Université Mohamed El Bachir El Ibrahimi
Bordj Bou Arréridj
Faculté de Mathématiques et Sciences

DÉPARTEMENT de MATHÉMATIQUES

MASTER

Domain : Mathematics

Spécialité : Recherche Opérationnelle

Par : Belbouab Amina - Kerfa Hanene

Thème

Optimisation des problemes de transports par les algorithmes génétique

Soutenu publiquement le : ...-10-2021

Devant les jurés :

Dr.	M.A. Université .... -	Président
Dr.	M.A. University .....- ....a	Examinateur
Dr. Djaafer zaouch	M.A. Université ..... -	Encadreur

Année universitaire 2020/2021

Remerciements

Tout d'abord, nous remercions Allah de nos avoir donné le courage de mener à bien modeste travail de fin d'étude.

Et nous ne pas oublier notre méres et notre famille qui nous donnent tous les soutiens de notre carriére scolaire.

nous tenons à remercier vivement notre encadreur **Dr. ....** d'avoir accéptée de diriger ce projet de fin d'étude, sa gentillesse, sa disponibilitée et ses précieux conseils.

Et nous remercions les membres de jury d'avoir accepté d'examiner notre travail.

Dédicaces 01

Je dédie ce modest travail á mes êtres les plus chers au monde
A mes parents qui ont tout sacrifié pour mes études.
A mon chère époux 'Mustapha' et ma fille 'Elyne'.
A mes fréres 'Mouhamed Amine' et 'Abd erraouf' et mes soeurs 'Rahma' et 'Ghoufrane'.
A tous mes amis et mes collégues.
Enfin, je dédie ce travail à tous ceux qui m'ont aidés de prés ou de loin.

M. Hanane

Dédicaces 02

Je dédie ce modest travail á mes êtres les plus chers au monde
A mes parents qui ont tout sacrifié pour mes études.
A mon frére 'Idris' et ma soeur 'Chaima'.
A tous mes amis et mes collégues.
A tous ma famille 'Belbouab'.
Enfin, je dédie ce travail à tous ceux qui m'ont aidés de prés ou de loin.

B. Amina

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Un probleme d'optimisation combinatoire (COP : Combinatorial Optimisation Problems) comprend un ensemble fini de solutions, ou chaque solution doit respecter un ensemble de contraintes relatives a la nature du probleme. On associe a chaque solution une valeur, nommele valeur de l'objectif qui est evalue a l'aide d'une fonction objective. les problemes d'optimisation combinatoire peuvent être mono-objectifs qui consistent a optimiser une seule fonction objective, ou multiobjectifs qui optimisent plusieurs fonctions objectives.

Les problèmes d'optimisations sont souvent faciles a definir, mais généralement difficiles a resoudre. En effet, la plupart de ces problemes appartiennent a la classe des problemes NP- completes et ne possedent donc pas a ce jour une solution algorithmique efficace et valable pour toutes les donnes, outre la classification selon le nombre de critere a optimiser les methodes d'optimisation combinatoire peuvent être classees aussi en methodes exactes et méthodes approchees (heuristiques et métaheuristiques).

Parmi les metaheuristiques, on trouve la famille des algorithmes evolutionnaires constituee de quatre types t les strategies d'évolution, la programmation evolutionnaire et génetique et les algorithmes génétiques (mono-objectifs, multiobjectifs).

Les methodes des algorithmes génetiques multiobjectifs sont particulierement bien adapteles au traitement de problemes multiobjectifs où l'on recherche un ensemble de solutions. Les algorithmes génetiques travaillent sur une population de solution,cette caractéristique leur permet de trouver plusieurs solutions potentielle-

ment Pareto optimales. Done I'approche des algorithmes génetiques peut etre capable de resoudre le probleme de controlle multiobjectif.

En outre, notre travail consiste a demontrer l'utilité de l'algorithme génetique multiobjectif pour résoudre un problème combinatoire (problème de voyageur de commerce).

Le Chapitre 1 nous essayons d'aborder quelques notions de base en optimisation combinatoire et on définit le problème d'optimisation MultiuObjectif , avec ses concepts fondamentaux : (la domination, l'efficacité et la convexité...). Puis on décrit les méthodes de résolution.

Le Chapitre 2 nous décrivons les algorithmes génétiques (principes , concepts de base et les opé- rateurs génétiques avec ses paramètres...etc). Ensuite les algorithmes génétiques multiobjectifs.

Le Chapitre 3 nous avons parlons tous d'abord les différents probléme de transport Comme probléme de voyageur de commerce routage vehicule et postier chinois. En suite on travail sur les travau dérigé par les chercheurs pour résoudre ces probléme par les algorithmes evolutionnaire en précise les algorithmes génétique et la programm.

Chapitre 1

Optimisation combinatoire

1.1 Introduction

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.
Dans ce chapitre, on définit c'est quoi un problème d'optimisation combinatoire en citant quelques problèmes NP-Difficiles avec une étude de sa complexité et ces notions, et aussi les problèmes d'optimisations multi-objectifs avec ses méthodes de résolution.

1.2 Notions de base en optimisation

Deux types de problèmes d'optimisation sont distingués : des problèmes de minimisation et des problèmes de maximisation. Un problème d'optimisation (de minimisation ou de maximisation) est défini par un ensemble de données et un ensemble de contraintes. Un ensemble de solutions S est associé au problème d'optimisation. Parmi les solutions S, un sous-ensemble $S \subseteq X$ représente des solutions réalisables

respectant les contraintes C du problème, à chaque solution s est associée une valeur f(s) qui représente sa qualité. La résolution du problème d'optimisation consiste à trouver une solution $s_* \in X$ qui minimise ou maximise la valeur f(s).

Quelque soit le type du problème d'optimisation, ce dernier est défini par le 6-uplet $< D, C, S, X, f,$ mode$>$ Où D représente les donnes du problème, C les contraintes que doit satisfaire une solution afin d'étre admissible, S l'ensemble des solutions possibles du problème traité, X un sous-ensemble de S représentant les solutions réalisables (admissibles), f une fonction du coût (aussi appelée fonction u objectif »n ou fonction fitness) qui associe à chaque solution s une valeur numérique f(s) (nombre réel ou entier) représentant la qualité de s, mode indique le type du problème, il permet de savoir est ce qu'on doit minimiser ou maximiser les valeurs des solutions de X. Dans ce qui suit nous présentons quelques définitions tirées de la littérature liées aux problèmes d'optimisation en général.

Définition 1.1 (Inspirée de : Papadimitriou et Steiglitz, 1982)**:**
Une instance d'un problème de minimisation (maximisation) est un couple (X, f). Où $X \subseteq S$ est un ensemble fini de solutions potentielles admissibles et f *est une fonction du coût (fonction objectif) à minimiser (à maximiser),* $\mathrm{f} : X \rightarrow 9$. *L'objectif est de trouver* $s* \in X$ *tel que* $\mathrm{f}(\mathrm{s}^*) \leq \mathrm{f}(\mathrm{s})$ *(Au cas de maximisation :* $\mathrm{f}(\mathrm{s}^*) \geq \mathrm{f}(\mathrm{s})$ *) pour n'importe quelle solution* $s \in X$

Définition 1.2 (Inspirée de : Sakarovitch, 1984)**:**
Un problème d'optimisation combinatoire est défini par l'ensemble S *des solutions possibles d'un problème.* $X \subseteq S$ *est l'ensemble des solutions réalisables.* $\mathrm{f} : X \rightarrow \mathfrak{R}$*, une fonction que l'on nomme fonction wobjectifs. La résolution du problème consiste à minimiser (maximiser) la valeur* f(s)*, où* $s \in X$*.*

Définition 1.3 (Inspirée de : Papadimitriou et Steiglitz, 1982)**:**
Un problème mono-objectif est décrit par la formule (1.1)

$$\textit{Tel que} \begin{cases} \textit{Mode}_{\,x} f(s) \\ C_i(s) \Delta 0, i = 1 \ldots \mathrm{m} \\ H_i(s) = 0, \mathrm{j} = 1 \ldots \mathrm{p} \\ s \in S \subset \mathfrak{R}^n \end{cases} \tag{1.1}$$

$$\Delta \textit{ est remplacé par } \begin{cases} \leq & \textit{au cas de problème de minimisation} \\ \geq & \textit{au cas de problème de maximisation} \end{cases}$$

f est la fonction du cout (la fonction wobjectifs) à minimiser (ou à maximiser). C_i *sont des contraintes d'inégalité et* H_i *sont des contraintes d'égalité.* S *est l'ensemble de solutions possibles (l'espace de recherche). s est une solution admissible, elle appartient à* S *et respecte les contraintes du problème.*

Définition 1.4 (Inspirée de : Fonseca et Fleming, 1993)**:**
Un probleme multi-objectif est décrit par la formule (**??**)

$$Tel\ que \begin{cases} Mode\ {}_{\mathrm{x}}\vec{f}(s) \\ \vec{C}_i(s)\Delta 0, \mathrm{i} = 1 \ldots \mathrm{m} \\ \vec{H}_i(s) = 0, \mathrm{j} = 1 \ldots \mathrm{p} \\ s \in S \subset \Re^n \end{cases} \tag{1.2}$$

Δ *est remplacé par*

$$\begin{cases} \leq & au\ cas\ de\ problème\ de\ minimisation \\ \geq & au\ cas\ de\ problème\ de\ maximisation \end{cases}$$

Dans le cas de problémes d'optimisation multi-objectifs, la fonction kobjectifs est représentée par un vecteur $\vec{f}$ *regroupant* N *fonctions wobjectify. Aussi les contraintes sont représentées par des vecteurs regroupant les contraintes de chaque fonction wobjectifs.*

Définition 1.5:
Une solution d'un probleme d'optimisation est un ensemble de quantités souvent numériques pour lesquelles des valeurs sont à choisir. L'ensemble de ces valeurs est généralement regroupé dans un vecteur représentant une solution. Supposant un problème de taille n*, le vecteur représentant la solution s est représenté par :*

$$\vec{s} = [s_1, s_2, s_3, \ldots, s_n] \tag{1.3}$$

Les différentes valeurs prises par les variables $(s_1, s_2, s_2, \ldots, s_n)$ *constituent l'ensemble des solutions envisageables. Selon le type des variables* $(\mathrm{s}_1, s_2, \mathrm{s}_3, \ldots, \mathrm{s}_\mathrm{a})$*, le type du problème d'optimisation peut être reconnu. Par conséquent, on distingue deux classes de problèmes d'optimisation : des problèmes continus et des problèmes discrets. Dans la première classe (i.e. la classe des problèmes continus), les variables composant une solution donnée sont de type réel. Tandis que, dans les problèmes de*

la deuxième classe (i.e. la classe des problèmes discrets), les variables composant une solution donnée peuvent être de type entier, naturel ou binaire.

Définition 1.6:
Une contrainte d'un problème est une restriction imposée par la nature et les caractéristiques du problème sur les solutions proposées.

Définition 1.7:
L'espace de recherche S *d'un problème est composé de l'ensemble de valeurs pouvant être prises par les variables* $(s_1, s_2, s_3, \ldots, s_n)$ *qui construisent la solution* s.

Définition 1.8:
Le voisinage V(s) d'une solution s est un sous ensemble de S*, dont les membres sont des solutions proches (voisines) de la solution s. En effet, on dit qu'une solution* $\langle s' \rangle$ *est une voisine de* s*, si elle peut etre obtenue en modifiant légèrement la solution s.*

Définition 1.9:
L'optimum local est la meilleure solution appartenant à un voisinage V(s) *de la solution s. Une solution* s' *(appartenant à* S *) est un optimum local de la structure du voisinage* V(s) *de la solution* s *si elle vérifie la condition suivante* (1.4) *:*

$$f(s') \leq f(s) \quad \forall s \in V(s) \tag{1.4}$$

Pour un problème de maximisation l'inégalité est inversée, c-à-d. la condition (1.5) *sera remplacée par la condition (1.5) :*

$$f(s') \geq f(s) \quad \forall s \in V(s) \tag{1.5}$$

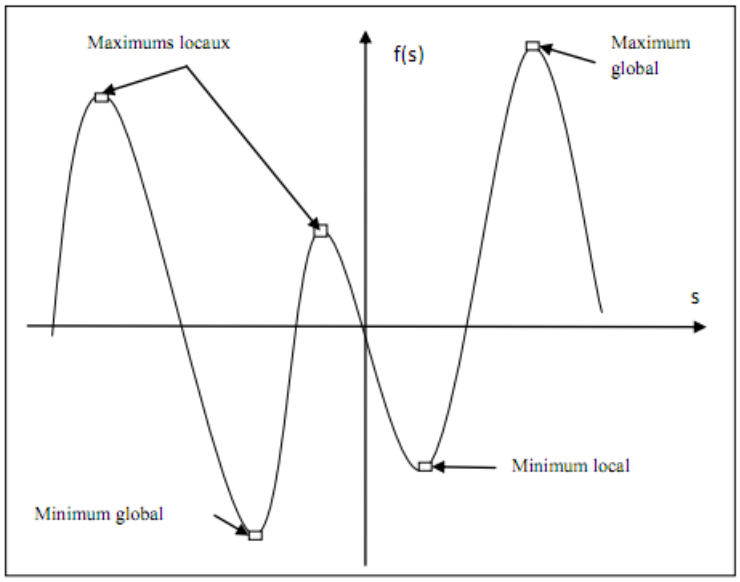


FIGURE 1.1 – Courbe représentant les optimums locaux et les optimums globaux.

Il est à noter que l'optimum local est aussi nommé maximum local au cas de problème de maximisation et minimum local au cas de problème de minimisation.

Définition 1.10:

Une solution optimale (L'optimum global) : Une solution s* *est dite optimale (ou optimum global) si les deux contraintes suivantes sont vérifiées :*

1. *Elle est réalisable : tirée de l'ensemble de solutions possibles (l'espace de recherche* S *) et respecte toutes les contraintes du problème posé.*
2. $\begin{cases} f(s^*) = \max_{s \in X}\{f(s)\} \text{ En cas de problème de maximisation} \\ f(s^*) = \min_{s \in X}\{f(s)\} \text{ En cas de problème de minimisation} \end{cases}$ *Où* X *est l'ensemble de solutions réalisables.*
   Autrement dit, l'optimum global est le meilleur optimum local. Ainsi, une solution est dite optimum global si :

$$\begin{cases} f(s^*) \geq f(s) \forall s \in E(S) \text{ au cas de problème de maximisation} \\ f(s^*) \leq f(s) \forall s \in E(S) \text{ au cas de problème de minimisation} \end{cases}$$

Où $E(S)$ *est l'ensemble d'optimums locaux. Il est à noter que l'optimum global est aussi nommé maximum global au cas de problème de maximisation et minimum global au cas de problème de minimisation.*

La figure 1.1 représente une courbe représentant les optimums locaux et les optimums globaux d'une fonction d'évaluation.

1.3 La complexité et la théorie de la complexité

Afin de mesurer la difficulté d'un problème donné et la comparer avec celles d'autres problèmes pour pouvoir dire qu'un tel problème est plus facile à résoudre que l'autre, nous pouvons calculer la complexité algorithmique de chacun d'entre eux. La complexité d'un problème donné est discutée sur deux cotés : coté temporel (complexité temporelle) et coté spatial (complexité spatiale). La complexité temporelle consiste à évaluer le temps de calcul nécessaire pour résoudre un problème donné. Tandis que la complexité spatiale permet d'estimer les besoins en mémoire (l'espace mémoire requis) pour la résolution d'un problème donné. La complexité d'un problème donné est estimée en fonction du nombre d'instructions permettant d'aboutir à la solution du problème posé. Elle est influencée par la taille du problème en question. En effet, elle exprime un rapport entre la taille du problème, le temps de calcul nécessaire et l'espace mémoire requis.

La théorie de la complexité s'intéresse à l'évaluation de la difficulté des problèmes via l'étude de la complexité de solutions algorithmiques proposées. Elle associe une fonction de complexité à chaque algorithme résolvant un problème donné et mesure les ressources nécessaires pour la résolution d'un problème posé. En effet, elle classe l'ensemble des problèmes selon deux critères qui sont le temps de calcul nécessaire et l'espace mémoire requis.

Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisations [Garey et Johnson, 1979]. Elle classe les problèmes selon leur complexité en deux classes principales : la classe P (Polynomial time) et la classe NP (Non deterministic Polynomial time). En outre, elle partage les problèmes de la classe NP en deux sous classes : NP-Complet et NP-Difficile.

1.3.1 Les problèmes de la classe *P*

Les problèmes appartenant à la classe P sont des problèmes pouvant être résolus par une machine de Turing déterministe en temps de calcul polynomial par rapport à la taille de l'instance du problème à traiter. Ils sont souvent faciles à résoudre par des algorithmes efficaces dont le nombre d'instructions nécessaire pour la résolution

d'un problème donné est borné par une fonction polynomiale par rapport à la taille du problème [Sakarovitch, 1984]. Plus formellement, les problèmes de cette classe sont définis comme suit [Alliot et Schiex, 1994] :

$$P = \{PL/PL \subseteq X^*, \exists MTD, PL = PL(MTD), \exists Ply, \forall w \in PL, |w| = n, T_{MTD}(w) < Ply(n)\}$$

Où Ply est une fonction polynomiale, MTD une machine de Turing déterministe, PL un problème et T_{MTD} le temps de reconnaissance du problème.
Le calcul du plus grand diviseur commun est un exemple d'un problème appartenant à la classe P.

1.3.2 Les problèmes de la classe NP

La classe des problèmes NP regroupe l'ensemble de problèmes qui peuvent être résolus avec une machine de Turing non-déterministe et admettent un algorithme polynomial pour tester la validité d'une solution du problème traité (i.e. il est possible de vérifier que la solution proposée est correcte en un temps polynomial par rapport à la taille du problème).
Le problème du sac à dos, le problème du voyageur de commerce, les problèmes d'ordonnancement, le problème de coloration de graphes sont des exemples de problèmes appartenant à la classe NP.

Les problèmes NP-Complets

La classe NP-Complet regroupe les problèmes de décision pour lesquels il n'existe pas d'algorithme permettant leur résolution en un temps polynomial. Selon Garey et Johnson, un problème X est un problème NP-Complet s'il appartient à la classe NP, et si quelque soit le problème X' qui appartient aussi à la classe NP, on peut le réduire au problème X en un temps polynomial [Garey et Johnson, 1979]. Selon Ferro et ces collègues, un problème X est un problème NP-Complet s'il appartient à la classe NP, et si on peut le réduire à un problème connu dans NP-Complet [Ferro et al, 2005]. Par conséquent, la NP-Complétude est un problème décisionnel. Le premier problème qui a été démontré comme étant NP-Complet a été celui de la satisfiabilité d'une formule logique binaire (SAT). Cela a été établi en 1971 par Stephen Cook dans son théorème publié dans [Cook, 1971].

Les problèmes NP-Difficiles

La classe de problèmes NP-Difficiles englobe les problèmes de décision et les problèmes d'optimisation. Les problèmes NP-Difficiles sont aussi difficiles que les problèmes NP-Complets. Si un problème de décision associé à un problème d'optimisation P est NP- Complet alors P est un NP-Difficile [Charon et al, 1996]. Par conséquent, afin de prouver qu'un problème d'optimisation est NP-Difficile, il suffit de montrer que le problème de décision associé à P est NP-Complet [Layeb, 2010]. Il est à noter que jusqu'à maintenant, aucun algorithme polynomial n'est connu pour résoudre ce type de problèmes (i.e. NP- Difficiles).

1.3.3 Les problèmes NP-Complets versus les problèmes NP-Difficiles

Les problèmes NP-Complets et les problèmes NP-Difficiles sont deux types de problèmes difficiles. Aucun algorithme polynomial permettant leur résolution n'a été trouvé. En fait, les algorithmes proposés dans la littérature sont de complexité exponentielle ou pire qu'exponentielle. La différence entre les problèmes de ces deux types de problèmes réside dans le type de la solution attendue. Les problèmes NP-Complets sont des problèmes décisionnels dont la réponse attendue pourra être « oui ou non ». Comme l'exemple du problème du cycle hamiltonien qui consiste à répondre à la question : Est-ce qu'il existe un cycle permettant de parcourir tous les sommets d'un graphe non orienté en passant par chacun une seule fois ? Ici on s'attend à une des deux réponses : oui il existe ou non il n'existe pas. Par contre, les problèmes NP-Difficiles concernent beaucoup plus les problèmes d'optimisation ou on cherche la solution optimale. Comme le cas du problème du voyageur de commerce dont on cherche le plus court chemin permettant de parcourir un ensemble de sommets (villes) en passant par chacun une seule fois.

1.3.4 La relation entre les problèmes P et NP

La question « P=NP ? » est une des questions les plus importantes qui n'ont pas encore été résolues en informatique théorique. En fait, la réponse à cette question a construit un champ de recherche ouvert. La réponse à la question « P=NP ? » par « oui », revient à montrer que tous les problèmes de la classe NP sont dans la

classe P. Autrement dit, la réponse à cette question revient à répondre à la question : Peut-on trouver en temps polynomial ce qu'on peut prouver en temps polynomial ? Cependant, aucune démonstration n'a été faite pour prouver que $P \subseteq NP$, ni que $P \nsubseteq NP$. La relation évidente c'est que $NP \subseteq P$. En effet, un problème qui peut être résolu en un temps polynomial par un algorithme déterministe, pourra aussi être résolu par un algorithme non déterministe. Ce qui est admis et connu jusqu'à maintenant, c'est que PNP [Garey et Johnson, 1979]. Néanmoins, aucune preuve n'a encore été prouvée jusqu'à ce jour. Si un jour on arrivera à trouver un algorithme polynomial permettant la résolution d'un problème NP-Complet, on pourra résoudre tous les problèmes NP-Complets en temps polynomial. Cook a prouvé dans [Cook, 1971] que tous les problèmes de la classe NP sont réductibles au problème de la satisfiabilité d'une formule logique, cela veut dire que si quelqu'un trouvera un algorithme polynomial pour le problème de SAT, alors la question « P = NP ? » sera résolu ! Selon la figure 1.2, les problèmes NP faciles à résoudre peuvent construire des problèmes de la classe P. Les autres problèmes, i.e. les plus difficiles à résoudre, peuvent être partagés en deux autres groupes : le groupe des problèmes NP-Complets et le groupe des problèmes dont le statut est indéterminé car ces problèmes n'ont pas été prouvés comme appartenant à la classe P ni à la classe NP-Complet [Lacomme et al, 2003], comme le cas du problème d'isomorphisme de deux graphes [Lacomme et al, 2003]. En effet, il semblait que ce problème appartient à la classe P [Sakarovitch, 1984]. Néanmoins, jusqu'à ce jour aucune preuve n'a été montrée [Benatchba, 2005].

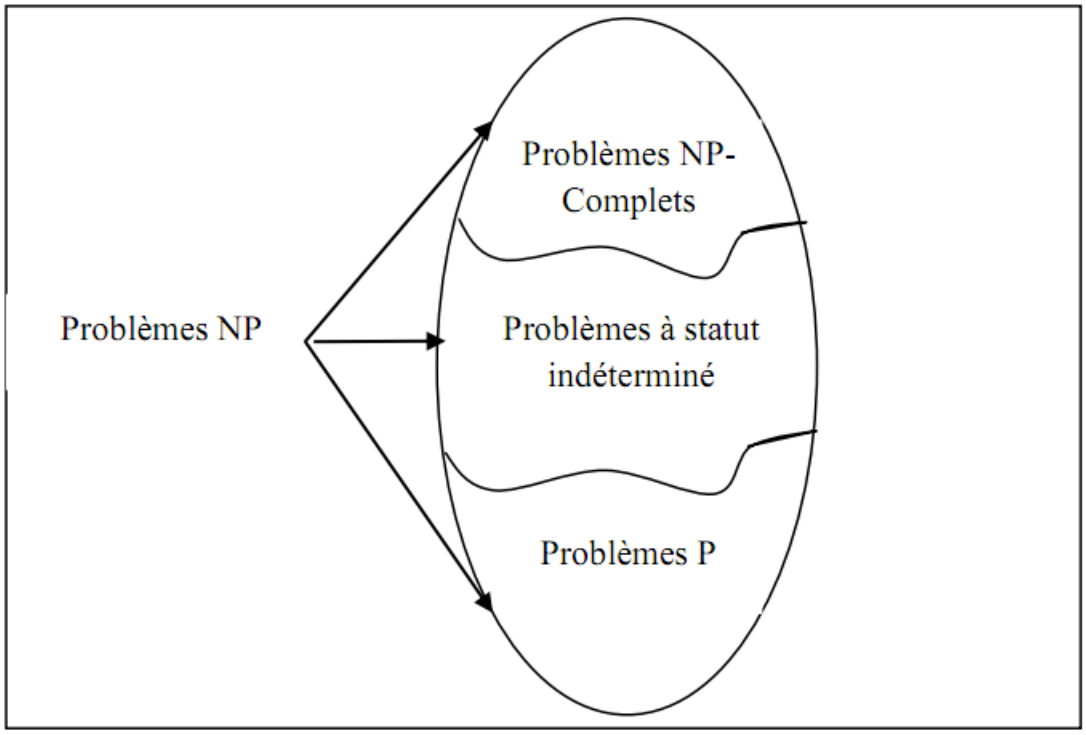


FIGURE 1.2 – La relation entre les problèmes P, NP et NP-Complets. [Lacomme et al, 2003].

1.3.5 Classification des problèmes d'optimisation

Les problèmes d'optimisation que l'on rencontre au monde réel sont classifiés selon leurs caractéristiques. Nous distinguons ainsi les problèmes selon :

— Le nombre de variables de décision :

- Mono-variable.
- Multi-variables.

— Le type de variables de décision :

- Variables continues : où les valeurs sont des nombres réels.
- Variables discrètes : où les valeurs sont nombres entiers.
- Variables combinatoires : où les valeurs sont des permutations sur un ensemble fini de nombre.

— Le type de la fonction-objectif

- Fonction linéaire.

- Fonction non linéaire.
- Fonction quadratique.

— La formulation du problème :

- Problème sans contraintes.
- Problème avec contraintes.

1.4 Formulation d'un problème d'optimisation multi-objectif

Un PMO peut être formulé comme suit :

$$\begin{aligned} \text{Minimiser } \overrightarrow{\mathcal{F}}(\vec{x}) &= \left(\overrightarrow{f_1}(\vec{x}), \overrightarrow{f_2}(\vec{x}), \ldots, \overrightarrow{f_k}(\vec{x})\right) \text{ avec } x \in R^n \rightarrow R^k \\ \vec{g}(\vec{x}) &\leq 0 \text{ avec } \vec{g}(\vec{x}) \in R^m \\ \vec{h}(\vec{x}) &= 0 \text{ avec } \vec{h}(\vec{x}) \in R^q \end{aligned}$$

où, k désigne le nombre d'objectifs, n est le nombre de variables de décision, m représente le nombre de contraintes d'inégalité, et q est le nombre de contraintes d'égalité.

1.5 Définition

L'espace de décision et l'espace de critère :
Deux espaces Euclidiens sont considérés en optimisation :

— **L'espace décisionnel** $F(X)$: de dimension n, n étant le nombre de variables de décision.

Cet espace est constitué par l'ensemble des valeurs pouvant être pris par le vecteur de décision.

— **L'espace objectif** $F(X)$: L'ensemble de définition de la fonction objective dans $\mathbb{R}$. La valeur dans l'espace objectif d'une solution est appelée coût, ou fitness.

Les fonctions objectives du problème d'optimisation forment un espace multidimension-nel appelé espace des fonctions objectives, en plus du traditionnel espace des variables de décision. Le schéma de la Figure 1.3 illustre

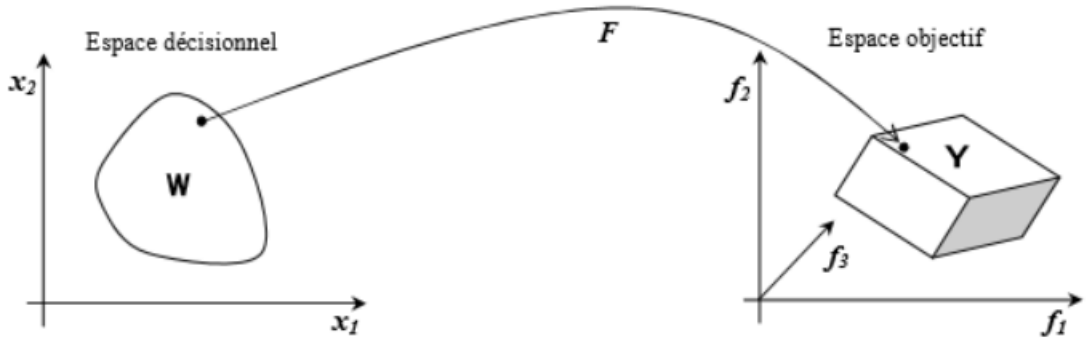


FIGURE 1.3 – problème d'optimisation multi-objectif (2 variables de décision et 3 fonctions objectifs)

les deux espaces ou, pour chaque solution $x = (x_1, x_2)$ dans l'espace des variables de décision, il existe un point dans l'espace des fonctions objectif tel que $F(x) = (f_1(x), f_2(x), f_3(x))$.

1.6 Dominance

La présence de plusieurs objectifs dans le problème d'optimisation multiobjectif modifie la terminologie employée, on ne parle plus de « d'optimum », mais de « Notion de domination »

Définition 1.11: ***la dominance notée par***$(x < y)$
On dit qu'une solution x *domine une solution* y *si seulement si les deux conditions suivantes sont vérifiées :*

- $\forall m \in [1, ..., M] \qquad f_m(x) \leq f_m(y)$
- $\forall m \in [1, ..., M] \qquad f_m(x) < f_m(y)$

Un exemple illustratif du principe de domination est donné par la figure suivante :

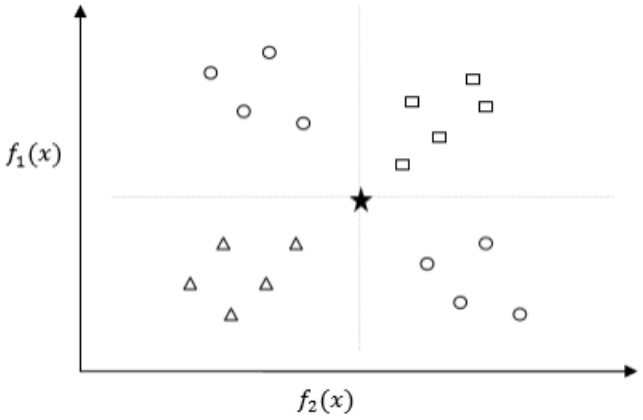


FIGURE 1.4 – Relation de dominance [3]

En observant la figure ci-dessus, on peut dire que l'étoile :

— *domine chaque carré.*

— *dominé par chaque triangle.*

— *équivalente à chaque anneau au sens de la dominance.*

Définition 1.12:
Dominance faible notée par : $(x \preceq y)$ *On dit qu'une solution* x *domine faiblement* y *si :*

$$\forall m \in [1 \dots M] \quad f_m(x) \leq f_m(y)$$

1.7 Optimalité de pareto

Soit un ensemble (P) de solution candidates d'un problème d'optimisation multiobjectif. L'ensemble $P' \subseteq P$, composé de tous les éléments de P qui ne sont dominés par aucun élément de P est dit sous-ensemble non dominé de l'ensemble P.

Définition 1.13:
optimalité locale au sens de Pareto : Un vecteur $\vec{x} \in \mathfrak{R}^n$ *est optimal localement au sens de Pareto s'il existe un réel* $\delta > 0$ *tel qu'il n'y ait pas de vecteur* $\vec{x}$ *qui domine le vecteur* $\vec{x}$ *avec* $\vec{x'} \in \mathfrak{R}^n \cap B(\vec{x}, \delta)$*, où* $B(\vec{x}, \delta)$ *représente une boule de centre* $\vec{x}$ *et de rayon* δ*.*

Définition 1.14: *Optimalité globale au sens de pareto*
Un vecteur $\vec{x}$ *est optimal globalement au sens de Pareto (ou optimal au sens de Pareto) s'il n'existe pas de vecteur* $\vec{x}$ *tel que* $\vec{x}$ *domine le vecteur* $\vec{x}$*.*

1.8 Front de Pareto et La surface de compromis

Définition 1.15: *Front de Pareto*
le front de Pareto est l'ensemble généralement non convexe constitué par l'image des solutions efficaces dans l'espace des critères. En effet, le front de Pareto est l'ensemble $Y_{ND} = F(X_E)$.

La figure (?? *montre le front de Pareto pour les différentes situations envisageables pour un problème deux critères. Notons que, le front de Pareto de chaque cas et la partie en gras de l'ensemble.*

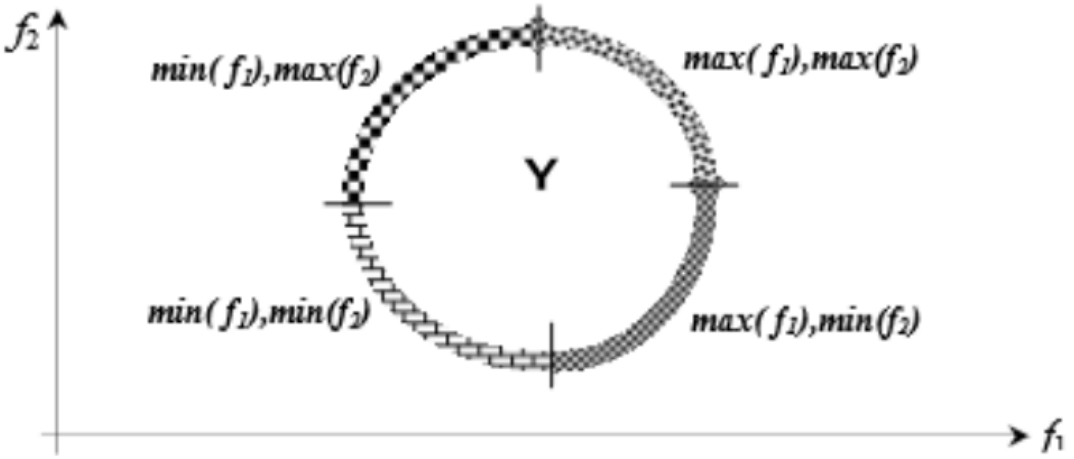


FIGURE 1.5 – Le front de Pareto [7]

1.8.1 Les points caractéristiques

Nous présentons quelques points caractéristiques très utilisent dans l'optimisation multiobjectif notamment le point idéal, le point nadir, le point anti-idéal, la matrice des gains.

Définition 1.16: ***point idéal***
Le point idéal $z* = (z*_1, z*_2, \ldots, z*p)$ *est le vecteur qui maximise (ou minimise) chacune des fonctions objectives* f_i

$$z_i^* = \{\max f_i(x), x \in S, i = 1, \ldots, p\}$$

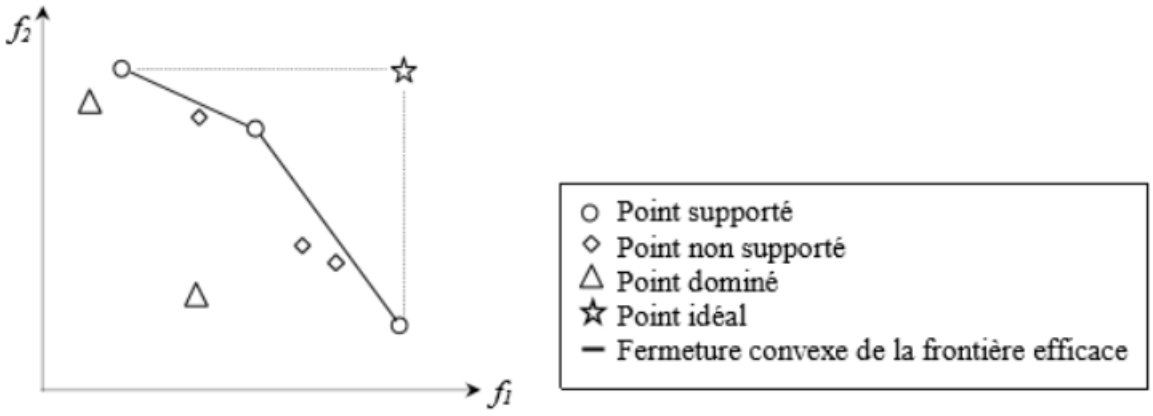


FIGURE 1.6 – Points caractéristiques d'un problème de maximisation [4]

Définition 1.17: *Point anti-idéal* [4]
Le point anti-idéal $z = (z_1, z_2, \ldots, z_p)$ *est le point qui minimise chacune des fonctions objectifs* f_i.

$$z_i = \{\max f_i(x), x \in S, i = 1, \ldots, p\}$$

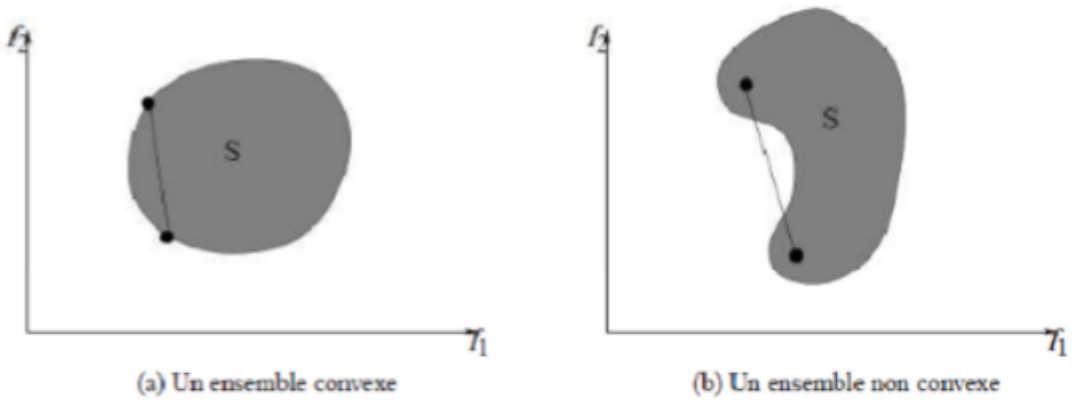
Définition 1.18: ***point Nadir****[5]*
Les coordonnées du point Nadir correspondent aux pires valeurs de chaque objectif des points ***du*** *front Pareto.*

Définition 1.19: *Matrice des gains[6]*
Nous appelons matrice de gains, une matrice dont les colonnes représentent les performances de k *points* $x_1^*, x_2^*, \ldots, x_k^*$:

$$\begin{pmatrix} z_1(x_1^*) & z_1(x_2^*) & \ldots & z_1(x_k^*) \\ z_2(x_1^*) & z_2(x_2^*) & \ldots & z_2(x_k^*) \\ . & . & . & . \\ z_k(x_1^*) & z_k(x_2^*) & \ldots & z_k(x_k^*) \end{pmatrix}$$

1.9 La convexité

Définition 1.20:
Un ensemble S *est convexe si, étant donné deux points distincts quelconques de cet ensemble, le segment qui relie ces deux points est contenu dans l'ensemble* S.



(a) Un ensemble convexe (b) Un ensemble non convexe

FIGURE 1.7 – Ensemble convexe/non convexe. [1]

1.10 Les Méthodes de résolution des problèmes d'optimisation combinatoire multi-objectives

1.10.1 Choix d'une méthode

La nature des variables,des domaines de définition et des critères à optimiser va influencer le choix de la méthode d'optimisation à utiliser. Il y a deux grandes familles de méthodes d'optimisation : **les méthodes exactes** et **les méthodes approchées**.

1.10.2 Les méthodes exactes

Les méthodes exactes cherchent à trouver de manière certaine la solution optimale en examinant de manière explicite ou implicite la totalité de l'espace de recherche. Elles ont l'avantage de garantir la solution optimale néanmoins le temps de calcul nécessaire pour atteindre cette solution peut devenir très excessif en fonction de la taille du problème (explosion combinatoire) et le nombre d'objectifs à optimiser. Ce qui limite l'utilisation de ce type de méthode aux problèmes bi-objectif de petite taille.
Ces méthodes génériques sont : " **Branch and Bound**", "**Branch and Cut** ","**Branch and Price**", "**Branch, Cut and Price**"," **Méthode à deux phases**".
D'autres méthodes sont moins générales, comme :"La programmation dynamique"," simplexe", "**La programmation linéaire en nombre entiers**"...etc.

1.10.3 Les méthodes approchées

Méthodes souvent inspirées de mécanismes d'optimisation rencontrés dans la nature. Elles sont utilisées pour les problèmes où on ne connaît pas d'algorithmes de résolution en temps polynomial et pour lesquels on espère trouver une solution approchée de l'optimum global. Elles cherchent à produire une solution de meilleure qualité possible dictée par des heuristiques avec un temps de calcul raisonnable en examinant seulement une partie de l'espace de recherche.Dans ce cas l'optimalité de la solution n'est pas garantie ni l'écart avec la valeur optimale.
Parmi ces heuristiques, on trouve les métaheuristiques qui fournissent des schémas de résolution généraux permettant de les appliquer potentiellement à tous les problèmes.
Nous distinguons parmi les méthodes approchées : "**les méthodes à solution unique**" et "**les méthodes à population de solution**"

- Les méthode à solution unique :
  Travaillent sur un seul point de l'espace de recherche à un instant donné en commençant par une solution initiale puis de l'améliorer itérativement en choisissant une nouvelle.
  a) **Les algorithmes de descente** :
  - **Descente simple** « simple descent » :
    Le principe est simple, à partir d'une solution existante, chercher aléatoirement une solution dans le voisinage et accepter cette solution si elle améliore la solution courante.
  - **Plus grande descente** « Deepest descent » :
    Est une version plus agressive de l'algorithme de descente. Au lieu de choisir une solution X' dans le voisinage de X, on choisit toujours la meilleure solution X' du voisinage de X. et on se trouve très rapidement bloqué dans un optimum local.[7]
  - **Multi-start descent** :
    Dans les deux algorithmes précédents, l'équilibre souhaité entre intensification et diversication n'existe donc plus. Un moyen très simple de diversifier la recherche peut consister à réexécuter un des algorithmes en prenant un autre point de départ. Comme l'exécution de ces algorithmes est souvent très rapide, on peut alors inclure cette

répétition au sein d'une boucle générale. On obtient alors un algorithme du type « Multi-start descent ».

b) **Recuit simulé** « Simulated annealing » :
Le recuit simulé a été Introduit par Kirkpatrick en 1982. Il tire son origine de la mécanique statistique en utilisant les critères de la méthode de Métropolis 1953. Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux basés sur les principes d'équilibre énergétique lors de la cristallisation des métaux. L'analogie avec une méthode d'optimisation est trouvée en associant une solution à un état du métal, on équilibre thermodynamique est la valeur de la fonction objective de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine. Le recuit simulé est une technique stochastique du typeMonte-Carlo généralisé pour la résolution approchée de problèmes de l'optimisation combinatoire basé sur un paramètre appelé température, qui sera ajusté au cours de la recherche. À partir d'une solution initiale il recherche dans son voisinage une autre solution de façon aléatoire qui peut être de moins bonne qualité permettant d'échapper aux optima locaux en acceptant temporairement une dégradation de la fonction objective. Initialement, la température est élevée autorisant une forte dégradation de qualité puis décroît pour diminuer la probabilité des dégradations importante.

c) **Recherche tabou** « Tabu search » :
Le principe de base est de poursuivre la recherche de solutions même lorsqu'un optimum local est rencontré en permettant des déplacements qui n'améliorent pas la solution et en utilisant le principe de mémoire pour éviter les retours en arrière (mouvements cycliques).
Contrairement au recuit simulé qui ne génère qu'une seule solution X' aléatoirement dans le voisinage $N(X)$ de la solution courante X, la méthode taboue, dans sa forme la plus simple, examine le voisinage $N(X)$ de la solution courante X. La nouvelle solution X' est la meilleure solution de ce voisinage (dont l'évaluation est parfois moins bonne que X elle-même). Pour éviter de cycler, une liste taboue (qui a donné le nom l'algorithme) est tenue à jour et interdit de revenir à des solutions déjà explorées.

— **Les méthode à population de solution :**

a) Les algorithmes évolutionnaires : On peut distinguer trois grandes classes d'algorithmes évolutionnaires :les algorithmes génétiques [Holland, 1975 ; Goldberg, 1989],les stratégies d'évolution [Schwefel, 1981] et la programmation évolutive [Fogel, 2000].es méthodes se distinguent par la manière de représenter l'information et par la façon de faire évoluer la population d'une génération à l'autre. Un algorithme évolutionnaire est typiquement composé de trois éléments fondamentaux :

   — une population : constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné.

   — un mécanisme d'évaluation des individus : permettant de mesurer l'adaptation de l'individu à son environnement.

   — un mécanisme d'évolution de la population : permettant, grâce à des opérateurs prédéfinis (tels que la sélection, la mutation et le croisement) d'éliminer certains individus et d'en créer de nouveaux.

b) **Les algorithmes de colonies de fourmis** :
   Les algorithmes à base de colonies de fourmis ont été introduits par Dorigo. Une des applications principales de la méthode originale était le problème du voyageur de commerce et depuis elle a considérablement évolué.
   Cette nouvelle métaheuristique imite le comportement de fourmis cherchant de la nourriture ; chaque fois qu'une fourmi se déplace, elle laisse sur la trace de son passage une odeur (la phéromone), avec plusieurs de ses congénères, elle explore une région en quête de nourriture. Face à un obstacle, le groupe des fourmis explore les deux côtés de l'obstacle et se retrouve, puis elles reviennent au nid avec de la nourriture. Les autres fourmis qui veulent obtenir de la nourriture elles aussi vont emprunter le même chemin.

c) Essaim de particulaire : Mets en jeu des groupes de particules sous forme de vecteurs se déplaçant dans l'espace de recherche. Chaque particule p est caractérisée par deux variables d'état (sa position courante $x(t)$ et sa vitesse courante $v(t)$). Cette technique repose sur deux règles :

— Chaque particule se souvient du meilleur pointe par lequel elle est passée au cours de ses évolutions et tend à y retourner,

— Chaque particule est informée du meilleur point connu au sein de la population et tend à s'y rendre..

1.11 Exemples de problèmes d'optimisation combinatoire

1.11.1 Le problème de sac à dos

Le "problème du sac-à-dos" est un problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Il doit son nom à l'analogie qui peut être faite avec le problème qui se pose au randonneur au moment de remplir son sac-à-dos : il lui faut choisir les objets à emporter de façon à avoir un sac le plus possible, tout en respectant son volume. Plus formellement, on peut le décrire de la façon suivante. Soit un ensemble de n éléments et une ressource disponible en quantité limitée, b. Pour j = 1 à n, on note pj le profit associé a la sélection de l'élément j et on note aj la quantité de ressource que nécessite l'élément j, s'il est sélectionné. Les coefficients pj et aj prennent des valeurs positives pour tout j = 1 à n. Le problème du sac-à-dos consiste à choisir un sous- ensemble des n éléments qui maximise le profit total obtenu, en respectant la quantité de ressource disponible.

On associe à chaque élément j une variable de sélection, x_j, binaire, égale à 1 si j est sélectionné, égale à 0 sinon.
Le profit total obtenu peut alors s'écrire comme la somme : $\sum_{j=1}^{n} p_j \ldots x_j$ et la quantité totale de ressource utilisée comme la somme : $\sum_{j=1}^{n} a_j \ldots x_j$. Le problème du sac-à-

dos se modélise donc sous la forme :

$$\begin{cases} \text{Max} \sum_{j=1}^{n} p_{j..} x_j \\ \text{S.C.} \quad \sum_{j=1}^{n} a_{j..} x_j \leq b \\ x_j \in \{0,1\} \quad \forall j = 1..n \end{cases}$$

Conclusion

Tout au long de notre chapitre, nous avons présenté les concepts fondamentaux de l'optimisation combinatoire, les classes des problèmes **NP** difficiles et leurs complexités, ensuite nous avons parlé de problèmes d'optimisation multiobjectif et les méthodes d'optimisation combinatoire de résolution exacte et approchée.

Chapitre 2

Les algorithmes génétique

2.1 Introduction

Le chapitre précédent discute les problèmes combinatoires, des exemples de ces derniers ainsi que les méthodes utilisées pour trouver des solutions adéquates. Parmi ces méthodes on trouve les plus répondues sont les algorithmes évolutionnaires.

Les algorithmes évolutionnaires (Evolutionary Algorithme , EA) sont des méta heuristiques basés sur des métaphores biologiques inspirées des mécanismes d'évolution darwinienne, qui simulent le processus de l'évolution naturelle. Les origines d'AE peuvent être rendues aux années 1950 et depuis les années 1970, la première fois par Darwin a inspiré bien plus tard les chercheurs en informatique. Plusieurs méthodes évolutionnaires ont été proposées, principalement il ya quatre types sont :

- programmation d'évolution
- Stratégies d'évolution
- Programmation Génétique
- Algorithme Génétique

Dans le reste de ce chapitre, on présente la description du principe de fonctionnement d'un algorithme génétique(mono-objectif et multi-objectif) et ces stratégies d'évolution.

2.2 Algorithmes génétiques

Les algorithmes génétiques sont des algorithmes d'explorations fondés sur les mécanismes de la sélection naturelle et de la génétique, utilisant le principe de la survie des structures-les mieux adaptées et les échanges d'information pseudo aléatoire. Les algorithmes génétiques sont des métaphores biologiques inspirées des mécanismes de l'évolution Darwinienne et de la génétique moderne et utilisées comme outils d'optimisation ou de recherche d'une solution de problème combinatoire.

2.2.1 Optimisation par les algorithmes génétiques

Définition 2.1:
L'optimisation consiste à rechercher la meilleure solution d'un problème au sens d'un ou de plusieurs critères choisis en respectant les caractéristiques du système et les contraintes qui lui sont imposées. Les AGs utilisés pour résoudre les problèmes d'optimisation nécessitent le codage de l'ensemble des paramètres d'origine du problème à optimiser en une chaîne de caractères d'alphabet, de longueur finie.

Ils n'utilisent que les valeurs de la fonction étudiée et pas celles de sa dérivée ou de toute autre fonction. Enfin, ils utilisent des règles de transition probabilistes et non déterministes.

Conclusion

Les AGs constituent une classe de stratégies de recherche réalisant un compromis entre l'exploration et l'exploitation. Ils représentent des méthodes qui utilisent un choix aléatoire comme outil pour guider une exploration intelligente dans l'espace des paramètres codés. Ce sont des algorithmes itératifs de recherche globale dont l'objectif est d'optimiser une fonction prédéfinie appelée fonction de coût ou fonction " fitness " F.

Vocabulaires des algorithmes génétiques :

Les algorithmes génétiques emploient un vocabulaire emprunté à la génétique naturelle. Ils travaillent sur un ensemble d'individus appelé population. nous allons

présenter quelques mots de vocabulaire relatifs à la génétique. Ces mots sont utilisés pour décrire un algorithme génétique :

Définition 2.2: *Gène*
Un gène est une suite de bases azotées (adénine (A), cytosine (C), guanine (G) et la thymine (T)) => la suite de symboles qui codent la valeur d'une variable.

Définition 2.3: *individu*
est une des solutions potentielles, représenté par un chromosome (génome).Un individu a deux représentations appelées phénotype et génotype.

Définition 2.4: *Génotype ou chromosome*
Le génotype est constitué de gènes situés sur des chromosomes stockés dans le noyau des cellules sous la forme d'une longue chaîne d'acide désoxyribonucléique (ADN). il donne une représentation codée d'une solution potentielle sous la forme d'un chromosome. Un chromosome est donc une suite de bits en codage binaire, appelé aussi chaîne binaire. Dans le cas d'un codage non binaire, tel que le codage réel, la suite A ne contient qu'un point, $A = [a, b]$ *tell que* $a, b \in <$.

Définition 2.5: *Phénotype*
Le phénotype est l'ensemble des protéines et des enzymes qui peuvent être fabriquées à partir de l'ADN.il représente une solution potentielle du problème à optimiser en utilisant la formulation originale du problème.le génotype et le phénotype sont similaires. Le génotype contient toutes les informations nécessaires à la définition complète du phénotype.

Définition 2.6: *Population*
C'est un ensemble d'individus codant un ensemble de solutions potentielles du problème à résoudre à l'aide d'un mécanisme approprié de codage.

Définition 2.7: *Génération*
Est un ensemble des opérations qui permettent de passer d'une population P à une population Q.

2.2.2 Principe de construction d'un algorithme génétique

Un algorithme génétique fonctionne typiquement à travers un cycle simple de quatre étapes :

1. On génère une population initiale de solutions de manière aléatoire ou avec des méthodes heuristiques qui essaient de fabriquer de bonnes solutions.
2. On évalue les solutions de la population en cours et on retient la meilleure de toutes les solutions construites.
3. On choisit les individus de la population courante qui vont survivre et se reproduire selon une fonction d'adaptation (fitness).
4. On passe de la population en cours à une nouvelle population en utilisant des opérateurs génétiques, généralement de "croisement" et/ou de "mutation". Certains individus étant conservés par simple recopie.
5. On retourne en (2) tant qu'un critère d'arrêt n'est pas vérifié, par exemple un nombre maximal de populations générées ou une durée maximale d'exécution sinon fin.

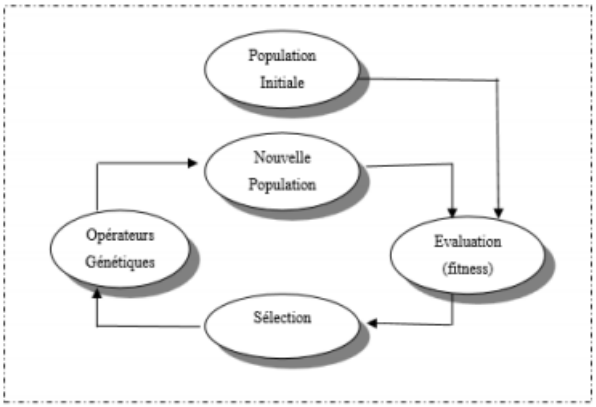


FIGURE 2.1 – Cycle génétique

2.2.3 Eléments de base pour la construction d'un AG

Codage

Le codage est une modélisation d'une solution d'un problème donné sous forme d'une séquence de caractères appelés chromosome ou chaque caractère, dit aussi gène, représente une variable ou une partie du problème. La tâche principale consiste à choisir le contenu des gènes qui facilite la description du problème et respecte ses contraintes.

La littérature définit quatre types de codage : binaire , réel , alphabétique et à base N.

- **codage binaire**

  Le codage binaire est le premier à être utilisé dans les algorithmes génétiques, car c'est le plus simple à mettre en oeuvre avec un alphabet réduit à deux éléments $\{0, 1\}$, rajouter à ça l'existence de fondements théoriques "théorie des schémas " et la facilité de mise en oeuvre des opérateurs génétiques.[3]

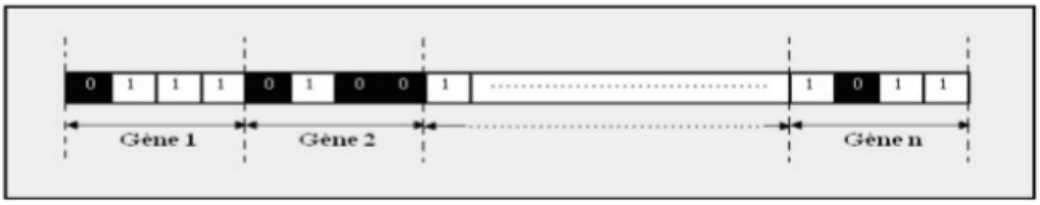


FIGURE 2.2 – Structure d'un chromosome en codage binaire [32].

- **codage réel**

  Contrairement au codage binaire, un gène est représenté par une suite de nombres entiers ou de nombres réels. Ce type de codage peut être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle.

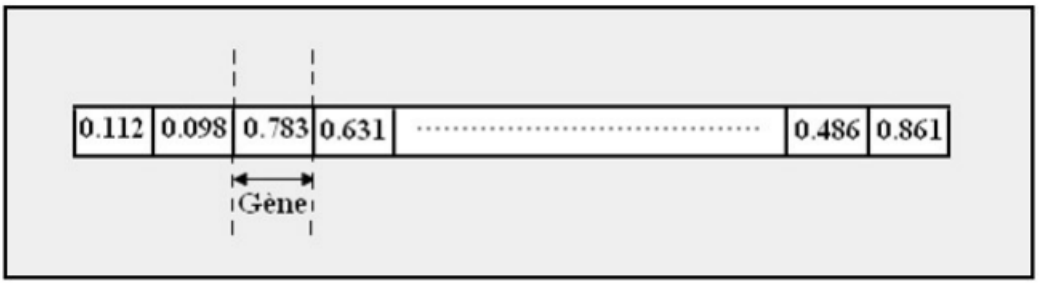


FIGURE 2.3 – Structure d'un chromosome en codage en nombres réels.

- **codage alphabétique**

  l'individu est représenté comme une suite de chaine de caractère A,G,T,C.

A	-	G	T	C

FIGURE 2.4 – codage alphabétique.

- **Le codage à base N**

  Un autre inconvénient majeur du codage binaire est présent : que se passe-t-il si les gènes correspondent à un ensemble discret de valeurs dont le cardinal n'est pas une puissance de 2 ? Il sera possible que des combinaisons binaires ne correspondant pas à un codage valide puissent apparaître. Pour surmonter ce problème, d'autres chercheurs ont inventé un nouveau codage appelé codage à base N où un chromosome est essentiellement constitué d'un ensemble d'éléments qui sont des chiffres exprimés dans une base de numération N en permettant de représenter N valeurs discrètes.

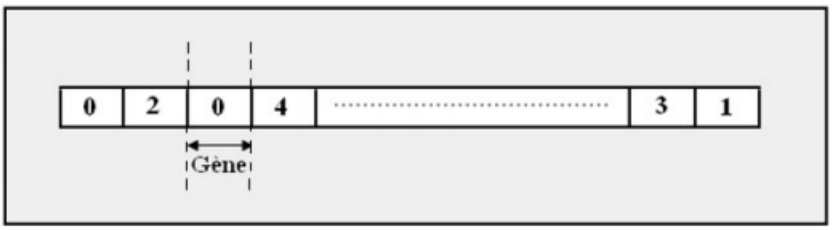


FIGURE 2.5 – Structure d'un chromosome en codage à base$n(n = 5)$.

Initialisation de la population

La population initiale est un ensemble de solutions choisies dans l'espace de recherche du problème d'optimisation .Le choix de la population initiale d'individus conditionne fortement la rapidité de l'algorithme .Si la position de l'optimum dans l'espace d'état est totalement inconnue, il est naturel de générer aléatoirement des individus en faisant des tirages uniformes dans chacun des domaines associés aux composantes de l'espace d'état en veillant à ce que les individus produits respectent les contraintes. Si par contre, des informations à priori sur le problème sont disponibles, il paraît bien évidemment naturel de générer les individus dans un sous-domaine particulier afin d'accélérer la convergence.

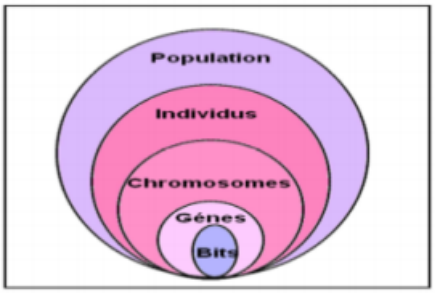


FIGURE 2.6 – Les cinq niveaux d'organisation de notre algorithme génétique

La fonction d'adaptation

La fonction d'adaptation, ou fitness, associe une valeur pour chaque individu. Cette valeur a pour but d'évaluer si un individu est mieux adapté qu'un autre à son environnement. Ce qui signifie qu'elle quantifie la réponse fournit au problème pour une solution potentielle donnée. Ainsi les individus peuvent être comparés entre eux. Cette fonction, propre au problème, est souvent simple à formuler lorsqu'il ya peu de paramètres. Au contraire, lorsqu'il ya beaucoup de paramètres ou lorsqu'ils sont corrélés, elle est plus difficile à définir.

2.2.4 Les opérateurs de l'algorithme génétique

Les opérateurs jouent un rôle prépondérant dans la possible réussite d'un AG. Nous en dénombrons quatre principaux : l'opérateur de sélection, de croisement, mutation et de remplacement.

2.2.5 Les opérateurs de l'algorithme génétique

Les opérateurs jouent un rôle prépondérant dans la possible réussite d'un AG. Nous en dénombrons quatre principaux : l'opérateur de sélection, de croisement, mutation et de remplacement.

L'opérateur de sélection

L'opérateur de sélection est chargé de " favoriser les meilleurs individus. Plus formellement, l'opérateur de sélection va générer à partir de la population courante une nouvelle population par copies des individus choisis de la population courante. La copie des chaînes s'effectue en fonction des valeurs de la fonction d'adaptation. Ce

procédé permet de donner aux meilleures chaînes, une probabilité élevée de contribuer à la génération suivante. Cet opérateur est bien entendu une version artificielle de la sélection naturelle, la survie darwinienne des chaînes les plus adaptées.

On trouve essentiellement quatre types de méthodes de sélection différentes :

— La sélection par roulette.

— la sélection par rang.

— La sélection par tournois.

— la méthode élitiste.

la sélection par roulette

elle consiste à créer une roue de loterie biaisée pour laquelle chaque individu de la population occupe une section de la roue proportionnelle à sa valeur d'évaluation. Ainsi, même les individus les plus faibles ont une chance de survivre. Si la population d'individus est de taille égale à N, alors la probabilité de sélection d'un individu xi notée p(xi) est égale à :

$$p(xi) = F(xi) \sum F(xi)$$

En pratique, on calcule pour chaque individu xi sa probabilité cumulée et on choisit aléatoirement un nombre r compris entre 0 et 1.

I la sélection par rang

elle consiste à ranger les individus de la population dans un ordre croissant (ou décroissant selon l'objectif) et à retenir un nombre fixé de génotypes. Ainsi, seuls les individus les plus forts sont conservés.

La sélection par rang

est la même que par roulette , mais les proportions sont en relation avec la rang plutôt qu'avec la valeur de l'évaluation , avec cette méthode tous les individus ont une chance d'être sélectionnés.

I La sélection par tournoi

elle consiste à choisir aléatoirement deux ou plusieurs individus et à sélectionner le plus fort. Ce processus est répété plusieurs fois jusqu'à l'obtention de N individus.

L'avantage d'une telle sélection est d'éviter qu'un individu très fort soit sélectionné plusieurs fois.

ILa méthode élitiste

Elle consiste à copier un ou plusieurs des meilleurs chromosomes dans la nouvelle génération (les meilleurs individus de la population Pt dans la population $Pt + 1$). Ensuite, on génère le reste de la population selon l'algorithme de reproduction usuelle . Cette méthode améliore considérablement les algorithmes génétiques, car elle permet de ne pas perdre les meilleures solutions.

L'opérateur de croisement

Il constitue la pièce maitresse de l'algorithme génétique, il agit sur la population des parents pour donner naissance à une population enfants. Au départ la population des parents est divisée en deux sous population de taille (N/2), ensuite un couple, où chaque parent issu d'une sous population est choisi pour participer au croisement avec une probabilité Pc. Si le croisement aura lieu, on tire alors aléatoirement une valeur (locus) représentant un point de croisement parmi les l allèles constituants le chromosome des parents. Finalement, on procède à un échange des séquences chromosomiques entre les deux parents.

La littérature définit plusieurs opérateurs de croisement.

croisement binaire Ce croisement peut avoir recours à plusieurs types en occurrence :

I Croisement en un point

On choisit au hasard un point de croisement, pour chaque couple. Notons que le croisement s'effectue directement au niveau binaire, et non pas au niveau des gènes. Un chromosome peut donc être coupé au milieu d'un gène, sauf si un bit représente une gêne.

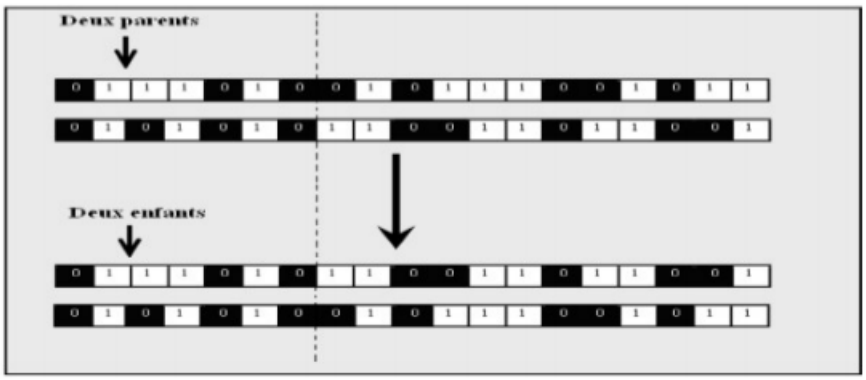


FIGURE 2.7 – Principe du croisement binaire en un point.

I Croisement en deux point

On choisit au hasard deux points de croisements successifs. Cet opérateur est généralement considéré comme plus efficace que le précédent.[1]

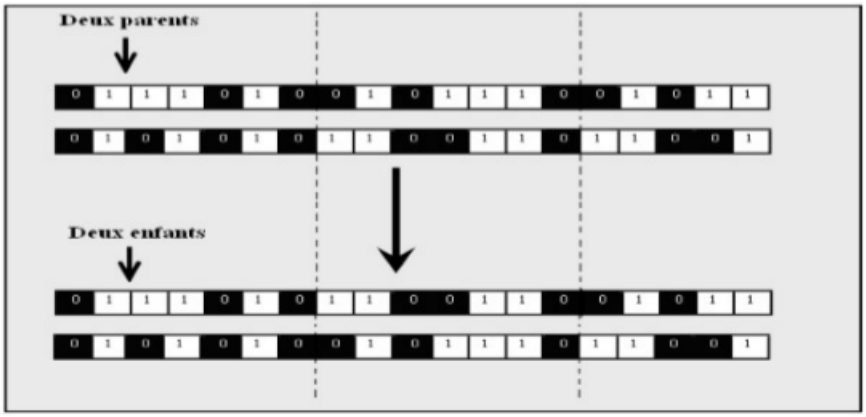


FIGURE 2.8 – Principe du croisement en deux points.

I Croisement uniforme

Ce type de croisement a été proposé par Syswerda . Il consiste à choisir avec la même probabilité un allèle de l'un ou de l'autre parent, pour transmettre sa valeur à la même position, aux enfants.

$$\begin{aligned} P1 &= 0000001 \\ P2 &= 1111110 \\ M &= 1101001 \end{aligned}$$

M représente le masque de transmission dont le principe est le suivant : si la valeur dans le masque est égale à 1, alors la valeur de l'allèle du parent 1 passe à l'enfant 1 (respectivement des parents 2 passes à l'enfant 2) ; sinon la valeur de l'allèle du parent 1 passe à l'enfant 2 (respectivement des parents 2 passes à l'enfant 1), tout en respectant la même position de l'allèle.

$$E1 = 0010111$$
$$E2 = 1101000$$

croisement réel Le codage réel requiert des opérateurs génétiques spécifiques pour la manipulation des chromosomes. Il est de plusieurs types : I l'opérateur de Croisement PMX(Partially Mapped Crossover) PMX est un opérateur de croisement à deux points de coupure qui définit un segment de même longueur dans chacun des parents $P1$ et $P2$. Les segments sont indiqués avec une trame foncée dans la partie (a) de la figure. Ces segments sont copiés vers les enfants $E1$ et $E2$. $E1$ a hérité du segment de $P2$ et E2 de P1, comme le montre la partie (b) de la figure.

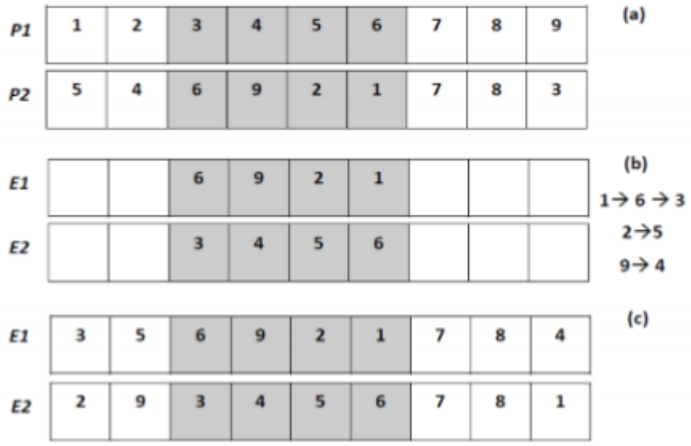


FIGURE 2.9 – l'opérateur de croisement PMX.

I l'opérateur de Croisement CX (Cycle Crossover)

CX est un opérateur qui satisfait la condition suivante : chaque gène d'un enfant provient de l'un des parents à la même position. Les enfants sont donc formés en copiant un gène d'un parent et en éliminant l'autre à la même position puisqu'il va appartenir au deuxième enfant. Une fois que les positions occupées sont copiées

par élimination, on a complété un cycle. Les places restantes des deux enfants sont complétées par les parents opposés.

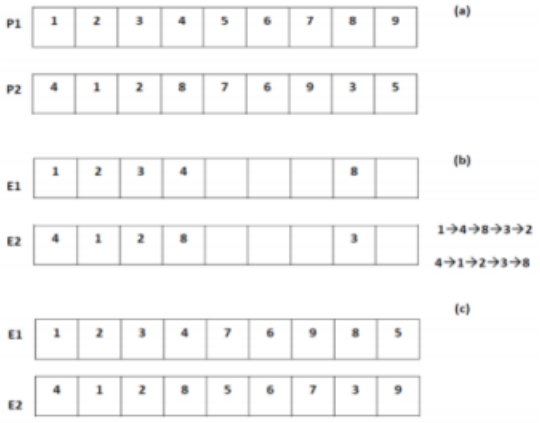


FIGURE 2.10 – l'opérateur de croisement CX.

I l'opérateur de Croisement OX (Order Crossover)

OX est un opérateur avec deux points de coupure qui copie le segment formé par ces deux points de coupure dans les deux enfants. Par la suite, il recopie, à partir du deuxième point de coupure ce qui reste des gènes dans l'ordre du parent opposé en évitant les doublons.

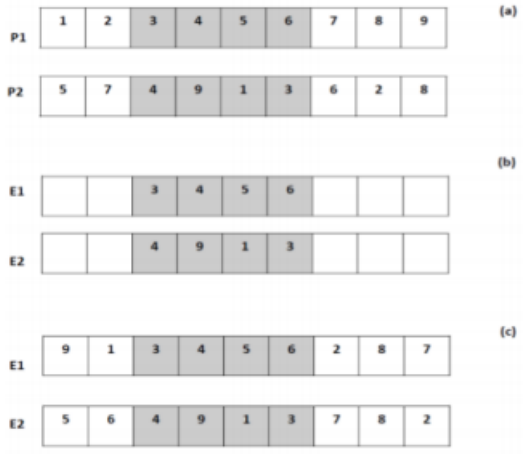


FIGURE 2.11 – l'opérateur de croisement OX.

L'opérateur de Mutation

Après le croisement, un opérateur de mutation, est un opérateur qui agit sur la population des enfants, son rôle est de garantir une diversité dans la population générée en introduisant de nouveaux gènes au patrimoine génétique de la population et ce avec une probabilité Pm souvent très faible.

Généralement, L'opérateur de mutation modifie de manière complètement aléatoire les caractéristiques d'un ou plusieurs individus de la population des enfants.

Mutation en codage binaire Dans un algorithme génétique simple, la mutation en codage binaire est la modification aléatoire occasionnelle de la valeur d'un caractère de la chaîne.

Mutation en codage réel Pour le codage réel, les opérateurs de mutation les plus connus et les plus utilisés sont les suivants [22] I Mutation par inversion L'inversion inverse l'ordre de visite des gènes entre deux points de coupure choisis aléatoirement.[1]

FIGURE 2.12 – Mutation par inversion.

I Mutation par déplacement

Une séquence est sélectionnée au hasard et déplacée vers une position elle- même tirée au hasard.

FIGURE 2.13 – Mutation par déplacement.

L'opérateur de remplacement

Cet opérateur est le plus simple, son rôle consiste à réintroduire les descendants obtenus par applications successives des opérateurs de sélection, de croisement et de mutation dans la population de leurs parents. On distingue essentiellement deux méthodes de remplacement différentes :

I Le remplacement stationnaire Dans ce cas, les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives, et le nombre d'individus de la population ne varie pas tout au long du cycle d'évolution simulé, ce qui implique donc d'initialiser la population initiale avec un nombre suffisant d'individus.

I Le remplacement élitiste Dans ce cas, on garde au moins l'individu possédant les meilleures performances d'une génération à la suivante. En général, on peut partir du principe qu'un nouvel individu (enfant) prend place au sein de la population que s'il remplit le critère d'être plus performant que le moins performant des individus de la population précédente.[26]

2.2.6 Les paramètres de l'algorithme génétique :

Les opérateurs de l'AG sont guidés par un certain nombre de paramètres généralement fixé à l'avance et dont dépend très fortement la " bonne " convergence de l'algorithme. Présentons brièvement les principaux paramètres de l'AG.

- I La taille de la population et la longueur du codage de chaque individu. Il est conseillé de prendre comme taille de la population la valeur correspondant à la longueur du codage des individus. En effet, si cette taille est très grande, l'évaluation de tous les individus de la population peut s'avérer trop longue. Par contre, si elle est très petite, l'algorithme peut converger trop rapidement .

- **I la probabilité de croisement** : La probabilité de croisement Pc dépend de la forme de la fonction d'évaluation. Son choix est généralement expérimental et sa valeur est très souvent prise entre 0.5 et 0.9. Plus elle est élevée , plus la population subit des changements importants. Ainsi, la convergence est très rapide si le taux de croisement est proche de 1.
- **I la probabilité de mutation** : le taux de mutation est généralement faible, le risque d'un taux élevé étant de modifier les meilleurs individus et ainsi, de s'éloigner de l'optimalité.

2.2.7 Domaine d'application

- Les applications des AG sont multiples : optimisation de fonctions numériques difficiles (discontinues, multimodales, bruitées...), traitement d'images (alignement de photos satellites, reconnaissance de suspects...), optimisation d'emplois du temps, optimisation de design, contrôle de systèmes industriels, apprentissage des réseaux de neurones, etc.
- Les AE peuvent être utilisés pour contrôler un système évoluant dans le temps (chaîne de production, centrale nucléaire...) car la population peut adapter des conditions changeantes. En particulier, ils supportent bien l'existence de bruit dans la fonction à optimiser.Ils peuvent aussi servir à déterminer la configuration d'énergie minimale d'une molécule ou à modéliser le comportement animal.
- Les AE sont également utilisés pour optimiser des réseaux (câbles, fibres optiques, mais aussi eau, gaz...), des circuits VLSI des antennes...
- Ils peuvent être utilisés aussi pour trouver les paramètres d'un modèle petit signal à partir des mesures expérimentales. Des commutateurs optiques adiabatiques ont été optimisé à l'aide des AE chez SIEMENS AG. On envisage l'intégration d'AE dans certaines puces électroniques afin qu'elles soient capables de se reconfigurer automatiquement en fonction de leur environnement (Evolving Hardware en anglais).
- Les AE peuvent être aussi utiles dans : l'étude du vivant, du monde réel (marchés économiques, comportements sociaux, systèmes immunitaires, etc ...), la programmation automatique (programmes LISP, automates cellulaires, etc ...), l'apprentissage (classiffication, prédiction, robotique, etc ...)

2.3 Algorithmes génétique multi-objectif

Plusieurs méthodes ont été proposées pour le traitement des problèmes multiobjectifs. Ces méthodes peuvent être classées principalement en deux catégories :

- Méthode agrégative :
  C'est l'une des premières méthodes utilisée pour résoudre les problèmes d'optimisation multi- objectifs (MO). Elle consiste à transformer le problème MO en un problème mo-noobjectif en combinant les composantes fi du vecteur objectif du problème en une seule fonction scalaire f. Il existe dans la pratique, différentes façons de construire la fonction f.
  La plus classique et la plus utilisée se ramène à une simple somme pondérée des objectifsfi (agrégation additive) :

$$\sum ni = 1Wi * fi$$

  Où les paramètres Wi sont les poids de pondération.

- Méthode non agrégative :

  Comme nous l'avons signalé précédemment, la méthode agrégative peut être utilisée de façon séquentielle pour obtenir le front de Pareto pour un problème d'optimisations multi- objectifs. Toutefois, cette approche n'est généralement pas satisfaisante car le nombre d'exécutions successives nécessaires pour déterminer les différents compromis conduit alors à un nombre d'évaluations de critères prohibitifs. Ainsi, pour surmonter cette difficulté, on préfère utiliser des méthodes permettant d'une part de trouver l'ensemble de solutions Pareto optimal en une seule exécution et d'autre part de s'affranchir des problèmes de mise à l'échelle des objectifs. Parmi ces méthodes on peut citer : I Vector Evaluated Genetic Algorithm (VEGA) I Multiple Objective Genetic algorithm (MOGA) I Niched Pareto genetic algorithm (NPGA) I Nondominated sorting genetic algorithm (NSGA) et sa deuxième version (NSGA-II) La figure suivante présente une taxonomie de ces algorithmes

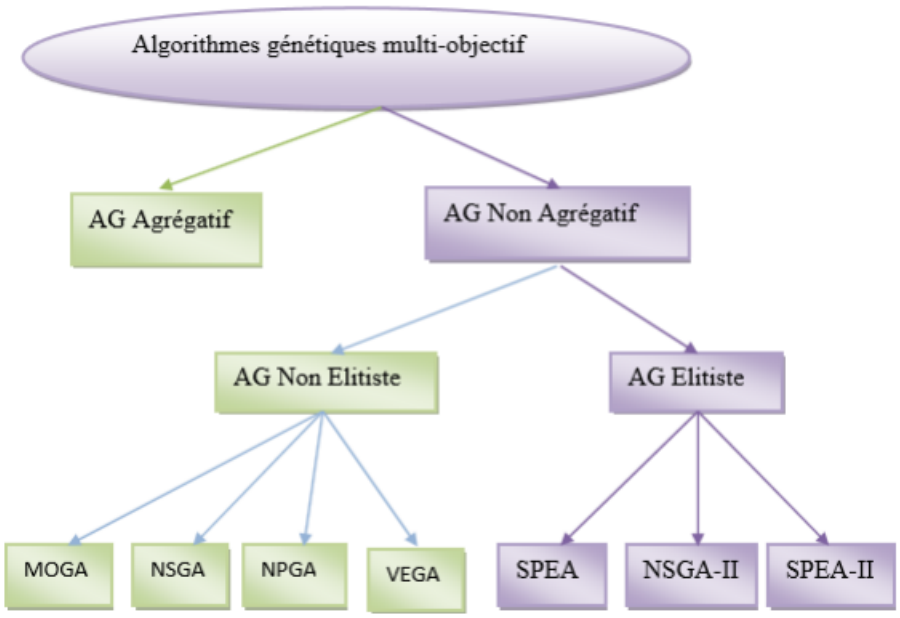


FIGURE 2.14 – Arborescence des algorithmes génétiques multi-objectif.

2.3.1 Algorithme génétique élitiste de tri non-dominé (NSGA-II)

Définition 2.8:

Le NSGA-II est l'un des algorithmes les plus populaires dans le domaine de l'optimisation multi-objectif, la notation NSGA-II est une abréviation de (Nondominated Sorting Genetic Algorithm), et le nombre deux, désigne le numéro de la version. Le NSGA-II a été une réponse aux critiques faites sur l'algorithme original le NSGA. La complexité de calcul O(MN3), (dont M désigne le nombre d'objectifs et N la taille de la population), le non-élitisme de l'algorithme et enfin le besoin de spécifier le paramètre de la procédure de sharin1

principe de l'algorithme NSGA-II

Dans cette procédure, on distingue quatre étapes :

a- Génération d'une population enfants Qt à partir de la population parente Pt par l'application des opérateurs génétiques (sélection, croisement et mutation). Cette manipulation est suivie par la fusion de deux populations enfants et parents dans une seule population globale Rt.

b- Classement des individus de la population globale Rt en plusieurs fronts de Pareto dans l'ordre croissant ($F1, F2, F3$,etc.).

c- Reconstruction de la population pare pour la génération suivante Pt+1 par la sélection d'individus appartenant aux premiers fronts de Pareto ($F1$,$F2$ et $F3$) tels que : g constituent l'ensemble des critiques faites sur le NSGA

$$Pt + 1 = \sum jK = 1FK \leq N$$

.où :

N : La taille de la population.

j : Nombre de fronts sélectionnés.

FK : Nombre d'individus dans le front de Pareto d'ordre k. d- Application de la technique de crowding distance sur le front de Pareto $Fj + 1$ pour compléter la population $Pt + 1$ à une taille de N.

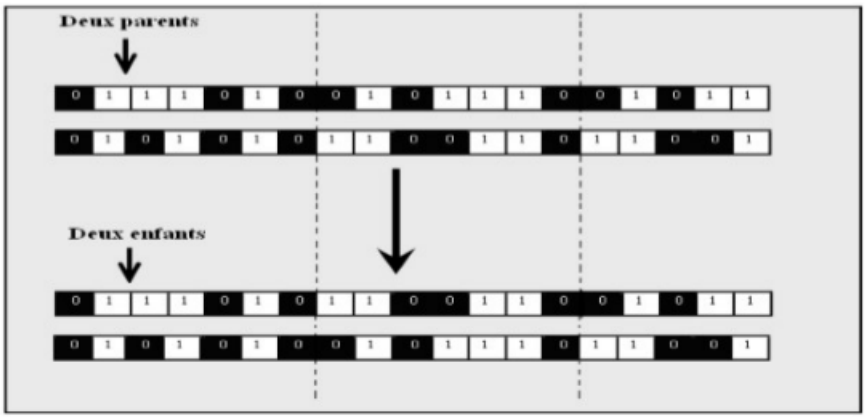


FIGURE 2.15 – Schéma de l'évolution de l'algorithme NSGA -II.

Classiffication des individus

Dans un premier temps, avant de procéder à la sélection, on affecte à chaque individu de la population un rang correspondant au front auquel il appartient. Tous les individus de la population non dominés appartenant au front optimal (1er front) reçoivent un rang égal à 1. Ces individus sont retirés de la population et l'opération est répétée ainsi de suite pour les fronts successifs suivants en incrémentant le rang.

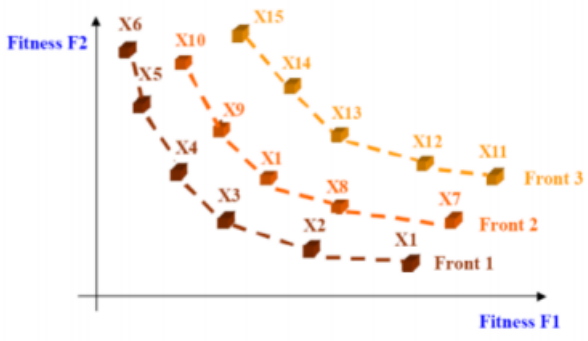


FIGURE 2.16 – Classiffication des individus suivant le rang de Pareto.

Distance de crowding

La distance de crowding d'une solution i (ou d'un individu) se calcule en fonction du paramètre formé par les solutions des mêmes fronts les plus proches de i sur chaque objectif. Le calcul de la distance de crowding nécessite avant tout le tri des solutions selon chaque objectif, dans un ordre ascendant. Ensuite, pour chaque objectif, les individus possédant les valeurs-limites (la plus petite et la plus grande valeur de fonction objective) se voient associés à une distance infinie(∞). Pour les autres solutions intermédiaires, on calcule une distance de crowding égale à la différence normalisée des valeurs de fonctions objectives de deux solutions adjacentes. Ce calcul est réalisé pour chaque fonction objective.

La distance de crowding d'une solution est calculée sous la forme suivante : Données :

I	:	Vecteur d'indice.
Fk	:	Sous-ensemble de Pareto. di : Crowded distance de la solution i.
Fi	:	Fonction d'objectif.
$fmax\ m$	:	Valeur maximale de la fonction d'objectif m.
$fmin\ m$	:	Valeur minimale de la fonction d'objectif m.

début initialiser $L = kFk$ pour chaque $zi \in F$ faire Initialiser $di = 0$ pour chaque fonction d'objectif fj faire Trouver le vecteur d'indice $Ij = Trier(fj, <)$ pour chaque fonction d'objectif fj faire Assigner $dmI = dmI = \infty$ pour chaque $zii = 2...(L-1)$ faire

Calculer :

$$dImj = dImj + |fmImj + 1 - fmImj - 1fmaxm - fminm|$$

end.

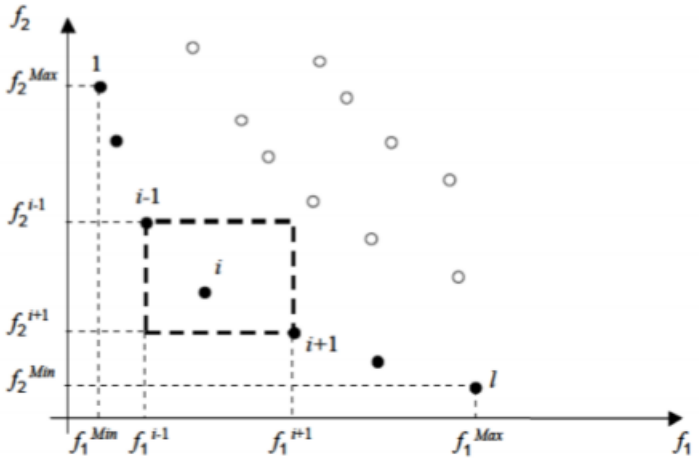


FIGURE 2.17 – Distance de crowding.

Convergence et diversité des solutions des algorithmes multi-objectifs

Au cours de ces dernières années, la recherche sur des algorithmes évolutionnaires a démontré leur capacité en résolvant les problèmes d'optimisation multiobjectifs, où le but est de trouver, en une seule exécution de l'algorithme, un certain nombre de solutions proches de l'optimale de Pareto. Beaucoup d'études ont été menées sur différents algorithmes évolutionnaires pour montrer leurs aptitudes à progresser les meilleures solutions, avec une bonne diversité (distribution homogène des solutions sur le front de Pareto) des solutions.
Ce pendant, aucun des algorithmes évolutionnaires multiobjectifs a une preuve formelle deconvegence avec une large diversité, aux véritables solutions en optimale de Pareto. D'après des études qui ont été mené sur plusieurs algorithmes multiobjectifs, le NSGA-II est classé parmi les meilleurs Et le plus populaire algorithme jusqu'à ce jour en terme de convergence et de diversité de solutions.

2.4 Critère d'arrêt

2.4.1 Mono-objectif

Pour les algorithmes génétiques mono-objectifs , trois types de critère d'arrêt de base sont connus :

- l'arrêt après un nombre fixe d'itérations (ou d'évaluations).
- L'arrêt dès que la valeur de la performance du meilleur individu a atteint un certain seuil.
- L'arrêt après qu'un certain nombre d'itérations a été faites sans amélioration.

2.4.2 Multi-objectif

Notons que le second des trois critères énumérés ci-dessus utilise les valeurs absolues de la performance des individus. Dans le cas des algorithmes génétiques multiobjectifs basés sur la domination de Pareto, la notion de qualité de chaque individu est relative aux autres membres de la population. En l'absence de préférences particulières, la performance absolue d'un ensemble non dominé ne peut être évaluée qu'en faisant référence à la surface de Pareto exacte. Il est donc nécessaire d'avoir une connaissance de la solution avant l'optimisation. Ce critère est donc valable dans le cas de fonctions teste, mais pas dans le cadre d'une application pratique.

Conclusion

Les algorithmes génétiques (AG's) semblent être une solution intéressante pour résoudre le problème d'optimisation multiobjectif. Dans ce chapitre, nous avons présenté le principe de fonctionnement et les différents opérateurs des algorithmes génétiques monoobjectifs, les algorithmes génétiques multiobjectifs, Ensuite, on a décrit l'algorithme génétique NSGA- II qui est utilisé dans ce mémoire comme algorithme de référence pour les problèmes d'optimisation multiobjectifs.

Chapitre 3

les algorithmes evolutionnaire pour les différents probléme de transport

Les problèmes de transport, appelés aussi problèmes de routage, modélisent des problèmes réels liés au transport de marchandises ou de personnes. A_n d'introduire le problème de routage de véhicules, nous parlerons de deux autres problèmes de transport : le problème du voyageur du commerce généralisé et le problème du postier chinois.

3.1 Le problème de tournées de véhicules

Les problèmes de tournées de véhicules sont largement considérés dans la littérature pour leurs intérêts applicatifs. Par ailleurs, des publications sont régulièrement proposées que ce soit sur la problématique initiale ou sur des variantes du problème. Dans ce chapitre nous présentons un rapide état de l'art sur les problèmes de tournées de véhicules et sur les approches de résolution que nous utilisons dans cette thèse. Nous nous contentons donc de présenter les principes et pour le détail nous renvoyons le lecteur à des publications spécialisées.

3.1.1 VRP : Définitions

Le problème de tournées de véhicules (VRP) consiste en la recherche des meilleures itinéraires pour une flotte de véhicules partant d'un dépôt afin de visiter un ensemble

de clients. L'évaluation de ces itinéraires se fait selon une fonction objectif prédéfinie, telle que le temps de parcours, une distance ou un cont global à minimiser. Ce problème est une extension classique du problème de voyageur de commerce, et tout comme ce dernier, il appartient à la classe des problèmes d'optimisation NP-Complets. La première formulation connue du VRP est celle de Dantzig et Ramser 25 sous le nom de "Truck Dispatching Problem" et depuis il est un problème largement étudié dans la littérature. Le VRP est modélisé par un graphe $G = (V, A)/V = C \cup \{0\}$ où $\{0\}$ est le dépôt, le point de départ de toutes les tournées. C est l'ensemble des clients à visiter et A l'ensemble des arcs reliant les clients. Le poids de chaque arc (i, j) de l'ensemble A correspond au cont (distance, temps ..) engendré par le véhicule en traversant cet arc. Les véhicules doivent commencer et finir leurs tournées au dépôt. L'objectif est donc de visiter chaque client exactement une fois tout en vérifiant les contraintes qui lui sont propres telles qu'une demande à satisfaire (livraison de marchandises) ou un service à effectuer (temps nécessaire). Un ensemble de visites faites par un véhicule est une tournée. Chaque tournée doit respecter les contraintes suivantes :

— Un client ne peut être servi que par un et un seul véhicule

— Chaque véhicule effectue une seule tournée

— Tous les clients doivent être desservis

— Capacité et/ou temps maximums à respecter et/ou nombre de véhicules à satisfaire. Cette dernière contrainte permet de distinguer le TSP (problème du voyageur de commerce) du VRP et est différente selon les auteurs. Ainsi la définition du sigle VRP lui même est source de confusion 85 . VRP de base.

Au delà de sa définition globale, le VRP possède plusieurs déclinaisons inspirées des différentes problématiques réelles. Nous définissons dans ce qui suit le cas classique du VRP puis présentons dans la section suivante brièvement d'autres variantes.

Il est commun dans la littérature de nommer "VRP" le cas du problème de tournées de véhicules avec contraintes de capacité ou Capacitated VRP (CVRP) qui modélise la problématique de livraison de marchandises. Chaque client ayant une demande qui est la quantité de marchandise à lui livrer. Les véhicules peuvent transporter jusqu'à une quantité maximale de marchandises. En plus des autres contraintes, il est nécessaire de vérifier qu'un véhicule puisse respecter les demandes de tous les

clients sur sa tournée et donc que la somme des demandes sur sa tournée ne dépasse pas sa capacité.

3.1.2 Domaines d'applications

Le VRP permet de modéliser de nombreux types d'applications dans le domaine du transport et de la distribution.
Les exemples donnés dans le tableau **??**

Secteur économique	**Applications**
Industrie automobile	Distribution de pièces de rechange
La livraison des produits	Carburant, gaz naturel, béton
Transport de nourriture	Grands détaillants ou petits magasins
La livraison de nourriture aux particuliers	Lait, aliments surgelés , plats préparés livrés à domicile
Vente au détail	La livraison d'appareils
Santé	Médicaments aux pharmacies
Presse	Des journaux et des magazines
Secteur bancaire	La livraison d'argent aux banques, perception d'argent des commerçants et courriers
Secteur public	Poubelles domestiques; poubelles publiques, nettoyage des rues, sablage des routes en hiver
Fabriquant	Organisation des mouvements d'une flotte de robots de transport
Industrie	Approvisionnement en parties et marchandises parmi différents emplacements
Agriculture	Collecte d'animaux, lait, céréales, livraison de nourriture animale
Industrie de transport	Entreprises de collecte et de livraison

FIGURE 3.1 – Différentes applications du VRP.

peuvent concerner la conception, d'un réseau de transport, et la gestion quotidienne de la collecte et de la livraison de produits. Le VRP avec fenêtres de temps (VRPTW) qui prend en compte des contraintes temporelles, permet d'approcher les problèmes réels d'une manière plus ne et plus réaliste que le problème VRP de base. Cependant, les diminutions incessantes des délais, la recherche des réductions de coûts, les règlements liés à des facteurs sociaux et environnementaux, imposent de prendre en compte dans la modélisation le contexte dynamique qui est une caractéristique incontournable des problèmes réels rencontrés dans le domaine du transport

et de la distribution.

3.1.3 Variantes du problème VRP

De multiples applications du VRP ont généré diverses variantes de ce problème et la littérature sur ces variantes est immense. Dans cette section nous présentons premièrement quelques variantes et dans un deuxième temps nous détaillerons plus particulièrement la variante périodique du VRP qui est la variante du VRP qui se rapproche le plus de notre problématique.

— Problèmes de tournées de véhicules avec fenêtres de temps (VRPTW).

— Le problème de tournées de véhicules multi-dépôts (MDVRP).

— Le problème de tournées de véhicules avec livraison et collecte (VRPB).

— Le problème de tournées de véhicules avec ramassage et livraison (VRPSD).

— Le problème de tournées de véhicules Multi-trips (VRPMT).

— Le problème de tournées de service (TRP).

3.1.4 Approches de résolution des problèmes de tournées : classiffication générale

Comme les autres problèmes d'optimisation combinatoire, le problème de routage de véhicules a été étudié et résolu par des méthodes exactes, des heuristiques spécifiques ainsi que par des métaheuristiques. Ces trois familles correspondent à la classication générale des méthodes de résolution ([Hao et al., 1999], [Gen-dreau and Potvin, 2005]) et sont illustrées dans la figure 3.2.

Méthodes exactes

Les méthodes exactes, appelées aussi méthodes complètes, permettent de trouver la solution optimale d'un problème d'optimisation en explorant exhaustivement l'ensemble des solutions (ou configurations) possibles [Solnon, 2010]. L'exploration énumérative (toutes les solutions sont évaluées une à une) est la technique la plus basique mais elle reste inappropriée aux problèmes combinatoires. Voilà pourquoi des méthodes exactes comme l'algorithme de ' Séparation and Évalua- tion' (Branch

and Bound [1]) explorent l'ensemble des solutions (ou configurations) possibles et éliminent des sous-ensembles de mauvaises solutions à l'aide de techniques d'élagage. L'avantage d'une telle approche est que les solutions éliminées ne sont pas évaluées 'à la main' mais de façon globale.

Méthodes approchées

Contrairement aux méthodes exactes, les méthodes approchées sont incom- plètes : elles permettent de trouver des bonnes solutions mais ne garantissent en aucun cas l'optimalité de celles-ci [2].

Les méthodes approchées sont composées de heuristiques et de métaheuristiques. Les heuristiques Par définition, une heuristique [3] est un moyen de guider les choix que doit faire un algorithme pour réduire sa complexité. Une heuristique est spécifique à un problème et ne peut pas être généralisée.

Les métaheuristiques

Les métaheuristiques peuvent être vues comme des heuristiques «puissantes et évoluées » dans la mesure où elles sont généralisables à plusieurs problèmes d'optimisation. Les métaheuristiques sont habituellement classées en fonction du nombre de solutions qu'elles manipulent : les métaheuristiques à solution unique telles que la recherche tabou et le recuit simulé ; et les métaheuristiques à population de solutions telles que que les algorithmes génétiques et les colonies de fourmis [Talbi, 2009].

Dilemme intensification versus diversification

Notons que la mise au point des méthodes appro chées doit trouver un équilibre entre deux tendances opposées lors de la re cherche dans l'espace des solutions : l'intensification (ou exploitation) et la diversification (ou exploration) [Solnon, 2010]. L'intensification de la recherche signifie que celle-ci se concentre autour des meilleures solutions rencontrées, considérées comme prometteuses ; alors que la diversification incite la recherche à explorer des nouvelles zones de l'espace de recherche en vue d'y trouver des bonnes solutions.

1. L'algorithme Branch et Bound est une généralisation de l'algorithme backtrack utilisé pour la résolution des problèmes de satisfaction de contraintes [Bensana and Verfaillie., 1995].
2. notons que par abus de langage, nous confondons ici les deux notions de complétude d'une méthode d'optimisation (sa faculté à trouver une solution s'il y en a) et de son optimalité (sa faculté à trouver la (ou les) meilleure(s) solution(s)) [Russell et al., 1995]
3. Étymologiquement, le mot heuristique signifie ' trouver, découvrir ' en grec ancien

Les métaheuristiques à solution unique ont plus tendance à l'exploitation du voisin age de la solution en question, et les approches à base de population de solutions ont plutôt tendance à l'exploration [Hao et al. 1999].

Dans la sous-section 3.2 de [], les heuristiques principales dédiées au problème de routage de véhicules a été présenté.

La sous-section 3.3, de même référence, elle été consacrée aux grandes lignes de l'application des algorithmes génétiques à la résolution du VRP. Pour une revue détaillée de l'application des métaheuristiques au VRP, on pourra se référer à Bräysy and Gendreau [2005b], Gendreau et al.[2003], Bräysy and Gendreau [2001a], Laporte [1992], Gendreau et al. [2001] et Gendreau et al. [2008].

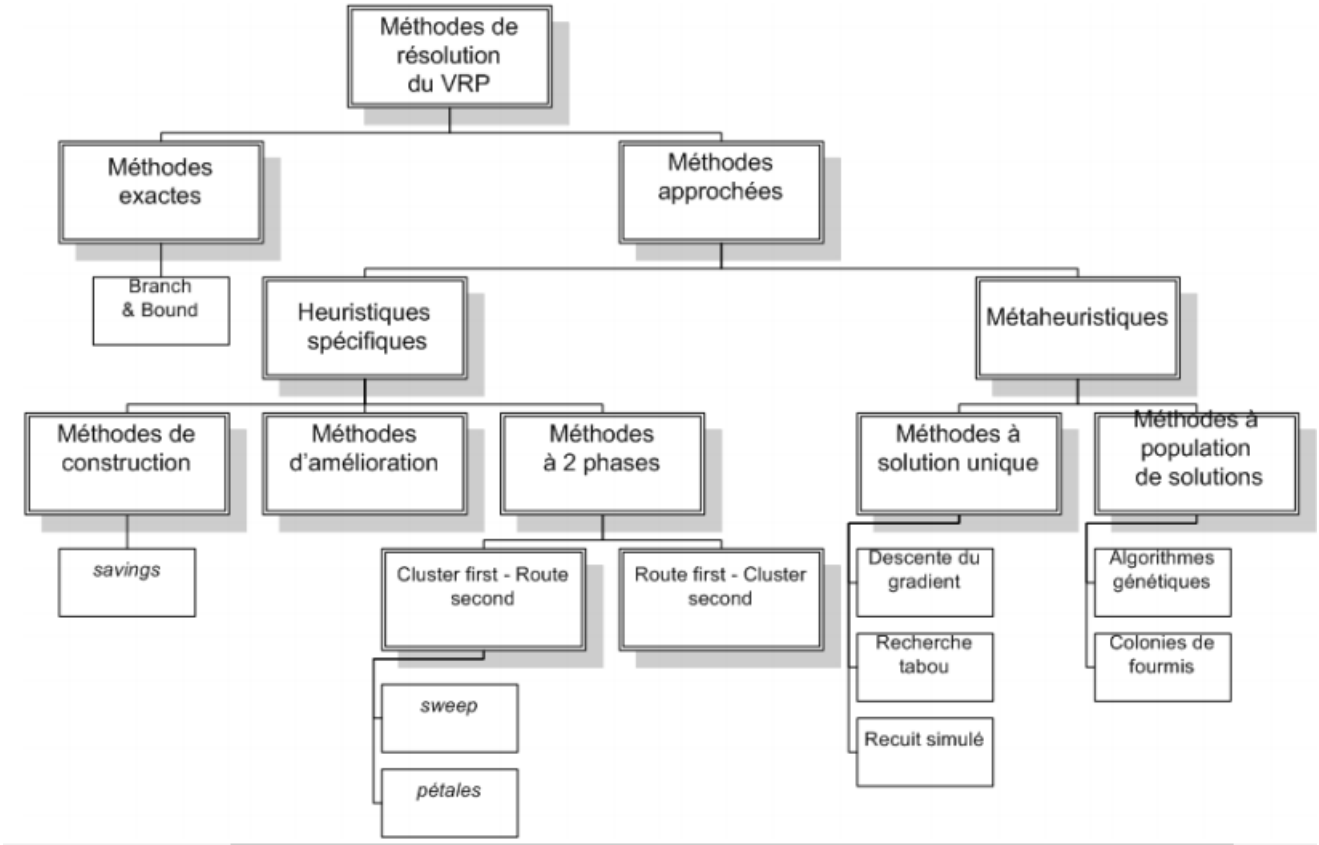


FIGURE 3.2 – Classication des méthodes de résolution du VRP.

3.2 VRP et multi-objectif

Les problèmes de tournées de véhicules académiques nécessitent généralement d'être adaptés lorsque l'on se trouve en présence d'un cas réel. Ces adaptations sont souvent des contraintes que l'on rajoute au modèle de départ, comme notamment les extensions vues précédemment. De plus, généralement dans les cas réels, il ne suffit pas d'optimiser un seul objectif, mais d'en prendre en compte plusieurs qui peuvent être contradictoires. Les objectifs les plus communs sont de minimiser la distance totale, le temps total de parcours, le coût global, la taille de la flotte, l'équilibrage des tournées, de maximiser la qualité de service ou le profit collecté, etc. De manière générale, Jozefowiez propose dans sa thèse de classer les objectifs selon leur appartenance à une composante du problème : la tournée (coût, travail,...), les noeuds ou les arcs (fenêtres de temps, service du client,...) et les ressources (taille de la flotte, marchandises,...). Chaque objectif sera ainsi défini par son implication dans l'une de ses catégories.

Toujours selon Jozefowiez, l'étude des problèmes de tournées de véhicules multi-objectif est principalement due à trois motivations : l'extension des problèmes classiques académiques, la résolution de problèmes réels ou bien encore l'étude d'objectifs complémentaires sans abandonner pour autant l'objectif principal classique qui est celui de la minimisation de la distance total.

Il existe de nombreuses références sur cette problématique en littérature, notamment en étude de problème bi-objectif, dans cette partie nous allons répertorier certaines études de manière non exhaustive.

La plupart des études sur le problème de VRP multi-objectif concernent des études bi-objectifs. Dans ces études, une façon, relativement commune, de prendre en compte plusieurs objectifs consiste à les agréger dans une fonction objectif finale. Une autre approche est apparue avec les colonies de fourmis. En effet, des recherches ont été menées de façon à utiliser une colonie par objectif comme dans. Il existe de nombreuses autres méthodes de résolution, il semblerait toutefois que les méthodes évolutionnaires aient été le plus utilisées et ont prouvé leur efficacité, comme par exemple la méthode NSGA-II. En effet, elles semblent bien se prêter à l'explora-

tion et sont souvent hybridées afin d'intensifier les recherches. Par ailleurs d'autres méthodes peuvent être utilisées comme des heuristiques de construction et d'amélioration. On peut également utiliser d'autres méta-heuristiques comme la méthode tabou ou le recuit simulé.

En résumé, les méthodes multi-objectif pour le VRP ont évoluées depuis quelques années. Tandis qu'elles se limitaient à des méthodes scalaires, dorénavant elles tendent à exploiter les méta-heuristiques. Nous avons également pu observer que de nombreuses études utilisent les algorithmes évolutionnaires, souvent hybridés avec des méthodes heuristiques afin d'explorer de façon plus efficace l'espace de recherche.

3.3 L'approche multicritères proposée pour l'optimisation de notre

problème Après l'obtention de toutes les solutions non dominées, nous illustrons dans cet algorithme notre approche multicritères basée sur les algorithmes génétiques, proposée pour l'optimisation de notre problème « MD-FSMVRPTW ».

L'objectif principal des algorithmes génétiques dans cette approche est de proposer une bonne approximation de la frontière de Pareto. Ceci, en explorant l'espace de solution, afin d'obtenir les populations les plus diverses et les plus possibles.

```
Début
  Etape 1: Création de la population initiale de taille = n);
  Etape 2: Affectation de tous les nœuds (clients) aux dépôts
           correspondants « création des clusters »;
  Tanque nombre de générations non atteint Faire
  Début
   Etape 3: Création de la population intermédiaire de taille = 2n
            avec de nouveaux individus de "croisement ou mutation"
   Etape 4: Application des procédures d'ordonnancement et de
            correction de capacité; «Appel les algorithmes
            Algo IV.2, Algo IV.3 »
   Etape 5 : Création de la deuxième population intermédiaire de
            taille = 2n * 2n représentant les différentes routes
            prises par chaque véhicule;
   Etape 6 : Déterminer la valeur de fitness pour chaque individu
            dans la population P_nœuds/véhicules (Le nombre de véhicules,
            la somme des retards et le coût de transport) pour
            obtenir par la suite la nouvelle population P_pareto-dominance;
   Etape 7 : Ordonner la population P_pareto-dominance en fonction des
            valeurs de fitness minimales (Le nombre de véhicules, la
            somme des retards et le coût de transport);
   Etape 8 : Copier les solutions non dominées;
   Etape 9 : Incrémenter le nombre de générations;
  FinTQ
Fin.
```

FIGURE 3.3 – Structure de l'algorithme génétique multicritères proposé pour l'optimisation de notre problème FSMVRPTW .

3.4 Schéma général de ProGenClust

Notre proposition est basée sur une hybridation entre un algorithme génétique que nous proposons, pour résoudre un problème MD-VRPTW, et un algorithme d'apprentissage « Kmeans ». Nous schématisons les différentes étapes de l'algorithme ProGenClust dans la figure suivante

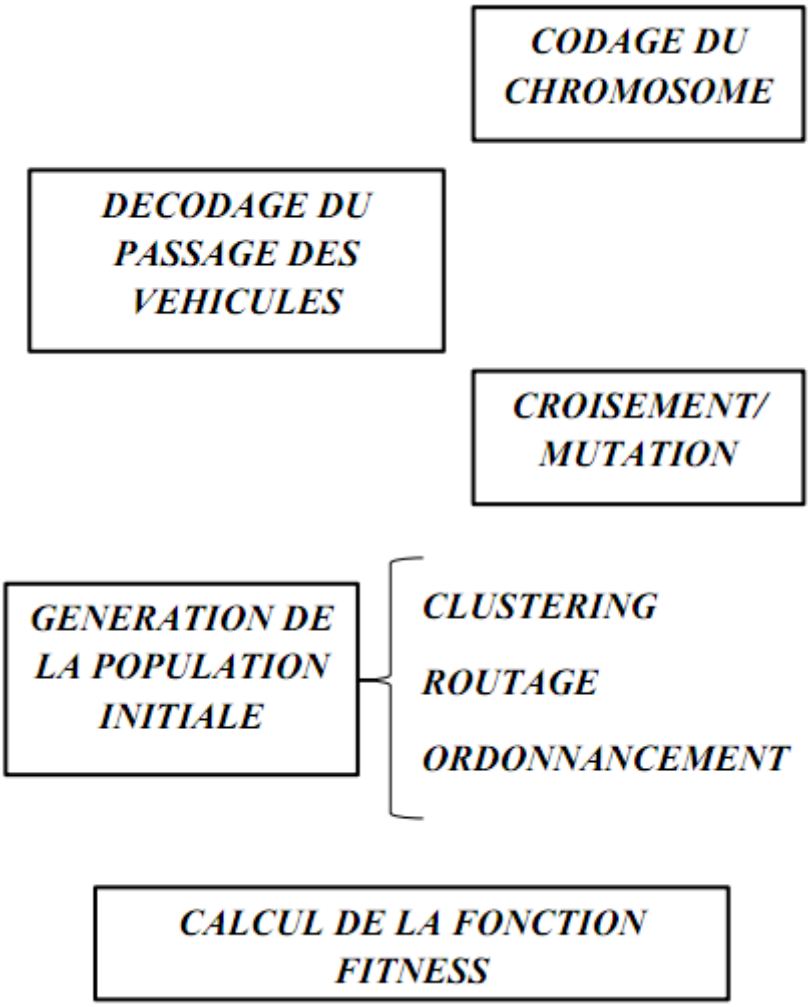


FIGURE 3.4 – Schéma général de ProGenClust.

3.5 Codage du chromosome

L'encodage permet de représenter les solutions en tant que chromosomes. Un chromosome est une séquence de noeuds, indiquant l'ordre dans lequel les véhicules doivent visiter tous les noeuds. Ce type de codage (Table IV.1) s'appelle codage par liste de permutation.

Position	Début	1 er	2ienut	3	4	5	6	7	8	9	10	Fin
Neud (i)		5	2	6	4	3	10	8	7	9	1	

TABLE 3.1 – Codage par liste de permutation

3.6 Décodage du passage des véhicules

Le décodage permet, à partir de chaque chromosome de la population, d'obtenir une solution initiale indiquant le passage de chaque véhicule sur les noeuds correspondants.

Position	Coût	Début	1ª	2 m	3 m/Fin	4me	Fin
V1	C1		2	4	1	5	
V2	C2		3	6	7	10	
V3	C3		8	9			

TABLE 3.2 – Ordre du passage des véhicules.

3.7 Croisement

Après la génération aléatoire de la population initiale, nous procédons à la phase de croisement qui assure la recombinaison des gènes parentaux pour former de nouvelles générations.

Dans ce travail, nous avons appliqué une nouvelle technique appelée the Best Cost Route Crossover (BCRC) développée par D. Fogel, pour le problème de routage des véhicules avec fenêtres de temps (VRPTW), qui permit de garantir que les solutions générées par l'évolution génétique sont réalisables. Nous montrons en outre l'applicabilité de BCRC en l'étendant au MDVRPTW, mais avec quelques légères améliorations.

Les étapes de la technique CRCB sont :

— Choisir les parents pour sélection par tournoi ;
— Sélectionnez au hasard un dépôt pour subir la reproduction ;
— Sélectionnez une route de chaque parent de manière aléatoire ;
— Supprimez tous les clients appartenant à la route 1 du parent 2 ;
— Supprimez tous les clients appartenant à la route 2 du parent 1 ;
— Pour chaque client appartenant à la route 1
    — Calculer le coût d'insertion de la route 1 dans chaque emplacement du parent 2 et stockez les coûts dans une liste ordonnée ;

- Pour chaque emplacement d'insertion, vérifié si l'insertion est possible ou non ;
- Générer un nombre aléatoire « r » (0= r =1).
- Choisir le premier emplacement d'insertion possible si rn = seuil ;
- Sinon, choisir la première entrée dans la liste ordonnée, quelle que soit la faisabilité ;

— Répéter l'étape précédente pour chaque client appartenant à la route 2.

3.8 Mutation

La mutation est un opérateur génétique utilisé pour maintenir la diversité génétique d'une génération de population de chromosomes à la suivante. Il modifie un ou plusieurs gênes dans un chromosome à partir de son état initial. La mutation est utilisée pour trouver une solution pour le MD-VRPTW en se basant sur les algorithmes génétiques. Une sous-chaîne du parent est sélectionnée de manière aléatoire et permutée dans le but de former un nouvel enfant.

Nous avons utilisé différents types de mutations pour permettre de multiples façons de se libérer des minimums-locaux et d'améliorer les coûts de la tournée lorsque cela est possible. Dans la littérature, nous avons trouvé deux types de mutations :

3.8.1 Mutation « Intra-Dépôt »

Les mutations appliquées à des routes à dépôts unique sont appelées mutations intra-dépôts, elles sont efficaces pour intégrer la diversité dans les routes de chaque dépôt.
Trois types de mutations Intra-Dépôts ont été étudiés :

Mutation d'inversion « Reversal mutation »

Nous proposons une adaptation d'une simple mutation largement utilisée, néralement appelée « Reversal mutation ». Deux points de coupe sont lectionnés dans le chromosome associé au dépôt déjà choisi et les valeurs des deux nes entre ces deux points de coupure sont inversées. Un exemple de mutation par version est

illustré dans la figure (Fig. IV.3) et qui consiste à choisir au hasard deux nes dans un chromosome et échanger leurs valeurs.

1	4	6	2	8	0	7	5	3	9	10

TABLE 3.3 – Avant mutation (Parent).

Avant mutation (Parent)

1	4	7	2	8	0	6	5	3	9	10

TABLE 3.4 – Après mutation (Enfant) .

Re-routage des clients individuels « Single customer re- routing »

Dans ce cas, nous sélectionnons au hasard un client, et nous l'éliminons de l'itinéraire stant. Ce client est ensuite inséré dans le meilleur emplacement d'insertion possible ns tout le chromosome. Cela implique de calculer le coût total d'insertion à chaque alisation d'insertion, qui réinsère finalement le client dans le lieu le plus réalisable.

Permutation « Swapping »

Cet opérateur de mutation simple sélectionne aléatoirement deux routes et échange un client choisi de manière aléatoire d'un itinéraire à un autre.

3.9 croisement Mutation Inter-Dépôts

La mutation Inter-Dépôts permet d'échanger des clients d'un dépôt a un autre, ce qui contribue à améliorer la qualité de la solution, puisque le regroupement statique initial (Clustering) qui permet l'affectation des clients aux dépôts disponibles, fait une hypothèse imposante sur le groupe où chaque client doit être intégré.

Chaque 10 générations, au lieu d'appliquer une des mutations intra-dépôt déjà décrites ci-dessus, on utilise un opérateur de mutation inter-dépôts (si c'est applicable bien sûr).

Dans la mutation inter-dépôts, les clients peuvent réduire les coûts totaux de l'itinéraire, la somme totale des retards ou le nombre de véhicules utilisés en les réaffectant à différents dépôts.

Notez ici que nous devons respecter la contrainte de capacité, afin d'assurer la non-surcharge de chaque véhicule.

3.10 Génération de la population initiale

Le mécanisme de génération de la population initiale permet la production d'une population d'individus comme base pour les futures générations. Nous jugeons important le choix de la population initiale car il peut rendre plus ou moins rapide la convergence vers l'optimum global.

Afin de converger rapidement vers l'optimum global, nous avons divisé la tâche de génération de la population initiale en deux phases :

La première phase appelée (Clustering) consiste à partager équitablement les noeuds sur les dépôts, de manière à affecter à chaque dépôt les noeuds les plus proches, comme indiqué dans la figure (Fig. IV.4).

Dans l'exemple illustré dans la figure (Fig. IV.5), il y a 10 clients désignés 1-10. Deux routes sont requises par les véhicules pour servir tous les clients pour l'instance (A 8 4 A 9 5 1 A). La première route part du dépôt A et se déplace vers les clients 8 et 4 et retourne au même dépôt. De même, la deuxième route commence à partir du dépôt A, sert les clients 9, 5, 1 et revient au même dépôt. La solution réalisable pour le processus d'optimisation est générée en trois étapes de base : le regroupement, le routage et l'ordonnancement.

3.10.1 Le regroupement (Clustering)

Dans la littérature, on trouve différentes heuristiques de voisinages dédiées clustering. Dans notre cas, nous avons appliqué le principe de l'algorithme Kmeans[63] avec une légère modification afin d'assurer le clustering. Nous avons procédé comme suit :

— Choisir k noeuds dans l'espace k ∈ S (les k nceuds = les dépôts utilisés) ;

— Tant que $\exists x \in$ S non visité Faire

- Calculer la distance euclidienne entre chaque ncud de l'ensemble S et les dépôts existants ;
- Affecter un noud au plus proche dépôt « selon la règle de distance détaillé ci-dessous » ;

— Une fois terminé, nous aurons k clusters.

Initialement, chaque client est affecté au plus proche dépôt en termes de distance euclidienne. Comme discuté ci-dessus, pour k dépôts dans le MD-VRPTW, le chromosome se compose de ´kˇ " clusters et les clients sont affectés à chacun de ces « k » clusters. Dans l'exemple (Fig. IV.4), il existe deux dépôts A et B, chaque client C_i doit être affecté à un seul dépôt exactement. Ce processus de regroupement est effectué en fonction du calcul de la distance selon la règle suivante :

— Si D (C_i, A) < D (C_i, B), le client C_i est affecté au dépôt A

— SiD (C_i, A) > D (C_i, B), le client C_i est affecté au dépôt B

— Si D (C_i, A) = D (C_i, B), le client C_i est affecté à un dépôt choisi arbitrairement entre A et B.

— D (C_i, P) : représente la distance entre le client C_i et le dépôt P, tels que :

$$D(C_i, P) = \sqrt{(X_{Ci} - X_P)^2 + (Y_{Ci} - Y_P)^2} \tag{3.1}$$

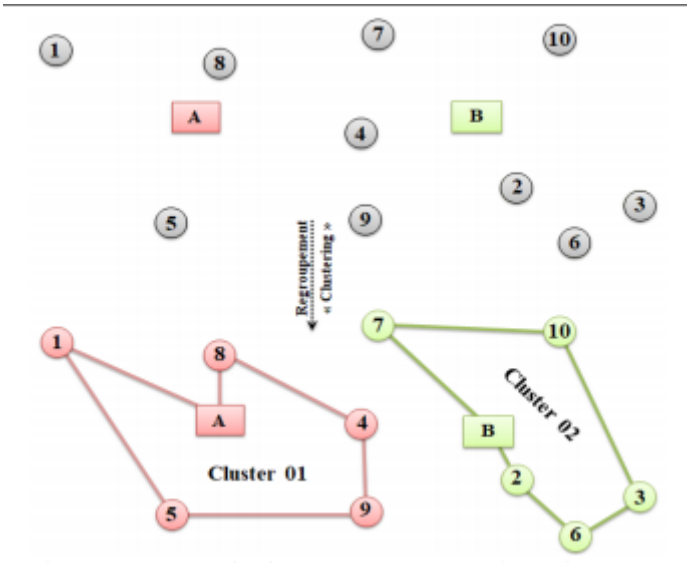


FIGURE 3.5 – Exemple de regroupement « clustering » .

3.10.2 Le routage

Dans un problème MD-VRPTW, un chromosome doit spécifier le nombre de routes (c'est-à-dire les véhicules) et l'ordre de livraison dans chaque itinéraire. Les clients du même groupe sont affectés à plusieurs itinéraires en utilisant la méthode Clarke et Wright Saving (G. Clarke et al. (1964)). Le processus de routage est basé sur la distance parcourue par les véhicules pour servir les clients. Une matrice d'économie $S\left(C_i, C_j\right)$ est construite pour chaque deux clients i et j dans le même lien. Tel que,

$$\mathrm{S}\left(\mathrm{C_i}, \mathrm{C_j}\right) = \mathrm{D}\left(\mathrm{P}, \mathrm{C_i}\right) + \mathrm{D}\left(\mathrm{P}, \mathrm{C_j}\right) - \mathrm{D}\left(\mathrm{C_i}, \mathrm{C_j}\right) \tag{3.2}$$

3.10.3 L'ordonnancement

A ce stade, un véhicule doit partir du dépôt et, à partir du premier client indiqué par le premier gène du chromosome, la séquence de livraison est choisie de telle sorte que le prochain client soit aussi proche du client précédent. Cette dernière est assurée en appliquant le principe de l'heuristique de voisinage 'le plus proche voisin' KPP, et renforcée par les deux algorithmes de correction de successeurs et de capacité « comme le montre l'(Algo 3.11) », en tenant compte du fait que les contraintes de longueur d'itinéraire et de capacité du véhicule ne soient pas violées avant d'ajouter un client à l'itinéraire actuel. Ce processus continu jusqu'à ce que chaque client soit

affecté à une seule route. Le principe d'ordonnancement est illustré dans (Algo 3.6).

```
I : Individu ;
n : Nombre de gènes par individu ;
Début
 m :=1 ;
 Tanque m<=n Faire
    i := m ;
    Tanque I[i] <> 0 et I < n Faire
     min:=i;
     j:=i+1;
     Tanque I[j] <> 0 et  j <= n Faire
      Si I[j] ≤ I[min]    Alors
        min:=j;
      FinSi
      j :=j+1 ;
     FinTQ
     i :=i+1 ;
    FinTQ
    m :=i+1 ;
 FinTQ
Fin.
```

FIGURE 3.6 – Algorithme d'ordonnancement .

```
I : individu ;
n : Nombre de gènes par individu ;
qt[] : Tableau des quantités à distribuer pour chaque client;
Début
 Q^T_max ; // Capacité maximale des véhicules de type T
 qt=0 ; // Capacité actuelle
 m :=1 ;
 Tanque m <= n Faire
  i :=m ;
  Tanque I[i] <>0 et i<=n Faire
   Si qt + qt[i] ≤ Q^T_max Alors
    qt = qt + qt[i] ;
   Sinon
    J :=i+1 ;
    Tanque I[j] <>0 et j<=n Faire
     Si qt + qt[j] ≤ Q^T_max Alors
      qt := qt + qt[j] ;
      N:=I[j]
      Pour l de j a i+1,-1 Faire
       I[l] = I[l-1] ;
      FinPour
      I[i] := N
     FinSi
     j :=j+1 ;
    FinTQ
   FinSi
   i :=i+1 ;
  FinTQ
  m :=i+1 ;
 FinTQ
Fin.
```

FIGURE 3.7 – Algorithme de correction de capacité.

A la fin de la phase d'ordonnancement, une solution réalisable de l'exemple du problème MD-VRPTW est construite comme le montre la figure (3.8).

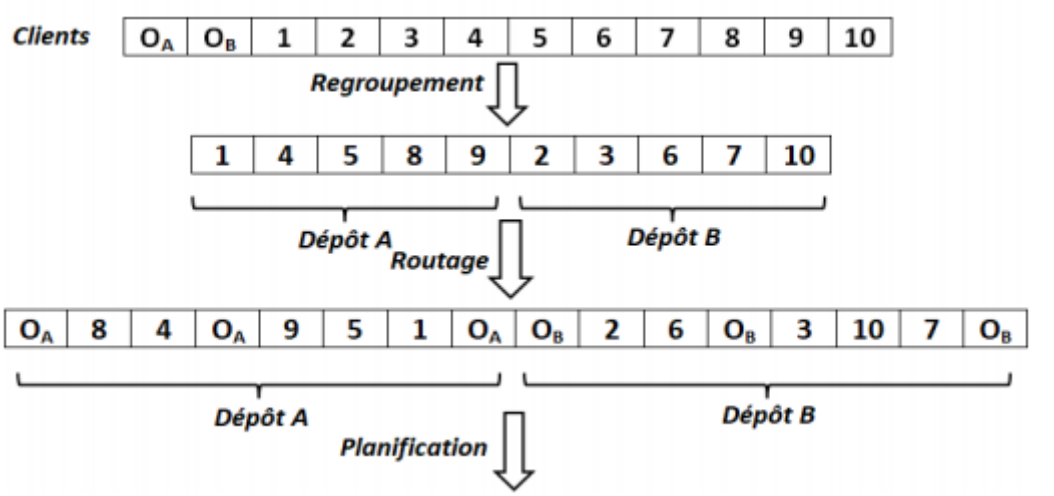


FIGURE 3.8 – Exemple d'un processus d'optimisation d'un problème MD-VRPTW.

3.11 Procédure de calcul du « fitness »

Chaque véhicule doit servir les noeuds dans sa tournée dans l'ordre d'affectation. Une fois de retour au dépôt, nous calculons le coût du transport correspondant à la distance parcourue, et nous répétons cela jusqu'à ce que tous les noeuds soient servis.

Nous reproduisons cette tâche pour chaque individu de la population, en prenant en compte les différentes combinaisons possibles entre les véhicules et les noeuds qui correspondent aux clients afin d'obtenir ultérieurement les individus qui minimisent notre fonction objective.

Une fois que chaque chromosome a été transformé en une topologie de réseau faisable en se basant sur les techniques de groupement, routage et ordonnancement donné ci-dessus, la valeur de fitness de chaque chromosome est déterminée en utilisant une fonction de fitness pondérée « weighted-sum ».

Cette méthode nécessite d'ajouter les valeurs des fonctions de fitness en utilisant des coefficients pondérés pour chaque objectif individuel. La fonction fitness $F(x)$

d'un individu x est définit comme suit :

$$\mathrm{F(x)} = \beta \cdot \sum_{k \in K} V_k$$
$$\mathrm{V_k} = \sum_{i \in Z} \sum_{j \in Z} d_{ij}$$

β est un paramètre de poids associé à la distance totale parcourue par les véhicules. La valeur du paramètre de poids utilisé dans cette fonction a été fixée à $\beta = 0.001$, ce qui concentre efficacement l'évolution de la minimisation de la longueur de la route.

3.12 Le problème du voyageur de commerce généralisé (TSP)

TSP (**Traveling Salesman Problem**) : le problème du voyageur de commerce est un cas particulier du problème VRP sans contrainte de capacité ; il consiste en la détermination d'un parcours de coût minimal (distance, temps, etc.) pour un seul véhicule partant d'une ocalité, visitant n autres localités et revenant à son point de départ [Rego et al. 1994].
Selon Laporte and Osman [1995] et Dhaenens et al. [2002], le problème du voyageur de commerce serait le problème le plus célèbre et le plus étudié en optimisation combinatoire. Dans ce problème, un voyageur de commerce doit visiter plusieurs villes (ou clients) en passant une et une seule fois par chacune d'entre elles, et en minimisant la distance totale parcourue [Lawler et al. 1985] .
Plus formellement, un TSP est modélisé sous forme d'un graphe où les sommets représentent les villes à visiter, et les arêtes les liaisons entre ces villes [Dhaenens et al. 2002]. La pondération ou le poids associé à chaque arête représente le coût de la liaison entre les deux villes et correspond généralement à la distance qui les sépare. L'objectif est de trouver un cycle hamiltonien, $c - \dot{a} - d$ un cycle passant une et une seule fois par tous les sommets du graphe, et de longueur minimale. En tant que problème d'optimisation, le TSP est un problème NP-difficile. En effet, dans sa version symétrique, $c - \dot{a} - d$ dans le cas où le graphe associé n'est pas orienté, le nombre total de solutions possibles est $\frac{(n-1)!}{2}$ où n est le nombre de villes. Avec une telle complexité factorielle, une résolution efficace du TSP nécessite donc le re-

cours à des heuristiques spécialisées voire même à des métaheuristiques. En effet, les méthodes exactes restent limitées aux problèmes de petite taille.

Une définition de problème de voyageur de commerce généralisé (TSP)

basée sur Laporte et Nobert (1983) et Noon et Bean (1991) suit :

Soit $G = (V, E)$ un graphe non orienté à n-noeuds dont les arêtes sont associées à des coûts non négatifs. Nous supposerons que w.l.o.g. que G est un graphe complet (s'il n'y a pas d'arête entre deux noeuds, on peut l'ajouter avec un coût infini).

Soit $V_1, \ldots, V_p$ une partition de V en p sous-ensembles appelés clusters (ie $V = V_1 \cup V_2 \cup \ldots \cup V_p$ et $V_l \cap V_k = \emptyset$ pour tout $l, k \in \{1, \ldots, p\}$). On note le coût d'une arête $e = \{i, j\} \in E$ par c_{ij}
Le GTSP demande de trouver un tour à coût minimum H couvrant un sous-ensemble de noeuds tel que H contienne exactement un noeud de chaque cluster $V_i, i \in \{1, \ldots, p\}$. Le problème implique deux décisions liées : choisir un sous-ensemble de noeuds $S \subseteq V$, tel que $|S \cap V_k| = 1$, pour tout $k = 1, \ldots, p$ et trouver un cycle hamiltonien de coût minimum dans le sous-graphe de G induit par S.
Un tel cycle est appelé cycle hamiltonien. Le GTSP est dit symétrique si et seulement si l'égalité $c(i, j) = c(j, i)$ est vérifiée pour chaque $i, j \in V$, où c est la fonction de coût associée aux arêtes de G.

3.12.1 Méthodes de résolution : Algorithme

Le problème du Voyageur de Commerce Généralisé (Generalized Traveling Salesman Problem, ou CTSP) a poussé de nombreux scientifiques à chercher des solutions proches de la solution optimale, toutes avec des temps de calcul différents, selon la taille de l'échantilion.
Il existe de nombreuses approches à la résolution du problème du Voyageur de Commerce Généralisé. Toutes possèdent leurs propres caractéristiques de temps de calcul et d'optimalité sur un échantillon de taille spécifique. Il s'agissait pour nous de trouver l'approche qui réduirait au mieux le temps de calcul afin de rendre notre site le plus efficace possible, ainsi que d'assurer un bon taux d'optimalité pour un échantillon de petite taille.

Les algorithmes de résolution du TSP peuvent être répartis en deux classes :

— Les algorithmes exacts permettent de trouver la solution optimale, mais leur complexité est exponentielle. Les algorithmes les plus efficaces sont "cutting-plane", "facetfinding" et "branch and bound".

— Les algorithmes d'approximation (heuristiques) obtiennent de bonnes solutions mais ne donnent aucune garantie sur l'optimalité de la solution trouvée. Nous présentons quelques un de ces heuristiques.

— Approche avec prétraitement des données.

— Approche branch-and-cut,

— Approche de réduction du problème pour se rapprocher d'un simple problème de Voyageur de Commerce,

— Approche avec heuristique composite de création de solutions partielles, et d'amélioration,

— Approche avec recherche locale approfondie des clusters,

— Approche d'adaptation de l'heuristique de Lin-Kemighan pour le Voyageur de Commerce Généralisé.

— Approche avec heuristique qui mime la sélection naturelle génétique.

Chacune de ces approches a ses propres caractéristiques : un certain taux d'erreur par rapport à la solution optimale, un temps d'exécution plus ou moins long, ou une précision variable par taille d'échantillon.

Une Algorithme exact pour le GTSP

Dans cette section, nous présentons un algorithme qui trouve une solution exacte au GTSP.
Étant donné une séquence $\left(V_{k_1}, \ldots, V_{k_p}\right)$ dans laquelle les clusters sont visités, nous voulons trouver le meilleur tour hamiltonien réalisable H^* (par rapport à la minimisation des coûts), visitant les clusters selon la séquence donnée. Cela peut être fait en temps polynomial en résolvant les problèmes de chemin le plus court $\left|V_{k_1}\right|$ comme décrit ci-dessous.
On construit un réseau en couches, noté LN, avec $p+1$ couches correspondant aux clusters $V_{k_1}, \ldots, V_{k_p}$ et en plus on duplique le cluster V_{k_1}. Le réseau en couches contient tous les noeuds de G plus quelques noeuds supplémentaires v' pour chaque

$v \in V_{k_1}$. Il existe un arc (i, j) pour chaque $i \in V_{k_l}$ et $j \in V_{k_{l+1}} (l = 1, \ldots, p-1)$, avec le coût c_{ij} et un arc $(i, h), i, h \in V_{k_l}, (\ l = 2, \ldots, p)$ avec le coûte c_{ih}. De plus, il existe un arc (i, j') pour chaque $i \in V_{k_p}$ et $j' \in V_{k_1}$ avec le coût $c_{ij'}$.

Pour tout $v \in V_{k_1}$ donné, sont considérés comme des chemins de v à $w', w' \in V_{k_1}$, que visite exactement un noeud de chaque cluster $V_{k_2}, \ldots, V_{k_p}$, ce qui donne un tour hamiltonien réalisable. Inversement, tout tour hamiltonien visitant les clusters selon la séquence $\left(V_{k_1}, \ldots, V_{k_p}\right)$ correspond à un chemin dans le réseau en couches d'un certain noeud $v \in V_{k_1}$ à $w' \in V_{k_1}$. Par conséquent, le meilleur tour hamiltonien (par rapport à la minimisation des coûts) H^* visitant les clusters dans une séquence donnée peut être trouvé en déterminant tous les chemins les plus courts de chaque $v \in V_{k_1}$ à chaque $w' \in V_{k_1}$ avec la propriété qui visite exactement un noeud du cluster. La complexité temporelle globale est alors de $\left|V_{k_1}\right| O(m+n \log n)$, soit $O(nm+n \log n)$ dans le pire des cas. Nous pouvons réduire le temps en choisissant $\left|V_{k_1}\right|$ comme cluster avec une cardinalité minimale. Il convient de noter que la procédure ci-dessus conduit à un algorithme exact en temps $O(nm + n \log n)$ pour le GTSP. Nous avons donc établi le résultat suivant :

Théorème 3.1: *La procédure ci-dessus fournit une solution exacte au GSTP en temps O((p-1) !(nm+nlogn)), où n est le nombre de noeuds, m est le nombre d'arêtes et p est le nombre de clusters dans le graphique d'entrée.*

Clairement, l'algorithme présenté est un algorithme de temps exponentiel à moins que le nombre de clusters p ne soit fixe.

A. *Représentation possibles du problème*

Dans cette partie, nous allons étudier différentes méthodes de représentation des données puis nous indiquerons laquelle a été retenue pour notre problème.

1. *La représentation adjacente*
   La représentation adjacente représente la tournée comme une liste de N villes commençant la ville 2. La ville j est listée à la position i si et seulement si le voyageur de commerce se dirige de la ville i à j (voir l'exemple figure 3.9).

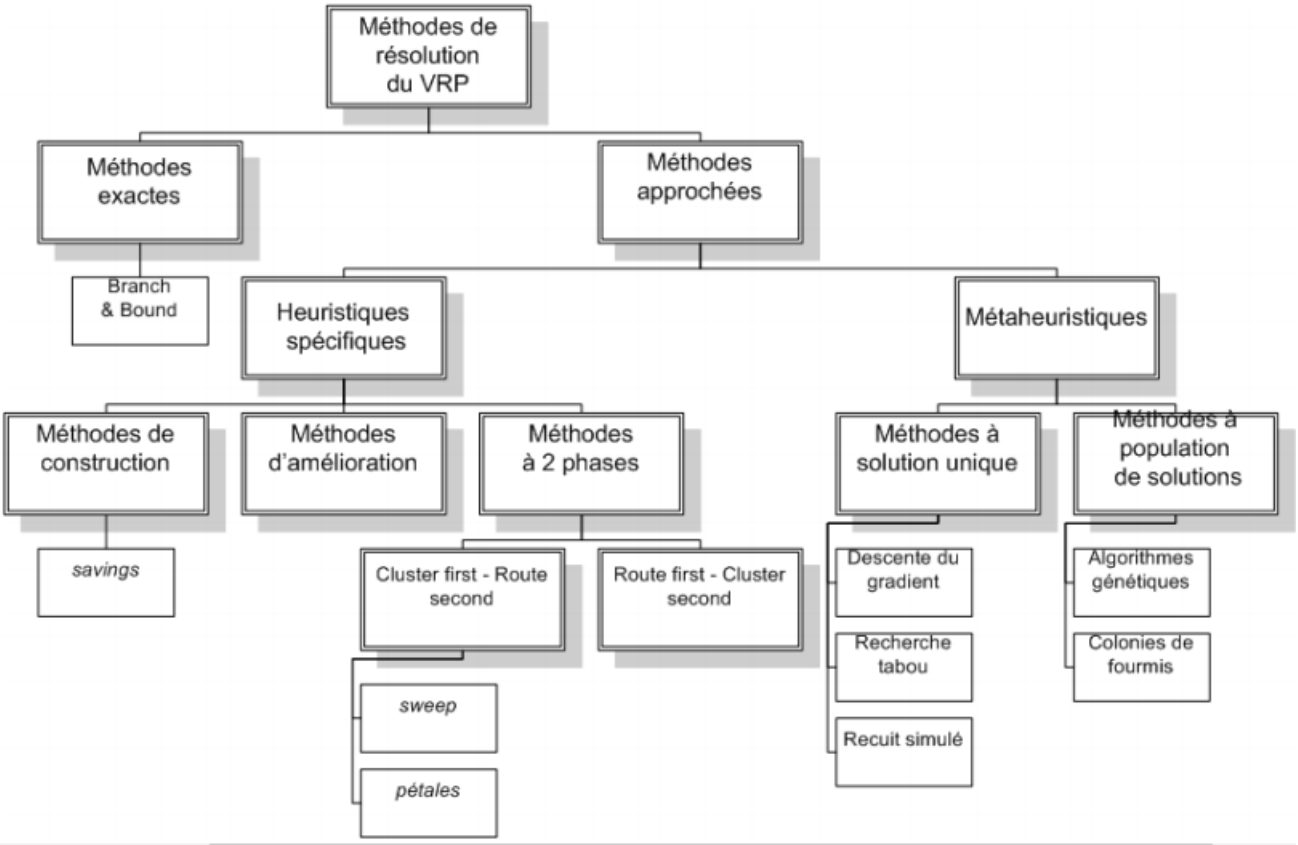


FIGURE 3.9 – Représentation adjacente de la tournée $1; 5; 2; 9; 7; 6; 8; 4; 1$.

Mais le problème principal est que la présentation adjacente ne supporte pas les Opérateurs classiques de croisement. Pour pallie à ce problème ; un mécanisme de réparation est nécessaire.

2. *La représentation ordinale*
   La représentation ordinale représente le parcours du voyageur de commerce comme une liste de N villes dont ième élément est un nombre compris entre 1 et $(N - i + 1)$.
3. *La représentation chemin*
   Un circuit est codé par un tableau des entiers qui représente le successeur et le prédécesseur de chaque ville.

3	5	2	9	7	6	8	4

FIGURE 3.10 – Codage du Chromosome.

Ainsi le tableau 3.10 représente le circuit suivant : le point de départ est

la ville 3 qui est aussi la ville d'arrivée, en passant par les viles selon leur ordre d'apparition dans le tableau $3 \rightarrow 5 \rightarrow 2 \rightarrow 9 \rightarrow 7 \rightarrow 6 \rightarrow 8 \rightarrow 4 \rightarrow 6$

B. *Génération de la population initiale*

La population initiale conditionne forcement la rapidité et la convergence de l'algorithme. Pour cela, nous avons appliqué plusieurs méthodes pour générer la population initiale :

- Génération aléatoire de la population initiale.
- Génération du premier individu au hasard, celui sera muté (N – 1) fois avec un opérateur de mutation.
- Génération du premier individu en utilisant un mécanisme heuristique. Le successeur de la première ville est celui qui se trouve à une distance plus petite par rapport aux autres. Après, nous utilisons un opérateur de mutation sur le parcours obtenu afin de générer (N-2) autres individus qui constitueront la population initiale.
- Une génération aléatoire du premier individu, solution du problème du T.S.P., puis nous complétons la population en essayant, au plus 50 fois, de diminuer la fonction objective de 2% par l'utilisation de l'Opérateur de mutation. Si la opération échoue à trouver un tel individu on prend le dernier individu de. la boucle.
- Cette méthode procède de la manière que la méthode précédente, mais la génération du premier individu s'effectue par un mécanisme heuristique.

C. *Sélection*

La méthode utilisée est la sélection par la roulette. Elle consiste à associer à un chromosome i de la population une portion proportionnelle à

$$\frac{1}{N-1} \times \left(1 - \frac{f_i}{\sum_{j \in \text{ Population}} f_j} \right)$$

où fi est la valeur de a fonction objectif de l'individu i.

Ainsi, les individus qui ont des valeurs faibles de la fonction de fitness peuvent avoir une forte chance d'être sélectionnés parmi les individus à croiser.

D. *Opérateur de croisement*

Ils manipulent la structure des chromosomes de deux parents, qui sont choisis au hasard, afin de produire deux enfants. Dans la littérature, nous avons choisi cing opérateurs de croisement que nous allons expliquer leurs manières de procéder pour construire deux chromosomes (solutions) appelés enfantl et enfant2 à partir de deux chromosomes appelés parentl et parent 2 .

L'opérateur chaoisis pour cette etude de comparaison, c'est le croisement ordonné (Crdered Crossover), présenté par Goldberg, est l'algorithme de base de NWOX, il crée des trous et les remplit en déplaçant tous les gènes vers la gauche. La différence est au niveau du roulement, il s'effectue en roulant les éléments n'étant pas de trous vers la gauche en ramenant l'el 'ement à l'extrême gauche à l'extrême droite, débuter avec un élément fixe à $b + 1$. L'algorithme de la figure 3.12 décrit en détail ce croisement :

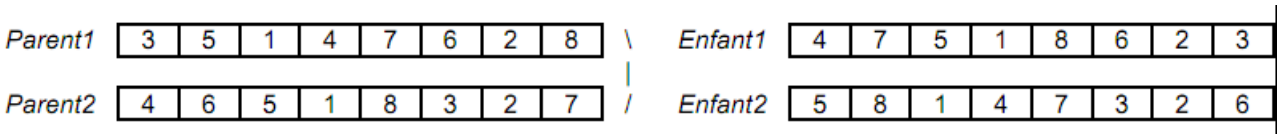


FIGURE 3.11 – Opérateur de croisement OX.

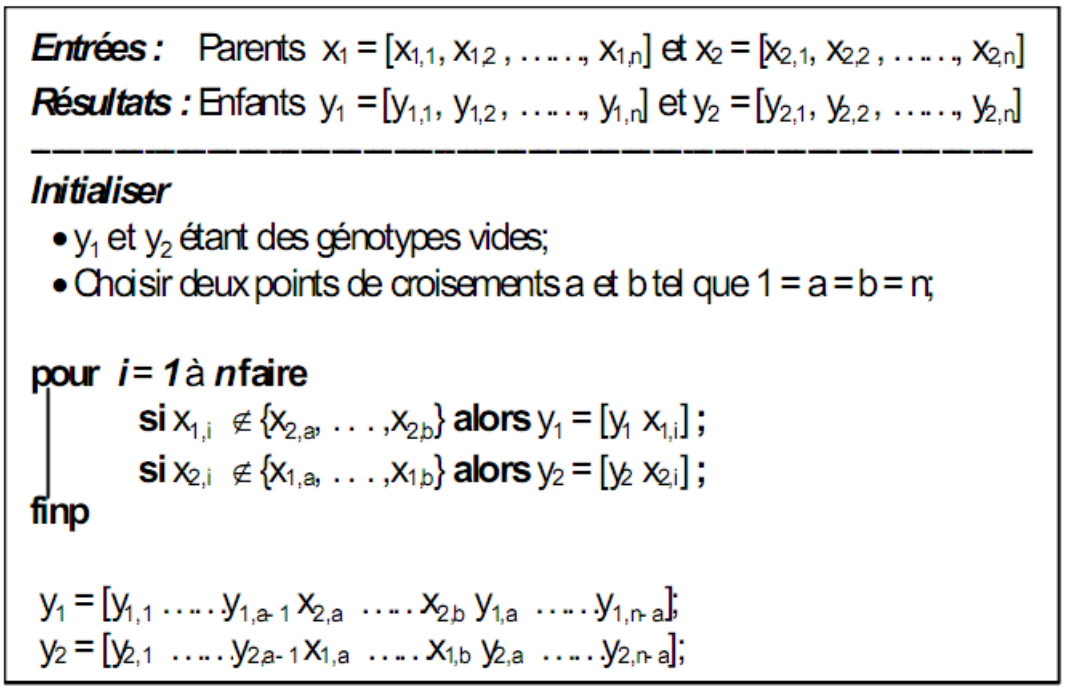


FIGURE 3.12 – Algorithme de croisement OX .

E. *Opérateur de mutation*

Il évite d'établir des populations uniformes incapables d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes. Nous utilisons les cinq Opérateurs de mutations suivants :

1. *Mutation Twors*

   C'est l'échange de deux positions de gènes choisi aléatoirement.

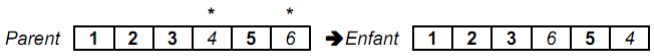


FIGURE 3.13 – Opérateur de mutation twors.

2. *Mutation de centre inverse (cim)*

   On choisit aléatoirement une position qui coupe le chromosome en deux séquences $S2$ et $S2$. On prend la première séquence S2 et on inverse l'ordre de ses gènes : le dernier devient premier, l'avant dernier devient second et ainsi de suite. On précède de la même manière sur S2.

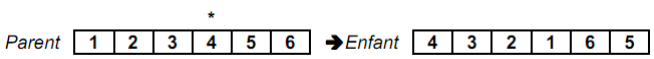


FIGURE 3.14 – Opérateur de mutation de centre inverse.

3. *Mutation Renversement d'une séquence (RSM)*

   Dans l'opérateur de mutation Renversement d'une séquence (reverse sequence mutation), on prend une séquence S limitée par deux positions $i < 1$ choisies aléatoirement. L'ordre des gènes de cette séquence sera inversé par la même manière que ce qui a été sur Sl dans l'opérateur précédent. L'algorithme illustré dans la figure 10 présente l'implantation de cet opérateur de mutation. Parent

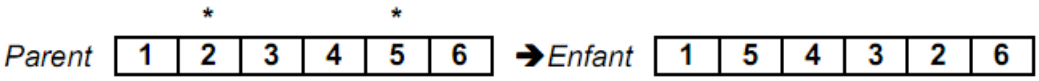


FIGURE 3.15 – Opérateur de mutation RSM.

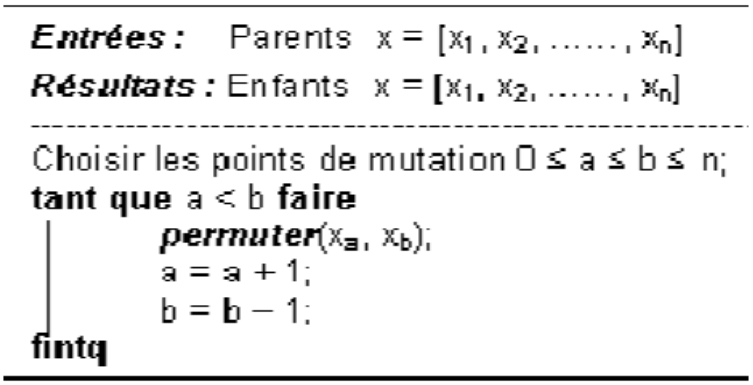


FIGURE 3.16 – Algorithme de Mutation Renverser une séquence (RSM).

4. *Mutation throas*

   On construit une séquence de trois gènes dont le premier est choisis au hasard et les deux autres ne sont que ces deux successeurs. Puis on précède de la manière suivante : le dernier devient premier de la séquence, le second devient dernier et le premier devient second dans la séquence. Parent |

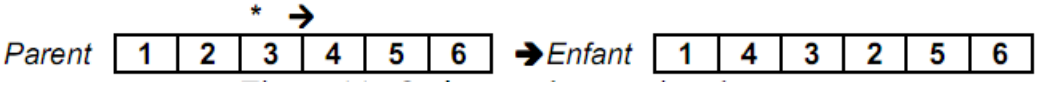


FIGURE 3.17 – Opérateur de mutation throas.

On choisit trois gènes aléatoirement qui prennent les positions distinctes non nécessairement successives i < j < 1 en procèdent ainsi : le gène de

la position i devient à la position j.

et celui qui était à cette position prendra la première position et le gène qui occupait cette position devient à la ième position.

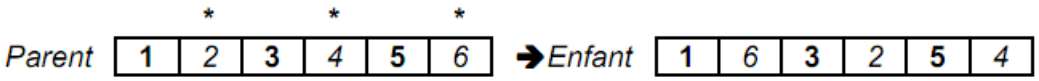


FIGURE 3.18 – Opérateur de mutation throas.

5. *Mutation Mélange Partiel (PSM)*

   Le mélange partiel du génotype (Partial Shuffle Mutation), comme son nom l'indique, change partiellement l'ordre d'apparition des gènes dans le génotype. L'algorithme 6 (figure 3.19) décrit en détail les étapes de mutation.

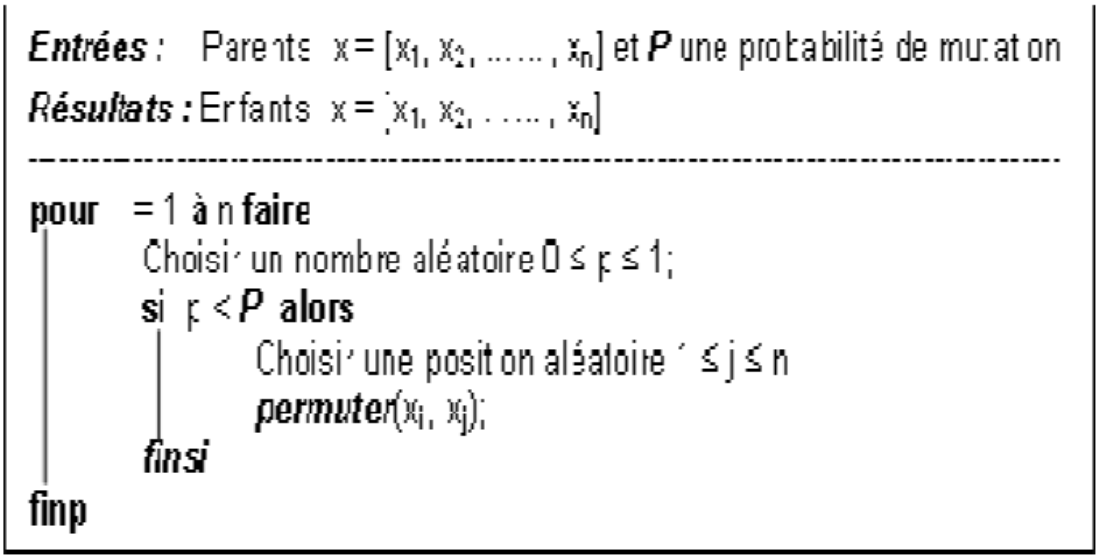


FIGURE 3.19 – Algorithme de Mutation Mélange Partiel (PSM).

F. *Méthodes d'insertion*

Nous avons utilisé, la méthode d'insertion élitisme qui consiste à recopier le meilleur chromosome de l'ancienne population dans la nouvelle. Celle-ci est complétée par les solutions résultantes d'opérations de croisement et de mutation en veillant à ce que la taille de la population reste fixe d'une génération à une autre.

3.12.2 RÉSULTATS NUMÉRIQUE ET DISCUSSION

Le problème du voyageur de commerce est l'un des problèmes les plus étudiés dans le domaine de l'optimisation de permutation de séquences de nombres. On utilise le problème test de BERLIN52 à 52 locations dans la ville de Berlin (3.20). Le seul critère d'optimisation est la distance parcourue pour compléter le trajet. La solution optimale à ce problème est connue et est de 7542 m (figure 3.24).

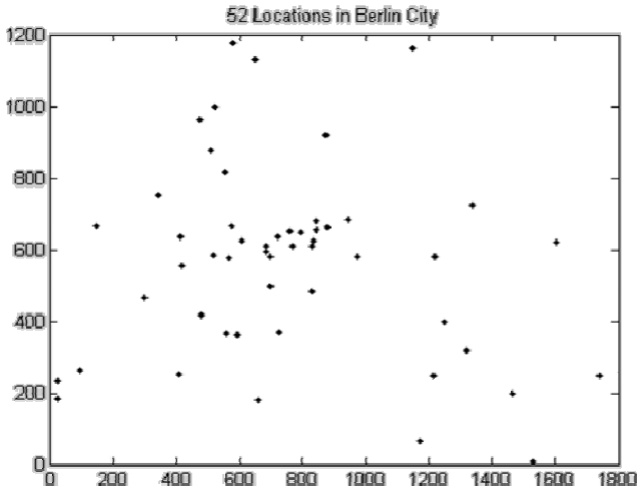


FIGURE 3.20 – Les 52 locations de la ville Berlin.

A. *Environnement*

Les opérateurs de mutation génétique et leurs différentes modalités, qui seront utilisées par la suite, sont regroupés dans ce tableau :

Opérateurs de mutation	PSM ; RSM ; THRORS ; THRAOS
Probabilité de mutation	1 ; 0.9 ; 0.8 ; 0.7 ; 0.6 ; 0.5 ; 0.4 ; 0.3 ; 0.2 ; 0.1 ; 0

FIGURE 3.21 – Les opérateurs utilisés.

Nous changeons à la fois qu'un seul paramètre et nous fixons les autres et nous exécutons cinquante fois l'algorithme génétique. La programmation a été faite en C++ sur une machine PC Core2Quad de CPU 2.4 Ghz et 2 Go en RAM ayant **Centos** 5.5 **Linux** comme un système d'exploitation.

B. *Résultats et Discussion*

Dans le but de comparer statistiquement les opérateurs, ils sont testés chacun sur 50 populations initiales différentes, en suivant l'algorithme évolutionnaire

(figure 3.22) auquel l'opérateur de variation est donné par l'algorithme de croisement OX (3.12) et suivi par l'un des opérateurs de mutation. La sélection se fait en choisissant par la roulette le plus court trajet.

Générer la population initiale P_0
pour $i = 0$ **à** Itr **faire**
P'_i = variation (P_i);
Évaluer (P'_i);
P_{i+1} = sélectionner ([P'_i, P_i]);
fin

FIGURE 3.22 – Algorithme évolutionnaire.

La figure 3.23 présente les résultats de l'application de différents opérateurs de mutation combinée à l'application de l'opérateur de croisement OX. On remarque que l'opérateur le plus performant est RSM suivi de prés par PSM, il est intéressant de noter que ce sont ces deux opérateurs qui perturbent le moins les individus en déplaçant ou en renversant un segment.

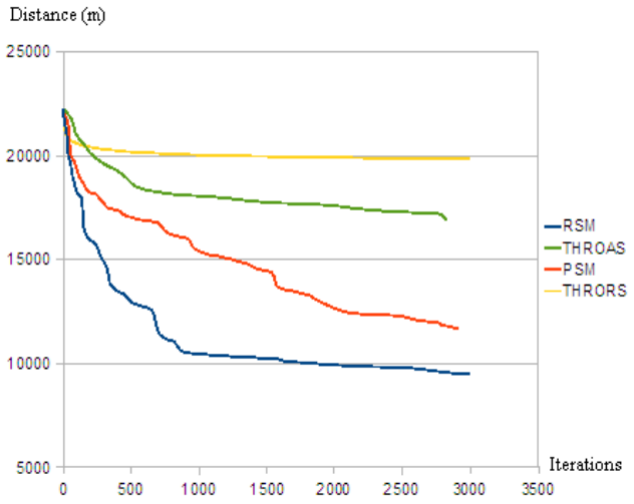


FIGURE 3.23 – La Comparaison entre les opérateur de Mutation.

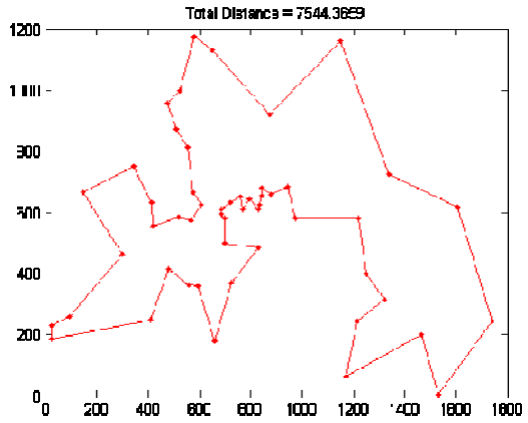


FIGURE 3.24 – La solution Optiamle de Berlin52.

3.13 Application de lapproche MOGP+S maths-crEA2

3.13.1 Les ressources utilisées

Les ressources utilisées pour réaliser ce travail et pour démontrer l'utilité de la programmation génétique sur les problèmes d'optimisation multi-objectifs sont :

1. Un micro-processeur Pentium III de vitesse 600 MHertz,
2. Une RAM de 192 Moctet,
3. Le système d'exploitation XP Pack 2 (PNX).

Le langage de programmation utilisé est l'environnement de simulation * Matlab : à cause de sa capacité de simuler les cas de programmation aléatoire, aussi que sa manière de présenter les résultats sous forme de graphe.

3.13.2 Le problème de TSP Bi-objectifs

Objectif	Trouver le meilleur programme permettant de résoudre le problème de TSP, c'est-à-dire trouver le meilleur chemin et réduisant le phénomène de Bloating.
L'ensemble de terminaux	City1, City2, City3, City4, City5, City6, City7, City8, City9, City10
L'ensemble de fonctions	CombinePath, SwapAndShift
La Première fitness.	La différence entre le chemin optimal et le chemin retourné par l'individu plus une pénalité si le chemin ne comporte pas toutes les villes.
La deuxième fitness	Pressure.
La fitness Brute	Réduire le Bloating : Adaptive Parsimony
Probabilité de mutation	SPEA2 : le principe de dominance
Probabilité de croisement	Variable
Nombre de copies à reproduire	Variable
Le succès	Le nombre de villes visitées (existant dans le chemin final)
La taille de population	$5, 10, 20, 50$
Le nombre de générations	$5, 10, 20, 50$
La population initiale	Full, Grow, Ramped
La méthode de sélection	Roulette, Tournament
L'élitisme	Replace, Keepset, Totalelitism
Les graphes	Les 5 types de graphes

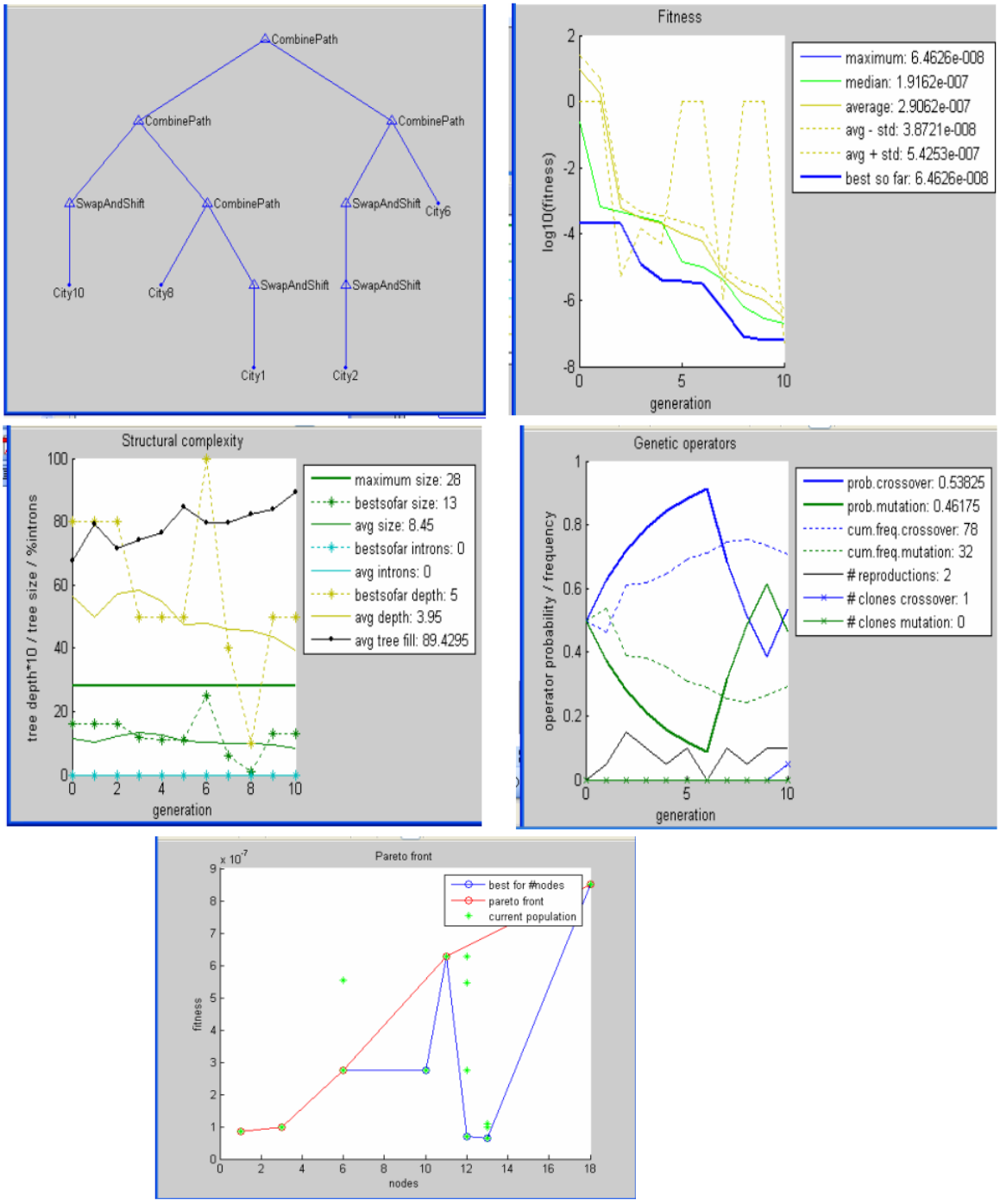


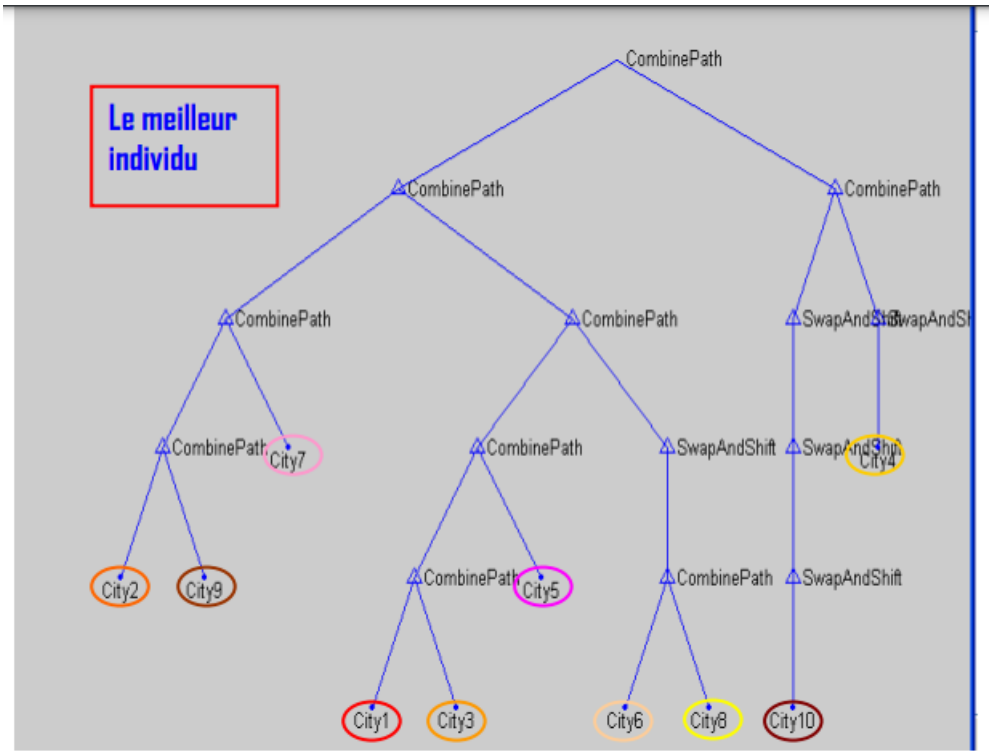
FIGURE 3.25 – Les résultats d'une exécution sur le problème de TSP.

Tout en appliquant ces paramètres au problème de TSP bi-objectifs, et après quelques éssaies on a abouti dans un essaie au meilleur programme dont on a :

— Le nombre de ville est exactement 10.

— Le meilleur chemin.

— Pas de phénomène de Bloating.



CombinePath(CombinePath(CombinePath(CombinePath(City2,Cyty3),City7),CombinePath(CombinePath(CombinePath(City1,City3),City5),SwapAndShift(CombinePath(City6,City8)))),CombinePath(SwapAndShift(SwapAndShift(SwapAndShif(City10),SwapAndShift(City4)))

3.14 Le problème du postier chinois (CPP)

Le problème du postier chinois a été introduit pour la première fois par le mathématicien chinois Meigu Guan (Mei-Ko Kwan) en 1962 [Gua62]. Etant donné un graphe $G = (V, E \cup A)$, il s'agit de déterminer un circuit de longueur totale minimum traversant au moins une fois chaque arc et arête du graphe.

Le problème du postier chinois (CPP) est bien connu et résolu dans sa version classique. Petit à petit, des contraintes supplémentaires sont venues s'ajouter pour obtenir une meilleure modélisation du monde réel.

Le PPC peut être résolu polynomialement dans les graphes non orientés, les graphes orientés et dans les graphes mixtes pairs. Dans les autres cas, le PPC est

un problème NP-dur.

3.14.1 Description du problème(Formulation)

Kwan Mei- Ko an faisant sa premiére publication on 1962 sur la théorie de graphes, vanait de génáraliser les rásultats de Lóonard Euler qui ne s'intéressait qu'aux cycles quitre. versent chaque arête du graphe exactement une fois. Les études de kwan étaient plutôt oriantées vers les cycles qui traverse sont chaque arête du graphe au moins une fois ; et son but était d'améliorer jus qu'á ce qu'ils soient optimaux. Avant d'aborder

les résultats de Kwan, nous nous proposons de donner la formulation mathámatique de ce problème.

Minimiser $\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij}^{[7]}$ (1)

Soumisas aux contraintes

$$\sum_{k=1}^{N} x_{ki} - \sum_{k=1}^{N} xik = 0, i, 1, \ldots N \quad (2)$$

$x_{ij} + x_{ji} \geqslant 1$ pour toutes les arêtes $(i, j) \in A$ (3)

$x_{ij} \geqslant 0$ est entier (4)

où

N = nombre de sommets du graphe

A. Ensemble des arêtes " "

X_{ij} = nombre de fois qu'une arête allant du sommet i au sommet j est traversé.

C_{ij} : distance de l'arête allant de sommet i au sommet j.

— L'expression (1) étant la fonction objective qui minimise la distance requise du trajet qui traverse chaeque arête au moins une fois.

— (2) exprime la continuité du flow requis, c'est-à-dire que le nombre d'arêtes incident de sa même sommet.

— (3) Traduit le fait que chaque arête doit être au moins traversée une fois.

— (4) exprime le non existence d'un cycle négatif.

- II faut remarquer que celte formulation nous aide á définir formellement le problème, mais ne peut être une méthode efficace de résolution ; car si on considére n'importe quelle muni Cipalité ou ville, elle possède au moins des centaines de rues et que le nombre de contraintes ainsi que le nombre de variables est au moins le double du nombre des rues.
C'est ce qui fait l'importance des résultats de kwan.

3.14.2 Algorithme

La tachnique étaient appliquée pour la première a fois á la dátermination de l'itinéraire d'un factaur qui souhaitait faire ses tournées et revenir á lo poste tout an minimisant la distance á parcourir.
Et à cause das rásultats de Kwan le problème : "Trouver un cyele de longueur minimale qui traverse chaque arête au moins une fois est devenu le "problème du postier chinois".

L'algorithme est le suivant :

1. Si tous les sommets sont pairs, le poids total du graphique est la distance la plus courte.
2. S'il n'y a que deux sommets impairs, associez le ou les arcs entre eux, puis trouvez le poids total du graphique.
3. S'il y a plus de deux sommets impairs, associez-les tous, puis trouvez le poids minimum ajouté.

3.14.3 Le problème du postier chinois non orienté

Le problème d'augmentation minimale est résolu sur un graphe qui ne contient que les sommets de degré impair du graphe original. Ce graphe contient également une arête entre chaque couple de sommets distincts, dont la longueur correspond à la longueur de la chaîne plus courte entre les deux sommets dans le graphe original. Le problème d'augmentation correspond alors à un problème de couplage de coût minimal.

3.14.4 Le problème du postier chinois orienté

Pour le cas orienté, le problème d'augmentation est défini sur les sommets dont le nombre d'arcs entrants est différent du nombre d'arcs sortants. Nous distinguons les sommets en déficit où le nombre d'arcs entrants est inférieur au nombre d'arcs sortants, et les sommets en surplus où le nombre d'arcs sortants est inférieur au nombre d'arcs entrants. L'objectif est d'ajouter des arcs à coût minimum entre les sommets en surplus et les sommets en déficit de façon à ce qu'ils soient en équilibre. Le problème d'augmentation correspond ici à unproblème de transport où les sommets en surplus sont des sources et sommets en déficit des puits.

3.14.5 Le problème du postier chinois mixte

Pour les graphes mixtes, le problème d'augmentation minimale est NP-difficile. Il existe pour le résoudre des méthodes exactes comme la méthode de Nobert et Picard.

3.15 L'algorithme génétique proposé pour la minimisation du coût total de transport

On suppose que « N ≫ est la taille de la population initiale, ´ $F_i\rangle$ la fonction objectif à minimiser (donnée par l'équation 3.3) correspondant à l'itération i. Nous déterminons ensuite :

$$f_{\text{recherche}} = \min_{i} f_i \tag{3.3}$$

L'algorithme génétique développé pour résoudre notre problème et pour minimiser le coût total du transport est illustré dans Algo 3.5.

```
Début
 Etape 1: Création de la population initiale de taille = n);
 Etape 2: Affectation de tous les nœuds (clients) aux dépôts
 correspondants « création des clusters »;
 Tanque nombre de générations non atteint Faire
 Début
  Etape 3: Création de la population intermédiaire de taille = 2n avec
  de nouveaux individus de "croisement ou mutation"
  Etape 4: Application des procédures d'ordonnancement et de
  correction de capacité; «Appelez les algorithmes Algo IV.2,
  Algo IV.3 »
  Etape 5 : Création de la deuxième population intermédiaire de
  taille = 2n * 2n représentant les différentes routes prises par
  chaque véhicule;
  Etape 6 : Déterminer la valeur de fitness pour chaque individu dans
  la nouvelle population (coût de transport);
  Etape 7 : Ordonner la population en fonction de la valeur de fitness
  minimale (coût de transport);
  Etape 8 : // Copier la meilleure solution;
  Si La solution actuelle est la meilleure Alors
        La meilleure solution = la solution actuelle;
  Sinon
      Incrémenter le nombre de générations;
 FinTQ
Fin.
```

FIGURE 3.26 – L'algorithme génétique proposé pour résoudre un mono-objectif MD-VRPTW.

Conclusion

Les travaux de recherche effectués afin de résoudre un problème TSP avec l'utilisation des algorithmes génétiques, ont donnés naissance à plusieurs mécanismes génétiques et en particuliers les opérations de mutation. Dans cet mémoire, nous avons étudié empiriquement l'impact de l'affiliation de ce type d'opérateurs génétique sur la résolution du problème de voyageur de commerce. Les résultats obtenus montrent que les opérateurs de mutation qui proposent les meilleures solutions sont RSM et PSM.

Il est important de garder en tête que pour chaque problème d'optimisation l'espace des solutions est diffèrent et donc les opérateurs génétiques, étant moins performants dans les présents problèmes, peuvent s'avérer plus efficaces avec d'autres problème.

Conclusion générale

Ce mémoire noustiepermis d'aborder une classe d'algorithmes très célèbre : les algorithmes évolutionnaires qui s'inspirent de la théorie de la sélection naturelle élaborée par Charles Darwin pour résoudre différents problèmes de transport comme probléme de voyageur de commerce, tournée vehicule, ...

L'objectif de ce mémoire était le développement d'un environnement de génération de tests pour la résolution des problèmes de transport de grand taille et sité les différents travaux des chercheurs Modifie qui on fait sur les algorithmes génétique pour trouvé meilleur solution approchée.

Finalement, cet algorithme est efficace quand il soit appliqué sur des problèmes de grande taille, car il permet de trouver de bonnes solutions approchées.

Bibliographie

[1] Collette, Y, et Siarry, P. (2002). Optimisation multiobjectif. Eyrolles. ISBN-13 : 978-2212111682.

[2] Marler, R.T., and Arora, J.S. (2004). 'Survey of multi-objective optimization methods for engineering'. Springer-Verlag. 26(6) : 369-395.

[3] S. Bandyopadhyay, A. Mukherjee, An algorithm for many-Objective optimization with reduced objective computations : A study in differential evolution. IEEE Transactions on Evolutionary Computation, 2014.

[4] N. Benhamed, Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés : sélection et pondération des primitives par algorithmes génétiques, Ecole de technologie supérieure : Génie de la production automatisée. Montréal : Université de Montréal, Québec. 137p, (2002).

[5] A. Berro, Optimisation multiobjectif et stratégies d'évolution en environnement dynamique, Thèse de doctorat : Informatique. Université des Sciences et Technologie de Toulouse I. France. 170p (2001).

[6] Borisovsky, P, Dolgui, A, Eremeev , A. Genetic algorithms for a supply management Problem : Mip-recombination vs greedy decoder. European Journal of Operational Research". 195 : 770-779, (2009).

[7] Chabane, B., Basseur, M., et Hao, J.K. (2015) Cas pratique pour le problème du sac-à-dos multiobjectif : Conception, modélisation, tests et analyse, 16$^{\text{éme}}$ conférence de la Recherche Opérationnelle et Aide à la Décision Française.

[8] Barichard, V. (2003). Approches hybrides pour les problèmes multiobjectifs, Thèse de doctorat : Informatique. Université d'Angers. France. 162p.

[9] Basseur, M. (2014). "Analyse et conception de recherches locales génériques pour l'optimisation combinatoire à un ou plusieurs objectifs. Mémoire d'habilitation à diriger des recherches : Informatique. Université d'Angers. France. 180p.

[10] Belhoul., L. (2014). Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée. Thèse de doctorat : Analyse et Modélisation de Systèmes pour l'Aide à la Décision. Université Paris-Dauphine. 140

[11] Jean-Michel Renders. Algorithmes génétiques et réseaux de neurones. Hermes, 1995.

[12] J. Achtnig. Particle Swarm Optimization with Mutation for High Dimensional Problems. Engineering Evolutionary Intelligent Systems. pp 423-439. Springer, 2008.

[13] G. B. Dantzig, R. Fulkerson, S. Johnson. Solution of a large-scale traveling salesman problem. Operations Research. Vol. 2, N° 4, pp 393- 410, 1954.

[14] Dorigo et L.M. Gambardella. Ant colony system : a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation. Vol. 1, N°. 1, pp.53-66. 1997.

[15] A. Otman, A. Essaadi J. Abouchabaka, Analyse des performances d'opérateurs de mutation génétique à la résolution du Problème de Voyageur de Commerce, October 2011.

[16] L. GUEZOULI, Développement d'une approche hybride multi-critères basée clustering pour la résolution d'un problème de distribution de produits, Master en informatique, option ingénierie de réseau et communication.

[17] R. Zaghdoud, Hybridation d'algorithme génétique pour les problèmes des véhicules intelligents autonomes : applications aux infrastructures portuaires de moyenne taille. Automatique, Ecole Centrale de Lille, 2015. Français.

[18] Camelia-M. Pintea, Petrica C. Pop et Camelia Chira, The generalized traveling salesman problem solved with ant algorithms, Complex Adaptive Systems Modeling, volume 5(8), 2017.

[19] S. Mostefa, LOCALISATION DES FUITES D'EAU PAE LE CALAGE DE MODELE FONDE SUR L'ALGORITHME GENETIQUE, MEMOIRE DE MASTER.

[20] J.Y. Potvin and J.M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operations Research, 66 :331-340, 1993.

[21] P. Shaw. Using constraint and local search methods to solve routing problems. In : CP-98, Fourth international conference on principles and practice of constraint programming, Lecture notes in computer science, 1520 :417-431, 1998.

[22] L.H. Shih and H.C. Chang. A routing and scheduling system for infectious waste collection. Environmental Modeling and Assessment, 6 :261-269, 2001.

[23] M.M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. Operations Research, 35 :254-265, 1987.

[24] T. Stützle. Local search algorithms for combinatorial problems-analysis, algorithms and new applications. DISKI-Dissertationen zur Künstliken Intelligenz. Inx, Sankt Augustin, Germany, 1999.

[25] E. Taillard. Parallel iterative search methods for vehicle routing problems. Networks, 23 :661-673, 2005.

[26] E.G. Talbi. A taxonomy of hybrid metaheuristics. Journal of Heuristics, 8(5) :541-564, 2002.