{ à } j \text{ est effectué par le véhicule } k \\ 0 & \text{sinon} \end{cases} \quad (1.4)$$

$$(\text{CVRP}) \begin{cases} \text{Min Z(X)} = \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=1}^{k} Cij\text{X}ijk \text{ ....(A)} \\ \sum_{i=0}^{n} \sum_{k=1}^{k} \text{X}ijk = 1 \quad \forall\, j \in \{1, \dots, N\} \text{ .... (1)} \\ \sum_{j=0}^{n} \sum_{k=1}^{k} \text{X}ijk = 1 \quad \forall\, j \in \{1, \dots, N\} \text{ ....(2)} \\ \sum_{i=0}^{n} \text{X}ipk - \sum_{j=0}^{n} \text{X}pjk \; \forall\, p \in \{1, \dots, N\}, k \in \{1, \dots, \text{m}\} \text{..(3)} \\ \sum_{j=0}^{n} \text{q}j(\sum_{i=0}^{n} \text{X}ijk) \leq \text{Q}k \quad \forall \text{ k} \in \{1, \dots, \text{m}\} \text{..........(4)} \\ \sum_{i=0}^{n} \sum_{j=0}^{n} dijXijk \leq \text{D}k \; \forall \text{ k} \in \{1, \dots, \text{m}\} \text{..........(5)} \\ \sum_{i=1}^{n} \text{X}0jk \leq 1 \forall \; k \in \{1, \dots, \text{m}\} \text{..........(6)} \\ \sum_{i=1}^{n} \text{X}i0k \leq 1 \forall \; k \in \{1, \dots, \text{m}\} \text{..........(7)} \\ \text{X}ijk \in \{0,1\} \quad \forall i,j,\text{k} \end{cases} \quad (1.5)$$

La fonction objectif consiste à minimiser le coût total de transport (Eq. A). Les contraintes (1) et (2) assurent que chaque client soit servi par un seul véhicule. La continuité d'une tournée est assurée par la contrainte (3) où si un véhicule arrive à un point de livraison (client), il doit également le quitter. La contrainte (4) est la contrainte de capacité du véhicule. La contrainte (5)

limite la distance maximale d'une tournée. Les contraintes (6) et (7) assure que chaque véhicule ne soit pas utilisé plus d'une seule fois.

Remarque : Notons que cette formulation mathématique de CVRP repose sur un modèle appelé modèle de flot à trois indices (*three-index vehicleflow formulation*) voir Regoand Roucairol[24], Toth and Vigo [25], Crainic and Semet[26]. Étant donné que dans (A) le coût de la solution est exprimé comme la somme des coûts des arêtes traversées, un modèle de flot à deux indices seulement ($x_{i,j}$) peut également être utilisé de manière aussi efficace. Cependant, l'absence de l'information quel véhicule k parcourt l'arête (i,j) ? devient limitante pour d'autres variantes du VRP, notamment si le coût de la solution dépend du type des véhicules utilisés par exemple (Toth and Vigo)[25].

Paramètres

Outre la version basique du Capacitated VRP, le problème de routage de véhicules a plusieurs autres variantes plus ou moins étudiées dans la littérature.

En effet, la définition la plus générale du VRP est la suivante : il s'agit de la conception de routes optimales par une flotte de véhicules, basée en un ou plusieurs dépôts, pour desservir un ensemble de clients (ou villes) dispersés géographiquement et ayant des demandes connues.

Cette définition généraliste met en évidence l'ensemble de paramètres qui caractérisent une variante du VRP : le réseau de transport, la clientèle et la flotte de véhicules. Des contraintes auxiliaires peuvent éventuellement s'ajouter à ces trois paramètres principaux. Un dernier paramètre réside dans la fonction objectif à optimiser.

Le réseau : Le réseau routier peut être symétrique ou asymétrique ; en conséquence, le graphe associé $G = (V, E)$ sera orienté ou non et les liaisons entre les sommets seront des arcs ou des arêtes.

Les tournées peuvent partir d'un seul dépôt ou de plusieurs dépôts.

La clientèle : La principale caractéristique de la clientèle est sa demande en marchandise. La livraison de celle-ci peut être contrainte à s'effectuer au cours de périodes de temps spécifiées, appelées fenêtres temporelles (*time windows*). Ces contraintes peuvent être dures ou souples. Dans le cas de contraintes dures, une arrivée avant la fenêtre temporelle impose une attente, et les retards sont interdits ; alors que les contraintes souples peuvent être violées et induisent ainsi des pénalités (Kallehauge et al) [27].

Les clients peuvent être livrés en marchandises mais peuvent aussi en remettre aux véhicules. On parle alors de tournées de livraison/ramassage ou de service mixte. Les temps de livraison et de ramassage peuvent être non négligeables et sont ainsi pris en compte dans le calcul des durées de tournées.

Finalement, l'accès à un client peut être limité à un sous-ensemble de véhicules seulement.

La flotte de véhicules est Le nombre de véhicules disponibles peut être fixe ou non. Notons que dans le cas d'un seul véhicule, le problème de tournées reste tout de même différent d'un problème de voyageur de commerce car la livraison des différents clients peut s'effectuer en plusieurs tournées.

Un véhicule peut être associé à un dépôt particulier ou pas.

Les véhicules peuvent avoir une capacité maximale en termes de marchandises transportées (volume, poids etc.). Dans le cas d'une flotte hétérogène, cette capacité peut différer selon le type de véhicules.

La fonction objectif : Les objectifs les plus communs sont soit la minimisation du nombre de véhicules utilisés soit la minimisation de la distance totale parcourue par les véhicules. D'autres objectifs peuvent être considérés :

- la minimisation de la durée totale des tournées
- la minimisation du coût total des tournées (en prenant en compte les coûts des véhicules, des chauffeurs etc.)
- la minimisation des pénalités liées aux violations des contraintes, notamment dans le cas de fenêtres temporelles
- la maximisation des gains engendrés par les tournées
- etc.

Selon Bräysy [67], les objectifs de minimisation du nombre de véhicules et de la distance (ou durée) totale des tournées sont conflictuels : la diminution du nombre de véhicules engendre le plus souvent une augmentation de la distance totale parcourue.

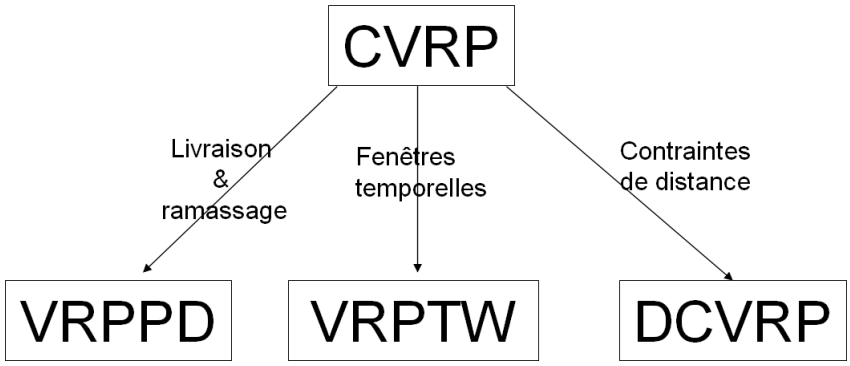


Fig 1.3 - Variantes de base du VRP avec contraintes de capacité.

Notons que dans des approches multi-objectifs, une somme pondérée de ces objectifs peut être considérée (Jozefowiez) [28].

Dans un état de l'art de différentes approches multi-objectives, Talbi [29] classifie cette méthode d'agrégation parmi les approches transformant le problème multi-objectif en un problème mono-objectif. Il distingue aussi deux autres types d'approches : les approches non-Pareto qui traitent les différents objectifs séparément et les approches Pareto qui cherchent des solutions diversifiées sur la frontière de Pareto. Cette dernière approche est basée sur la notion de Pareto-dominance entre les solutions : une solution est dite Pareto-optimale (ou optimale au sens de Pareto) si elle est optimale sur au moins un objectif, et que toute amélioration d'un objectif entraînerait la dégradation d'un autre.

4.2 Les différents problèmes de tournées de véhicules

Les cas d'application réels exigent souvent des contraintes additionnelles auxquelles sont soumis les opérationnels. On retrouve par exemple des problèmes impliquant plusieurs dépôts, une flotte de véhicules hétérogène, ou des fenêtres de temps associées aux clients. Ces attributs se traduisent en une vaste littérature, et il serait impossible de donner ici un état de l'art exhaustif de ces problèmes et des méthodes de résolution associées. Nous recommandons l'ouvrage de référence de Toth and Vigo [30] qui présentent en détail les différentes extensions du VRP ainsi que les méthodes associées, exactes et approchées.

Problème de tournés avec fenêtres de temps

Le VRPTW (*Vehicle Routing Problem With Time Windows*) est la variante du VRP la plus étudiée [30]. Des fenêtres de temps de visite sont associées aux clients et au dépôt. Chaque arc

étant dans ce contexte caractérisé par une durée de trajet généralement assimilé à un cout. La disponibilité d'un client i est représentée par une fenêtre de temps définie par une date de service au plus tôt e_i et une date de service au plus tard l_i. Un véhicule se rendant au client i plus tôt que e_i doit attendre jusqu'au début de la fenêtre de temps. Le temps d'attente résultant est pris en compte dans la contrainte de durée maximum de route. Une arrivée tardive après l_i rend la solution associée irréalisable. Certaines variantes, nommées "fenêtres de temps souples" autorisent des arrivées tardives et éventuellement en avance, sanctionnées au moyen de pénalités.
De nombreuses formulations et méthodes exactes sont proposées pour le VRPTW [37]. Nous pouvons citer quelques méthodes récentes de résolution exacte : le *branch-and-cut* [31], le *branch-and-price* [32] et le *branch-and-price-and-cut* [33]. Les métaheuristiques proposées pour le VRPTW sont les algorithmes de recherche tabou [34], recuit simulé [35] et les algorithmes génétiques [36].

Problème de tournés avec flotte limité

Contrairement au cas de la flotte illimitée où une solution triviale consiste à associer une tournée à chaque client, Le m-VRPTW [37] impose une contrainte sur le nombre de véhicules disponibles. En effet, les moyens logistiques étant rarement infinis, une restriction de la flotte à m véhicules peut être introduite à la définition du problème. Cette contrainte a pour effet de rendre plus difficile la production de solutions réalisables. S'il existe une affectation permettant de satisfaire toutes les demandes sur l'horizon de planification, le problème est dit satisfiable et l'objectif est de trouver un ensemble des tournées respectant les contraintes de capacité des véhicules et des fenêtres de temps, tout en minimisant le coût total de transport lié aux arcs empruntés par les véhicules. Une recherche tabou a été proposée par Lau et al. [37] pour résoudre le m-VRPTW. Cette recherche est basée sur le concept de Holding List qui est une tournée particulière comportant l'ensemble des clients non routés. Les opérations de transfert et d'échange de clients considérés pour le voisinage sont effectuées sur la Holding List au même titre que sur les autres tournées. On retrouve le même principe sous forme de d'Ejection Pool dans les travaux de Lim and Xingwen [38]. Un modèle mathématique amélioré a été proposé par Wang et al. [39]. Le modèle contribue à l'initialisation d'un algorithme génétique conçu pour le m-VRPTW.

Problème de tournés avec contrainte de distance maximale

Dans le DVRP (*Distance Constrained VRP*), une contrainte sur la distance maximale que peut parcourir chaque véhicule est posée. Ce type de contrainte permet de prendre en compte les

possibilités d'épuisement du carburant ou l'autonomie limitée des véhicules imposant leur retour au dépôt pour ravitaillement. Les problèmes avec fenêtres de temps prennent implicitement ce type de contrainte en compte puisque la fenêtre de temps associée au dépôt limite la longueur des tournées. On ne trouve pas beaucoup de travaux traitant spécifiquement le DVRP. Laporte, Desrocher et Nobert en [1989, 1990] ont produit des méthodes exactes pour ce problème. Prins [40] a notamment proposé un algorithme mémétique très efficace basé sur un codage de tournées sans séparateur et sur le découpage optimal de Beasley pour le décodage. Il convient de mentionner que la plupart des heuristiques et métaheuristiques pour le VRP fonctionnent également pour le DVRP, ou sont très facilement adaptables.

Problème de tournés avec profits

Une entreprise est souvent amenée à traiter certaines demandes dans une période de temps donnée (par exemple une journée). Cependant, les ressources personnelles et matérielles de l'entreprise ont souvent des limites par rapport à la demande des clients. Les entreprises sont donc confrontées à une demande dépassant leurs moyens et doivent par conséquence identifier les clients à servir dans la période actuelle et ceux à traiter ultérieurement, en vue de maximiser leur profit. Cette problématique forme une nouvelle classe des problèmes de tournées appelée problèmes de tournées avec profits (*Routing Problems with Profits* - PRPs [41]). Dans ce type de problème, il est impossible de servir tous les clients. Afin de les discriminer, une valeur P_i, qualifiée de profit, est associée à chaque client. Cette grandeur vise à quantifier l'intérêt que représente le service d'un client donné.

Tout comme les problèmes de tournées classiques, les routes construites doivent être de coût minimal. C'est pourquoi des tournées avec profits sont par nature bi-objectifs : les gains de profits à maximiser et les coûts des trajets à minimiser.

Plusieurs variantes peuvent figurer dans cette classe en jouant sur la considération des deux objectifs déjà mentionnés. Le *Problème de Tournées Sélectives* (PTS) [41] s'intéresse à maximiser le gain récolté par tous les véhicules et à limiter, au niveau des contraintes, la longueur de chaque tournée par une distance maximale. Par opposition aux problèmes sélectifs, les *problèmes de tournées avec quotas* (PTQ) cherchent à minimiser la somme des trajets en se fixant un profit minimal à atteindre. Une autre variante du problème considère simultanément les deux objectifs [42]. La fonction à optimiser est donnée sous forme agrégée.

5 Conclusion

Nous avons présenté dans ce premier chapitre les notions et les outils nécessaires à la bonne compréhension de notre travail. Nous avons rappelé certaines notions de base en théorie de la complexité des algorithmes et des problèmes. Puis, nous avons met l'accent sur les principales définitions et notions liées à l'optimisation combinatoire. Un troisième volet a été consacré à l'étude formelle du problème de tournées de véhicules avec contrainte de capacité ainsi que les différents problèmes de tournées de véhicules.

CHAPITRE II

Algorithmes métaheuristiques

1 Introduction

Le terme métaheuristique a été inventé par Fred Glover en 1986[43], lors de la conception de la recherche tabou. Les métaheuristiques peuvent être considérées comme des algorithmes stochastiques itératifs, où ils manipulent une ou plusieurs solutions à la recherche de l'optimum. Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement à atteindre le temps d'exécution ou une précision demandée. Ces méthodes tirent leur intérêt de leur capacité à éviter les optimums locaux, soit en acceptant des dégradations de la fonction objectif lors du traitement, soit en utilisant une population de points comme méthode de recherche.

Les métaheuristiques forment une famille d'algorithmes d'optimisation pour résoudre des problèmes d'optimisation difficiles, pour lesquels nous ne connaissons pas de méthodes classiques plus efficaces. Ils sont généralement utilisés comme des méthodes génériques qui peuvent optimiser un large éventail de problèmes différents, d'où le qualificateur de méta. Leur capacité à optimiser un problème à partir d'une quantité minimale d'informations est contrebalancée par le fait qu'ils n'offrent aucune garantie quant à l'optimalité de la meilleure solution trouvée. Cependant, du point de vue de la recherche opérationnelle, cette constatation n'est pas nécessairement un inconvénient, puisque l'on préfère toujours une approximation de l'optimum global trouvée rapidement à une valeur exacte trouvée dans un temps inacceptable [44].

2 Métaheuristiques à solution unique

Les métaheuristiques à mono solution ou à solution unique manipulent une solution à la fois.

2.1 Recuit simulé

Cette méthode trouve ses origines dans la *thermodynamique*. Elle est issue d'une analogie entre le phénomène physique de refroidissement lent d'un corps en fusion, qui le conduit à un état solide de basse énergie. Il faut abaisser lentement la température, en marquant des paliers suffisamment longs pour que le corps atteigne *l'équilibre thermodynamique* à chaque palier de température. Pour les matériaux, cette énergie basse se manifeste par l'obtention d'une structure régulière comme dans les cristaux ou l'acier. L'analogie exploitée par le recuit simulé consiste à considérer une fonction f à minimiser comme fonction d'énergie. Une solution x peut être considérée comme un état donné de la matière dont $f(x)$ est l'énergie.

Le recuit simulé exploite généralement le critère défini par l'algorithme de Metropolis (voir le Pseudo-Code de la règle de Metropolis) pour l'acceptation d'une solution obtenue par perturbation de la solution courante. Pour une "température" T donnée, à partir d'une solution courante x, on considère une transformation élémentaire qui changerait x en $s(x)$. Si cette perturbation induit une diminution de la valeur de la fonction objectif f, $\Delta f = f(s(x)) - f(x) < 0$, elle est acceptée, ceci dans le cas d'une minimisation. Dans le cas contraire, si $\Delta f = f(s(x)) - f(x) \geq 0$, la perturbation est acceptée avec une certaine probabilité

$p = exp^{\frac{-\Delta f}{T}}$ qui décroît avec le temps (Thangiah et al) [45].

Le paramètre de contrôle T de cet algorithme est la "Température" du système, qui influe sur la probabilité d'acceptation d'une solution moins bonne. A une température élevée, la probabilité d'acceptation d'un mouvement quelconque tend vers 1 : presque tous les changements sont acceptés. L'algorithme équivaut alors à une marche aléatoire dans l'espace des solutions. Cette température est diminuée lentement au fur et à mesure du déroulement de l'algorithme pour simuler le processus de refroidissement des matériaux, et sa diminution est suffisamment lente pour que l'équilibre thermodynamique soit maintenu (voir le Pseudo-Code de l'algorithme du recuit simulé).

La génération des voisins $S(x)$ se fait par la technique de la recherche locale.

Algorithme Pseudo-Code de la règle de Metropolis

```
SI f(s(x)) ≤ f(x)
   f(x) = f(s(x))
   x = s(x)
SINON
   p = exp -Δf/T
   r = solution aléatoire entre [0, 1]
   SI r ≤ p
      f(x) = f(s(x))
      x = s(x)
   FIN SI
FIN SI
```

Algorithme Pseudo-Code de l'algorithme du recuit simulé

$x =$ solution aléatoire
$f_{min} = f(x)$
$x_{min} = x$
Initialiser température T (assez élevée)
REPETER
REPETER
générer un voisin $s(x) \in$ voisinage $S(x)$
Appliquer la règle de Metropolis
SI $f(x) < f_{min}$
$f_{min} = f(x)$
$x_{min} = x$
FIN SI
JUSQU'A équilibre thermodynamique atteint
décroître température T
JUSQU'A conditions d'arrêt satisfaites

L'efficacité du recuit simulé dépend fortement du choix de ses paramètres de contrôle, dont le réglage reste lui aussi très empirique. Les principaux paramètres de contrôle sont les suivants :

- la valeur initiale de la température,
- la fonction de décroissance de la température,
- le critère de changement de palier de température,
- les critères d'arrêt.

Le rôle de la température T au cours du processus de recuit simulé est très important. Une forte décroissance de température risque de piéger l'algorithme dans un minimum local, alors qu'une faible décroissance au début du processus entraîne une convergence très lente de l'algorithme. Un compromis pour adapter la décroissance de la température à l'évolution du processus consiste à utiliser une variation logarithmique. La loi logarithmique de décroissance de la température, qui assure la convergence théorique du recuit simulé, est la suivante :

$$T_k = \frac{\mu}{Log(1 + k)}$$

Où k est le nombre de paliers de température effectués, et μ est une constante positive. En pratique, on adopte souvent une décroissance géométrique $T_{k+1} = \alpha.T_k$, avec $(0 < \alpha < 1)$, car la loi précédente induit un temps de calcul prohibitif.
Nous trouverons dans Rego et al [46] une utilisation de cette méthode pour résoudre le problème VRP.

2.2 Recherche tabou

La recherche tabou (RT) est une métaheuristique originalement développée par Glover [43]. Elle est basée sur des idées simples, mais elle est néanmoins très efficace. Cette méthode combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes permettant à celle-ci de surmonter l'obstacle des optima locaux, tout en évitant les problèmes de cycle. Elle a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire dont le problème de routage de véhicules (VRP).

Dans une première phase, la méthode de recherche tabou part d'une solution quelconque x appartenant à l'ensemble de solutions X, puis on se déplace vers une solution $s(x)$ située dans le voisinage $S(x)$ de x. L'algorithme va explorer itérativement un sous-ensemble de l'espace de solutions X. Afin de choisir le meilleur voisin $s(x)$ dans le voisinage $S(x)$, l'algorithme évalue la fonction objectif f en chaque point $s(x)$, et retient le voisin qui améliore la valeur de la fonction objectif f ou si ce n'est pas possible celui qui la dégrade le moins.

L'originalité de la méthode de recherche tabou par rapport aux méthodes de recherches locales réside dans le fait que l'on retient le meilleur voisin, même si celui-ci est plus mauvais que la solution d'où l'on vient. Ce caractère autorisant les dégradations de la fonction objectif évite à l'algorithme d'être piégé dans un minimum local. Cependant il induit un risque de création de cycles. En effet, lorsque l'algorithme quitte un minimum local par acceptation de la dégradation de la fonction objectif, il peut revenir sur la même trace déjà suivie, à l'itération suivante. Pour régler ce problème, l'algorithme a besoin d'une mémoire pour conserver la trace des dernières meilleures solutions déjà visitées.

Ces solutions sont déclarées *tabous*, d'où le nom de la méthode. Elles sont stockées dans une liste de longueur L donnée, appelée *liste tabou*. Une nouvelle solution n'est acceptée que si elle n'appartient pas à cette liste tabou. Ce critère d'acceptation d'une nouvelle solution évite la formation de cycles de longueur inférieure ou égale à la longueur de la liste tabou. Il dirige l'exploration de la méthode vers des régions du domaine de solutions non encore visitées. Le Pseudo-Code de l'algorithme tabou classique est présenté dans l'algorithme suivant.

Algorithme Pseudo-Code de la méthode de recherche tabou

x = solution aléatoire
$f_{min} = f(x)$
$x_{min} = x$
TABOU est vide // liste de solutions $s(x)$, de longueur L
REPETER
Générer 1 solution **TEL QUE** $s_i(x) \in$ voisinage $S(x)$ et $\{x, s_i(x)\} \notin$ TABOU
$f(s(x)) = min_i[f(s_i(x))]$, $1 \leq i \leq N$
Ajouter ($\{x, s_i(x)\}$, TABOU)
$x = s(x)$
SI $f(x) < f_{min}$
$f_{min} = f(x)$
$x_{min} = x$
FIN SI
JUSQU'A conditions d'arrêt satisfaites

On élimine à chaque itération la solution tabou la plus ancienne, en la remplaçant par la nouvelle solution retenue. Mais le codage d'une telle liste est encombrant, car il faudrait garder en mémoire tous les éléments qui définissent une solution. Pour pallier cette contrainte, il est possible de remplacer la liste tabou de solutions interdites par une liste de "*transformations interdites*", en interdisant la transformation inverse d'une transformation faite récemment.

Cette méthode métaheuristique est largement utilisée pour le problème VRP voir Rego et al [46], Gulay et al [47] et Rego [48]. Nous trouvons aussi une version parallèle qui traite ce problème (Caricato et al) [49].

2.3 Recherche locale itérée

La Recherche locale itérée (*Iterated Local Search* - ILS[50]) est une métaheuristique de trajectoire. Son principe consiste à alterner entre une recherche locale et une procédure de perturbation. Lorsque la méthode découvre un nouvel optimum, elle choisit soit de le prendre comme démarrage référence soit de garder l'ancien. La solution choisie sert de point de départ pour la procédure de perturbation. Il existe plusieurs variantes d'ILS, par exemple certaines acceptent des solutions de qualité détériorée sous certaines conditions, d'autres considèrent plusieurs types de perturbation et elles sont appliquées à certaines étapes appropriées de l'algorithme.

2.4 Recherche à voisinage variable

Recherche locale à voisinage variable (*Variable Neighborhoods Search* - VNS [51]) Cette métaheuristique nécessite la définition d'un ensemble de voisinages (V_1, V_2, ..., V_k). Une relation d'ordre entre ces voisinages est à définir en fonction du problème. L'idée principale est de changer itérativement la meilleure solution en explorant différents voisinages dans l'espoir d'améliorer la solution courante ou bien de la diversifier. A partir d'une solution courante *s*, le voisinage V_i est sélectionné et une recherche locale est appliquée. Si la solution obtenue est meilleure que la solution *s* alors la recherche reprend à partir de cette solution avec le voisinage de départ V_1. Sinon, la recherche passe à un voisinage de portée supérieur V_{i+1}. De nombreuses variantes de VNS existent, nous pouvons citer la version réduite (*Reduced* VNS), la version biaisée (*Skewed* VNS) et la méthode de descente à voisinage variable (*Variable Neighborhood Descent* -VND).

3 Métaheuristiques à population de solutions

Les métaheuristiques à solutions multiples manipulent une population de solutions et sont pour cela appelées métaheuristiques à population de solutions.

3.1 Algorithmes génétiques (GAs)

Les principes fondamentaux de ces algorithmes ont été exposés par Holland [52]. Ces algorithmes s'inspirent du fonctionnement de l'évolution naturelle, notamment la sélection de Darwin, et de la reproduction selon les règles de Mendel. La sélection naturelle, que Darwin appelle "élément propulseur" de l'évolution, favorise les individus d'une population qui sont le mieux adaptés à un environnement. La sélection est suivie de la procréation. Elle est réalisée à l'aide de croisements et de mutations au niveau du patrimoine génétique des individus (ou "génotypes") qui est constitué d'un ensemble de gènes. Ainsi deux individus "parents", qui se croisent, transmettent une partie de leur patrimoine génétique à leurs descendants. Le génotype de l'enfant fait que celui-ci est plus ou moins bien adapté à l'environnement. S'il est bien adapté, il a une plus grande chance de se reproduire dans la génération future. Au fur et à mesure des générations, on sélectionne les individus les mieux adaptés, et l'augmentation du nombre des individus bien adaptés fait évoluer la population entière vers un optimum (local ou global).

Dans les algorithmes génétiques, nous essayons de simuler le processus d'évolution d'une population. Nous partons d'une population de *N* solutions du problème représentées par des

individus (chromosomes). Cette population choisie aléatoirement est appelée population parent. Le degré d'adaptation d'un individu à l'environnement est exprimé par la valeur de la fonction coût $f(x)$ (ou fitness), où x est la solution que l'individu représente. Nous disons qu'un individu est d'autant mieux adapté à son environnement, que le coût de la solution qu'il représente est plus faible (pour un problème de minimisation). Au sein de cette population, intervient alors la sélection au hasard d'un ou deux parents qui produisent une nouvelle solution, à l'aide des opérateurs génétiques, tels que le croisement et la mutation. La nouvelle population, obtenue par le choix de N individus parmi les parents et les enfants de l'étape courante, est appelée génération suivante. En itérant ce processus, nous produisons une population plus riche en individus mieux adaptés. L'algorithme suivant montre le Pseudo-Code de l'algorithme génétique de base.

Algorithme Pseudo-Code d'un algorithme génétique de base

POP, POP' : deux tableaux de taille N
Initialiser la population POP
Evaluer la population POP
POP' est VIDE
Rechercher $x \mid f(x) = min_i[f(x_i)],\ 1 \leq i \leq N$
$f_{min} = f(x)$
$x_{min} = x$
REPETER
  Evaluer la population POP
  REPETER //phase de reproduction génétique
    sélection
    croisement
    mutation
  JUSQU'A POP' remplie par les nouveaux individus
  Sélectionner la nouvelle population à partir de (POP, POP')
  Rechercher $x \mid f(x) = min_i[f(x_i)]\ ,\ 1 \leq i \leq N$
  SI $f(x) < f_{min}$
  $f_{min} = f(x)$
  $x_{min} = x$
  FIN DE SI
JUSQU'A conditions d'arrêt satisfaites
Résultat (x_{min}, f_{min})

Où, POP représente la population courante et POP' représente la population des individus générés par l'étape de reproduction.

Cet algorithme comporte trois phases distinctes :

1. La sélection de la population d'individus la mieux adaptée pour contribuer à la reproduction de la génération suivante (version artificielle de la sélection naturelle) ; elle peut être mise en œuvre sous plusieurs formes algorithmiques.
2. La phase de reproduction, qui exploite essentiellement les opérateurs de croisement et de

mutation sur les individus sélectionnés précédemment

3. La stratégie de remplacement des populations parent et enfant par la génération suivante. Elle pourra être mise en œuvre sous plusieurs formes.

Codage : Chaque individu de la population est une solution, cet individu est représenté par un *chromosome*. Ce chromosome est constitué de *gènes* qui peuvent prendre des valeurs appelées *allèles*. Il existe plusieurs manières de coder une solution au problème VRP.

Opérateurs de reproduction : La phase de reproduction exploite principalement deux opérateurs le croisement et la mutation. Elle comporte aussi l'opération de sélection et l'opération de production de la génération suivante, à partir des populations parent et enfant.

- ***Sélection*** : La sélection consiste à choisir les individus qui vont participer à la reproduction de la population future. La fonction de sélection choisit, de façon déterministe ou selon une méthode probabiliste (roulette, tournoi, etc.), un individu pour qu'il participe à l'étape de reproduction pour former la nouvelle population.
- ***Croisement***: Le principal opérateur agissant sur la population des parents est le croisement, qui est appliqué avec une certaine probabilité appelée taux ou probabilité de croisement P_c (typiquement proche de l'unité). Le croisement consiste à choisir deux individus tirés au hasard dans la population courante, et à définir aléatoirement un ou plusieurs points de croise- ment. Les nouveaux individus sont alors créés en interchangeant les différentes parties de chaque chaîne. Cet opérateur permet de bien **exploiter** le domaine de variation des individus, et de diriger la recherche vers des régions intéressantes de l'espace d'étude en utilisant la connaissance déjà présente dans la population courante.
- ***Mutation***: L'opération de mutation protège les algorithmes génétiques des pertes prématurées d'informations pertinentes. Elle permet d'introduire une certaine information dans la population, qui a pu être perdue lors de l'opération de croisement. Ainsi elle participe au maintien de la diversité, utile à une bonne **exploration** du domaine de recherche. L'opérateur de mutation s'applique avec une certaine probabilité, appelée taux ou probabilité de mutation P_m, typiquement faible.
- ***Elitisme* :** Cette méthode consiste à copier un ou plusieurs des meilleurs chromosomes dans la nouvelle génération avec un pourcentage prédéfini appelé "*taux d'élitisme*". Ensuite, on génère le reste de la population selon l'algorithme de reproduction usuel. Cette méthode améliore considérablement les algorithmes génétiques, car elle évite de perdre les meilleures solutions trouvées jusqu'ici (Davis) [68].

L'efficacité de l'algorithme génétique dépend fortement du choix de ces paramètres de contrôle (les

paramètres ci-dessus), dont le réglage reste très empirique, elle dépend aussi fortement de la représentation choisie pour le codage des solutions et des opérateurs de reproduction utilisés.

Pour le réglage des paramètres de l'algorithme génétique. Les principaux paramètres sont les suivants :

1. la taille de la population N ,
2. le type et taux de croisement,
3. le type et le taux de la mutation,
4. le type de la sélection,
5. le taux d'élitisme.

Il existe beaucoup de travaux relatifs au VRP et à ses extensions ; parmi eux nous pouvons citer les travaux de Rego et al [46] qui présentent plusieurs méthodes de résolution (heuristiques et métaheuristiques) du problème VRP et TSP de Pereira et al [52] qui proposent un algorithme génétique avec une nouvelle représentation chromosomique du VRP et de Prins [53] et Baker et al [69] qui utilisent l'algorithme génétique comme méthode de résolution avec aussi une nouvelle représentation des solutions, etc.

3.2 Optimisation par essaim de particules (PSO)

(Particle Swarm Optimization - PSO [54]) Cette méthode est inspirée du comportement social des animaux évoluant en essaim. Elle repose sur un ensemble d'individus appelés particules. Une particule représente une solution potentielle qui se déplace dans l'espace de recherche, en quête de l'optimum global. L'intelligence globale de l'essaim est donc la conséquence directe des interactions locales entre les différentes particules qui le constituent. Chaque particule dispose d'une mémoire concernant sa meilleure solution visitée a ainsi que la capacité de communiquer avec les particules de son entourage. A partir de ces informations, la particule calcule une vitesse de mouvement qui indique le degré de changement de la solution dans la prochaine itération. La mise à jour de la vitesse est influencée par les trois composantes suivantes :

- **Une composante physique :** la particule tend à suivre sa direction courante de déplacement. Ceci est contrôlé par le paramètre w appelé le facteur d'inertie ;
- **Une composante cognitive :** la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée. Le paramètre *c1* contrôlé le comportement cognitif de la particule ;
- **Une composante sociale** : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins. Le paramètre *c2* contrôle l'aptitude sociale de la particule.

La combinaison des paramètres *w*, *c1* et *c2* permet de régler la balance entre les phases de

diversification et d'intensification du processus de recherche. L'optimisation par essaim particulaire est à l'origine développée pour les problèmes d'optimisation à variables continues. Il est difficile d'appliquer le schéma PSO dans l'optimisation combinatoire à cause de diverses façons de définir les présentations et les opérateurs nécessaires, et à cause de la difficulté de trouver ceux qui sont adaptés au problème en question [55].

3.3 Optimisation par colonies de fourmis (ACO)

Les algorithmes de fourmis sont issus des travaux de Colorni, Dorigo et Maniezzo (Colorni et al) [70] au début des années 90. Ils reposent sur une analogie avec le comportement collectif des fourmis à s'organiser pour la recherche de nourriture.

Schématiquement, les fourmis explorent leur environnement immédiat en laissant derrière elles des traces chimiques, la *phéromone*. Elles se servent de la phéromone pour se guider et tendent naturellement à suivre les traces existantes. En l'absence de phéromone, leur exploration devient complètement aléatoire. Elles possèdent donc une vision très limitée de leur environne- ment. Une fois la nourriture trouvée, elles se servent des traces qu'elles viennent de déposer pour retrouver le chemin du retour vers le nid. Durant le trajet, elles laissent de nouveau sur leur passage de la phéromone, en quantité proportionnelle à l'intérêt de la source de nourriture (pondération). Le chemin est donc fortement imprégné de phéromones et constitue une piste de choix pour la colonie de fourmis. Plus la source de nourriture a été jugée intéressante, plus les fourmis auront tendance à suivre ce chemin. Ainsi, peu à peu, les traces vers les sources de nourriture seront de plus en plus marquées. Il est intéressant de noter que les fourmis ont la remarquable propriété de suivre naturellement le chemin le plus court vers la source de nourriture. En effet, Goss et al [71] a observé qu'en présence de deux routes possibles vers la nourriture, les four- mis adoptent rapidement la plus courte. Ceci est dû au fait que le retour à l'embranchement des fourmis ayant pris la bonne option s'opère avant celles ayant effectué le mauvais choix. De fait, dans l'intervalle de temps, la bonne route se trouve favorisée par une plus grande concentration de phéromones. Le biais est ensuite renforcé par le phénomène d'évaporation de la phéromone, qui tend à effacer progressivement les traces les moins fréquentées. Colorni, Dorigo et Maniezzo (Colorni et al) [70] ont adapté ce principe au domaine de l'Optimisation Combinatoire. Pour ce faire, ils ont associé le voisinage du nid à l'espace des solutions. Chaque solution s'apparente à une source de nourriture dont la qualité est fournie par la fonction d'évaluation. Chaque fourmi est assimilée à un processus répétitif de construction de solutions. La construction est guidée par un ensemble de données globales "*la phéromone*". Cet ensemble est typiquement une mémoire sur les attributs de solution, régulièrement mise à jour par les processus de construction (en fonction de la qualité de la

solution) et par un mécanisme simulant l'évaporation de la phéromone. Le problème majeur de ce modèle réside dans sa grande sensibilité vis-à-vis du principe de la phéromone. En fait, la qualité des résultats fournis par la métaheuristique dépend très fortement de l'importance accordée à la phéromone. Lorsque la pondération est trop élevée, le mécanisme de construction tend à favoriser la génération de solutions trouvées dans les itérations précédentes. A l'inverse, lorsqu'il est trop faible, il tend à produire des solutions aléatoires.

Le pseudo-code de l'algorithme de la colonie de fourmis est présenté dans l'algorithme suivant Nous pouvons améliorer la performance en tenant en compte des modifications suivantes qui touchent pratiquement tous les aspects de la métaheuristique :

Algorithme Pseudo-Code de la méthode de colonie de fourmis

1. Initialiser les traces,
2. Tant qu'un critère d'arrêt n'est pas satisfait, répéter en parallèle pour chacune des *p* fourmis :
   (a) Construire une nouvelle solution à l'aide des informations contenues dans les traces et une fonction d'évaluation partielle.
   (b) Evaluer la qualité de la solution.
   (c) Mettre à jour les traces.

(a) - le processus de génération de solutions s'est vu intégrer une phase d'amélioration itérative afin de fournir des solutions de meilleure qualité.

(b) - l'actualisation des traces a évolué pour permettre l'application explicite de stratégies de fouille ou de prospection. Seules les meilleures solutions donnent lieu à une mise à jour des traces.

(c) - la coordination des différentes fourmis a été sensiblement améliorée par l'apparition d'une fourmi particulière, la reine. Cette dernière prend en charge la gestion de la mémoire collective de la colonie (les traces) et prend les décisions stratégiques en fonction de l'évolution globale de la recherche. Elle constitue de fait une sorte d'agent "intelligent".

3.4 L'algorithme de TLBO (Teaching-Learning-Based Optimization algorithm)

TLBO est une méthode d'optimisation globale à l'origine développée par Rao et al [56], Rao et al [57] et Rao & Savsani [58]. C'est un algorithme inspiré du processus d'enseignement-apprentissage et est basé sur l'effet de l'influence d'un professeur sur la production des élèves dans une classe. L'algorithme décrit deux modes de base de l'apprentissage : (i) par l'intermédiaire du professeur, connu sous le nom de phase de professeur (teacher phase) et (ii) par l'interaction avec les autres élèves, phase dite des élèves (learner phase). Dans cet algorithme d'optimisation, un groupe des

élèves est considéré comme population et les différentes matières proposées aux élèves sont considérés comme étant les variables du problème d'optimisation (les paramètres impliqués dans la fonction objectif) et le résultat d'un élève est considéré comme étant la valeur de la fonction fitness (fonction d'évaluation). La meilleure solution dans l'ensemble de la population, qui correspond à la meilleure valeur de la fonction objectif, est affectée au professeur.

Principe de l'algorithme TLBO :

Dans tous les problèmes d'optimisation il y a un certain nombre de paramètres à ajuster. Ces paramètres à ajuster dans la méthode TLBO sont constituées par les matières enseignées aux élèves et le résultat des élèves est la valeur de la fitness du problème d'optimisation. Cet algorithme est basé sur un groupe d'élèves constituant la population dans la méthode TLBO. On distingue deux phases dans la méthode TLBO : phase professeur (apprentissage par le professeur) et phase élèves (apprentissage par interaction d'élèves entre eux). Les sous-sections ci-dessous expliquent le principe de la méthode TLBO [14].

Initialisation :

m : Nombre des matières enseignées aux élèves.
n : Nombre des élèves dans la classe.
i : Nombre des itérations.
k : Variation de la population.($k = 1,2,\ldots,n$).
j : Variation des variables de la conception. ($j = 1,2,\ldots,m$).
$M_{j,i}$: Résultat moyen de la classe dans le module particulier j.
$X_{totale\_kbest,i}$: Résultat du meilleur élève (le majorant).
La population est initialisée aléatoirement dans l'espace de recherche en utilisant l'équation suivante

$$X_{(j,i)} = X^{min}_{\ j} + \text{rand } X\ (X^{min}_{\ j} - X^{max}_{\ j}) \qquad (2.1)$$

Ou $X^{min}_{\ j}$ et $X^{max}_{\ j}$ sont les valeurs minimales et maximales des variables de conception.

Phase de l'enseignant (Teacher) :

C'est la première partie de l'algorithme d'optimisation. Pendant cette phase les élèves apprennent à travers le professeur, donc le professeur essai d'améliorer la moyenne de la classe dans les matières enseignées par lui. Le meilleur apprenant identifié est considéré par l'algorithme comme le professeur. La différence entre le résultat du professeur pour chaque matière et le résultat moyen existant de chaque matière est donné par :

$$Difference\_Mean_{j,k,i} = r_i\ (X_{j,kbest,i} - T_F M_{j,i}) \qquad (2.2)$$

Où, $X_{j,kbest,i}$ est le résultat du meilleur élève dans la matière j.

r_i : Nombre aléatoire entre 0 et 1.

T_F : Est le facteur d'enseignement qui décide la valeur de la moyenne à échangé, sa valeur est choisie aléatoirement, elle peut prendre les valeurs 1ou 2. L'équation suivante est utilisée pour générer T_F :

$$T_F = round[1 + rand(0,1)\{2-1\}] \quad (2.3)$$

Sur la base de $Difference\_Mean_{j,k,i}$, la solution existante est mise à jour dans la phase de professeur selon l'expression suivante :

$$X'_{j,k,i} = X_{j,k,i} + Difference\_Mean_{j,k,i} \quad (2.4)$$

Où, $X'_{j,k,i}$ est l la valeur mise à jour de $X_{j,k,i}$

$X'_{j,k,i}$ est accepté s'il nous donne la meilleure valeur de la fonction à optimiser. Ces valeurs deviennent l'entrée de la phase d'élèves. La phase d'élèves dépend de la phase de professeur.

Phase des élèves (Learners) :

C'est la deuxième partie de l'algorithme où les élèves augmentent leurs connaissances par interaction entre eux. Un élève interagit de manière aléatoire avec d'autres élèves pour améliorer ses connaissances.

Considérant une taille de population de N, le phénomène d'apprentissage de cette phase est expliqué ci-dessous :

Choisis aléatoirement deux élèves $p\ et\ Q$ tels que $X'_{total-p,i} \neq X'_{total-Q,i}$ (Où, $X"_{total-p,i}$ et $X"_{total-Q,i}$ sont les valeurs mises à jour de $X'_{total-p,i}$ et $X'_{total-Q,i}$ de p et Q respectivement à la fin de la phase professeur).

$$X"_{j,p,i} = X'_{j,p,i} + r_i\,(X'_{j,p,i} - X'_{j,Q,i}), \text{ if } X'_{total-p,i} < X'_{total-Q,i} \quad (2.5)$$

$$X"_{j,p,i} = X'_{j,p,i} + r_i\,(X'_{j,Q,i} - X'_{j,p,i}), \text{ if } X'_{total-Q,i} < X'_{total-p,i} \quad (2.6)$$

$X"_{j,p,i}$ est accepté s'il nous donne la meilleure valeur de la fonction à optimiser.

Les équations (5) et (6) sont utilisées pour des problèmes de minimisation. Dans le cas des problèmes de maximisation, les équations. (7) et (8) sont utilisées.

$$X"_{j,p,i} = X'_{j,p,i} + r_i\,(X'_{j,Q,i} - X'_{j,p,i}), \text{ if } X'_{total-Q,i} < X'_{total-p,i} \quad (2.7)$$

$$X"_{j,p,i} = X'_{j,p,i} + r_i\,(X'_{j,p,i} - X'_{j,Q,i}), \text{ if } X'_{total-p,i} < X'_{total-Q,i} \quad (2.8)$$

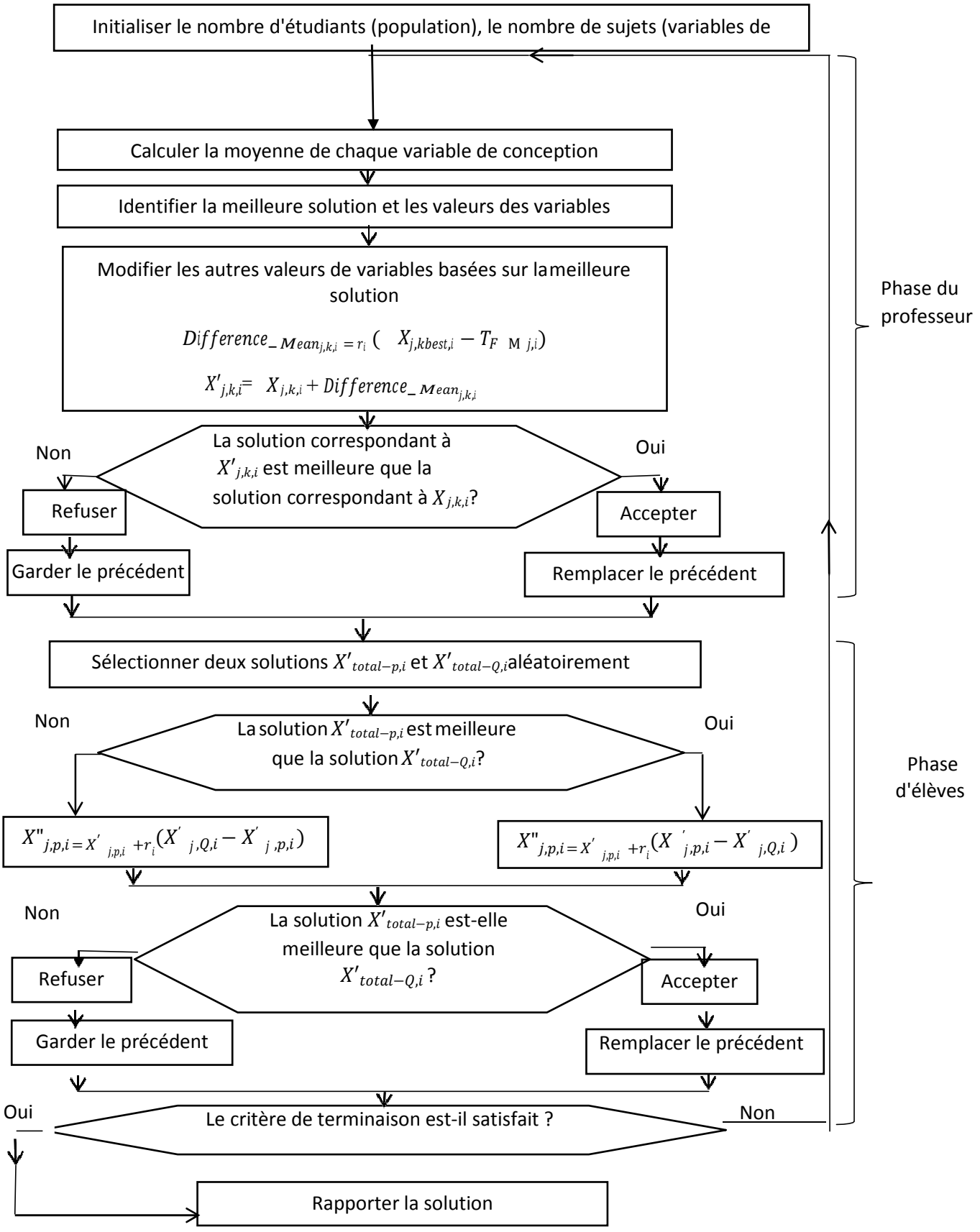


Fig 2.1 Organigramme d'algorithme TLBO

Exemples d'optimisation en utilisant la méthode TLBO

Dans cette partie, pour montrer l'efficacité de l'algorithme TLBO, nous avons utilisé le logiciel Matlab pour coder utilisons l'algorithme, ensuite nous avons considéré l'optimisation de plusieurs fonctions mathématiques. Les fonctions que nous avons considéré sont les suivantes :

$f_1(x) = x^2$ Min $(f_1) = 0$ pour $x = 0$ (figure 2.2)

$f_2(x) = [100(x^2 - x)^2 + (1 - x)^2]$ Min $(f_2) = 0$ *pour* $x = 1$ (figure 2.3)

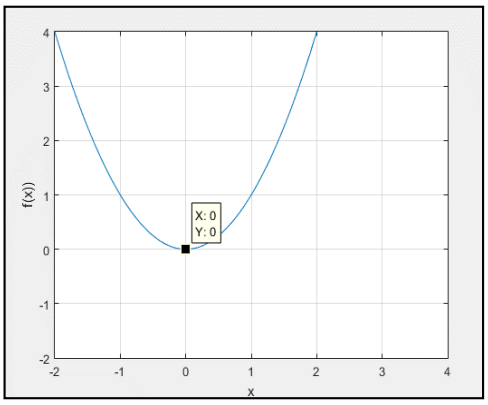


Fig 2.2 courbe de la fonction coût $f1$

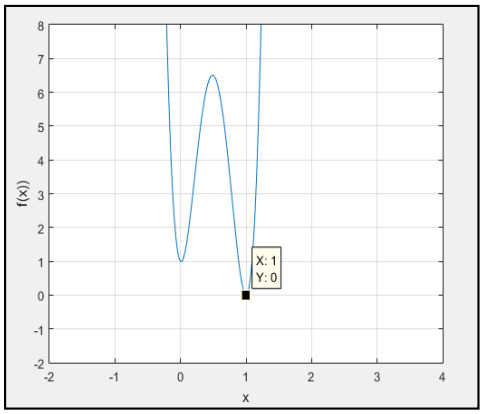


Fig 2.3 courbe de la fonction coût $f2$

Les tableaux suivants illustrent les résultats de l'application de la méthode TLBO pour trouver le minimum global des deux fonctions ($f1$) et ($f2$). Dans le premier cas nous avons fixé le nombre d'itérations à 20 itérations et fait varier la taille de la population (tableau 2.1). Dans le deuxième cas, nous avons fixé la taille de la population à 5 étudiants et fait varier le nombre d'itérations (tableau 2.2).

Population	**3**		**5**		**7**	
Fonction	$f1$		$f1$	$f2$	$f1$	$f2$
F min	8.7185e -36	0.09899	4.8824e -36	9.7835e-7	5.8062e -39	9.8914e-10
Valeur de x	9.3373^{e} -17	0.102	2.2096e-18	0.88925	7.6198e-20	0.90183

Tableau 2.1 Résultats d'optimisation en TLBO pour plusieurs valeurs de la population

Itérations	5		15		25	
Fonction	$f1$	$f2$	$f1$	$f2$	$f1$	$f2$
F min	5.8834e -36	9.8891 e-8	5.8062e -39	9.9183e -11	2.3821e -43	9.99e -14
Valeur de x	3.2098e-18	0.90512	7.6198e -20	0.99985	4.8807e -22	1.000

Tableau 2.2 Résultats d'optimisation en TLBO pour plusieurs valeurs du nombre maximal d'itérations

Les résultats obtenus, montrent qu'il y a une amélioration de la précision lorsqu'on augmente le

nombre d'itérations et la taille de la population.

4 Métaheuristiques hybrides

La majorité des métaheuristiques publiées dans la littérature sont d'une façon ou d'une autre hybrides. Le théorème du no free lunch (Wolpert et Macready en 1997) [59] indique qu'une métaheuristique ne peut prétendre être plus efficace qu'une autre sur tous les problèmes possibles. Néanmoins, en l'implémentant et en la paramétrant d'une certaine façon, elle peut être plus adaptée à certaines classes de problèmes.

Le principe donc des métaheuristiques hybrides est de combiner des métaheuristiques avec d'autres techniques d'optimisation. Ces techniques peuvent être d'autres métaheuristiques, des heuristiques, des méthodes exactes, etc. dans le but de créer des algorithmes plus performants. Plusieurs manières d'hybrider les métaheuristiques peuvent être envisagées selon la nature du problème. Le premier livre consacré entièrement au sujet d'hybridation des métaheuristiques a été publié par Blum et Roli en 2008 [60].

4.1 Classification hiérarchique des méthodes hybrides

La Classification hiérarchique des méthodes hybrides a été proposée par Jourdan et al. [61]. Elle les classe selon le niveau qui peut être soit bas (Low-Level) soit haut (HighLevel) et le mode de l'hybridation qui peut être soit le mode relais ou le mode co-évolutionnaire.

Dans le niveau bas, une métaheuristique remplace un opérateur d'une autre méthode qui l'englobe. Par contre, dans le niveau haut de l'hybridation, chaque métaheuristique garde sa structure entière et fonctionne de manière indépendante.

L'hybridation à bas niveau a pour but d'utiliser les propriétés de diversification de la métaheuristique de base et d'intensifier la recherche avec l'hybridation embarquée.

L'hybridation à haut niveau consiste à combiner une métaheuristique avec d'autres méthodes sans que leurs fonctionnements internes ne soient en relation. Le but est d'exécuter une séquence de méthodes.

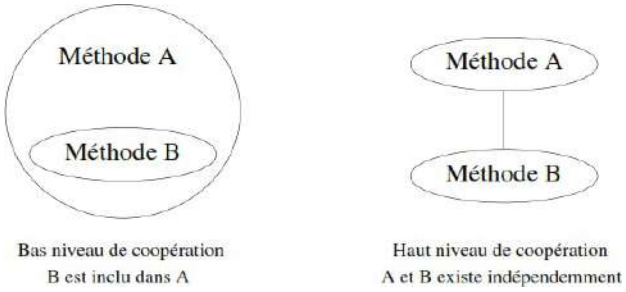


Fig 2.4 – Les différents niveaux d'hybridation

On dit que l'hybridation est séquentielle si les méthodes hybridées s'exécutent l'une après l'autre et le résultat de la méthode exécutée est communiqué (comme entrée) à la méthode qui s'exécute juste après. Quand les différentes méthodes fonctionnent en parallèle pour explorer l'espace de recherche, on parle de mode co-évolutionnaire. Ces classes sont comme suit :

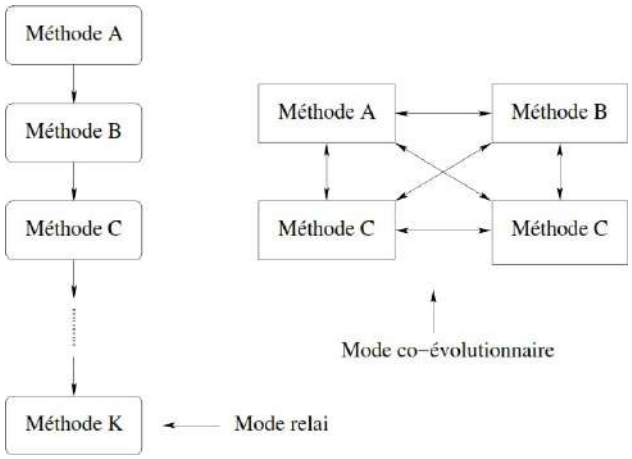


Fig 2.5 – Les différents modes d'hybridation

4.1.1 L'hybridation relais de bas niveau (BNR)

Low-level Relay Hybrid method (LRH), Dans cette classe on regroupe les méthodes hybrides qui sont formées par un Algorithme maitre et un algorithme esclave c'est à dire une des méthodes sera la principale et l'autre sera incorporée dans cette dernière. Le fonctionnement donc la méthode principale dépend de la méthode incorporée qui doit lui communiquer son résultat pour pouvoir fonctionner. Comme exemple on peut citer celui de Martin et Otto [72] qui ont inséré la méthode de descente dans un algorithme de recuit simulé.

4.1.2 L'hybridation co-évolutionnaire de bas niveau (BNC)

Low-level Teamwork Hybrid methode (LTH). Dans cette classe la méthode incorporée doit pouvoir être exécutée en parallèle avec la méthode globale. L'avantage de ce type d'hybridation est de compenser la puissance d'exploitation d'une recherche locale et celle d'exploration d'une recherche globale. Stûtzle et Hoos [63] incorporent une fonction de recherche locale dans un algorithme de colonie de fourmis pour résoudre le problème du voyageur de commerce et celui de partition de graphes. Cotta et al [64] ont incorporée une méthode de *branch and bound* dans un algorithme génétique, la méthode de *branch and bound* a joué le rôle d'un opérateur de croissement.

4.1.3 L'hybridation relais de haut niveau (HNR)

High-level Relay Hybrid (HRH). Souvent utilisée et consiste à hybrider des métaheuristiques qui

fonctionnent de manière séquentielle c'est-à-dire la ou (les) solution(s) finale(s) de la première métaheuristique représente la ou (les) solution(s) initiale(s) de la métaheuristique suivante. Dans cette procédure, toutes les méthodes gardent leur intégrité. Dans cette classe on retrouve plusieurs méthodes dans la littérature notamment celles qui hybrident une métaheuristique à population avec une métaheuristique à solution unique, ou même l'utilisation d'une heuristique pour la construction de solution de démarrage pour une métaheuristique à solution unique. Lin, Kao et al. [65] ont proposé une hybridation où la méthode du recuit simulé crée une population initiale pour un algorithme génétique.

4.1.4 L'hybridation co-évolutionnaire de haut niveau (HCH)

High level co-evolutionary Hybrid Method, cette classe contient des algorithmes formés d'hybridation de méthodes qui ne sont pas incorporées les unes dans des autres, et qui fonctionnent en parallèles. V. Nwana et al. [66] ont proposé une méthode hybride formée d'un algorithme de recuit simulé et de la méthode de *branch and bound* où les deux méthodes hybridées fonctionnent en parallèle.

5 Conclusion

Dans ce chapitre quelques approches métaheuristiques à solution unique et à population de solution ont été présentées, une description détaillée de ces algorithmes avec leurs pseudo codes ont été dévoilées, enfin l'algorithme de TLBO originale a été expliqué en détail pour l'adapté à notre solution qui sera proposé pour résoudre le problème de CVRP dans le chapitre suivant.

CHAPITRE III

Adaptation de l'algorithme TLBO pour la résolution du problème de tournées de véhicules

1 Introduction

Dans ce chapitre, nous allons présenter notre algorithme TLBO proposé pour la résolution approchée du problème de CVRP. Nous commençons par présenter deux travaux récents dans la littérature qui ont une relation avec notre travail. Par la suite, nous décrivons les différents éléments de l'algorithme TLBO proposé. Enfin, des prises d'écran avec explication sur le fonctionnement de l'algorithme, une étude expérimentale et des comparaisons des résultats obtenus avec la littérature seront élaborés.

2 Deux travaux récents

2.1 Enhanced Intelligent Water Drops algorithm for solving the Capacitated Vehicle Routing Problem (CVRP)

Shah-Hosseini a proposé un algorithme métaheuristique basé sur la population inspirée du comportement des rivières, appelé *Intelligent Water Drops* (IWD). Depuis lors, l'algorithme IWD a été appliqué à de nombreux problèmes d'optimisation combinatoire tels que TSP, sac-à-dos multiple, routage de véhicule et planification d'atelier, où le résultat était impressionnant.

L'hypothèse de base de l'algorithme IWD est que la solution d'un problème peut être présentée sous forme d'un graphe. En d'autres termes, il existe différents chemins possibles d'une source à une destination et le chemin le plus court est préféré. La structure du graphe dépend du problème à résoudre. En outre, il existe des gouttes d'eau intelligentes (IWD) qui peuvent se déplacer le long des chemins de graphe. Chaque IWD contient une certaine quantité de sable qui indique la dureté du chemin. Le chemin avec une plus petite quantité de sable est préféré. Il y a une interaction entre les quantités de sable de chaque IWD et les chemins qu'elle traverse, c'est-à-dire que les IWD peuvent charger/décharger les sables sur les chemins en fonction des circonstances, telles que la vitesse de IWDs.

L'algorithme IWD est un algorithme constructif et itératif qui commence par des gouttes d'eau intelligentes IWD, dont chacune crée un chemin qui représente une solution du problème traité. Chaque IWD commence par former une source, choisit un chemin, puis se déplace le long de ce chemin vers le prochain nœud non visité. Durant ce mouvement, la goutte d'eau charge/décharge le sol et sa vitesse peut changer. Ce processus se poursuit jusqu'à ce qu'un chemin complet de la source à la destination soit créé. La quantité de sable qu'un IWD peut ramasser est proportionnelle

au temps nécessaire pour parcourir le chemin et ce temps est calculé en fonction de la loi physique. Lorsqu'un chemin est utilisé par plus d'une IWD, le chemin aura moins de terre ; par conséquent, ce chemin sera préféré. Le mécanisme de choix du chemin suivant à partir du nœud actuel est similaire au mécanisme de roue de roulette dans l'algorithme génétique (GA) ; cependant, ici, cela dépend du sol des sentiers non visités. Lorsque toutes les gouttes ont terminé leur chemin, une itération de l'algorithme est terminée. Après chaque itération, le sol des meilleures solutions ou chemins, *Se*, appelés solutions élites, est mis à jour. Ce processus est appelé propagation globale du sol. En outre, la mise à jour des meilleures solutions trouvées jusqu'à présent est appelée Global Best Solution (GBS) dans cette étude.

L'algorithme IWD a deux types de paramètres : statique et dynamique [74]. Les paramètres statiques, tels que *la vitesse initiale des IWD* et *le sol initial des chemins*, sont constants dans toutes les itérations de l'algorithme tandis que les paramètres dynamiques, tels que *les nœuds visités*, doivent être réinitialisés à chaque itération.

Après l'achèvement d'une itération, les paramètres dynamiques seront réinitialisés et le processus ci-dessus se poursuivra jusqu'à ce que la condition d'arrêt, soit le nombre maximum d'itérations de l'algorithme (*MaxItr*), soit *le temps de calcul* soit satisfait.

Un algorithme amélioré de gouttes d'eau intelligentes (IIWD) est proposé pour résoudre le problème de routage de véhicule capacitif (CVRP) [74], dans lequel une flotte de véhicules devrait servir un ensemble prédéfini de clients avec des demandes connues. L'objectif est d'affecter et de séquencer les clients dans le circuit de chaque véhicule de sorte que le coût total encouru par rapport aux limites de capacité soit minimisé. Afin d'appliquer l'algorithme IIWD à CVRP, le structure de graphe $G(N, E)$ doit d'abord être défini. Il existe n nœuds (clients) où chaque paire de nœuds est connectée via une arête. Par conséquent, il y a $n\ (n-1)\ /\ 2$ arêtes qui présentent un graphe complet.

2.2 Cuckoo search algorithm for solving the Capacitated Vehicle Routing Problem (CVRP)

Comme les coucous ont de beaux sons et qu'ils exploitent une stratégie de reproduction agressive, ce sont des oiseaux attrayants. Parmi les nombreuses caractéristiques fascinantes des espèces de coucous, un attribut frappant des coucous est que certaines espèces se livrent à un soi-disant «parasitisme du couvain», car elles pondent leurs œufs dans les nids d'autres espèces de sorte que les oiseaux hôtes éclosent et couvent de jeunes poussins de coucou. Afin de repousser les coucous intrus, certains oiseaux hôtes s'engagent dans un conflit direct. Si un oiseau hôte découvre que les

œufs ne sont pas les siens, il réagit de deux manières: (1) soit il jette les œufs étranges, soit (2) il abandonne simplement son nid et en fabrique un nouveau ailleurs. Certaines espèces de coucous ont évolué de sorte que les coucous parasites femelles pondent des œufs souvent très spécialisés pour l'imitation de la couleur et du motif de quelques oiseaux hôtes sélectionnés. Cette adaptation diminue le risque que leurs œufs soient jetés et améliore en conséquence la reproduction du coucou. De plus, le moment de la ponte de certaines espèces de coucous s'est également adapté. C'est-à-dire que les coucous parasites optent fréquemment pour un nid dans lequel les oiseaux hôtes pondent simplement leurs propres œufs. Dans l'ensemble, les œufs des coucous éclosent un peu plus tôt que les œufs hôtes. Dès l'éclosion du premier poussin coucou, sa première action instinctive est d'expulser les œufs de l'hôte en poussant aveuglément les œufs hors du nid. Cela augmente la part de nourriture du poussin coucou fournie par son oiseau hôte. Pour obtenir plus de nourriture, un poussin coucou est également capable d'imiter l'appel des poussins hôtes.

L'algorithme Cuckoo search est basé sur:

Il y a trois principes idéalisés que nous allons décrire afin de faciliter les concepts d'algorithme de recherche de coucou comme suit:

(a) Chaque coucou pond un œuf à la fois et s'efforce de le jeter dans un nid choisi au hasard. En d'autres termes, puisque chaque œuf est situé dans un nid, cela équivaut à une solution.

(b) Afin de préserver l'élitisme, les meilleurs nids avec des œufs (solutions) de la plus haute qualité passeront à la génération suivante.

(c) L'œuf pondu par un coucou peut être découvert via l'oiseau hôte avec une probabilité Pa et le nombre de nids d'hôtes disponibles est donné et fixé. Dans cette condition, l'œuf pourrait être jeté par l'oiseau hôte ou le nid est abandonné par l'oiseau hôte afin de construire un nid entièrement nouveau dans un nouveau site.

La dernière hypothèse mentionnée ci-dessus pourrait être considérée d'une manière plus facile qui pourrait être approchée par une fraction Pa des n nids étant substituée au moyen de nouveaux nids avec de nouvelles solutions aléatoires à de nouvelles positions. Différentes représentations pourraient également être appliquées pour montrer la valeur de la fonction de fitness. Par exemple, dans les problèmes de minimisation, la valeur inverse d'une fonction objectif peut être considérée comme la valeur d'aptitude. Comme déjà souligné, puisque chaque œuf est situé dans un nid, cela équivaut à une solution, et un œuf de coucou signifie une nouvelle solution dans le but d'utiliser les nouveaux et potentiellement meilleurs coucous (solutions) dans les nids. De cette manière, les termes «œuf», «nid» et «solution» ont le même sens.

Un vol de Lévy est effectué chaque fois qu'une nouvelle solution $x\ (t\ +\ 1)$ pour coucou *i* doit être

générée comme suit :

$$xi(t+1) = xi(t) + \alpha \oplus Lévy(s,\delta)$$

Le CS de base est un algorithme polyvalent avec peu de paramètres; cependant, il est possible de l'améliorer et de l'ajuster, en particulier face à des problèmes discrets combinatoires *NP*-difficile. De la même manière, Ehsan Teymourian, Vahid Kayvanfar, GH.M.Komaki et M.Zandieh ont présenté un algorithme CS avancé pour résoudre le CVRP avec des performances compétentes [74]. Pour ce faire, voici quelques améliorations apportées au CS classique: envisager un nouveau groupe de coucous pour effectuer une recherche locale intelligente, régler dynamiquement les paramètres du CS, les mouvements adoptés et le vol de Lévy en utilisant des structures de voisinage concernant la nature du CVRP et du CS discret. Ces changements accompagnent d'autres ajustements des principaux éléments du CS pour faire face au CVRP.

3 L'apprentissage compréhensif

L'apprentissage compréhensif (*Comprehensive Learning*) PSO (CLPSO) propose par Liang et al[77], pour améliorer le PSO continu. Dans cette variante la meilleur position personnel($Pbest$) de toutes les autres particules peut être utilisé pour mettre à jour le vecteur d'exemplaire d'une particule, il est montré que cette stratégie d'apprentissage garde la diversité dans l'essaim et minimise la convergence prématuré vers une région sous optimale de l'espace de recherche.

Dans CLPSO, toute particule met à jour son vecteur d'exemplaire pour chaque dimension, que ce soit selon son $Pbest$ personnel ou selon le $Pbest$ des autres particules. La décision de choisir un ou un autre $Pbest$ dépend de la probabilité d'apprentissage *Pc* (*Learning Probability*) de la particule.

La valeur Pc_i de chaque particule *i* est définit selon l'expression empirique suivante :

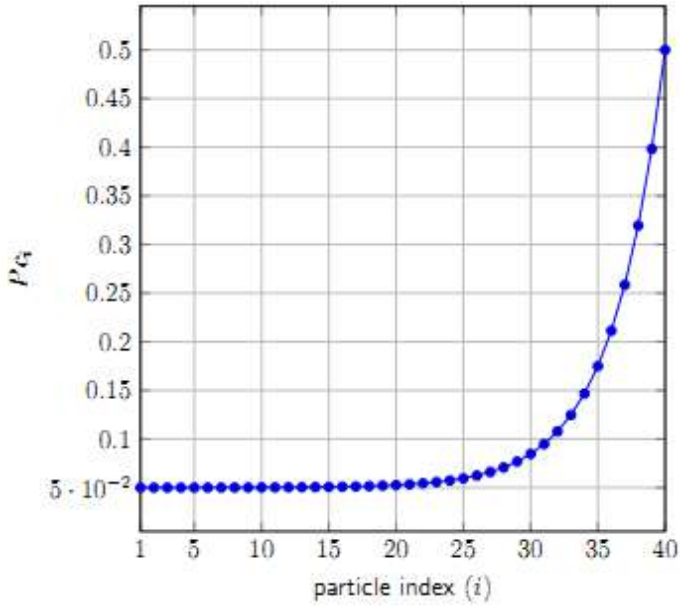
$$Pc_i = Pc_{min} + (Pc_{max} - Pc_{min})\frac{\exp\frac{10(i-1)}{m-1} - 1}{\exp(10) - 1} \qquad (3.1)$$

Où Pc_{min} et Pc_{max} correspond respectivement aux valeurs permissibles minimum et maximum de Pc. Des bonnes performances ont été obtenues en fixant Pc_{min} à 0.05 et Pc_{max} à 0.5 (3.1), des petites valeurs de Pc encouragent une forte aptitude d'exploration (la particule est attirée par son propre succès), alors que les grandes valeurs suggèrent une forte aptitude d'exploration (la particule suit la meilleure expérience de recherche des autres particules).

Dans CLPSO, chaque particule i utilise un vecteur d'exemplaire $E_i=(E_{i,1}, E_{i,2}, \ldots E_{i,n})$, chaque élément $1 \leq E_{i,j} < m$ donne quel $Pbest$ (index) de l'essaim de particule *i* qui doit suivre dans la dimension *j*, E_i est assigné comme suit :

Pour chaque dimension j, un nombre aléatoire P est généré initialement entre [0,1],
Si P est supérieur ou égale à Pci alors Ei,j est affecté à i, sinon elle est affectée à l'index $k(k \neq i)$ du $Pbest$ vainqueur du tournoi de sélection entre deux particules choisis aléatoirement. Pour assurer une bonne convergence des propriétés, chaque particule réaffecte toutes ces exemplaires si son $Pbest$ n'est pas amélioré pour certain nombre d'itérations successives, qui s'appelé (*refreshing gap*) g, avec cette stratégie d'apprentissage le vecteur d'exemplaire d'une particule i dans une dimension j est met à jour comme suit :

$$V_{i,j}\ (t+1) = V_{i,j}(t) + r_j.c.(pbest_{Ei,j}(t) - X_{i,j}(t)) \tag{3.2}$$



Pc_{min} = 0.05, et Pc_{max} = 0.5.

Fig 3.1 – Valeurs de probabilités d'apprentissage (Pc) pour un groupe de 40 individus.

4 L'algorithme TLBO Proposé

L'algorithme TLBO est une métaheuristique de population de solution relativement nouvelle, proposée à l'origine par Rao et al. [56, 57] pour les problèmes d'optimisation continue. Il imite le processus classique de transmission des connaissances en classe, de l'enseignant aux étudiants et entre étudiants eux mêmes. Depuis sa création en 2011, TLBO a été efficacement étendu et appliqué pour résoudre une variété de problèmes d'optimisation continue et combinatoire.

Dans TLBO, les individus (X_i, i = 1,2, ..., m) au sein de la population (P) sont considérés comme des *apprenants* ou des *étudiants*. Ils représentent des solutions potentielles (vecteurs réels dans le

TLBO original) dans l'espace de recherche du problème, $X_i = (X_{i,1}, X_{i,2}, ..., X_{i,n})$ où n est le nombre de variables de décision. A chaque itération de TLBO, le meilleur apprenant selon la fonction d'évaluation (*fitness*) joue le rôle du professeur (X^*) de tous les autres apprenants. Après la génération d'une première classe d'apprenants, le processus de recherche itérative de TLBO se compose de deux étapes: *la phase d'enseignement* et *la phase d'apprentissage*.

Pendant la phase d'enseignement, les connaissances sont transférées de l'enseignant à ses étudiants, tandis que pendant la phase d'apprentissage des étudiants aux étudiants eux-mêmes.

Un nouvel ensemble (classe) d'apprenants est généré au cours de chaque phase via des opérateurs de variation suivi d'une étape de remplacement. Dans les deux phases, chaque apprenant nouvellement généré (X'_i) remplace l'apprenant actuel (X_i) dans la classe s'il a une meilleure valeur de fitness. Les deux phases de recherche sont répétées jusqu'à ce qu'un nombre maximum d'itérations soit atteint.

La structure de base de l'algorithme TLBO est illustrée à la figure 3.2. Les opérateurs de variation utilisés pour la génération de nouveaux apprenants sont décrits ci-dessous.

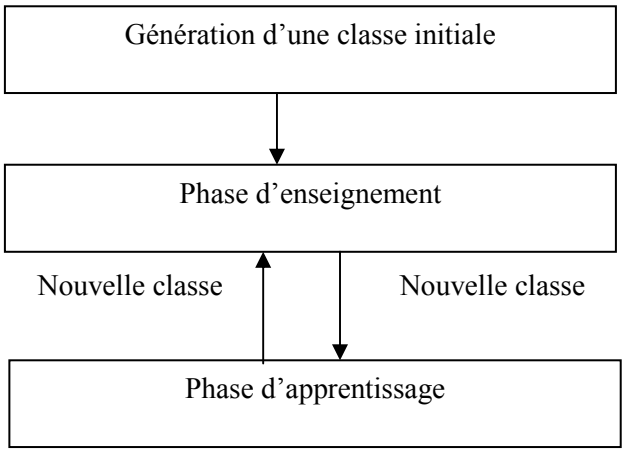


Fig 3.2 : Forme Générique de l'algorithme TLBO

4.1 Les éléments de l'algorithme TLBO proposé

4.1.1 Structure de l'enseignant (Teacher)

Au cours de cette phase, les apprenants reçoivent des connaissances de leur enseignant. Un nouvel apprenant (X'_i) est généré en recombinant l'apprenant actuel (X_i) avec la différence de vecteur de position entre l'enseignant (X^*) et la moyenne actuelle de tous les apprenants (M). Ce vecteur moyen et obtenu en prenant la valeur moyenne de chaque variable de décision. Il juge la qualité des apprenants existants dans la classe. La mise à jour de l'apprenant i sur chaque dimension j se fait selon l'expression suivante:

$$X'_{i,j} = X_{i,j} + r_j \cdot (X^*_j - TF_j \cdot M_j) \quad \text{.....(3.3)}$$

où r_j est un nombre aléatoire généré uniformément pour chaque dimension à partir de [0,1], et TF_j est un facteur d'enseignement qui prend aléatoirement la valeur 1 ou 2 avec une probabilité égale. TF_j décide le pourcentage de changement qui sera appliqué à la valeur moyenne.

4.1.2 Structure de l'apprenant (Learner)

Pendant la phase d'apprentissage, chaque apprenant (X_i) essaie d'augmenter son niveau de connaissances grâce à l'interaction avec un apprenant (X_k) sélectionné au hasard dans la classe, où k ≠ i. Si X_k a une meilleure valeur de fitness que X_i, alors X_i est déplacé vers X_k (Eq 3.4), sinon, il est déplacé dans la direction opposée (Eq 3.5).

$$X'_{i,j} = X_{i\,j} + r_j \cdot (X_{k,j} - X_{i,j}) \quad \text{..... (3.4)}$$

$$X'_{i,j} = X_{i,j} + r_j \cdot (X_{i,j} - X_{k,,j}) \quad \text{..... (3.5)}$$

Lors de la génération d'un nouvel apprenant à l'aide des équations. (3.4) et (3.5), des dimensions différentes ont des valeurs différentes de $r_j \in [0,1]$.

4.2 Procédure de la construction de nouvelle solution

Pour générer des nouvelles solutions pour le problème de tournées de véhicules avec capacité (CVRP) par l'application de notre algorithme proposé, nous avons opté pour le choix de la stratégie de l'apprentissage compréhensif précédemment décrite [75], pour le remplissage des vecteurs d'exemplaires dans les deux phases de l'algorithme TLBO proposé. Dans les deux phases de l'algorithme, nous avons défini la méthode de construction d'une nouvelle solution, soit entre l'apprenant et l'enseignant et entre l'apprenant et ses camarades.

La procédure est comme suit : à partir d'une solution vide, on fait le choix d'un apprenant de manière aléatoire, on cherche dans son voisinage soit l'enseignant ou le vecteur d'exemplaire par l'application de la roulette, on vérifié les contraintes de la route, si elles sont vérifiées on allant vers le site suivant, et ainsi de suite jusqu'à l'obtention d'une solution complète qui contient plusieurs routes, et chaque route a une distance et une capacité.

Un exemple de procédure de la construction de nouvelle solution

Initialisation des paramètres de l'exemple

Commande de chaque client

N client (ville)	1	2	3	4	5	6	7	8	9	10
Commande client	22	13	10	20	31	35	15	08	24	11

i, j ∈ {1,…, n}, et n =10. La Capacité de véhicule Q = 80,

le nombre de véhicule m = 3.

Teacher phase

Teacher : X*

3	8	2	4	9	0	1	5	7	0	6	10	0

Learner X_i

5	6	8	0	9	1	4	10	0	2	3	7	0

Construction des solutions Xi '

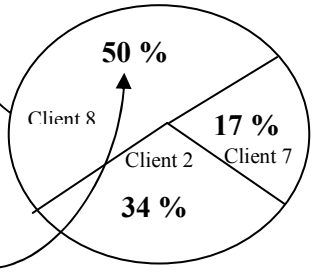
X_i '

3	8											

{8 , 2, 7 },

r_i =Random [0,1]()

La roulette



supposant que r_i =0.29

[1-2-3] ----------→ [1/6- 2/6 -3/6]

|---------0.50---------|----0.33----|--0.17--|

0 Client 8 0.50 Client 2 0.83 C7 1.0

La valeur r_i = 0.29 est situe a l'intervalle [0.00 à 0.50], donc on va choisir le client **8**

Etape suivante : vérification de la satisfaction des contraintes de la route

La somme des commandes des clients 8 et 3 égale 18 est < 80 donc on passe au choix d'autres clients. Il y a deux client {2, 6}, on applique la fonction de la roulette pour le choix d'un client, sachant que le client 2 occupe 2/3 (67 %) de la roulette, et le client 6 occupe 1/3 (33 %) de la

roulette. Supposant que r_i =0.86, donc on choisit le client 6, et ainsi de suit jusqu'à la construction d'une solution complète X_i' qui contient plusieurs routes et tous les clients sont servis.
On va vérifie la valeur de fitness de X_i' par rapport X_i , si la fitness de X_i' est meilleur on écrase X_i, et la remplacé par X_i', sinon on garde X_i et passer un autre apprenant (learner).

Learner phase :

Dans cette phase la même procédure décrite précédemment est appliquée, avec la différence suivante : l'interaction entre les apprenants eux-mêmes se faite on associant un vecteur d'exemplaire pour chaque apprenant, le but de ce vecteur est que chaque apprenant compare ces connaissances avec un groupe d'apprenants au lieu d'un seul apprenant dans la même étape de construction d'une solution. A condition de ne pas dépassé la valeur de '*refreshing gap'* qui est égale à 7. C.-à-d., si l'apprenant, après l'interaction avec sept apprenants qui appartiennent à son vecteur d'exemplaire, n'a pas pu améliorer ces connaissances, on rafraîchit son vecteur d'exemplaire.

Learner X_i

3	**8**	**2**	**4**	**9**	**0**	**1**	**5**	**7**	**0**	**6**	**10**	**0**

Learner X_j sachant que *j* appartient au vecteur d'exemplaire de l'apprenant *i*

2	**5**	**9**	**0**	**8**	**3**	**7**	**0**	**10**	**6**	**4**	**1**	**0**

On applique la fonction de la roulette pour choisir un des clients 8 et 7, supposant que la roulette a choisit 7

Construction des solutions X_i'

X_i'

3	**7**											

Dans cette étape l'interaction se fait entre l'apprenant X_i et un autre apprenant X_k appartient a son vecteur d'exemplaire,

Learner X_i

3	8	2	4	9	0	1	5	7	0	6	10	0

Learner X_k tel que ***k*** appartient au vecteur d'exemplaires de l'apprenant ***i***

2	5	9	0	8	3	7	0	10	6	4	1	0

On applique la même procédure précédente, on choisit le client 5, car le client 3 déjà apparait dans la solution. Et ainsi de suite jusqu'à l'apparence de tous les clients dans la solution.

Construction des solutions X_i '

Xi '

3	7	5										

4.2.1 Affectation des exemplaires

Le vecteur d'exemplaires TE_i (resp. LE_i) définit pour chaque dimension quelle solution ($X_k, k = 1,2, ..., m$) l'apprenant i doit suivre pour générer une nouvelle solution pendant la phase d'enseignement (resp. phase d'apprentissage).

La probabilité d'enseignement Pt_i (resp. La probabilité d'apprentissage Pl_i) est utilisée pour choisir des exemplaires d'enseignement (resp. des exemplaires d'apprentissage) pour chaque dimension de l'apprenant.

L'exemplaire d'enseignant (TE) est le vecteur d'exemplaires pour acquérir des connaissances pendant la phase d'enseignement. Les éléments de ce vecteur sont soit l'index de l'apprenant lui-même, soit l'index du meilleur apprenant de la classe (l'enseignant).

L'exemplaire d'apprenant (LE) est le vecteur d'exemplaires pour acquérir des connaissances pendant la phase d'apprentissage. Les éléments de ce vecteur peuvent prendre l'index de n'importe quel apprenant de la classe, y compris l'apprenant lui-même.

Deux variables représentent la probabilité d'enseignement ($0 < Pt < 1$) et la probabilité d'apprentissage ($0 < Pl < 1$) de l'app*renant.* Pt *(r*esp. Pl) est utilisé pour choisir des exemplaires d'enseignement (resp. Des exemplaires d'apprentissage) pour chaque dimension de l'apprenant.

Exemplaires d'enseignant :

Dans cette section, tous les apprenants ont la même valeur de probabilité d'enseignement (Pt), c'est-à-dire que tous les apprenants ont une chance égale d'apprendre de leur enseignant. Les exemplaires d'enseignement TEi, j ($j = 1,2, \ldots, n$) de l'apprenant i sont choisis comme suit:

pour chaque dimension j, une valeur aléatoire ρ est générée entre 0 et 1. Si ρ est inférieur à Pt , $TE_{i,j}$correspond à l'indice du meilleur apprenant (enseignant) de la classe; sinon, il est mis à l'index i de l'apprenant. Pour permettre à l'apprenant d'apprendre de l'enseignant sur différentes dimensions, nous renouvelons son vecteur d'orientation LE_i lorsque son compteur de stagnation d'enseignement St_i atteint une limite prédéterminée α, appelée écart de rafraîchissement. L'algorithme 1 illustre l'algorithme associé à la sélection d'exemplaires pour l'apprenant i.

Algorithm 1: Assignment of teaching exemplars

input : Learner ID *i*
output: Vector of examplars TE_i
Let *idx_teacher* the index of the best learner in the class
for *j*=1 **to** *n* **do**
  $\rho \leftarrow rand[0.1]( )$
  If $\rho < Pt_i$ **then**
    $TE_{I,j} \leftarrow idx\_teacher$
  else
    $TE_{I,j} \leftarrow i$
  end
end

Exemplaires de l'apprenant :

Dans ce travail, nous adoptons la stratégie adaptative pour fixer les probabilités d'apprentissage (Pl) de tous les apprenants. Pour ce faire, à chaque itération de notre Algorithme TLBO, les apprenants doivent être triés en fonction de la valeur de fitness d'une façon décroissante. Les exemplaires d'apprentissage $LE_{i,j}$ $(j = 1,2, \ldots, n)$ de chaque apprenant i sont choisis comme suit :

Pour chaque dimension j, une valeur aléatoire ρ est d'abord choisie aléatoirement de [0,1]. Si ρ est supérieur à Pl_i, $LE_{i,j}$est fixé à l'indice i de l'apprenant; sinon, $LE_{i,j}$ est fixé à l'indice k du vainqueur entre deux apprenants sélectionnés au hasard. Pour inciter l'apprenant à apprendre à partir de différents vecteurs d'exemplaires sur différentes dimensions, son vecteur d'exemplaire TE_i est réaffecté lorsque son compteur de stagnation d'apprentissage Sl_i compte un nombre prédéterminé d'itérations α (*refreching gap* α est pris le même que pour la phase d'enseignement). L'algorithme 2 montre l'algorithme associé à la sélection d'exemplaires d'apprentissage.

Input : Learner ID *i*
Output: Vector of exemplars LE_i
for *i=1* **to** *n* **do**
  $p \leftarrow rand[0,1]()$
  if $p > Pl_i$ **then**
    $LE_{i,j} \leftarrow i$
  else
    /* Binary tournement */
    $l1 \leftarrow random[0..m]()$
    $l2 \leftarrow random[0..m]()$
    if *fitness* (X_{l1}) *is better than fitness* (X_{l2}) **then**
      $LE_{i,j} \leftarrow l1$
    else
      $LE_{i,j} \leftarrow l2$
    end
  end
end

4.3 Implémentation

Dans cette section, nous allons présenter le code des principales fonctions de notre application.

On a utilisé le logiciel de programmation RAD Studio EMBARCADERO, le langage de programmation C++ Builder 2010, exécuté dans une machine ayant les caractéristiques suivantes :

Processeur : Intel Core I5-3230M 2.6 GHz.

RAM :4 GBDDR3

Les principales procédures et fonctions utilisées dans notre implémentation sont décrites comme suit :

1- TLBO teaching phase method

```
Void teaching_phase(int t)
{
        double new_fitness;
        for (int i = 0; i < S; i++)
        {
                learner *l = Class[i];
                // re-assign exemplars (i.e., no change number == refreshing_gap)
                if (l->teaching_stagnation_cpt > refreshing_gap)
                {
                        set_exemplar_teaching(i);
                        l->teaching_stagnation_cpt = 0;
                }
                // create a new tour ...
                build_solution(l);

                // Local search application
                if (ls_flag && rand01(&seed) < 0.01) // Local search
                {
                        local_search(l);
                }
                // Evaluation of VRPSolution object
                l->solution->evaluation_fitness();
                new_fitness = l->solution->fitness;

                // replacement step ...
                if (new_fitness < l->fitness)
                {
                        // create the tour of VRPSsolution solution
                        l->create_tour();
                        // replace the fitness value
                        l->fitness = new_fitness;
                        // Update learning structures
                        l->set_city_position_in_learner();
                        set_exemplar_teaching(i);
                        l->teaching_stagnation_cpt = 0;
                        // marke a change
                        learner_changed = true;
                        if (new_fitness < fitness_teacher)
                        {
                                fitness_teacher = new_fitness;
                                idx_teacher = i;
                                Class[i]->solution->copySolution(teacher_VRPSolution);
                        }
                }
                else {
                        l->teaching_stagnation_cpt += 1;
                }
        }
}
```

2- TLBO learning phase method

```
void   learning_phase(int t)
{
        double new_fitness;
        for (int i = 0; i < S; i++)
        {
                learner *l = Class[i];

                // re-assign exemplars (i.e., no change number == refreshing_gap)
                if (l->learning_stagnation_cpt > refreshing_gap)
                {
                        set_exemplar_learning(i);
                        l->learning_stagnation_cpt = 0;
                }
                // create a new tour ...
                build_solution(l);

                // Local search application
                if (ls_flag && rand01(&seed) < 0.01) // Local search
                {
                        local_search(l);
                }
                // Evaluation of VRPSolution object
                l->solution->evaluation_fitness();
                new_fitness = l->solution->fitness;

                // replacement step ...
                if (new_fitness < l->fitness)
                {
                        // create the tour of VRPSsolution solution
                        l->create_tour();
                        // replace the fitness value
                        l->fitness = new_fitness;

                        // Update learning structures
                        l->set_city_position_in_learner();
                        set_exemplar_learning(i);
                        l->learning_stagnation_cpt = 0;
                        // marke a change
                        learner_changed = true;

                        if (new_fitness < fitness_teacher)
                        {
                                fitness_teacher = new_fitness;
                                idx_teacher = i;
                                Class[i]->solution->copySolution(teacher_VRPSolution);
                        }
                }
                else {
                        l->learning_stagnation_cpt += 1;
                }
        }
}
```

3- Local search method

```
void local_search(learner *l)
{
	Multi_Neigh_LS_IntraRoute *NN_LS_IntraRoute;
	NN_LS_IntraRoute = new Multi_Neigh_LS_IntraRoute(l->solution, n / 5);
	VRP_CROSS_Exchange *CROSS_Exchange;
	CROSS_Exchange = new VRP_CROSS_Exchange(l->solution, n / 5, 2);
	NN_LS_IntraRoute->execute(false);
	CROSS_Exchange->execute(false);
	NN_LS_IntraRoute->execute(false);
	delete NN_LS_IntraRoute;
	delete CROSS_Exchange;

}
```

4- TLBO iteration: Teaching and learning phases

```
void	udpate_class(int t)
{
	learner_changed = false;
	// TEACHING PHASE
	phase = 0;
	teaching_phase(t);
	// ranking the learners if there is a change
	if (learner_changed)
	{
		ranking_learners();
		assign_learning_probabilities();
	}
	// LEARNING PHASE
	phase = 1;
	learning_phase(t);

	learner_changed = false;
	// ranking the learners if there is a change
	if (learner_changed)
	{
		ranking_learners();
		assign_learning_probabilities();
	}
}
```

Prise d'écran de la fenêtre principale de notre programme

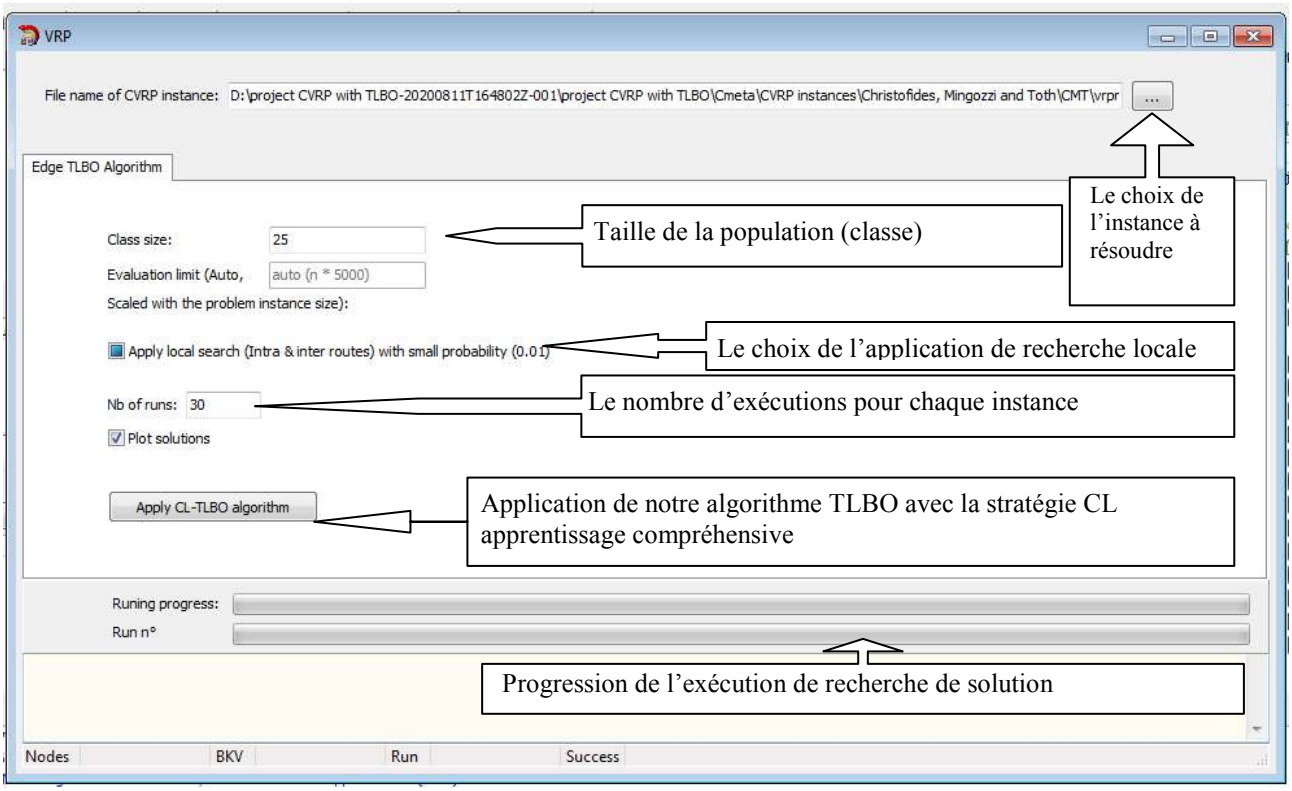


Fig 3.3 - fenêtre principale de notre programme

Prise d'écran de progression de recherche d'une solution optimale parmi les solutions trouvées

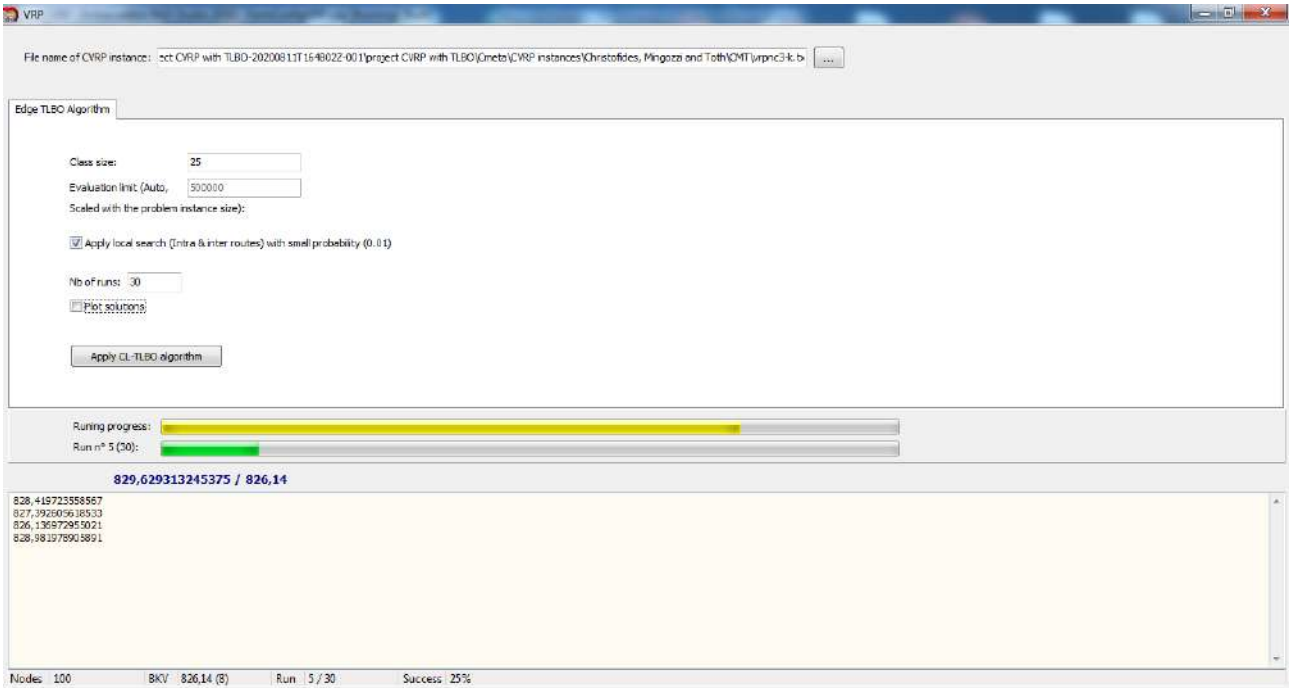


Fig 3.4 - progression de recherche d'une solution optimale

Plots ou Graphe de solution

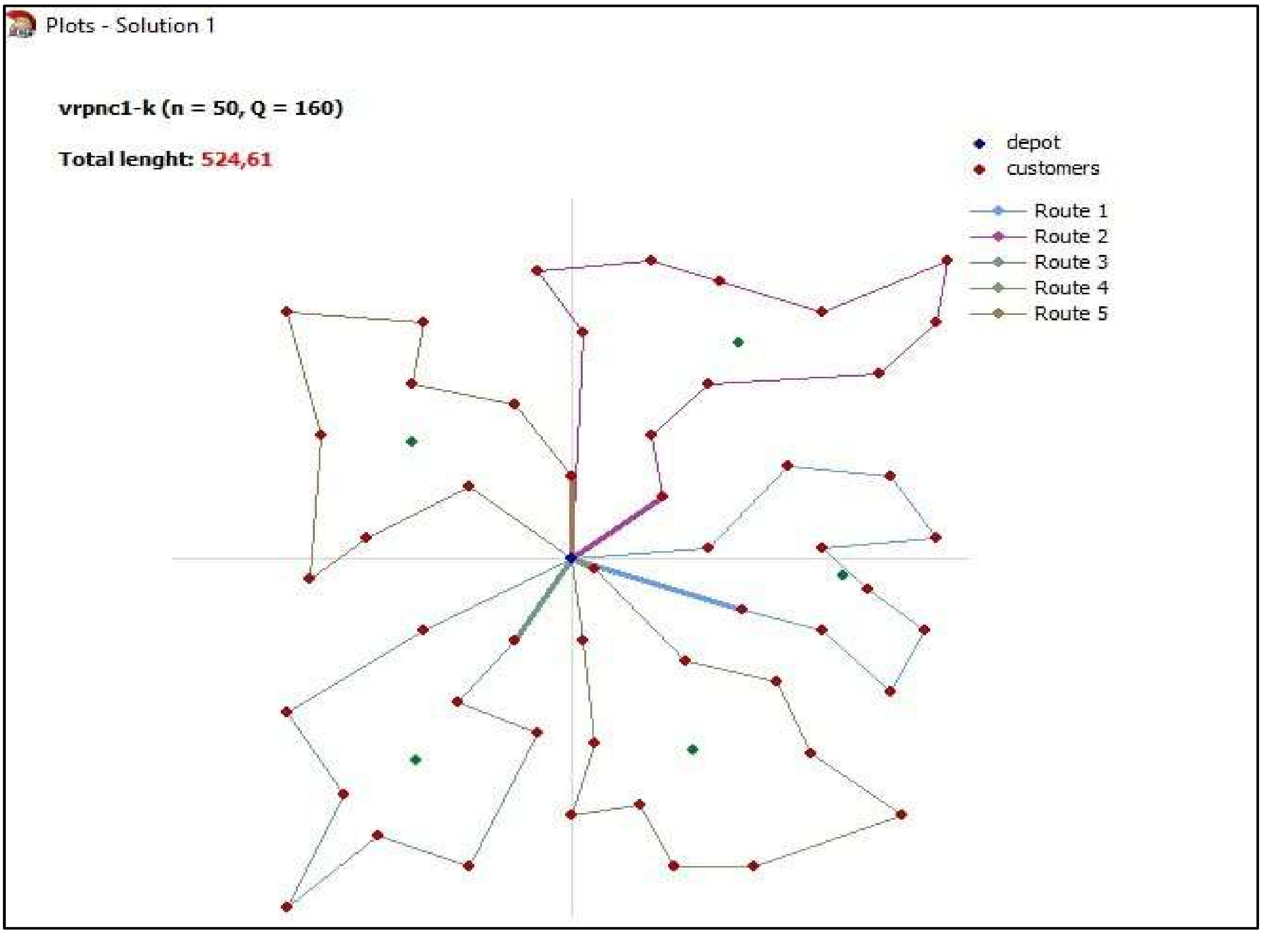
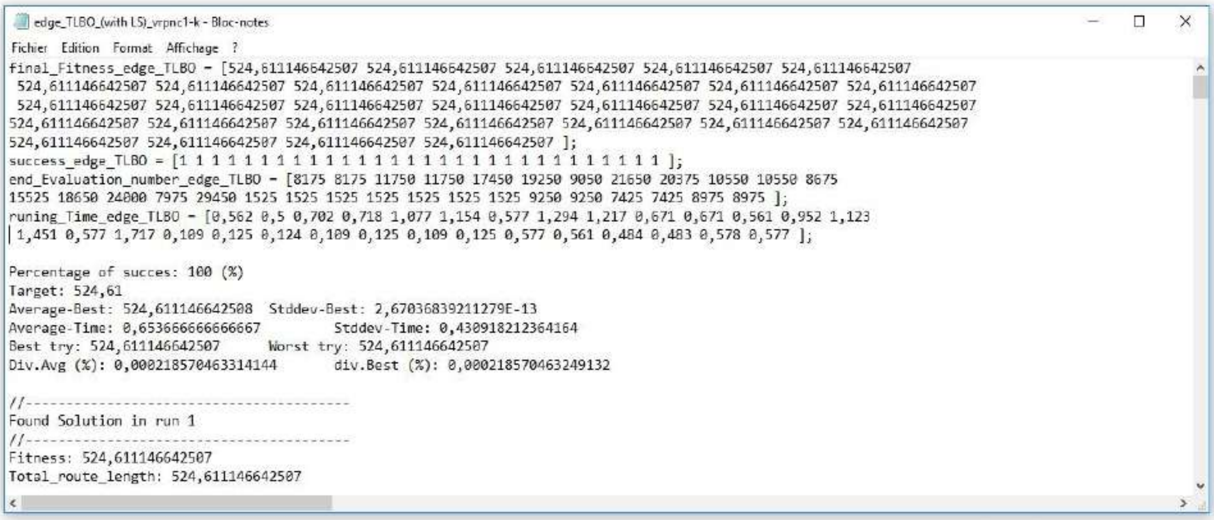


Fig 3.5 – représentation graphique d'une solution trouvée

Prise d'écran du fichier de résultats obtenus



edge_TLBO_(with LS)_vrpnc1-k - Bloc-notes

Fichier Edition Format Affichage ?

```
final_Fitness_edge_TLBO = [524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507
 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507
 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507
524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507 524,611146642507
524,611146642507 524,611146642507 524,611146642507 524,611146642507 ];
success_edge_TLBO = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];
end_Evaluation_number_edge_TLBO = [8175 8175 11750 11750 17450 19250 9050 21650 20375 10550 10550 8675
15525 18650 24800 7975 29450 1525 1525 1525 1525 1525 1525 1525 9250 9250 7425 7425 8975 8975 ];
runing_Time_edge_TLBO = [0,562 0,5 0,702 0,718 1,077 1,154 0,577 1,294 1,217 0,671 0,671 0,561 0,952 1,123
 1,451 0,577 1,717 0,109 0,125 0,124 0,109 0,125 0,109 0,125 0,577 0,561 0,484 0,483 0,578 0,577 ];

Percentage of succes: 100 (%)
Target: 524,61
Average-Best: 524,611146642508  Stddev-Best: 2,67036839211279E-13
Average-Time: 0,653666666666667         Stddev-Time: 0,430918212364164
Best try: 524,611146642507      Worst try: 524,611146642507
Div.Avg (%): 0,000218570463314144       div.Best (%): 0,000218570463249132

//----------------------------------------
Found Solution in run 1
//----------------------------------------
Fitness: 524,611146642507
Total_route_length: 524,611146642507
```

Activer Windows

Fig 3.6 – fichier texte du résultat obtenu

5 Expérimentations et résultats

instance	*n*	*k*	*Q*	*Our BS*	*TLBO Proposé Sans LS*			*TLBO Proposé avec LS*		
					avg	*best*	*Std.Dev*	*avg*	*best*	*Std.Dev*
PR1	50	5	160	**524,61**	537,96	531,84	0,074	524,61	**524,61**	0,43
PR2	75	10	140	**835,26**	965,61	937,65	0,763	838,92	**835,26**	4,57
PR3	100	8	200	**826,13**	826,14	942,96	0,640	828,49	**826,13**	16,926
PR4	150	12	200	**1034,20**	1456,62	1392,86	1,527	1041,18	**1034,20**	2386,93
PR5	199	17	200	**1334.45**	1824.81	1718.12	2444	1344,02	**1334,45**	16199
PR6	50	6	160	**555.43**	561,17	555.43	0,87	555,72	**555.43**	8,66
PR7	75	11	140	**909,67**	948,58	918,41	1,91	909,67	**909,67**	11,42
PR8	100	9	200	**865,94**	917,12	894,45	0,95	865,94	**865,94**	9,74
PR9	150	14	200	**1162,88**	1332,16	1301,27	9,61	1163,73	**1162,88**	22,25
PR10	199	18	200	**1402,45**	1680,91	1574,81	11,23	1408,79	**1402,45**	1866,87
PR11	120	7	200	**1042,11**	1231,98	1191,13	1,36	1042,11	**1042,11**	26,11
PR12	100	10	200	**819,55**	877,65	865,33	5,43	819,55	**819,55**	4,50
PR13	120	11	200	**1541,14**	1593,56	1564,17	248,68	1544,44	**1541,14**	53,01
PR14	100	11	200	**866,36**	869,73	867,73	4,62	866,36	**866,36**	3,01

TABLEAU 3.1 *Résultats de l'algorithme proposé pour les instances de benchmark Christofides (avec et sans local Search) avec un nombre de (run) exécution égal à 30.*

NB : LS : Local Search, n : nombre de client, k : nombre de véhicule, Q : la capacité de véhicule, Our BS : notre meilleur solution, avg : average (moyen), best : meilleur solution, Std.Dev : variance (temps).

Le tableau 3.1 présente les résultats obtenus après l'exécution de notre algorithme TLBO proposé, avec l'utilisation des quatorze (14) instances de benchmark de Christofides, la première colonne représente le nom de l'instance utilisée, la deuxième et la troisième colonnes représentent respectivement le nombre et la capacité des véhicules, cependant, la cinquième colonne *Our BS* définit le meilleur résultat obtenu après la comparaison entre les résultats obtenus par l'exécution de notre algorithme TLBO sans et avec l'utilisation de recherche locale.

Les colonnes Avg, best et std.dev représentent respectivement le moyen de solution obtenu, la meilleure solution et la variance en terme temps d'exécution des solutions.

On constat que l'algorithme TLBO proposé avec l'utilisation de recherche locale donne toujours des meilleurs résultats par rapport l'exécution du même algorithme sans l'utilisation de recherche locale.

L'utilisation de recherche locale avec l'algorithme proposé améliore de façon remarquable l'efficacité de recherche des solutions.

Comparaison des résultats obtenus (avec l'application de recherche locale) avec les quelques résultats de la littérature sur les benchmarks Christofides

instance	*n*	*k*	*Q*	*Meilleure solution connue*	*Vidal et al 2014*	*Xiao et al 2014*	*IIWD 2016*	*ACS 2016*	*Notre TLBO*
PR1	50	5	160	**524.61**	524.61	524.61	524.61	524.61	**524,61 ***
PR2	75	10	140	**835.26**	835.26	835.26	835.74	835.26	**835,26 ***
PR3	100	8	200	**826.13***	826.14	826.14	826.14	831.16	**826,13 ***
PR4	150	12	200	**1028.42**	1028.42	1031.44	1029.13	1028.42	1034,20
PR5	199	17	200	**1291.29**	1291.45	1305.08	1305.49	1291.83	1334.45
PR6	50	6	160	**555.43**	555.43	- -	555.43	555.43	**555.43***
PR7	75	11	140	**909,68**	909,68	- -	910,40	911,23	**909,67***
PR8	100	9	200	**865,94**	865,94	- -	865,94	865,94	**865,94***
PR9	150	14	200	**1162,55**	1162,55	- -	1170,25	1178,22	1162,88
PR10	199	18	200	**1395,85**	1395,85	- -	1417,90	1412,01	1402,45
PR11	120	7	200	**1042,11**	1042,11	1042,12	1042,11	1049,12	**1042,11***
PR12	100	10	200	**819,56**	819,55	819,56	819,56	819,56	**819,55***
PR13	120	11	200	**1541,14**	1541,14	- -	1548,23	1546,78	**1541,14***
PR14	100	11	200	**866,37**	866,37	- -	866,37	866,37	**866,36***

TABLEAU 3.2 Comparaison des résultats obtenus avec les meilleurs résultats de littérature pour les instances de Christofides

Le tableau 3.2 présente les meilleurs résultats précédemment rapportés dans la littérature pour les quatorze (14) instances de Christofides. La première ligne donne les références des travaux comparés et les lignes suivantes montrent les résultats de chaque travail sur chaque instance de test. Dans ce tableau, les meilleurs résultats de notre algorithme utilisé sont également présentés. De plus, nous identifions celui de notre algorithme qui donne le meilleur résultat par le symbole *.

Les meilleures solutions trouvées à ce jour dans la littérature sont incluses dans le tableau 3.2, qui sont tirées du site Web de la bibliothèque de problèmes de tournées de véhicules avec capacité (CVRPLIB).

Comme illustré dans le tableau 3.2, dans dix (10) instances, les solutions optimales sont trouvées, indiquées par le symbole *. Sur la base des valeurs en gras de la colonne (Meilleurs résultats de notre algorithme), dans tous les cas, avec l'utilisation de la procédure de recherche locale, nous avons pu atteindre les meilleures solutions trouvées dans la littérature à l'exception de PR4, PR5, PR9 et PR10. Autrement dit, dans **71.42 %** des instances de Christofides, les meilleures solutions obtenues dans la littérature sont atteintes. Ce qui montre que l'algorithme proposée est atteint un niveau acceptable dans la résolution du problème de CVRP est notamment quand la valeur n (nombre de clients) est inférieur à 150.

6 Conclusion

Dans ce dernier chapitre, les résultats obtenus ont révélé que la résolution du problème de tournées de véhicules avec capacité par l'algorithme TLBO proposé a atteint un niveau acceptable de succès et notamment dans les valeurs inférieures à 150 villes. Ceci nous permet de conclure que l'utilisation de TLBO en combinaison avec une procédure de recherche locale et la stratégie d'apprentissage compréhensif (originalement utilisé dans l'algorithme métaheuristique PSO) est peut-être reconnue comme une nouvelle métaheuristique pour la résolution du problème de CVRP.

Conclusion générale et perspectives

Dans ce mémoire, le problème de tournées de véhicules avec capacité (CVRP) a été étudié en tant qu'un problème NP-difficile. Pour résoudre ce problème complexe, nous avons implémenté un nouvel algorithme TLBO adapté à l'optimisation combinatoire avec l'application d'une procédure de recherche locale (*Local search*) et l'utilisation de la stratégie d'apprentissage compréhensif (*comprehensive learning*) originalement utilisé dans l'algorithme PSO (*Particle Swarm Optimization*). À notre connaissance, aucune approche similaire n'a été proposée dans la littérature.

Notre nouvel algorithme TLBO est basé sur les notions d'enseignement et d'apprentissage, et se déroule en deux phases principales qui sont la phase d'enseignant (Teacher phase) et la phase d'apprenant (Learner phase). C'est l'un des algorithmes métaheuristiques les plus récemment utilisés pour la résolution des problèmes d'optimisation dans différents domaines. Le but d'utilisation de l'algorithme TLBO est d'améliorer les caractéristiques de recherche de solution et résoudre le problème de tournées de véhicule plus efficacement. Ces deux derniers sont les aspects de l'état de l'art de notre approche.

Un ensemble d'instances standard est utilisé pour évaluer notre algorithme adapté, incluant quatorze instances de benchmark de Christofides et al. Les résultats obtenus ont été comparés avec les meilleurs résultats de la littérature. Les résultats expérimentaux ont révélé que notre algorithme TLBO adapté et couplé avec la procédure de recherche locale et par le recours à la stratégie d'apprentissage compréhensif nous a permis de traiter efficacement ces instances benchmarks. À cet égard, il convient de souligner que notre algorithme proposé a atteint les solutions optimales de 10 instances de Christofides, a l'exception de quatre (04) instances pour lesquels l'algorithme donne des résultats acceptables et proches aux meilleurs valeurs connues. C.-à-d., dans 71,42 % des cas, les meilleures solutions connues sont atteintes.

Comme perspectives, il est intéressant d'essayer d'utiliser cet algorithme pour la résolution d'autres variantes de problème VRP, comme le VRP avec fenêtre de temps (VRPTW), VRP avec livraison et ramassage, etc. Par ailleurs, des améliorations et ajustements pourraient être étudiés afin d'améliorer les performances de notre algorithme proposé.

Références bibliographiques

[1] Richard M Karp. Reducibility among combinatorial problems. In Complexity of computer computations, pages 85–103. Springer, 1972.

[2] A. Freville and G. Plateau, "*An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem*", Discrete Applied Mathematics, vol. 49, pages 189-212, 1994

[3] S. Hanafi, A. Freville and A. El Abdellaoui, "*Meta-Heuristics : Theory and Application*", Chapter Comparison of heuristics for the 0-1 multidimensional knapsack problem, pages 446-465, Kluwer Academic, 1996.

[4] V. Boyer, D. El Baz, M. Elkihel, "*Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound*," European J. Industrial Engineering, Vol. 4, n° 4 : 434–449, 2010.

[5] M. Hifi, H. M'Halla, S. Sadfi: "*An exact algorithm for the knapsack sharing problem*". Computers & OR 32: 1311-1324 .2005

[6] M. Hifi, H Mhallah, S. Sadfi. "*Adaptive algorithms for the knapsack problem*". European Journal of Industrial Engineering, 2 (2) : 134-152, 2008.

[7] V. Boyer, D. El Baz, M. Elkihel, "*Dense dynamic programming on multi GPU*," in Proc. of the 19th International Conference on Parallel Distributed and networked-based Processing, PDP 2011, Ayia Napa, Cyprus, 545–551, February 2011.

[8] M.S. Hung, J.C. Fisk. "*An algorithm for the 0-1 multiple knapsack problem*", Naval Research Logistics Quarterly, Vol 24, pp.571–579. 1978.

[9] S. Martello, P. Toth. "*Solution of the zero-one multiple knapsack problem*". European Journal of OperationalResearch, Vol. 4, pp. 276–283. 1980.

[10] T. Yamada, S. Kataoka. "*Heuristic and exact algorithms for the disjunctively constrained knapsack problem*". EURO 2001, Rotterdam, Netherlands; July 911, 2001.

[11] M. Hifi, M. Michrafy."*Reduction strategies and exact algorithms for the disjunctively knapsack problem*". Computers and Operations Research, 34 (0) : 2657-2673, 2007.

[12] A. Bekrar, I. Kacem, C. Chu, C. Sadfi, "*An improved heuristic and an exact algorithm for the 2D strip and bin packing problem*", International Journal of Product Development 10 ,pp. 217-240, 2010.

[13] M. Hifi, I. Kacem, S. Nègre, L. Wu. "*A Linear Programming Approach for the Three-Dimensional Bin-Packing Problems*", Electronic Notes in Discrete Mathematics 2010, 36: 1, 993-1000, 2010.

[14] S. Cheng, J.A. Stankovic and K. Ramamritham, „„Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey"", pp. 150-173 in Hard Real-Time Systems: Tutorial, ed.J.A. Stankovic and K. Ramamritham, IEEE (1988).

[15] Frédéric GUINAND "Ordonnancement avec communications pour architectures multiprocesseurs dans divers modèles d'exécution". Thèse doctorat Juin 95.

[16] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986

[17] G. Laporte. The traveling salesmanproblem : An overview of exact and approximate algorithms. *European Journal of OperationalResearch*, 59(2) : 231–247, 1992.

[18] [88] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(2) : 231–247, 1992.

[19] G. B. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954.

[20] T. Bektas. The multiple traveling salesman problem : an overview of formulations and solution procedures. *Omega*, (34) :209–319, 2006

[21] C. Dhaenens, M.L. Espinouse, and Bernard Penz. Problèmes combinatoires classiques. In Recherche opérationnelle et réseaux : méthodes d'analyse spatiale. Hermès Science Publications, 2002. 6, 7, 17, 18, 20

[22] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling- salesman problem. Journal of the Operations Research Society of America, 2 (4) :pp. 393 410, 1954. 7

[23] M. Gendreau, A. Hertz, and G. Laporte. New insertion and post optimization procedures for the traveling salesman problem. Operations Research, 40(6) : 1086 1094, 1992. 16

[24] C. Rego and C. Roucairol. Le problème de tournées de vehicules : Étude et résolution approchée. Technical Report 2197, INRIA - Institut National de Recherche en Informatique et en Automatique, 1994. 5, 7, 10, 13, 17, 22, 31

[25] P. Toth and D. Vigo. The vehicle routing problem. Society for Industrial Mathematics (SIAM) Monographs on Discrete Mathematics and Applications, Philadelphia, 2001a. 5, 7, 42

[26] G. Crainic and F. Semet. Recherche opérationnelle et transport de marchandises. In Optimisation combinatoire. 3, Applications. Hermès Science : Lavoisier, 2006. 7, 10, 17, 19, 20, 25, 26

[27] B. Kallehauge, J. Larsen, O.B.G. Madsen, and M.M. Solomon. Vehicle routing problem with time windows. Column Generation, pages 67 98, 2005. 11

[28] N. Jozefowiez. Modélisation et résolution approchées de problèmes de tournées multi-objectif. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, December 2004. 11

[29] Talbi. Métaheuristiques pour l'optimisation combinatoire multi-objectif : Etat de l'art. Technical report, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille 1, France, 2001. 12

[30] P. Toth and D. Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001.

[31] J. F. Bard, G. Kontoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, (36) : 250–269, 2002.

[32] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. *Networks*, (44) :216–229, 2004.

[33] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.*, 56(2) :497–511, 2008.

[34] J-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon F., and Soumis. The vehicle routing problem. Chapter VRP with Time Windows, pages 157– 193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[35] J. Z. Czech. Parallel simulated annealing for the vehicle routing problem with time windows. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 376–383, 2002.

[36] J. Berger and M. A. Ayechew. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 31 :2037–2053, 2004

[37] 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*, 148(3) :559–569, 2003.

[38] A. Lim and Z. Xingwen. A two-stage heuristic for the vehicle routing problem with time windows and a limited number of vehicles. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, page 82c, jan. 2005.

[39] X-P. Wang, C l. Xu, and X-P. Hu. Genetic algorithm for vehicle routing problem with time

windows and a limited number of vehicles. In *Management Science and Engineering, 2008. ICMSE 2008. 15th Annual*

[40] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & amp, Operations Research*, 31(12) :1985–2002, 2004. ISSN 0305-0548.

[41] D. Feillet. *Routing problems with profits: Study and application to a freight transportation problem*. PhD thesis, Ecole centrale des arts et manufactures, Chatenay-Malabry, France, 2001.

[42] H. Bouly, A. Moukrim, D. Chanteur, and L. Simon. Un algorithme de destruction/construction itératif pour la résolution d'un problème de tournées de véhicules spécifique. In *MOSIM'08*, 2008.

[43] Fred Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, Comput. & Ops. Res.Vol. 13, No.5, pp. 533-549, 1986.

[44] ABBAS El Dor. Perfectionnement des algorithmes d'optimisation par essaim particulaire : applications en segmentation d'images et en électronique. Autre [Cs. OH]. Université Paris- Est, 2012.

[45] S. R. Thangiah. Vehicle routing with time windows using genetic algorithms. *Application hanbook of genetic algorithms : new frontiers*, II :253–277, 1995.

[46] C. Rego and C. Roucairol . « Le Problème de Tournées de Véhicules : Etude et Résolution Approchée ». Technical Report, inria, Février 1994.

[47] B. Gulay and D. Ozgur . « A tabu search algorithm for the vehicle routing problem ». Computers and Operations Research 26 (3), pages 255–270, 1999.

[48] C. Rego . « Node ejection chains for the Vehicle Routing Problem: Sequential and Parallel Algorithms ». Parallel Computing 27 (3), pages 201–222, 2001.

[49] P. Caricato , G. Ghiani , A. Grieco , and E. Guerriero . « Parallel Tabu Search For A Pickup And Delivery Problem Under Track Contention ». Parallel Computing 29 (5), pages 631–639, 2003.

[50] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[51] P. Hansen and N. Mladenovi. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3) :449–467, 2001

[52] F. Pereira , J. Tavares , P. Machado , and E. Costa . « GVR : a New Genetic Representation for the Vehicle Routing Problem ». Artificial Intelligence and Cognitive Science: 13th, Irish Conference Proceedings, pages 95–102, 2002.

[53] C. Prins . « A simple and effective evolutionary algorithm for the vehicle routing problem ». MIC'2001 (4th Metaheuristics International Conference), 143-147, 16-20/07/01, Porto, Portugal, 2001.

[54] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[55] K. Kameyama. Particle swarm optimization - a survey. *IEICE Transactions*, 92-D(7) :1354–1361, 2009

[56] Teaching-Learning-Based Optimization (TLBO) algorithm and its codes developed by Dr. R. Venkata Rao and his team at S.V. National Institute of Technology

[57] R. Venkata Rao, Vivek Patel, an improved teaching- learning- based optimization algorithm for solving unconstrained optimization problems [en ligne], 740p. Format PDF. Disponible : «https://www.sciencedirect.com/science/article/pii/S0010448510002484 »

[58] S. Goss , S. Aron , J.-L. Deneubourg , and J.M. Pasteels . « Self organized shortcuts in the argentine ant ». Naturwissenschaften, Vol. 76, pages 579–581, 1989.

[59] J-Y. Potvin and S. Bengio. The vehicle routing problem with time windows part ii : Genetic search. *INFORMS Journal on Computing 8*, pages 165–172, 1996.

[60] J-Y. Potvin, T. Kervahut, B-L. Garcia, and J-M. Rousseau. The vehicle routing problem

with time windows part i : Tabu search. *INFORMS Journal on Computing*, 8(2) :158–164, 1996.
[61] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & amp, Operations Research*, 31(12) :1985–2002, 2004. ISSN 0305-0548.
[62] C. Prins, N. Labadie, and M. Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2) :507– 535, 2009.
[63] G. Reinelt. A traveling salesman problem library. *ORSA Journal on Computing*, 1991.
[64] G. Righini and M. Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & amp; Operations Research*, 36(4) : 1191–1203, 2009.
[65] Y. Rochat and R. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1 :147–167, 1995.
[66] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: a generic branch-and-price algorithm. Preprint accepted for publication in INFORMS Journal on Computing, 2012.
[67] Bräysy. Local search and variable neighborhood search algorithms for the vehicle routing problem with time windows. PhD thesis, University of Vaasa, Finland, 2001. 11.
[68] L. Davis . « Handbook of genetic algorithms ». Technical Report, Van Nostrand Reinhold, 1991.
[69] B. M. Baker and M.A. Ayechew . « A genetic algorithm for the vehicle routing problem ». Computers and Operations Research 30, pages 787–800, 2003.
[70] A. Colorni , M. Dorigo , and V. Maniezzo . « Distributed optimization by ant colonies ». Proceedings of the first European Conference on Artificial Life (ECAL 91), pages 134–142, 1992.
[71] S. Goss , S. Aron , J.-L. Deneubourg , and J.M. Pasteels . « Self organized shortcuts in the argentine ant ». Naturwissenschaften, Vol. 76, pages 579–581, 1989.
[72] Olivier C Martin and Steve W Otto. Combining simulated annealing with local search heuristics. Annals of Operations Research, 63(1) :57–75, 1996.
[73] Matej C repinšek , Shih-Hsi Liu , Luka Mernik "*A note on teaching–learning-based optimization algorithm*" information sciences –journal-elsevier vol 212 , 1 December 2012, Pages 79-93
[74] EhsanTeymourian, VahidKayvanfarb, GH.M.Komaki, M.Zandieh "*Enhanced intelligent water drops and cuckoo search algorithms for solving the capacitated vehicle routing problem*" " journal Elsevier – InformationSciences 334–335(2016) pages354–378
[75] Abdelkamel Ben Ali , Gabriel Luque , Enrique Alba "*An efficient discrete PSO coupled with a fast local search heuristic for the DNA fragment assembly problem*" journal Elsevier -Information Sciences vol512 (2020) pages 880–908
[76] H.W.Kuhn: *The Hungarian Algorithm for the assignment problem*. Naval Research Logistic Quarterly, vol. 2, pages 83-97, 1955
[77] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. *Comprehensive learning particle swarm optimizer for global optimization of multimodal functions*. IEEE transactions on evolutionary computation, 10(3):281–295, 2006