N° d'ordre : 15/2023-C/MT

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIÈNE

FACULTÉ DE MATHÉMATIQUES

THÈSE DE DOCTORAT

PRÉSENTÉE POUR L'OBTENTION DU GRADE DE DOCTEUR

En Mathématique

Spécialité : Optimisation Stochastique

Présentée par : AMINA GUERROUMA

THÈME :

Le Problème de Sac à Dos Quadratique Stochastique Multi objectif

Soutenue publiquement le 02/ 03 /2023 devant le jury composé de :

Mr. MOULAÏ Mustapha	Professeur à l'USTHB	Président
Mr. AÏDER Méziane	Professeur à l'USTHB	Directeur de thèse
Mr. CHAABANE Djamal	Professeur à l'USTHB	Examinateur
Mme. ADICHE Chahrazad	Maître de Conférences/A à l'Univ. UMB-Boumerdes	Examinatrice
Mme. DAHMANI Isma	Maître de Conférences/A à l'USTHB	Examinatrice

RÉSUMÉ :

Le problème du sac-à-dos est fondamental dans l'optimisation combinatoire et possède plusieurs variantes. Dans notre étude, nous nous concentrons sur le problème de sac-à-dos quadratique stochastique multi-objectifs avec des poids aléatoires. Nous proposons l'Algorithme Memetique Avec Recherche Locale de Pareto par Voisinage de Sélection (MASNPL). À chaque itération de cet algorithme une série de croisement, de mutation et de recherche locale sont appliqués à une population de taille N pour générer de nouvelles solutions (enfants) dont nous appliquons à cette dernière l'algorithme SNPLS basé sur la comparaison entre une solution actuelle (enfant) et une nouvelle solution obtenue par l'algorithme de gradient. Ensuite, nous utilisons un opérateur de sélection afin de sélectionner les meilleurs individus (c'est-à-dire qui ont une meilleure répartiti- on sur le front Pareto optimal) parmi les individus de la population combinée de parents et de progénitures. Le principe de l'opération de sélection repose sur le choix des individus appartenant aux meilleurs fronts non-dominé et possédants un voisinage le moins dense.

Quant à la deuxième étude, nous nous concentrons sur le problème de sac-à-dos multiple stochastique multi-objectifs à deux étapes et à poids aléatoires, dans le but de résoudre un tel probléme, nous proposons l'algorithme memetique avec MTHM et MTHMn. À chaque itération de cet algorithme, une série de croisement, de mutation et de recherche locale sont appliqués à la population initiale réalisable (parents) obtenue par l'heuristique gloutonne et améliorée par l'heuristique de Martello et Toth *MTHM* et MTHMn pour générer de nouvelles solutions (enfants). Ensuite, un opérateur de sélection est appliqué à la population combinée (parents et enfants) afin de sélectionner les meilleurs individus (c'est-à-dire qui ont une meilleure répartition sur le front Pareto optimal et possédant un voisinage le moins dense).

Remerciements

Je remercie Dieu le tout puissant qui m'a bénis et permis d'arriver là où je suis..

Tout d'abord, Je tiens à exprimer ma gratitude et mes remerciements les plus sincères à mon Directeur de Thèse, Monsieur. AÏDER Méziane, pour avoir dirigé et encadré mon travail. Je le remercie pour sa disponibilité, son soutien et ses conseils tout au long de ces années de préparation de mon projet de thèse, pour sa patience, sa motivation.

Je remercie également Monsieur. MOULAÏ Mustapha, Professeur à l'USTHB pour avoir accepté de présider ce jury et pour l'intérêt qu'il porte à ce travail.

J'exprime ma gratitude et mes remerciements à Monsieur. CHAABANE Djamal Professeur à l'USTHB avoir accepté de participer à ce jury de thèse en tant qu'examinateur.

J'exprime ma gratitude et mes remerciements à Madame. ADICHE Chahrazad, Maître de Conférences A à l'université UMB-Boumerdes pour avoir fait partie de ce jury de thèse en tant qu'examinatrice.

J'exprime ma gratitude et mes remerciements à Madame. DAHMANI Isma, Maître de Conférences A à l'USTHB pour avoir fait partie de ce jury de thèse en tant qu'examinatrice.

Mes pensées vont également à mes amis, Hassiba, Hicham que j'ai eu le plaisir de les côtoyer et de faire leurs connaissances Je les remercie pour leurs soutien et encouragements.

Je tiens à exprimer ma profonde reconnaissance et remerciements à tous les membres de ma famille, en particulier mes très chers parents et ma chère sœur et mes chers frères et mes petits neveux.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

Liste des algorithmes

INTRODUCTION

De nombreux problèmes réels, touchant des domaines variés (industriels, économiques, militaires, etc.) peuvent être modélisés comme des problèmes d'optimisation combinatoire. Ces derniers consistent à chercher dans un ensemble discret un sous-ensemble parmi les meilleurs sous ensembles de façon à maximiser ou minimiser un critère (ou plusieurs critères) sous certaines contraintes.

Certains problèmes réels peuvent être modélisés sous la forme de programmes linéaires, de manière à ce que cette modélisation soit la plus proche possible d'un modèle théori- que de la littérature pour lequel différentes approches de résolution efficaces ont été proposé- es. L'étude de ce genre de modèles théoriques est d'une grande importance pour les pren- eurs de décisions, dans la mesure où elle met à leur disposition un ensemble de méthodes efficaces, capables de résoudre leurs problèmes particuliers. Définir une méthode de résolution efficace, capable d'obtenir une solution optimale, avec une complexité spati- ale et temporelle raisonnable, est un objectif difficile à atteindre, en particulier si l'instance traitée est de grande taille.

En général, les méthodes de résolution que l'on met en œuvre pour résoudre ce type de problème doivent prendre en considération deux facteurs : la qualité des solutions et le temps de résolution. Bien que ces deux facteurs soient généralement liés, parfois il est nécessaire de faire le choix entre trouver une solution (des solutions) optimale(s) ou de se contenter d'une solution (des solutions) approchée(s). Souvent, ce choix est influencé par la nature du problème traité. Toutefois, l'efficacité d'une méthode de résolution dépend de sa complexité temporaire. Si la complexité temporaire est bornée par une fonction polynomiale d'un paramètre caractérisant la taille du problème alors la méthode est dite *efficace* ou *polynomiale*. Dans la nomenclature de la complexité algorithmique, les problèmes d'optimisation combinatoire pour lesquels on ne connait pas d'algorith- mes de résolution polynomiaux, sont dits **NP-difficiles**. Ces problèmes sont réputés les plus difficiles parmi les problèmes d'optimisation combinatoire.

Les méthodes de résolution des problèmes d'optimisation combinatoire peuvent être classées en deux catégories :

Les méthodes de résolution capables de générer une solution optimale sont appelées *méthodes exactes*. Elles consistent en général à énumérer l'ensemble des solutions de l'espace de recherche de manière implicite. Cependant, elles ne permettent de résoudre

que des problèmes de tailles modérées, puisque leur temps d'exécution augmente exponentiellement avec la taille du problème. En général, les difficultés liées à la complexité de ce genre de méthodes rendent leur application difficile, voire impossible. Et ce, malgré le développement fulgurant des logiciels de résolution de problèmes d'optimisation. Face à un tel constat, une autre catégorie d'approches de résolution est apparue, à savoir *les méthodes de résolution approchées* qui constituent une alternative intéressante pour la résolution des problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale.

Généralement, les méthodes de résolution approchées qui perdent en optimalité pour gagner en efficacité et permettent de trouver rapidement des solutions réalisables du problème traité, mais sans garantie sur la qualité de celle-ci. Parmi ces méthodes on peut citer *les heuristiques*, *les méthodes gloutonnes* et *les métaheuristiques* représentées essentiellement par les méthodes de voisinage telles que la recherche tabou, le recuit simulé et les algorithmes évolutifs (les algorithmes génétiques et les recherches dispersées). Il existe aussi d'autres méthodes de résolution appelées *méthodes hybrides*, ces méthodes combinent des méthodes approchés avec des méthodes exactes et représentent un outil assez puissant pour résoudre les problèmes combinatoires.

L'objectif du travail présenté dans cette thèse est de proposer plusieurs méthodes de résolution approchées et exactes pour obtenir des solutions de bonne qualité en un temps d'exécution raisonnable pour une classe des problèmes d'optimisation combinat- oire, Nous nous concentrons principalement sur quelques variantes du problème du sac-à-dos qui fait l'objet de nombreuses études dans la littérature. Les différentes varia- ntes du problème du sac-à-dos restent des problèmes particuliers et difficiles à résoudre. Ils permettent de formuler de nombreux problèmes réels et sont souvent utilisés comme sous-problèmes d'autres problèmes plus complexes. Ils appartiennent à la classe des problèmes NP-difficiles.

Dans cette thèse, nous avons étudié le problème de sac-à-dos et quelques unes de ses variantes, plus précisément nous focalisons sur le problème de sac-à-dos stochastique quadratique multi-objectifs ainsi que le problème de sac-à-dos stochastique multi-objectifs multiples, dans le but de résoudre ces problèmes, nous avons utilisé de nouveaux algorithmes : hybrides et évolutionnaire.

Notre travail est structuré de la façon suivante :

Nous présentons dans le premier chapitre, un ensemble de définitions et de concepts de base liés au problème de sac-à-dos. Nous nous intéressons aux différentes variantes du problème de sac-à-dos : stochastique, quadratique, multidimensionnel et nous décrivons les concepts de base liés à l'optimisation multicritère utilisés.

Le deuxième chapitre est dédié à la présentation de notre état de l'art sur le problème de sac-à-dos et ses différentes variantes. Nous rappelons dans ce chapitre les différentes méthodes de résolutions existantes dans la littérature permettant de positionner notre travail par rapport aux travaux existants.

Dans le troisième chapitre, nous décrivons les concepts de base liés à l'optimisation multicritère utilisés ainsi que le problème de sac-à-dos stochastique, quadratique dans le cadre de cette thèse. Une présentation d'un nouvel algorithme hybride qui combine les métaheuristiques les plus performantes pour la résolution des problèmes multicritère, une méthode qui servira comme outil d'amélioration de la qualité de la solution existante ainsi qu'une aide à la sélection des meilleures solutions parmi les solutions obtenues. Un ensemble d'expériences numériques a été présenté et analysé, où nous compar- ons les performances de notre algorithme hybride avec une méthode exacte ainsi que NSGA-II.

Dans le quatrième chapitre, nous décrivons les concepts de base liés au problème sac-à-dos multiples stochastique et multi-objectifs. Nous exposons l'approche adoptée pour la génération d'une population initiale réalisable et l'algorithme évolutionnaire ainsi qu'une nouvelle heuristique permettant d'améliorer et de sélectionner les meilleures solutions parmi les solutions obtenues. Nous avons présenté et analysé un ensemble d'expériences numériques dans lesquelles nous avons comparé les performances de notre algorithme évolutionnaire avec celles d'une méthode exacte et de l'heuristique de Martello et Toth *(Martello and Toth Heuristic Method)* MTHM.

CHAPITRE 1

CONCEPTS FONDAMENTAUX

"La musique est une mathématique sonore, la mathématique une musique silencieuse.",
Edouard Herriot.

Sommaire

1.1 Introduction

DANS ce chapitre, nous présentons un ensemble de définitions et de concepts de base liés aux problèmes de sac-à-dos. Dans la première partie, nous rappelons quelques définitions de base sur le problème de sac-à-dos. Ensuite nous décrivons quelques variantes du sac-à-dos et nous rappelons brièvement la programmation stochastique ainsi que les concepts de base liés à l'optimisation multicritère.

1.2 Programmation stochastique

La notion *d'incertitude* dans la programmation mathématique est apparue pour la premiére fois dans les années 50 avec les travaux de Dantzig [**16**], Cooper & Charnes [**11**], Beale [**6**], et elle a rencontré depuis un développement rapide.

L'incertitude dans les problèmes d'optimisation touche notamment les coûts de production, les prix des marchés, les pénalités en cas des violations des contrats, aussi bien que la demande de clients, les délais de livraison, les temps de traitement, la disponibilité des machines et d'autres coefficients technologiques.

Il se peut que l'on connaisse partiellement certains aspects du phénomène à travers soit des scénarios, soit un historique, soit une loi de probabilité, soit des moments de la variable aléatoire (par exemple espérance mathématique, variance). Des scénarios peuvent être crées à partir des historiques, (comme par exemple les ventes des dernières années, les températures de la dernière décennie, la mortalité d'une certaine population), ou bien à partir de l'opinion d'experts qui peuvent prévoir le comportement du phéno- mène incertain. Si la loi de probabilité de la variable aléatoire est connue de façon analytique, on peut soit créer des méthodes analytiques (la plupart des fois, ceci est un travail non-trivial) qui tiennent compte de cette loi de probabilité, soit générer un ensemble de scénarios suivant cette loi.

Le but que l'on se fixe dans un problème d'optimisation est de prendre la meilleure décision vis à vis des situations qui comportent de l'incerti- tude. Le regard vers l'incertitude reste subjectif et varie d'un décideur à d'autre. On peut par exemple, souhaiter que dans l'espace de tout événement possible, la solution adoptée soit telle que son coût soit minimal en moyenne. Ainsi, les approches de l'incertitude comprennent des techn- iques orientées vers la minimisation (maximisation) de l'espérance mathématique (ou d'autres moments), la minimisation de la déviation par rapport aux cibles (deviation from goals), la minimisation des coûts maximaux, la minimisation de l'écart entre le meilleur et le pire cas, ou même la simple satisfaction des contraintes avec une probabilité donnée d'avance qui représente le taux de fiabilité du système.

Lorsque nous parlons de la programmation stochastique, on entend la programmation mathématique sous incertitude. La programmation stochastique n'est pas une famille de problèmes, de modèles et d'outils différents des autres domaines d'optimisation (programmation linéaire, non-linéaire, dynamique) ; au contraire, elle constitue un com- plément de ces familles lorsque la notion d'incertitude intervient. Nous nous permettons donc de parler de programmation stochastique linéaire.

Les approches qui dominent sur la modélisation et la résolution des problèmes de la programmation stochastique sont les suivantes :

- Modèles avec des contraintes probabilistes.
- Modèles de recours.
- Modèles robustes.

1.2.1 Modèle sous contraintes probabilistes

Des problèmes dont la modélisation appartient à la catégorie des modèles probabilistes sont apparus pour la première fois dans **[5]** sous le nom de programmation sous contraintes probabilistes *"Chance constrained programming"*.

L'idée de modélisation dans ce type de modèle consiste à imposer une probabilité pour violation des contraintes aléatoires. Soit nous imposons un seuil de probabilité pour chaque contrainte aléatoire p_i, $i = 1, \ldots, m$ avec $0 \leq p_i \leq 1$. Nous admettons que la fonction de coût ne dépend pas de la variable aléatoire ξ.

$$\begin{cases} \min c^t x \\ \mathbb{P}(T_i(\xi)x \geq h_i(\xi)) \geq p_i \quad i = 1, \ldots, m \\ Ax \leq b \end{cases} \tag{1.1}$$

p_i étant une probabilité réglée par l'utilisateur qui reflète une tolérance acceptable pour la violation des contraintes aléatoires.
Soit nous imposons un seuil de probabilité p, pour l'ensemble des contraintes. Le problème donc est le suivant :

$$\begin{cases} \min c^t x \\ \mathbb{P}(T(\xi)x \geq h(\xi)) \geq p \\ Ax \leq b \end{cases} \tag{1.2}$$

$$D\left(p_i\right) = \left\{x \in \mathbb{R}^n \mid P\left[T_i\left(\xi\right) \geq h_i\left(\xi\right)\right]\right\}$$

$$D\left(p\right) = \left\{x \in \mathbb{R}^n \mid \mathbb{P}\left[T\left(\xi\right) \geq h\left(\xi\right)\right]\right\}$$

Le probléme qui se pose dans cette approche est que l'ensemble des solutions des problèmes (1.1) et (1.2) n'est pas généralement convexe, de plus ce problème de convexité des ensembles $D(p_i)$et $D(p)$ dépend de la nature aléatoire de T, h et aussi des seuils de probabilité p, p_i, pour résoudre ce problème, nous citons quelques conditions de convexité des ensembles $D(p_i)$ ou $D(p)$.

- T et h sont aléatoires de distribution normale.
- T et h ne sont pas indépendantes

Nous supposons que (T_i, h_i) est un vecteur aléatoire normalement distribué de moyenne $\mu_i \in \mathbb{R}^{n+1}$ et de matrice de variance-covariance S.

$$D(\alpha_i) = \{x / P(\xi(x) \geq 0) \geq \alpha_i\}$$

$$D(\alpha_i) = \left\{x / P\left(\frac{\xi(x) - m_{\xi}(x)}{\sigma_{\xi}(x)} \geq \frac{-m_{\xi}(x)}{\sigma_{\xi}(x)}\right) \geq \alpha_i\right\}$$

$$= \left\{x / 1 - \Phi\left(\frac{-m_{\xi}(x)}{\sigma_{\xi}(x)}\right) \geq \alpha_i\right\}$$

$$= \left\{x / \Phi\left(\frac{-m_{\xi}(x)}{\sigma_{\xi}(x)}\right) \leq 1 - \alpha_i\right\}$$

$$= \left\{x / \frac{-m_{\xi}(x)}{\sigma_{\xi}(x)} \leq \Phi^{-1}(1 - \alpha_i)\right\}$$

$$= \left\{x / -m_{\xi}(x) - \sigma_{\xi}(x)\Phi^{-1}(1 - \alpha_i) \geq 0\right\}$$

Comme $m_{\xi}(x)$ est affine en x et $\sigma_{\xi}(x)$ est convexe en x la contrainte $\left\{-m_{\xi}(x) - \sigma_{\xi}(x)\Phi^{-1}(1 - \alpha_i) \geq 0\right\}$ est convexe si est seulement si $\left\{\Phi^{-1}(1 - \alpha_i) \geq 0\right\}$ et c'est le cas si $\alpha_i \geq 0,5$

- T et h sont indépendantes.

Soit les distributions des variables T et h

$$T_{ij} \rightsquigarrow N\left(\mu_{ij}, V_{ij}^2\right), h \rightsquigarrow N\left(m_i, \sigma_i^2\right)$$

Soit la variable $y_i = T_{ix} - h_i$ a pour distribution :

$$y_i \rightsquigarrow N\left(\sum_{j=1}^{n} \mu_{ij} - m_i, \sum_{j=1}^{n} V_{ij}^2 x_j^2 + \sigma_i^2\right)$$

D'où

$$D(\alpha_i) = \left\{ x \in \mathbb{R}^n \diagup \sum_{j=1}^{n} \mu_{ij} x_j - m_i + \Phi^{-1}(\alpha_i) \sqrt{\sum_{j=1}^{n} V_{ij}^2 x_j^2 + \sigma_i^2} \leq 0 \right\}$$

- T est déterministe et h est aléatoire.

Dans ce cas le problème est simple. Soit F_i la fonction de répartition de h_i, alors

$$D(\alpha_i) = \{x \diagup P(T_i(\xi)\, x \geq h_i(\xi)) \geq \alpha_i\}$$

$$D(\alpha_i) = \{x \diagup F_i(T_i x) \geq \alpha_i\}$$

$$D(\alpha_i) = \left\{x \diagup T_i x \leq F_i^{-1}(\alpha_i)\right\}$$

$D(\alpha_i)$ est un ensemble de contraintes linéaires en x, donc il est convexe.

1.2.2 Programmation stochastique avec recours

La programmation stochastique s'intéresse essentiellement à des problèmes où on doit prendre une décision sur le champ sous présence d'incertitude, sans attendre la réalisation des certaines variables aléatoires ; on fait souvent appel à ce type de problèmes sous le nom de *"here and now problems"*. Dans la famille *"here and now"* on est obligé de déterminer au moment t_0 certaines valeurs x_0^j qui ne peuvent plus changer dans la suite, quoi qu'il arrive aux instants ultérieurs. Il se peut qu'une décision à l'instant initial t_0 coûte trop cher dans le futur.

L'idée est d'optimiser les décisions que l'on doit prendre au moment t_0 tout en tenant compte des conséquences de ces décisions pour toute réalisation des aléas qui pourrait se produire aux instants ultérieurs. Le mot *"recours"* révèle la possibilité dont on dispose de remédier à une décision éventuellement trop optimiste à l'instant t_0, en mettant en œuvre des actions correctives, évidemment plus chères, qui satisfont pourtant les contraintes posées aux instants ultérieurs t_i , $i > 0$. On précise qu'après avoir pris la décision au moment t_0, on passe au moment t_1 où tous les événements inconnus jusqu'à cet instant se révèlent. Par conséquent, l'étude de tels cas a un intérêt seulement si on dispose de quelques informations sur les réalisations des événements qui nous sont inconnus au moment t_0 (information comme par exemple la distribution de proba- bilité, des scénarios, la moyenne, la variance etc.).

Le problème peut s'étendre en plusieurs étapes suivant l'horizon d'étude, mais considérons d'abord le cas de deux étapes seulement. Les variables de la première étape sont considérées comme étant des variables structurantes, stratégiques qui ne peuvent plus changer à la seconde étape. La décision de la première étape doit être prise en respectant des contraintes propres à cette étape. La seconde étape comprend des actions (pénalis- antes) qui peuvent être entreprises afin de satisfaire des contraintes qui font

intervenir des variables de la première étape. Toutes ces idées sont résumées dans le programme suivant :

$$\begin{cases} \min c^t x + Q(x) \\ s.c.\ \ Ax \leq b \\ x \geq 0 \\ t.q\ \ Q(x) = \mathbb{E}\left[Q(x,\xi)\right] = \sum_j p^j Q\left(x,\xi^j\right) \end{cases} \tag{1.3}$$

$$Q(x,\xi) = \min\left\{q(\xi)^t y \mid W(\xi)\, y = h(\xi) - T(\xi)\, x, y \geq 0\right\} \tag{1.4}$$

Où p^j est la probabilité pour $\widetilde{\xi} = (\xi)^j$, la $j^{éme}$ réalisation de $\widetilde{\xi}$, $W(\xi)$ est dite matrice de recours de dimension (m_0, n_0)
$h(\xi) = h_0 + H\xi = h_0 + \sum_i h_i\xi_i$, $T(\xi) = T_0 + \sum_i t_i\xi_i$ et $q(\xi) = q_0 + \sum_i q_i\xi_i$ et $Q(x,\xi)$ représente la valeur optimale du recours et représente l'espérance mathématique du coût de recours.
Le problème (1.3) constitue le premier niveau, qui doit être résolu sans les contraintes aléatoires et le problème (1.4) constitue le problème du second niveau pour une décision x, déterminée et une réalisation de ξ.

Les différents types de recours

dans la littérature, il existe plusieurs type de recours dont nous citons :

Recours fixe

Définition : le cours est considéré comme étant fixe ou déterministe si les valeurs q et W d'un programme avec recours (1.3) ne dépendent pas de ξ, leur valeurs sont à priori connues.

Recours complet

Définition : le recours fixe est complet si :
$\forall x \in \mathbb{R}^*$ *il existe* $y \in Y$ tels que $Q(x,\xi) < +\infty; \forall \xi \in \Xi$.

Recours relativement complet

Définition : le recours fixe est relativement complet si :

$\forall x \in \chi \subseteq \mathbb{R}^*$ *il existe* $y \in Y$ tels que $Q(x,\xi) < +\infty; \forall \xi \in \Xi$. La différence entre les deux types de recours porte sur l'ensemble des x pour lesquels il existe une solution au second niveau. Si cet ensemble est le domaine de définition de x à la première étape est relativement complet, tandis que si c'est alors le recours est complet.

Recours simple

Définition : c'est un cas particulier de recours fixe complet, il correspond au cas où la matrice de recours $W = (I, -I)$ où I représente la matrice d'identité d'ordre m_0.

1.2.3 Méthode de décomposition L-Shaped

Dans cette partie nous décrivons la méthode de décomposition L-Shaped de Van Slyke et Wets, aussi connue sous la forme de la méthode de décomposition de Benders.
Le principe de cette méthode consiste à décomposer le problème en deux étapes. Les variables du premier niveau sont les variables dont la valeur doit être fixée avant la réalisation des variables aléatoires et ne peut en aucun cas être modifiée suivant la réalisation des variables aléatoires. D'autre part, les variables du second niveau (les variables de recours) sont autorisées à ddépendre des variables et variable du premier niveau. Le principe de la méthode de L-Shaped est que nous tenons compte de toute information disponible afin de prendre les décisions, mais en même temps, il est irréaliste de tenir compte des événements inconnus, qui n'existent, au moment de la prise de décision, que dans un espace probabiliste.

Test de faisabilité

Nous introduisant une autre définition pour la faisabilité. Si nous disposons d'une solution du premier niveau $x = x^0$ du problème (1.3), comment décidons que cette solution est réalisable pour le problème (1.4) et pour toutes les valeurs possibles de ξ, dans le cas où nous ne pouvons pas sovoir si le recours est relativement complet.

Lemme : Lemme de Farkas

$$\{y / Wy = h, y \geq 0\} \neq \varnothing$$

Si est seulement si

$$W^t u \geq 0 \Longrightarrow h^t u \geq 0$$

Nous changeons le signe de u, la seconde partie de l'équivalent peut être réécrite comme suit :

$$W^t u \leq 0 \Longrightarrow h^t u \leq 0$$

Ou de façon équivalente :

$$h^t u \geq 0 \text{ quand } \; t \in \left\{u / W^t u \geq 0\right\}$$

Cependant, cela peut être reformulée comme suit :

$$\left\{u / W^t u \leq 0\right\} = \left\{u / u^t W y \leq 0 \quad \forall y \leq 0\right\}$$

$$\left\{u \mid W^t u \leq 0\right\} = \left\{u \mid u^t h \leq 0, \forall h \in pos\, W\right\} \tag{1.5}$$

L'expression (1.5) définit le cône polaire de $posW$

$$polposW = \left\{u / u^t h \leq 0 \quad \forall h \in pos\ W\right\}$$

Dans le cas où nous ne connaissons pas toutes les valeurs de $polposW$, et nous ne sommes pas au courant du recours relativement complet, pour un x^0 donné et pour tous ξ nous devons vérifier la faisabilité.

Nous voulons trouver σ tels que :

$$\sigma^t t \leq 0 \quad \forall t \in pos\ W$$

Ceci est équivalent à imposer que les $\sigma^t W \leq 0$. en d'autres termes,σ doit être dans le cône $polposW$, nous devrions en même temps imposer que $\sigma^t \left[h\left(\xi\right) - T\left(\xi\right) x^0\right] \geq 0$ car si nous pouvons rajouter la contrainte $\sigma^t \left[h\left(\xi\right) - T\left(\xi\right) x\right] \leq 0$ au problème (1.6).

$$\begin{cases} \min f = c\left(\xi\right) x \\ s.c.\ T\left(\xi\right) x = h\left(\xi\right) \\ x \in D \end{cases} \tag{1.6}$$

Où $x \in \mathbb{R}^n$, (T, C, h) de dimension respective $\left(m_0 \times n\right)$, $\left(1 \times n\right)$, et $\left(m_0 \times 1\right)$.
Les coefficients de $T\left(\xi\right)$, $c\left(\xi\right)$ ainsi que $h\left(\xi\right)$ sont des variables aléatoires définies sur un espace de probabilité (Ω, Ξ, P) et D est un polyèdre convexe déterministe $D = \{x / Ax \leq b, x \geq 0\}$
Nous pouvons exclure $\left[h\left(\xi\right) - T\left(\xi\right) x^0\right]$ sans exclure des solutions réalisables, ainsi nous résolvons le problème suivant :

$$\max_{\sigma} \left\{\sigma^t \left(h\left(\xi\right) - T\left(\xi\right)\right) x^0 / \sigma^t W \leq 0, \|\sigma\| \leq 1\right\} \tag{1.7}$$

La dernière contrainte est introduite pour borner σ, sinon la valeur maximale sera égale à $+\infty$, et cela ne nous intéresse pas car nous cherchons une direction bien définie par σ. si pour un certain ξ_i nous avons trouvé que $\left[h_i - T_i x^0\right] \sigma > 0$ alors pour ce ξ_i la solution du premier niveau $x = x^0$ ne gère pas le problème du second niveau réalisable, par conséquent nous devons exclure cette solution x^0, en créant alors la coupe d'admissibilité :

$$\sigma^t \left[h_i - T_i x\right] \leq 0 \tag{1.8}$$

Test d'optimalité Nous supposons que nous disposons d'un recours relatif complet ou bien des coupes d'admissibilité, le problème de la deuxième étape :

$$\min \left\{ q\left(\xi\right)^{t} y / Wy = h\left(\xi\right) - T\left(\xi\right) x, y \geq 0 \right\} \tag{1.9}$$

Est réalisable et son problème dual est donnée par :

$$\max_{\pi} \left\{ \pi^{t} \left(h\left(\xi\right) - T\left(\xi\right) x / \pi^{t} W \leq q\left(\xi\right)^{t} \right) \right\} \tag{1.10}$$

Le problème (1.3) peut être réécrit en introduisant une nouvelle variable θ

$$\begin{cases} \min c^{t} x + \theta \\ s.c. \quad Ax \leq b \\ \sigma^{t} T x \geq \sigma^{t} h \\ \theta \geq Q\left(x\right) \\ x \geq 0 \end{cases} \tag{1.11}$$

x^{0} une solution admissible du premier niveau et θ^{0} initialement fixé à $-\infty$, la valeur de $Q\left(x^{0}\right)$ est calculée à partir du problème (1.9)

$$Q\left(x^{0}\right) = \sum_{i}^{N} p^{i} Q\left(x^{0}, \xi^{i}\right) = \sum_{i}^{N} p^{i} \pi_{i}^{t} \left(h_{i} - T_{i} x^{0} \right) \tag{1.12}$$

Si $Q\left(x^{0}\right) < \theta^{0}$, alors x^{0} est une solution optimale, sinon nous excluons la solution x^{0} et nous créons une coupe d'optimalité

$$\theta \geq \sum_{i}^{N} p^{i} \pi_{i}^{t} \left(h_{i} - T_{i} x \right) \tag{1.13}$$

1.3 programmation linéaire stochastique multi-objectifs

1.3.1 Formulation du problème linéaire stochastique multicritère

Le problème linéaire stochastique multicritèree est définie, en général comme suit :

$$\begin{cases} \min F_{k} = c_{k}\left(\xi\right) x \quad k = 1, ..., m \\ s.c. \quad T\left(\xi\right) x = h\left(\xi\right) \\ Ax \leq b \\ x \geq 0 \end{cases} \tag{1.14}$$

Avec (c, T, h) de dimension respective (m, n) , (m_0, n) , et $(m_0, 1)$. Les coefficients (T, c, h) sont des variables aléatoire de distributions connues, définies sur un espace de probabilité (ω, Ξ, P)

1.3.2 Différentes méthodes de résolution

La résolution des problèmes multi-objectifs stochastiques nécessite deux transformations, la première étape consiste à transformer le problème multicritères en un problème monocritère et la deuxième transforme le problème stochastique en un problème déterm- iniste.

1. Approche multicritère Dans cette approche on utilise les critères de la programmation linéaire stochastique unicritère *E-modèle, V-modèle, E.V-modèle, P-modèle, K-modéle*.

2. Approche stochastique Cette approche consiste à appliquer une des méthodes d'obtention des solutions efficaces de la programmation multicritères déterministe qui conduise généralement à la résolution d'un programme stochastique avec un seul critère.

Concepts de solutions efficaces

Dans ce paragraphe nous donnons quelques définitions de solutions efficaces. Ces définitions coïncident avec l'approche objective et elles sont déterminées avec transfor- mation de problème multi-objectifs stochastique en un problème multi-objectifs déterministes. La résolution des problèmes déterministes donne un ensemble de solutions efficaces considérées comme étant des solutions efficaces du problème initial.

Définition.(Solution efficace en moyenne) x^* est dite solution efficace en moyenne pour le problème (1.14), si elle est efficace pour le problème

$$\begin{cases} \min \widetilde{F}_k = \widetilde{c_k}(\xi)\, x \quad k = 1, ..., m \\ tel\ \ que. \quad \widetilde{F}_k = E(F_k) = \sum_{i=1}^{N} p_i F_{ki} = \sum_{i=1}^{N} p_i c_k(\xi_i)\, x = E(c_k(\xi)\, x) \end{cases}$$

Définition :(Solution efficace avec variance minimum) Une solution x^* est dite **efficace avec variance minimum** pour le problème (1.14) si elle est efficace pour le problème :

$$\min_{x \in S} \left(\sigma_1^2(x), ..., \sigma_m^2(x)\right)$$

$\sigma_k^2(x)$ est la variance du $K^{ème}$ objectif.

Définition :(Solution efficace avec un risque minimum) Une solution x^* est dite **efficace avec un risque minimum** pour le problème (1.14) si elle est efficace pour le problème.

$$\max_{x \in S} \left(P(F_1 \leq \alpha_1), ..., P(F_m \leq \alpha_m)\right)$$

L'application de ce critère nécessite la connaissance des fonctions de répartitions ou bien les lois de probabilité des objectifs stochastiques, ainsi que le niveau d'aspiration qui est fixé par le décideur.

Définition :(Solution efficace avec probabilités) x^* est une solution efficace avec probabilités $(\alpha_1, ..., \alpha_m)$ pour le problème (1.14) s'il existe $\beta^* = (\beta_1^*, ..., \beta_m^*)$ telle que (x^*, β^*) est efficace pour le programme suivant :

$$\begin{cases} \min_{(x,\beta)} (\beta_1, ..., \beta_m) \\ S.C. \quad P\left(c^t(\xi) \leq \beta_k\right) \geq \alpha_k \quad k = 1, ..., m \end{cases}$$

Où $(\alpha_1, ..., \alpha_m)$ sont les seuils de probabilité qui sont fixés à priori par le décideur.

1.4 Problème d'optimisation multi-objectifs

Dans plusieurs problèmes de la vie réelle, il y a généralement plus d'un objectif à atteindre. Les problèmes d'optimisation combinatoire multi-objectifs MCOP sont la désignation commune des problèmes d'optimisation combinatoire avec plus d'un objectif à optimiser. Ces problèmes sont caractérisés par des solutions multiples, c'est-à-dire qu'il n'y a pas une seule meilleure solution *optimum global*, La résolution de tels problèmes consiste à trouver l'ensemble de ces solutions optimales. Ces dernières sont appelées *solutions efficaces* et l'ensemble de solutions réalisables particulières appelées *ensemble de Pareto* ou *solutions non-dominées*, qui sont supérieures aux autres lorsqu'on considère tous les objectifs. Les solutions multiples s'expliquent par le fait que les objectifs sont contradictoires. Par exemple, lors du choix d'une nouvelle voiture à acheter, plusieurs objectifs peuvent être considérés. D'une part, il est souhaitable d'acq- uérir une voiture qui présente de bonnes performances (en termes de vitesse maximale et de capacité d'accélération). D'autre part, la voiture doit être sûre, économique et présente le coût d'acquisition le plus faible possible. Où $(\alpha_1, ..., \alpha_m)$ sont les seuils de probabilité qui sont fixés à priori par le décideur.

Un problème d'optimisation multi-objectifs MO consiste à minimiser (cas de minimisation) simultanément un ensemble de m fonctions objectifs :

$$\min_{x \in E} f(x) = \min_{x \in E} (f_1(x), f_2(x), ..., f_m(x),) \tag{1.15}$$

où

- E désigne l'espace des solutions réalisables,
- m est le nombre de fonction objectif,

- Chaque solution x a pour image dans l'espace des objectifs un point $y : y = f(x)$ avec $y \in \mathbb{R}^m$,
- x est le vecteur de variables de décisions, i.e., $x = (x_1, x_2, ..., x_n)^T$.

1.4.1 Définition de la dominance

Une solution $x \in E$ domine $\overline{x} \in E$ si elle vérifie :

$\forall k \in \{1, ..., m\}, f_k(x) \leq f_k(\overline{x})$.
$\exists k \in \{1, ..., m\}, f_k(x) < f_k(\overline{x})$.

On note cette relation de dominance $x \preceq \overline{x}$.

Cette définition permet d'introduire le concept de *Pareto-optimalité*.

1.4.2 Définition de pareto-optimal

Une solution est dite *efficace* ou *Pareto-Optimal* si elle n'est dominé par aucune autre solution appartenant à E. l'ensemble de ces solutions sont également appelées *solutions non-dominées*.

1.4.3 Définition de front de pareto

L'image des solutions efficaces forme dans l'espace des objectifs un ensemble de points non-dominés, communément appelé *Front De Pareto*.

1.4.4 Définition de solution efficiente

Une solution réalisable $x \in E$ est dite *efficiente* s'il est impossible de trouver une autre solution réalisable $y \in E$ qui soit au moins aussi meilleure que x pour tous les objectifs et strictement meilleure pour au moins un critère, c'est-à-dire. une solution x est dite *efficiente*, si et seulement s'il est impossible de trouver une autre solution réalisable $y \in E$ telle que $f_k(y) \geq f_k(x)$ pour tous les $k \in \{1, \ldots, m\}$ et $f_k(y) > f_k(x)$ pour un $k \in \{1, \ldots, m\}$.

1.5 Définitions du problème de sac-à-dos

Le problème de sac-à-dos, connu également sous le nom de (*Knapsack Problem* (KP)), est un problème fondamental en recherche opérationnelle et l'un des problèmes d'optimisation combinatoire les plus examinés dans la littérature. Notons que c'est un problème appartenant à la classe des problèmes *NP-difficiles*. Son principe se base sur la sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Il tire ses origines du choix du randonneur lors du remplissage de son sac-à-dos. Il lui faut choisir les articles à emporter de telle façon à avoir les articles les

plus utiles (critère de qualité) tout en respectant le volume du sac-à-dos (capacité de la ressource), la question qui se pose est de savoir quels articles il doit mettre dans le sac. Il existe plusieurs variantes du problème de sac-à-dos. Elles diffèrent dans la distribution des articles et des sacs. Les méthodes de résolution exacte pour ce problème sont la programmation dynamique et la procédure d'évaluation et de séparation. Les méthodes de résolution approchées regroupent les algorithmes génétiques, les algorithmes de colonies de fourmis, et bien d'autres méthodes.

Dans le problème de sac-à-dos en variables binaires, chaque article peut être pris dans le sac au plus une fois. En revanche, dans un problème de sac-à-dos borné, nous avons une quantité bornée de chaque type d'article. Dans le problème de sac-à-dos à choix multiple, les articles sont choisis de façon disjointe. S'il existe plusieurs sacs à remplir simultanément, le problème de sac-à-dos est multidimensionnel. Notons que toutes les variantes de problème de sac-à-dos sont NP-difficile. Par ailleurs, les différentes variantes de problème de sac-à-dos, bien qu'elles semblent très proches, ne font pas du tout appel aux mêmes méthodes de résolution (exactes ou approchées) pour une variante donnée.

En pratique, les problèmes de type sac-à-dos ont été étudiés depuis les travaux de pionniers de Dantzig [**16**] à la fin des années cinquante. Ces problèmes possèdent de nombreuses applications dans l'industrie, la planification financière, la gestion de portefeuille, le transport ainsi que d'autres problèmatiques liées au quotidien. Ces problèmes sont aussi d'une grande importance théorique vu qu'ils interviennent comme sous-problèmes dans plusieurs problèmes en nombres entiers.

En effet, le problème de sac-à-dos unidimensionnel intervient, par exemple, comme sous-problème dans le problème de découpe (voir Gilmore & Gomory [**22**], Hifi & Roucairol [**25**], lors de la génération d'un pivot du Simplexe. Il intervient également dans la résolution du problème d'affectation généralisée, ce qui fait de lui un modèle théorique particulièrement intéressant. Il a été largement étudié (Balas & Zemel [**6**], Fayard & Plateau [**20**], Horowitz & Sahni [**27**], Kellerer and al. [**28**], Martello & Toth [**40**]), Elkihel and al. [**19**]. Bien qu'elles semblent trés proches, ces formulations ne font pas appel aux mêmes techniques de résolution.

1.6 Problème de sac-à-dos

Le problème de sac-à-dos *Knapsack (KP)* à variables binaires $(0-1)$ est défini comme suit : On considère un ensemble d'articles numérotés par l'indice i variant de 1 à n. Chaque article i dispose d'un poids w_i et d'un profit r_i. La capacité du sac est notée C. On désire le remplir de façon à maximiser la somme des articles placés, et ce en respectant la contrainte de capacité.

Le problème, noté (KP) appelé aussi ($0-1$ knapsack problem) peut s'écrire de la manière suivante :

$$\begin{aligned} \max & \sum_{i=1}^{n} r_i x_i \\ & \sum_{i=1}^{n} w_i x_i \leq C \\ & x_i \in \{0,1\}, \quad \forall i \in \{1,\ldots,n\} \end{aligned} \tag{1.16}$$

Le vecteur $x := (x_i, 0 \leq i \leq n) \in \{0,1\}^n$ est une solution du problème où $x_i = 1$ si l'article i est mis dans le sac et $x_i = 0$ si cet article n'a pas été mis dans le sac. Le problème consiste donc à choisir un sous-ensemble d'articles parmi la liste d'articles initiale afin de maximiser la fonction objectif suivante :

$$f(x) = \sum_{i=1}^{n} r_i x_i \tag{1.17}$$

Tout vecteur binaire $(x_1, x_2, ..., x_n)$ est appelé solution du problème *KP*. On dit qu'une solution $\overline{x}$ est *réalisable* ou admissible si elle vérifie la contrainte de capacité, c'est-à dire :

$$\sum_{1 \leq i \leq n} w_i \overline{x}_i \leq C \tag{1.18}$$

Par ailleurs, la solution notée x^* est dite *optimale*, si elle est à la fois réalisable et maximise la somme des profits des articles mis dans le sac. En d'autres termes, pour toute solution réalisable $\overline{x}$, on a :

$$\sum_{i=1}^{n} r_i \overline{x}_i \leq \sum_{i=1}^{n} r_i x_i^* \tag{1.19}$$

1.7 Quelques variantes du problème de sac-à-dos

1.7.1 Problème de sac-à-dos quadratique

Dans toutes les variantes du problème de sac-à-dos examinées jusqu'à présent, le bénéfice du choix d'un élément donné était indépendant des autres éléments choisis.

Dans de nombreuses applications de la vie réelle ainsi que dans les problèmes qui ont pour origine la théorie des graphes, il est naturel de supposer que le profit d'un emballage devrait également refléter la façon dont les éléments donnés s'assemblent. Une formulation possible d'un tel problème est le problème de sac-à-dos *Quadratique* (Quadratic knapsack *QKP*) dans lequel, chaque élément a un profit correspondant et un profit supplémentaire est obtenu si l'élément est choisi avec un autre élément. Le problème de sac-à-dos quadratique *QKP* a été introduit pour la première fois par Gallo and al. [**21**] et a été étudié de manière intensive au cours de la dernière décennie.

Le problème de sac-à-dos quadratique est défini comme suit :

Considérons un ensemble d'articles numérotés par l'indice i variant de 1 à n. Chaque article i dispose d'un poids entier positif w_i. En outre, nous disposons d'une matrice $n \times n$ d'entiers non négatifs $r = r_{ij}$, où r_{ii} est le profit obtenu si l'élément i est sélectionné et $r_{ij} + r_{ji}$ est un profit obtenu si les deux éléments i et j sont sélectionnés, pour $i < j$. Le principe du QKP est de sélectionner un sous-ensemble d'articles dont le poids global ne dépasse pas une capacité C donnée du sac-à-dos, de manière à maximiser le profit global.

Soient $N := \{1, ..., n\}$ l'ensemble des articles. En introduisant une variable binaire $x_i = 1$ pour indiquer si l'article i est sélectionné, et $x_i = 0$ si l'article i ne l'est pas, le problème peut être formulé comme suit :

$$\begin{aligned} \max \sum_{i \in N} \sum_{j \in N} r_{ij} x_i x_j + \sum_{i \in N} r_i x_i & \\ \sum_{i \in N} w_i x_i &\leq C \\ x_i &\in \{0,1\}, \quad \forall i \in N \end{aligned} \tag{1.20}$$

Sans perte de généralités nous supposons que :

$$\max_{i \in N} w_i \leq C < \sum_{i \in N} w_i \tag{1.21}$$

La matrice de profits est symétrique, c'est-à-dire $r_{ij} = r_{ji}$ pour tout $i, j \in N$.

Nous remarquons que :

- S'il existe des poids négatifs $w_i < 0$ alors nous inversons la variable bivalente x_i comme suit : $1 - x_i$.
- Si $w_i > C$, alors $x_i = 0$.

1.7.2 Problème de sac-à-dos multi-objectifs

Nous considérons le problème d'optimisation avec m fonctions objectifs. Si $\overline{x}$ est une solution réalisable, alors $f_k(\overline{x})$ désigne la valeur de $\overline{x}$ par rapport à la $k^{éme}$ fonction objectif, avec $1 \leq k \leq m$.

Le but de trouver un ensemble de solutions qui couvre tous les compromis possibles entre les différents objectifs, en recherchant les solutions « *efficaces* ». Une solutions $\overline{x}_1$ domine faiblement une solution réalisable $\overline{x}_2$ si :

$$f_k(\overline{x}_1) \geq f_k(\overline{x}_2), \quad 1 \leq k \leq m \tag{1.22}$$

Une solution réalisable x^* est *efficace ou Pareto optimale* s'il n'y a pas d'autre solution réalisable qui domine x^*. L'ensemble $\mathbb{P}$ des solutions efficaces est appelé *frontière de Pareto (ou courbe de Pareto)*. Un ensemble de solutions réalisables est dit *réduit* s'il ne contient pas deux solutions différentes $\overline{x}_1$ et $\overline{x}_2$ telles que $\overline{x}_1$ domine faiblement $\overline{x}_2$.

Le problème de sac-à-dos multi-objectifs (MOKP) est obtenu à partir du problème de sac-à-dos classique en introduisant m valeurs de profit au lieu d'un profit pour chaque article. Plus précisément, le (MOKP) est défini comme suit :

Soit $N = \{1, ..., n\}$ l'ensemble des articles, C est la capacité du sac-à-dos. Chaque article i ($i = 1, \ldots, n$) a m profits $r_i^k (k = 1, \ldots, m)$ et un poids w_i. La variable bivalente est définie par : $x_i = 1$ si l'article i est mis dans le sac et $x_i = 0$ sinon.

Le problème de sac-à-dos multi-objectifs en variables bivalentes $(0 - 1)$ est formulé comme suit :

$$\begin{aligned} &\max f_k(x), && k = 1, \ldots, m, \\ &\sum_{i=1}^{n} w_i x_i \;\leq\; C && \\ &\qquad x_i \;\in\; \{0,1\}, && i = 1, \ldots, n, \end{aligned} \tag{1.23}$$

où

$$f_k(x) = \sum_{i=1}^{n} r_i^k x_i,\ k \in \{1, \ldots, m\} \tag{1.24}$$

représente la $k^{éme}$ fonction objectif,

- n est le nombre d'articles.
- m est le nombre de fonctions objectifs.
- C est la capacité du sac.

Dans ce problème, nous supposons que : r_i^k, C, w_i sont des entiers positifs et $w_i \leq C$ pour $i \in \{1, \ldots, n\}$ et $k \in \{1, \ldots, m\}$.

1.7.3 Problème de sac-à-dos multidimensionnel

Le problème de sac-à-dos multidimensionnel MKP est une généralisation du problème standard de sac-à-dos KP, d'un seul sac-à-dos à s sacs-à-dos avec des capacités (éventuellement) différentes. L'objectif est d'affecter chaque article à au plus un des sacs-à-dos de telle sorte qu'aucune des contraintes de capacité ne soit violée et que le profit total des articles placés dans les sacs-à-dos soit maximisé.

Le problème de sac-à-dos multidimensionnel MKP a de nombreuses applications pratiques, telles que l'allocation des processeurs, le chargement des cargaisons ou la budgétisation des capitaux.

Nous considérons un problème de sac-à-dos multidimensionnel MKP de la forme suivante :

Étant donné un ensemble de $N = \{1, \ldots, n\}$ d'articles à charger dans S sacs-à-dos de capacité $C_j, j \in \{1, \ldots, s\}$. Chaque article $i \in N$ est caractérisé par son poids w_i, son profit r_i et sa variable de décision x_{ij} qui vaut 1 si l'article i est chargé dans le sac j

et qui vaut 0 dans le cas contraire. Il s'agit de trouver un sous-ensembles disjoints de N (où chaque sous-ensemble correspond au remplissage d'un sac) qui maximisent le profit total formé par la somme des articles sélectionnés.

Le problème MKP est formulé comme suit :

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{s} r_i x_{ij} & \\ \sum_{i=1}^{n} w_i x_{ij} &\leq C_j, \qquad j = 1, \ldots, s \\ x_{ij} &\in \{0,1\} \quad i = 1, \ldots, n, j = 1, \ldots, s, \end{aligned} \tag{1.25}$$

où r_i, C_j et w_i sont des entiers positifs.

CHAPITRE 2

ETAT DE L'ART

"Obvious" is the most dangerous word in mathematics,
[*"Évident" est le mot le plus dangereux en mathématiques*],
Eric T., Bell.

Sommaire

2.1 Introduction

Dans ce chapitre nous donnons un aperçu sur le problème de sac-à-dos ainsi que ses variantes et nous présentons les méthodes de résolution les plus performantes en termes de taille d'instance et de temps d'exécution existantes dans la littérature. Il ressort de cet état de l'art que les méthodes exactes semblent réellement limitées pour résoudre de tel problème. Plusieurs méthodes permettent de déterminer des bornes supérieures efficaces à l'aide de relaxations lagrangiennes par exemple, mais cela ne permet toutefois pas de définir un processus de résolution exacte aussi performant que certaines heuristiques.

2.2 Problème de sac-à-dos stochastique

Le problème de sac-à-dos stochastique avec des poids aléatoires a fait l'objet de plusieurs études dans la littérature, dont il est défini comme suit :

Soit C la capacité du sac et de poids fixe, n un ensemble d'élément. Le poids de chaque élément est inconnu dans la première étape, et qui sera connu avant que la décision de la deuxième étape ne soit prise. Nous considérons les poids comme des variables

aléatoires et supposons que les poids χ_i de l'élément i suit la loi normale de manière indépendante avec une moyenne $\mu_i > 0$ et un écart type σ_i.

En effet, chaque élément i a un profit fixe par unité de poids $r_i > 0$.

Le problème d'optimisation stochastique peut être traité de deux manières différentes :

Soit comme un problème à une étape, ce qui entraine que la décision finale doit être prise avant que les paramètres aléatoires ne soient révélés, soit comme un problème à deux ou à plusieurs étapes qui permettent d'apporter des corrections ultérieures de la décision prise à l'avance (première étape).

Lors de la première étape du problème de sac-à-dos à deux étapes les articles peuvent être placés dans le sac-à-dos. En effet, si le poids total de ces articles dépasse la capacité du sac, certains d'entre eux doivent être retirés lors de la deuxième étape.
Par conséquent, nous introduisons une contrainte de chance afin de réduire le pourcentage d'avoir une surcharge causée par les éléments choisis lors de la première étape.

Au début de la deuxième étape, les poids de tous les articles sont révélés. En cas de surcharge, les articles doivent être retirés et une pénalité d doit être payée pour chaque unité de poids de l'article retiré du sac. Notre objectif est donc de minimiser la pénalité totale.

Dans le problème de sac-à-dos avec recours simple, la contrainte de capacité est introduite dans la fonction objectif en utilisant la fonction de pénalité $[.]^+$ ainsi que le facteur de pénalité $d > 0$. Cela sera interprété comme suit : Dans le cas où le choix des articles conduit à un excès de capacité, une pénalité est imposée par unité de surpoids.

Cohn & Barnhart. (1998) **[15]**, Ont étudié le problème de sac-à-dos stochastique, ils ont focalisé sur la recherche d'une heuristique robuste pour une variante du problème de vehicle routing problem dont ils ont défini leur problème comme suit : une entreprise de livraison de gaz à chauffage pour usage particulier, elle sélectionne un sous-ensemble des clients réguliers afin de leur faire une livraison quotidienne du gaz, puis elle affecte les clients aux conducteurs et conçoit une tournée de livraison pour chaque conducteur. Les demandes des clients ne sont pas connues avec précision, mais sont basées sur des prévisions. Une pénalité doit être payée si le client est non desservi. Puis ils ont déterminé le problème déterministe du sac-à-dos stochastique à poids aléatoires. Ils ont présenté une approche algorithmique repose en grande partie sur l'établissement de meilleures bornes et de relations de dominance, ainsi que sur le choix judicieux de l'ordre dans lequel les différentes branches de l'arbre doivent être parcourues. Dans le pire des cas, notre approche peut prendre un temps exponentiel car le problème déterministe du sac- à-dos est complexe sur le plan informatique.

Andrieu and al.(2007)**[3]**, ont étudié le problème de sac-à-dos stochastique sous contrainte de défaillance. Cette contrainte est exprimée en termes d'une condition qui entraine une défaillance, représentant une panne physique ou technique. Ils ont formulé

le problème en termes de contrainte de probabilité, où le niveau de "fiabilité" est un paramètre de modélisation dont la probabilité de défaillance ne doit pas dépasser ce niveau. L'application de l'algorithme stochastique d'Arrow-Hurwicz pose deux difficultés : l'une est que la contrainte de probabilité est concave, l'autre est l'estimation du gradient de la contrainte de probabilité. Afin d'estimer le gradient de cette dernière, ils ont développé deux méthodes : la première est appelé la méthode de convolution et une autre appelé la méthode de Finite Differences (FD).

Kosuch & Lisser (2009) [**31**], ont étudié deux variantes du problème de sac-à-dos à poids aléatoire.

La première variante est le problème de sac-à-dos à recours simple, ils ont utilisé l'Algorithme du gradient stochastique afin de résoudre cette variante. Une méthode d'approximation par convolution est appliquée pour estimer le gradient d'une fonction non différentiable.

La seconde variante est le problème de sac-à-dos stochastique avec contrainte probabiliste, ils ont proposé deux formulations pour la même variante, dont la première est le problème de sac-à-dos sous contrainte d'espérance, avec l'approximation par convolution, ils ont pu estimer le gradient de la contrainte :

$$1_{\mathbb{R}^+}\left(C - \sum_{i=1}^{n} \chi_i x_i\right) \tag{2.1}$$

Cela permet de résoudre le problème de sac-à-dos sous contrainte d'espérance à l'aide de l'algorithme stochastique d'Arrow-Hurwicz. Puis ils ont comparé les méthodes de résolutions présentées au dessus avec la méthode de branch and bound (B& B algorithm).

Lisser & Lopez (2010) [**37**], ont considéré le problème de sac-à-dos quadratique stochastique à deux étapes avec recours et à poids aléatoires, dans la première étape ils ont introduit une contrainte de probabilité sur la capacité du sac-à-dos. Ils ont proposé de résoudre ce problème à grandes instance d'utiliser des techniques basées sur les relaxations semi-définies.

Kosuch & Lisser (2011) [**32**], ont étudié le problème de sac-à-dos stochastique à deux étapes qui l'ont considéré comme une version particulière du problème de sac-à-dos stochastique avec des poids normalement distribués. Contrairement au problème de sac-à-dos à une étape, dont les éléments peuvent être ajoutés ou retirés du sac-à-dos au moment où les poids réels sont connus (deuxième étape). De plus, une contrainte de chance est introduite dans la première étape afin de réduire le pourcentage de cas où les articles choisis mènent à une surcharge dans la deuxième étape. Ils ont proposé des méthodes dans le but d'évaluer la fonction objectif basée sur le calcul des bornes supérieures et inférieures, ces dernières sont introduites dans l'algorithme branch and bound pour pouvoir trouver l'espace de solution de la première étape.

Kosuch (2014) **[33]**, a étudié le problème de sac-à-dos à deux étapes avec des poids aléatoires χ_i est discrètement distribué avec L résultats possibles $\chi^1, ..., \chi^L$. Il désigne par $p^1, ..., p^L$, où L est le nombre de scenarios dans la deuxième étape, les probabilités de ces L scénarios sont non nulles et suppose que tous les poids sont strictement positifs. Il a étudié des cas particuliers dans la deuxième étape qui sont : les éléments peuvent être soit rajoutés ou retirés et non pas les deux à la fois.

Kosuch and al. (2017) **[34]**, ont défini le problème de sac-à-dos stochastique comme suit :

Etant donné un ensemble de n éléments, il s'agit de choisir un sous-ensemble, sans connaître la valeur exacte de leur poids. Ils ont considéré les poids comme des variables aléatoires et ont supposé que le poids χ_i de l'élément i suit la loi normael de moyenne $\mu_i > 0$ et d'écart type σ_i. Chaque élément a un profit fixe par unité de poids $r_i > 0$. Le choix d'un profit par unité de poids peut être justifié par le fait que la valeur d'un élément dépend souvent de son poids, que l'on ne connaît pas à l'avance. Par conséquent, ils ont étudié le problème de sac-à-dos stochastique avec une contrainte d'espérance. Afin de résoudre ce problème, ils ont proposé de résoudre la version relaxée de ce problème en utilisant un algorithme de gradient stochastique afin de déterminer les bornes supérieures pour être utilisées dans l'algorithme branch-and-bound. Ils ont proposé deux approches pour estimer les gradients, l'une basée sur l'intégration par parties et appelée hereafter IP-method et l'autre sur les différences finies et appelée FD-method. La méthode des différences finies est une approche simple et robuste qui donne des résultats efficaces malgré le fait que les gradients estimés sont biaisés, tandis que l'intégration par parties est basée sur une analyse plus théorique et permet d'élargir le champ des applications.

2.3 Problème de sac-à-dos multiple

Le problème de sac-à-dos quadratique multiple consiste à affecter les n articles ayant des profits individuels et par paire à un ensemble de s sacs-à-dos limités afin de maximiser le profit total.

Martello & Toth. (1981) **[38]**, ont proposé trois types de procédures afin de pouvoir résoudre le problème multiple sac de grande taille dans un temps d'exécution raisonnable, le but de la première procédure est de trouver une solution initiale réalisable en appliq- uant l'algorithme glouton, afin de pouvoir améliorer la solution initiale ils ont proposé une procédure basée sur le réarrangement des articles déjà sélectionnés, et la troisième procédure consiste à remplacer les articles déjà sélectionnés par d'autres non déjà choisis et qui maximise le profit.

Hiley & Julstrom. (2006) **[25]**, ont proposé trois méthodes de résolutions afin de résoudre le problème de sac-à-dos multiple quadratique : la première est l'algorithme glouton dont le principe est de remplir les sacs-à-dos qui sont initialement vide un par

un avec des articles dont la capacité est susceptible d'être importante par rapport à leur poids. Et les deux autres sont respectivement, l'heuristique hill-climber qui combine le codage évolutionnaire avec l'heuristique de mutation qui génère les voisins de la solution actuelle obtenues à base du codage évolutionnaire dont le principe de cette dernière est de coder les solutions possibles sous forme de chaines de caractères, dont sa longueur est égale au nombre d'articles. Où $c\left[i\right] = k$ indique que l'article i est assigné au sac-à-dos s et la seconde est l'algorithme génétique.

Pisinger. (2007) **[42]**, a donné un aperçu sur les bornes supérieures présentées dans la littérature. Les techniques de calcul des limites comprennent la relaxation à partir de plans supérieurs, la linéarisation, la reformulation, la relaxation lagrangienne, la décomposition lagrangienne et la programmation semi-définie. Un bref aperçu des heuristiques, des techniques de réduction, des algorithmes de branch and bound, suivi d'un aperçu des inégalités valides pour le polytope quadratique du sac-à-dos.

Chen & Hao (2014) **[12]**, ont présenté une approche de recherche itérative à seuil réactif (iterated responsive threshold search IRTS) pour résoudre le QMKP, la méthode est basée sur l'utilisation combinée de trois voisinages, l'algorithme alterne entre une phase d'exploration basée sur le seuil où les transitions de solution sont autorisées parmi celles qui respectent le seuil réactif et une phase d'amélioration basée sur la descente où seules les solutions améliorées sont acceptées. Lorsque les deux phases précédentes s'arrêtent, une stratégie de perturbation est appliquée pour assurer la diversification globale de la procédure de recherche en déplaçant la recherche vers une nouvelle région éloignée.

Chen & Hao (2016) **[13]**, ont proposé une généralisation du problème de sac-à-dos quadratique multiple (QMKP) à un seul objectif à bi-objectifs, de sorte qu'ils maximisent simultanément le profit total des articles chargés dans les sacs-à-dos. Pour résoudre ce type de problème, ils ont proposé un algorithme hybride à deux étapes (HTS) pour une approximation du front de pareto du QMKP bi-objectif. L'algorithme HTS combine deux méthodes de recherche différentes et complémentaires : pour la première étape ils ont proposé une recherche scalarizing memetic et pour la deuxième étape ils ont utilisé la recherche locale de Pareto.

Laalaoui & Mhalla (2015) **[35]**, ont proposé une heuristique qui résout le problème multiple sacs à très grandes instances (un million d'éléments). La méthode proposée est basée sur l'heuristique IRT qui est une technique basée pricipalement sur la recherche locale qui tente d'améliorer un sous-ensemble d'éléments au lieu de l'ensemble complet de n éléments. Chaque fois, un intervalle d'articles est sélectionné de manière aléatoire pour choisir l'ensemble d'articles qui peut maximiser le profit total.

2.4 Problèmes d'optimisation multi-objectifs

Considérons le problème de maximisation à m objectifs suivant :

$$\begin{aligned} &\max F(x) = (f_1(x), \ldots, f_m(x))^T \\ &\text{s. c.} x \in \Omega, \end{aligned} \tag{2.2}$$

où

- $f_k(x)$ est la $k^{éme}$ fonction objectif pour maximiser.
- x est le vecteur de décision.
- Ω est l'espace réalisable.

Deb and al. (2002) [**17**], ont proposé un algorithme évolutionnaire EA multi-objectifs (MOEA) basé sur le tri non-dominé, appelé algorithme génétique de tri non-dominé II (NSGA-II) qui est une approche rapide. Ils ont présenté également un opérateur de sélection de meilleures solutions (en termes de fitness et de répartition) combinant les populations de parents et d'enfants. Les résultats de simulation sur des problèmes difficiles montrent que la NSGA-II proposée, dans la plupart des problèmes, est capable de trouver une bien meilleure répartition des solutions et une meilleure convergence vers le front Pareto-optimal par rapport à la stratégie d'évolution dérivée de Pareto et à l'EA de strength-Pareto.

Bhuvana & Aravindan. (2015) [**7**], dans le but de résoudre des problèmes d'optimisation multi-objectifs sans contraintes, ils ont proposé un algorithme appelé algorithme memetique avec recherche locale préférentielle utilisant des poids adaptatifs (MAPLS-AW) incorporé dans l'algorithme NSGA-II existant, ont proposé une méthode de recherche locale préférentielle pour améliorer les solutions optimales globales ainsi qu'une méthode de poids adaptatif dans le but de combiner plusieurs objectifs.

Chu & Yu. (2018) [**14**], l'algorithme génétique de tri non-dominé II (NSGA-II) est efficace pour traiter les problèmes multi-objectifs. Lors de l'évaluation de la fiabilité d'un algori- thme pour des problèmes multi-objectifs, deux types d'indices sont souvent considérés simul- tanément, la convergence vers le front de Pareto ainsi que la caractéristique de distribution des solutions. Crowding distance dans la NSGA-II standard a la propriété que les solutions au sein d'un cube ont la même crowding distance, ce qui ne contribue pas à la converg- ence de l'algorithme. Plus une solution est proche du front de Pareto, plus sa priorité doit être élevée. Les auteurs ont redéfini crowding distance tout en conservant presque tous les avantages de la distance originale. De plus, la vitesse de convergence vers le front de Pareto est plus rapide.

CHAPITRE 3

PROBLÈME DE SAC-À-DOS QUADRATIQUE STOCHASTIQUE MULTI-OBJECTIFS

"Obvious" is the most dangerous word in mathematics,
[*"Évident" est le mot le plus dangereux en mathématiques*],
Eric T., Bell.

Sommaire

3.1 Introduction

Le problème de sac-à-dos est l'une des classes d'optimisation combinatoire largement étudiée et classé comme NP-difficile. Il est couramment utilisé pour modéliser de nom-

breux problèmes du monde réel dans différents domaines. Ce problème consiste à sélectionner un sous-ensemble d'articles parmi un ensemble donné, en veillant à ne pas dépasser la capacité fixe et positive du sac-à-dos et en maximisant le profit total. Chaque article a un poids et un profit positif.

Par exemple, imaginons qu'une entreprise ait un budget fixe pour investir dans plusieurs projets, chacun ayant un coût et générant un profit en cas de réussite. Le défi des décideurs est de sélectionner un sous-ensemble de projets à investir de manière à maximiser le profit total, tout en respectant le budget disponible.

Deux monographies sont entièrement consacrées aux problèmes de sac-à-dos avec ses différentes versions et variantes. La première, de Martello & Toth (1990) [**40**], décrit en détail les algorithmes ainsi que les tests numériques, sur 250 pages avec 200 références utilisées, les principaux résultats dans ce domaine publiés au cours des trois décennies précédentes. La deuxième monographie, de Kellerer and al. (2004)[**27**], s'étend plus de 500 pages, et contient environ 500 références, Les deux tiers de ces références ont été publiés après la première monographie, démontrant ainsi l'intérêt croissant des chercheurs pour ce domaine. En plus de ces beaux ouvrages, des surveys sont régulièrement publiés pour mettre à jour les connaissances dans ce domaine. Cacchiani and al. (2022a,b) [**9**], [**10**], ont récemment proposé une survey sur les problèmes de sacs-à-dos et l'ont structurée en deux parties. La partie I est consacrée aux problèmes dont le but est d'assigner de manière optimale des articles à un seul sac-à-dos et passe aux problèmes avec des contraintes spéciales et des domaines d'investigation relativement récents, comme les problèmes robustes a deux niveaux. La partie II couvre les problèmes de sac-à-dos multiples et quadratiques, et comprend un traitement succinct des problè- mes de sac-à-dos multi-objectifs.

Dans notre étude, nous nous interessons au problème de sac-à-dos *quadratique (Quadratic Knapsack Problem (QKP))*. Ce type de problème est définit comme suit : Étant donné un sac-à-dos de capacité fixe et un ensemble d'articles, chaque article a un poids entier positif, et un profit individuel. Nous avons associé un profit relatif à chaque paire d'articles sélectionnés. Notre objectif est de choisir un sous-ensemble d'articles dont le poids total n'excède pas la capacité maximale du sac-à-dos donné, tout en maximisant le profit total. Une étude bien connue de Pisinger (2007) [**42**] a proposé une survey détaillée concernant ce problème ainsi que des méthodes de résolution.

Gallo and al. (1980) [**20**] ont introduit le problème de sac-à-dos quadratique et ont déterminé les premières bornes en utilisant le concept de " upper plane ". Les auteurs ont présenté les différentes utilisations de ce concept dans un schéma de branch-and-bound pour résoudre un tel problème.

Aïder and al. (2022a) [**1**] ont considéré un algorithme basé sur des stratégies de branchement pour résoudre des problèmes de sac-à-dos quadratiques multiples à grande échelle. Ils ont développé une procédure améliorée de résolution de type "fix and solve"

et l'ont intégrée dans la méthode locale basée sur le branching, où les branches reflètent l'intensification et la diversification de la recherche locale autour d'une solution.

Les mêmes auteurs Aïder and al. (2022b) [**2**] ont utilisé l'algorithme hybride à base de population (HBPA) pour résoudre efficacement le problème de sac-à-dos quadratique multiple bi-objectifs.

Letocart and al. (2014) [**36**] ont proposé pour résoudre le problème de sac-à-dos $(0-1)$ quadratique avec k-objets, une heuristique rapide et efficace qui donne à la fois des bonnes bornes inférieures et supérieures en un temps raisonnable. Plus précisément, ils ont intègré une heuristique primaire et une phase de réduction de la programmation semi-définie dans une heuristique duale de substitution.

Pour résoudre le problème de sac-à-dos quadratique multiple généralisé (GQMKP), Zhou and al.(2022)[**50**] ont proposé un algorithme de recherche évolutionnaire et hybride (HESA) efficace qui utilise un opérateur de croisement pour générer de nouvelles solutions (enfants), dans le but d'améliorer ces dernières, ils proposent d'appliquer un algorithme de recherche tabou pour toutes les solutions trouvées (réalisables et non réalisables). D'autres nouvelles caractéristiques de HESA comprennent une stratégie adaptée pour assurer la diversité la qualité supérieure de la population initiale et quant à la technique streamlining est utilisée afin d'accélérer les évaluations des solutions candidates.

Cependant, dans la vie réelle, les problèmes ne sont pas déterministes, c'est-à-dire que certains paramètres sont connus sans précision au moment de la prise de décision. Ces paramètres peuvent être modélisés par des variables aléatoires distribuées de manière continue ou discrète, ce qui transforme le problème de sac-à-dos fondamental en un problème d'optimisation stochastique.

Dans ce cas, nous nous référençons aux travaux de Kosuch & Lisser (2011) [**32**] qui ont présenté deux modèles de problème de sac-à-dos stochastique avec des poids aléatoires. Le premier modèle, nommé problème de sac-à-dos stochastique avec recours simple, est un problème sans contraintes.
Les auteurs ont proposé différentes méthodes de résolution pour le problème relaxé correspondant comme l'algorithme branch-and-bound, tandis que le second, comme un problème à deux ou plusieurs étapes qui permettent d'apporter des corrections ultérieures de la décision prise à priori. Les auteurs ont proposé une méthode pour calculer les bornes supérieures et inférieures. Ces bornes sont utilisées dans le cadre de l'algorithme branch-and-bound.

Kosuch and al. (2017) [**34**] ont étudié le problème de sac-à-dos stochastique avec des contraintes d'espérance et ont proposé de résoudre le problème relaxée en utilisant un algorithme de gradient stochastique pour établir les bornes supérieures pour le branch-and-bound.

Blado & Toriello (2021) [**8**] ont examiné un problème de sac-à-dos stochastique multiple à deux étapes présentant un ensemble de perturbations (scénarios) possibles avec des

probabi- lités d'occurrences connues. Les auteurs ont proposé deux approches basées sur la technique de branch-and-price pour ce problème.

Pour le même problème, Tonissen and al. (2021) [**47**] ont utilisé un schéma de branch-and-price et ont proposé une comparaison entre deux approches de décompositions différentes dont la première est la décomposition de récupération séparée (SRD) et la seconde est la décomposition de récupération combinée (CRD).

Range and al. (2018) [**43**] ont considéré le problème de sac-à-dos stochastique et ont combiné le problème de sac-à-dos sous contrainte de chance et le problème de sac-à-dos stochastique avec recours simple. Ils ont formulé le modèle résultant comme étant un problème de réseau et ont montré qu'il peut être résolu par une approche de programmation dynamique par étiquetage pour le problème du plus court chemin avec des contraintes de ressources.

Tonissen & Schlicher (2021) [**48**] ont étudié le problème de sac-à-dos stochastique d'impression 3D à deux étapes pour lequel ils ont présenté une formulation de programmation stochastique à deux étapes et l'on reformulé par la suite en un programme linéaire en nombres entiers équivalent.

Dans notre cas, nous étudions le cas où les poids des articles sont aléatoires et sont supposés être indépendants, normalement distribués, avec une moyenne et une varian- ce connues, quant à la capacité et les profits sont déterministes.
Ce cas entraîne la problématique suivante : les articles choisis a priori (c'est-à-dire avant la révélation des poids réels) peuvent ne pas respecter la contrainte de capacité du sac-à-dos. Nous avons référencé aux travaux relatifs à Andrieu and al.(2007) [**3**] et Kosuch & Lisser (2009) [**31**] ont utilisé l'algorithme de gradient pour résoudre le problème stochastique, et ont introduit la méthode de convolution pour obtenir le gradient de la fonction objectif qui n'est par différentiable.

Le problème de sac-à-dos quadratique stochastique (SQKP) est NP-difficile.
Pour résoudre un tel problème, les algorithmes exacts et heuristiques constituent deux méthodes de résolution principales et complémentaires dans la littérature. D'une part, le temps d'exécution de l'algorithme exact pour atteindre l'optimum peut devenir impossible pour les problémes de grandes instances.
D'autre part, les heuristiques et les métaheuristiques donnent des solutions sous-optimales satisfaisantes (pour les problèmes de grandes instances) en un temps d'exécution acceptable.
Pour résoudre le problème de sac-à-dos quadratique stochastique, Lisser & Lopez (2010) [**37**] ont utilisé les techniques de résolution basées sur les relaxations semi-définies.
Song and al. (2018) [**45**] ont étudié le problème de sac-à-dos multiple quadratique stochastique dont l'objectif est de trouver une solution de meilleure qualité pour tous les cas possibles. Les auteurs ont utilisé la technique appelée *recoverable robustness* qui

consiste à trouver d'abord une solution réalisable pour le problème statique associé, puis l'adapter en fonction des scénarios stochastiques existants.

La plupart des problèmes du monde réel tels que l'ordonnancement, l'allocation de ressources, ... doivent optimiser simultanément plusieurs critères. Ces problèmes sont appelés *problèmes d'optimisation multi-objectifs (MOOP)* et peuvent être formulés mathématiquement comme suit :

$$\begin{aligned} &\max F(x) = (f_1(x), \ldots, f_m(x))^T \\ &\text{subject to } x \in \Omega \end{aligned} \tag{3.1}$$

où :

— m est un nombre entier qui représente le nombre de fonctions objectifs.

— Ω est l'espace des solutions réalisables.

Notre travail est inspiré principalement des travaux de Chen & Hao (2016) **[16]** ont étudié le problème de sac-à-dos multiple quadratique bi-objectif BOQMKP et ont proposé un algorithme hybride HTS à deux étapes pour estimer le front de Pareto.

Les méthodes d'optimisation classiques (y compris les méthodes d'aide à la décision multicritères) suggèrent de convertir un problème d'optimisation multi-objectifs en un problème d'optimisation mono-objectif pour le résoudre. Nous nous sommes inspirés des méthodes utilisées pour résoudre le problème multi-objectifs de Capacitated Arc Routing (MO-CARP), dont Tang and al. (2009) **[46]** ont proposé un algorithme memetique avec recherche de voisinage avancée (MAENS), et Mei and al. (2011) **[41]** ont développé un nouvel algorithme memetique (MA) appelé Decomposition-based Memetic Algorithm with Extended Neighborhood Search D-MAENS. Shang and al. (2014) **[44]** ont proposé une autre version de D-MAENS avec deux améliorations consistent d'une part à accélérer la vitesse de convergence, et d'autre part ils ont implémenté une techn- ique utilisant l'archive pour maintenir la meilleure solution actuelle dans sa direction de décomposition pendant la recherche.

Zhang & Li (2007) **[49]** ont proposé un algorithme évolutionnaire multi-objectifs basé sur la décomposition MOEA-D, dont ils ont décomposé le problème d'optimisation multi-objectifs en de nombreux sous-problèmes d'optimisation scalaire et les optimisent simultanément, chaque sous-problème est optimisé en utilisant uniquement les informations de ses plusieurs sous-problèmes voisins.

Bhuvana & Aravindan (2016) **[7]** ont introduit un mécanisme de recherche locale préférentielle pour améliorer davantage les solutions optimales globales, ainsi qu'un mécanisme de poids adaptatif pour combiner plusieurs objectifs.

Kim & Liou (2012) **[28]** ont proposé une nouvelle méthode de partage de fitness pour l'Algorithme Génétique Multi-Objectifs en combinant une nouvelle fonction de partage et des dégradations latérales dans le processus de partage, avec une préférence pour l'une ou l'autre des deux solutions proches.

Arshad and al. (2009) [**4**] ont présenté un algorithme génétique basé sur les séquences (SBGA) pour résoudre le problème symétrique de voyageur de commerce (TSP). L'ensemble des séquences est sélectionné à partir des meilleurs individus, qui sont utilisés pour orienter la recherche du SBGA. De plus, certaines opérations sont appliquées pour maintenir la diversité en divisant les séquences sélectionnées en sous-tours si le meilleur individu de la population ne s'améliore pas.

Certaines de ces idées ont été intégrées dans la méthode NSGA-II afin d'obtenir un nouvel algorithme memetique pour la résolution des problèmes d'optimisation multi-objectifs. Deb and al. (2002)[**17**] ont proposé un algorithme évolutionnaire multi-objectifs MOEA basé sur le tri non-dominé, appelé algorithme génétique de tri non-dominé II NSGA-II, et Chu & Yu (2018) [**14**] ont proposé d'introduire le crowding-distance dans la méthode NSGA-II.

3.2 Formulation Mathématique

Nous considérons un problème de sac-à-dos quadratique stochastique multi-objectifs de la forme suivante : Étant donné un sac-à-dos de capacité fixe $C > 0$ ainsi qu'un ensemble de n, articles, $i = 1, \ldots, n$, chaque article a un poids qui n'est pas connu à l'avance. C'est-à-dire que la décision sur les éléments à choisir doit être prise sans la connaissance exacte de leurs poids. Par conséquent, nous traitons les poids comme des variables aléatoires χ_i, $i = 1, \ldots, n$, qui suivent la loi normale de moyenne $\mu_i > 0$, et d'écart type $\sigma_i, i = 1, \ldots, n$. De plus, chaque élément $i = 1, \ldots, n$ a un profit fixe par unité de poids de type m-vecteur, $r_i = (r_i^1, \ldots, r_i^m)^T$, $r_i^k \in \mathbb{Z}^+, k = 1, \ldots, m$, nous associons à chaque paire d'articles i et j, $1 \leq i \neq j \leq n$, un m-vecteur de profits conjoint par unité de poids $r_{ij} = (r_{ij}^1, \ldots, r_{ij}^m)^T$, pour chaque fonction objectf $k, k = 1, \ldots, m$.

Dans le cas d'un surpoids, les articles doivent être retirés, et une pénalité d doit être payée pour chaque unité de poids retirée. Notre objectif est donc de minimiser la pénalité totale.

La sélection de l'article i est indiquée par une variable de décision binaire x_i qui prend la valeur 1 si l'article i est inclus dans la sélection et 0 sinon.

Le problème de sac-à-dos quadratique stochastique multi-objectifs peut être formulé mathématiquement comme suit :

$$\max_{x \in \{0,1\}^n} \mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j (\chi_i + \chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i \chi_i\Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i \chi_i - c\Big]^+, k = 1, \ldots, m. \quad (3.2)$$

Le but de l'équation (3.2) est de maximiser le profit total de tous les articles assignés.

Nous appelons problème de sac-à-dos binaire bi-objectifs si $k = 2$ et est noté par $0-1$ BOKP.

Problème de sac-à-dos quadratique sous Contrainte

1. Problème de sac-à-dos quadratique sous contrainte d'espérance

$$\begin{cases} \max\limits_{x\in\{0,1\}^n} \mathbf{E}\Big[\sum\limits_{i=1}^{n-1}\sum\limits_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum\limits_{i=1}^{n} r_i^{(k)} x_i\chi_i\Big],\ k=1,\dots,m \\ \text{s.t.} \quad \mathbf{E}\big[\mathbf{1}_{\mathbb{R}^+}\big(c-g(x,\chi)\big)\big] \geq p. \end{cases} \tag{3.3}$$

2. Problème de sac-à-dos quadratique sous contrainte de chance

$$\begin{cases} \max\limits_{x\in\{0,1\}^n} \mathbf{E}\Big[\sum\limits_{i=1}^{n-1}\sum\limits_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum\limits_{i=1}^{n} r_i^{(k)} x_i\chi_i\Big],\ k=1,\dots,m \\ \\ \text{s.t.} \quad \mathbf{P}\big[g(x,\chi)\leq c\big] \geq p. \end{cases} \tag{3.4}$$

Où :

- $\mathbf{P}[A]$: représente la probabilité d'un événement A.
- $\mathbf{E}[.]$: représente l'espérance.
- $\mathbf{1}_{\mathbb{R}^+}$ représente la fonction indicatrice de l'intervalle réel positif.
- $g(x,\chi) = \sum\limits_{i=1}^{n} x_i\chi_i$,
- $d \in \mathbb{R}^+$: représente la pénalité.
- $p \in [0.5, 1]$ est une probabilité dont sa valeur est fixée par le décideur.

Dans la formulation du problème de sac-à-dos quadratique stochastique multi-objectifs avec recours simple, la contrainte de capacité est incluse dans la fonction objectif en utilisant la fonction de pénalité $[.]^+$ et le facteur de pénalité $d > 0$.

Dans le cas d'une surcharge, les articles doivent être retirés et une pénalité d doit être payée pour chaque unité de poids retirée.

Nous pouvons réécrire la fonction (3.2) comme suit :

$$\mathbf{J}^{(k)}(x,\chi) = \mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i\chi_i\Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i\chi_i - c\Big]^+,\ k=1,\dots,m. \tag{3.5}$$

Comme la fonction **J** n'est pas différentiable, nous devons faire une approximation de son gradient en utilisant une méthode proposée par Andrieu and al. (2007) **[3]** appelée *"Méthode d'approximation par convolution "*.
L'idée de base de cette méthode, est de faire une approximation de la fonction indicatrice $(\mathbf{1}_{\mathbb{R}^+})$ par sa convolution avec une fonction approximant le Dirac lorsqu'un paramètre t tend vers 0 :

$$h_t(x) := \frac{1}{t}h(\frac{x}{t}) \tag{3.6}$$

Considérons une fonction $h : \mathbb{R} \longrightarrow \mathbb{R}$ avec les propriétés suivantes :

1. h a un unique maximum lorsque $x = 0$,
2. $\forall x \in \mathbb{R},\ \ h(x) \geq 0$,
3. $\forall x \in \mathbb{R},\ \ h(x) = h(-x)$,
4. $\int_{-\infty}^{+\infty} h(x)dx = 1.$

Pour toute autre fonction $\rho : \mathbb{R} \longrightarrow \mathbb{R}$ lorsque t tend vers 0, nous pouvons déterminer la convolution de ces deux fonctions ρ et h comme suit :

$$(\rho.h_t)(x) = \frac{1}{t}\int_{-\infty}^{+\infty} \rho(y)h(\frac{y-x}{t})dy \tag{3.7}$$

Ce qui s'interprète comme l'espérance de ρ pour une variable aléatoire y et de distribution $h(./t)/t$ centrée autour de x, donc de distribution de plus en plus concentrée autour de x lorsque $t \longrightarrow 0$. L'approximation ρ_t est différentiable et la théorie des distributions donne :

$$(\rho_t)'(x) = \rho * (h_t)'(x) = \rho' * h_t\,(x) = \delta_0 * h_t\,(x) = h_t\,(x)$$

La fonction $\rho_t(x)$ est différentiable avec :

$$(\rho.h_t)'(x) = \frac{1}{t^2}\int_{0}^{+\infty} h'(\frac{x-y}{t})dy = -\frac{1}{t}h(\frac{x}{t}). \tag{3.8}$$

Lorsque $\rho = \mathbf{1}_{\mathbb{R}^+}$, nous avons :

$$(\rho.h_t)\,(x) = \frac{1}{t}\mathbf{E}\Big[\int_{\infty}^{+\infty} \mathbf{1}_{\mathbb{R}^+}(y)h(\frac{x-y}{t})dy\Big] = \frac{1}{t}\mathbf{E}\Big[\int_{0}^{+\infty} h(\frac{y-x}{t})dy\Big] \tag{3.9}$$

$$(\rho.h_t)\,(x) = \mathbf{E}\Big[\rho_t(x)\Big]$$

$$\rho_t(x) = \frac{1}{t}\mathbf{E}\Big[\int_{0}^{+\infty} h(\frac{y-x}{t})dy\Big]$$

En se basant sur la méthode expliquée ci-dessus, nous obtiendrons l'approximation du gradient $\nabla(J_t)$ de la fonction J :

$$\nabla(\mathbf{J}_t^{(k)})(x,\chi) = \Big[(r_1^{(k)}\chi_1,\ldots,r_n^{(k)}\chi_n)^T + \Big(\sum_{\substack{j=1\\j\neq 1}}^{n} r_{1j}^{(k)}(\chi_1+\chi_j)x_j,\ldots,\sum_{\substack{j=1\\j\neq n}}^{n} r_{nj}^{(k)}(\chi_n+\chi_j)x_j\Big)^T\Big]$$
$$-d\Big(-\frac{1}{t}h\Big(\frac{g(x,\chi)-c}{t}\Big).\chi.\big(g(x,\chi)-c\big) + \mathbf{1}_{\mathbb{R}^+}\big(g(x,\chi)-c\big).\chi\Big), k=1,\ldots,m. \tag{3.10}$$

Andrieu and al. (2007) [**3**] et Kosuch & Lisser (2011) [**32**] ont proposé diverses fonctions qui peuvent être choisies pour h, dont Ils ont calculé pour chaque fonction une valeur de référence pour l'erreur quadratique moyenne de la fonction approximative du gradient déjà obtenu et les ont comparées. Par conséquent, en se basant sur les résultats présentés par Andrieu and al. (2007) [**3**], la fonction ci-dessous nous permet d'avoir une valeur minimale de l'erreur quadratique moyenne :

$$h(x) = \frac{3}{4}\left(1-x^2\right)(\mathbf{1}_{\mathbb{R}^+}) \tag{3.11}$$

La fonction indicatrice $(\mathbf{1}_1)$ est définie comme suit :

$$(\mathbf{1}_1) = \begin{cases} 1 & \text{Si} \quad -1 \leq x \leq 1 \\ 0 & \text{sinon} \end{cases}$$

En basant sur les résultats de Kosuch & Lisser (2011) [**32**], nous obtiendrons l'estimation suivante du gradient de J :

$$\nabla(\mathbf{J}_t^{(k)})(x,\chi) = \Big[(r_1^{(k)}\chi_1,\ldots,r_n^{(k)}\chi_n)^T + \Big(\sum_{\substack{j=1\\j\neq 1}}^{n} r_{1j}^{(k)}(\chi_1+\chi_j)x_j,\ldots,\sum_{\substack{j=1\\j\neq n}}^{n} r_{nj}^{(k)}(\chi_n+\chi_j)x_j\Big)^T\Big]$$
$$-d.\Big(-\frac{3}{4t}\Big(1-\Big(\frac{g(x,\chi)-c}{t}\Big)^2\Big)\cdot\mathbf{1}_1\cdot\Big(\frac{g(x,\chi)-c}{t}\Big).\chi.\big(g(x,\chi)-c\big) - \mathbf{1}_{\mathbb{R}^+}\big(g(x,\chi)-c\big).\chi\Big),$$
$$k=1,\ldots,m. \tag{3.12}$$

3.2.1 Problème déterministe équivalent

Nous définissons une nouvelle variable aléatoire comme suit :

$$X := \sum_{i=1}^{n} x_i\chi_i. \tag{3.13}$$

Qui suit la loi normale de moyenne :

$$\widehat{\mu} = \sum_{i=1}^{n} \mu_i x_i \tag{3.14}$$

Et un écart type :

$$\widehat{\sigma} = \sqrt{\sum_{i=1}^{n} \sigma_i^2 x_i^2}. \tag{3.15}$$

La fonction de densité est :

$$\varphi\left(X\right) = \frac{1}{\widehat{\sigma}} f\Big(\frac{X-\widehat{\mu}}{\widehat{\sigma}}\Big). \tag{3.16}$$

La fonction suivante représente la fonction de distribution cumulative :

$$\Phi(X) = F\Big(\frac{X-\widehat{\mu}}{\widehat{\sigma}}\Big). \tag{3.17}$$

Cohn & Barnhart (1998) [**15**] et Kosuch & Lisser (2009) [**31**] ont proposé de réécrire la fonction objectif en une fonction déterministe en utilisant les calculs suivants :

$$\begin{aligned}
\mathbf{E}\Big[\big[X-C\big]^{+}\Big] &= \int_{-\infty}^{\infty} [X-C]^{+}\varphi(X)dX = \int_{C}^{\infty}(X-C)\varphi(X)dX \\
&= \int_{C}^{\infty} X\varphi(X)dX - C\int_{C}^{\infty}\varphi(X)dX \\
&= \widehat{\mu}\int_{C}^{\infty}\varphi(X)dX - \widehat{\sigma}^2\int_{C}^{\infty}\varphi'(X)dX - C\int_{C}^{\infty}\varphi(X)dX \\
&= \widehat{\sigma}^2\varphi[(X)]_C^{\infty} + (\widehat{\mu}-C)[\Phi(x)]_C^{\infty} = \widehat{\sigma}^2\varphi(C) + (\widehat{\mu}-C)[1-\Phi(C)] \\
&= \widehat{\sigma} f\Big(\tfrac{C-\widehat{\mu}}{\widehat{\sigma}}\Big) + (\widehat{\mu}-C)\Big[1-F\Big(\tfrac{C-\widehat{\mu}}{\widehat{\sigma}}\Big)\Big]
\end{aligned} \tag{3.18}$$

En utilisant les calculs présentés ci-dessus, nous écrivons la fonction objectif déterministe équivalente comme suit :

$$J_{det}^{(k)}(x) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\mu_i+\mu_j) + \sum_{i=1}^{n} r_i^{(k)}\mu_i x_i - d\Big[\widehat{\sigma}F\Big(\frac{C-\widehat{\mu}}{\widehat{\sigma}}\Big) - \Big(C-\widehat{\mu}\Big)\Big[1-F\Big(\frac{C-\widehat{\mu}}{\widehat{\sigma}}\Big)\Big]\Big], \quad k = 1,\ldots,m. \tag{3.19}$$

3.3 Méthode de résolution

Dans le processus de décomposition, le MO-SQKP original est d'abord décomposé en plusieurs sous-problèmes de sac-à-dos quadratiques stochastiques à objectif unique (SO-SQKP). Etant donné le vecteur objectif :

$$F(x) = \big(f^1(x), f^2(x), \ldots, f^m(x)\big)^T . x \in \{0,1\}^n$$

La contrainte de capacité est incluse dans la fonction objectif en utilisant la fonction de pénalité $[.]^+$ et le facteur de pénalité $d > 0$. Cela peut être interprété comme suit :
Dans le cas d'une surcharge, les articles doivent être retirés et une pénalité d doit être payée pour chaque unité de poids retirée Comme déjà présenté dans (3.5).

3.3.1 Population Initiale

La population initiale est construite selon l'heuristique constructive gloutonne.
Initialement, le sac-à-dos est vide, les articles sont mis dans le sac-à-dos un par un, un article i est sélectionné et choisi pour être mis dans le sac si et seulement si sa valeur de densité $v_d\,(i, S)$ est la plus élevée par rapport aux autres valeurs de densité relatives aux articles restants.

La valeur de densité d'un article i est calculée à partir de la formule suivante Hiley & Julstrom (2006)**[25]** :

$$v_d(i, S) = \frac{r_i + \sum_{j \in S} r_{ij}}{w_i}. \tag{3.20}$$

Avec S est l'ensemble des articles non assignés.

Algorithme 1 Heuristique Constructive Gloutonne

Input :
r_q : Profit quadratique.
r : Profit linéaire.
w : Le poids de chaque article.
C : La capacité maximale du sac-à-dos.
Output : P : a greedy solution
S $\longleftarrow$ tous les objects
sac-à-dos $\longleftarrow \emptyset$
Tant que $S \neq \emptyset$ **faire**
    b $\longleftarrow$ L'article qui a $v_d\,(b, S)$ maximum
    Si $w_b \leq C$ **alors**
        Ajouter b au sac-à-dos ;
        $C = C - w_b$;
    fin Si
    retirer b de S ;
fin Tant que

3.3.2 Génération de sous-populations

A chaque itération, pour créer un nouvel individu dans la population, nous utilisons un vecteur de profit appelé V qui représente le ratio de la fonction objectif que nous souhaitons favoriser. Si nous voulons privilégier la première fonction objectif, le vecteur sera défini comme suit : $V = (1, 0, 0, \ldots, 0)$. Chaque valeur de V est ensuite multipliée par le profit, c'est-à-dire que $new\_profits^m = V(m) * \text{profit}^m$, où $k = 1, .., m$ est le nombre de fonctions objectif. Pour créer une sous-population, nous favorisons la $k^{\text{ème}}$ fonction objectif en donnant la valeur 1 à l'élément m du vecteur V et 0 aux autres éléments. Ensuite, nous utilisons ces nouveaux profits dans notre heuristique constructive glout- onne pour obtenir un nouvel individu. Pour les autres individus de la population, nous évaluons d'abord les individus précédents déjà dans notre population, puis nous cher- chons la valeur maximale max^* parmi toutes les valeurs évaluées. Nous définissons ensuite un nouveau vecteur *vecteur_max* qui est basé sur les évaluations de la fonction objectif. vecteur_max = ($\frac{max(f^1)}{max^*}$, $\frac{max(f^2)}{max^*}$, $\ldots$, $\frac{max(f^m)}{max^*}$)
Où f_k^* est un vecteur d'évaluation de la $k^{\text{ème}}$ fonction objectif. En utilisant le vecteur *vecteur_max*, nous créons un nouveau vecteur de profit $V = \frac{vecteur\_max}{\sum vecteur\_max}$ pour générer de nouveaux profits. Nous utilisons ensuite ce vecteur V pour générer un nouvel individu en répétant le processus déjà expliqué ci-dessus dans notre heuristique constructive gloutonne. Nous répétons ce processus jusqu'à ce que nous ayons une population de taille N.

3.3.3 Algorithme NSGA-II

L'algorithme génétique de tri non-dominé II *Elitist Non-Dominated Sorting Genetic Algorithm NSGA-II* est l'un des algorithmes d'optimisation multi-objectifs les plus connus, il est plus efficace que sa version précédente NSGA et tend à se converger rapidement. Son principal avantage est la stratégie de préserver la diversité des solutions. La NSGA-II peut être détaillée comme suit : :

Étape 1 **: Population Initiale :** En vue de résoudre notre problème, nous devons d'abord générer une population initiale, Pour ce faire, nous appliquons une heuristique constructive gloutonne qui nous permet d'obtenir une population initiale de taille N dont toutes les solutions sont réalisables.

Étape 2 **: Tri Non-Dominé :** La population initiale obtenue est triée selon le niveau de son non-domination et est obtenu en appliqant L'algorithme de tri non-dominée décrit dans la section 3.3.4.

Étape 3 **: Crowding Distance :** nous attribuons à chaque individu de la population initialement trouvée une valeur de crowding-distance en appliquant l'algorithme crowding-distance décrit dans la section 3.3.5.

Algorithme 2 Génération de sous-populations

Input :
N : La taille de la Population.
r_q : Le profit quadratique.
r : Le profit linéaire.
w : Poids des articles.
C : La capacité du sac-à-dos.
Output : P : La population de taille N
$P \longleftarrow \emptyset$
Pour i=1 au *Nombre de fonctions objectifs* **faire**
    V = Zeros(1,*Nombre de fonctions objectifs*)
    V(i)=1
    Pour j=1 au *Nombre de fonctions objectifs* **faire**
        nouveaux_profits$_q^j$=$V(j) * r_q^j$
        nouveaux_profits^j=$V(j) * r^j$
    fin Pour
    X $\longleftarrow$ Heuristique constructive gloutonne(nouveaux_profits$_q$,nouveaux_profits,w,C)
    $P \longleftarrow P \cup X$
fin Pour
Tant que $|P| \leq N$ **faire**
    max*=la valeur maximale parmi toutes les valeurs évaluées de P
    Pour j=1 to *Nombre de fonctions objectifs* **faire**
        vecteur_max(j) = valeur maximale de f^j across all individuals in P
    fin Pour
    V=$\frac{vecteur\_max}{\sum vecteur\_max}$
    Pour j=1 au *Nombre de fonctions objectifs* **faire**
        nouveaux_profits$_q^j$=$V(j) * r_q^j$
        nouveaux_profits^j=$V(j) * r^j$
    fin Pour
    X $\longleftarrow$ Heuristique constructive gloutonne(nouveaux_profits$_q$,nouveaux_profits,w,C)
    $P \longleftarrow P \cup X$
fin Tant que

Étape 5 **: Opérateurs génétiques :** Le codage GA utilise un croisement binaire simulé et une mutation polynomiale.

Étape 6 **: Recombinaison & sélection :** La population de la progéniture et celle de la génération actuelle sont combinées, et les individus de la génération suivante sont choisis par le processus de sélection. Les individus de la nouvelle génération de taille N sont ceux appartenant aux meilleurs fronts non-dominés.

Algorithme 3 Algorithme De Tri Non-Dominé

Input : N : La taille de la Population ;
        P : La population initiale
Output : F_i : Les fronts non-dominés ;
Pour chaque $p \in P$ **faire**
    $S_p = \emptyset$
    $\eta_p = 0$
    $\eta_q = 0$
    Pour chaque $q \in P$ **faire**
        Si $p \succ q$ **alors** ▷ Si p domine q
            $S_p = S_p \cup \{q\}$ ▷ Ajouter q à l'ensemble des solutions dominé par p
            $\eta_q = \eta_q + 1$
        Sinon Si $q \succ p$ **alors**
            $\eta_p = \eta_p + 1$ ▷ Incrémenter le compteur de domination de p
        fin Si
    fin Pour
    Si $\eta_p = 0$ **alors** ▷ p appartenant au premier front non-dominé
        $p_{\text{rank}} = 1$
        $F_1 = F_1 \cup \{p\}$
    fin Si
fin Pour
$i = 1$
Tant que $F_i \neq \emptyset$ **faire**
    $Q = \emptyset$ ▷ Utiliser pour construire le prochain front non-dominé
    Pour each $p \in F_i$ **faire**
        Pour each $q \in S_p$ **faire**
            $\eta_q = \eta_q - 1$
            Si $\eta_q = 0$ **alors** ▷ q appartenant au prochain front non-dominé.
                $q_{\text{rank}} = i + 1$
                $Q = Q \cup \{q\}$
            fin Si
        fin Pour
    fin Pour
    $i = i + 1$
    $F_i = Q$
fin Tant que

3.3.4 Algorithme de tri non-dominé

Dans cette section, nous détaillons les deux étapes de l'algorithme de tri non-dominé dont le principal objectif est de trier tous les individus de la population en fonction de leurs appartenances aux différents niveaux de non-domination.

La première étape de l'algorithme de tri non-dominé a pour but de déterminer tous les individus du premier niveau de non-domination de la population de taille N. Nous

vérifions pour chaque individu q de la population de taille N s'il est dominé par p. Si q est dominé par p alors nous rajoutons q à l'ensemble Sp et nous incrémentons le compteur de η_q de 1. Sinon nous incrémentons le compteur de η_p de 1 qui veut dire : p est dominé par q, et nous rajoutons p à l'ensemble Sq.
Par conséquent pour chaque individu q de la population est désigné par :

- η_q : Un indicateur qui compte le nombre des individus qui dominent q.
- S_q : Représente l'ensemble des individus dominés par q.

Nous répétons le processus (lignes $3-17$ de l'algorithme 2) jusqu'à ce que l'ensemble des individus du premier front non-dominé de la population soit déterminé.

Après la première étape de la procédure de tri non-dominé, le premier niveau non-dominé est déterminé et les deux entités (η_q,S_q)sont attribuées à chaque individu. Avec le numéro de dominance de chaque individu p appartenant au premier front non-dominé est égal à zéro $\eta_p = 0$.

La deuxième étape de notre algorithme consiste à déterminer les autres fronts non-dominés. Initialement, nous réduisons le nombre de dominance η_p de un pour chaque membre q appartenant à l'ensemble S_p visité par p avec $\eta_p = 0$.

Si le nombre de dominance de q devient nul $\eta_q = 0$, alors q sera rajouté à la liste séparée Q, par conséquent ces membres appartiennent au deuxième niveau non-dominé. Nous refaisons le processus ci-dessus afin d'identifier tous les autres fronts non-dominés et que chaque individu est affecté une seule fois à un seul front non-dominé.

3.3.5 Algorithme crowding-distance

Crowding distance est l'estimation de la densité au voisinage d'un individu i particulier de la population. Nous calculons la distance moyenne entre une solution donnée i et ses deux points les plus proches dans l'espace des résultats (l'espace des objectifs). Cette quantité représente une estimation de la taille du plus grand rectangle enfermant le point i sans y inclure d'autres points de la population (Figure. 3.1)

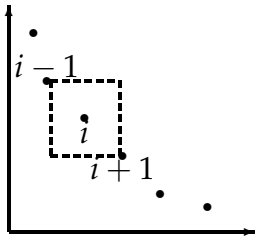

FIGURE 3.1 – Calcul de crowding distance

La Figure. 3.1 est une représentation à deux dimensions associée à la $i^{éme}$ solution.

Le calcul de crowding distance pour chaque membre de la solution nécessite de trier la population en fonction de chaque valeur de la fonction objectif dans un ordre décroissant. Ensuite, pour chaque fonction objectif, nous attribuons une valeur de crowding distance infinie aux solutions limites ayant des valeurs de fonction objectif les plus faibles et les plus élevées. Et pour les solutions intermédiaires, la valeur de crowding distance est égale à la différence absolue normalisée des valeurs des fonctions de deux solutions adjacentes.

Ce processus est effectué pour chaque fonction objectif. La valeur globale de crowding distance de chaque individu est déterminée en additionnant les valeurs individuelles de crowding distance correspondantes à chaque fonction objectif.

L'algorithme 3 ci-dessous décrit la procédure de calcul de la valeur de crowding distance de chaque individu de la population de taille N.

Algorithme 4 Algorithme crowding-distance

Input. m : Le nombre de fonction objectif ;
        P : Population initiale
Output. La valeur de crowding distance de chaque individu $P[i]_{dist}$;
$\mathcal{L} = |P|$ ▷ Le nombre de solutions dans P
Pour chaque $i \in P$ **faire**
    $P[i]_{dist} = 0$ ▷ Initialiser la distance
fin Pour
Pour pour chaque fonction objectif k **faire**
    $P = Sort(P, k)$ ▷ Trier en utilisant chaque valeur de la fonction objectif
    $P[1]_{dist} = P[\mathcal{L}]_{dist} = \infty$ ▷ Les individus ayant des valeurs de fonction objectif faibles et élevées sont toujours sélectionnés
    Pour $i = 2$ **jusqu'à** $\mathcal{L} - 1$ **faire** ▷ Pour tous les autres individus
        $$P[i]_{dist} = P[i]_{dist} + \frac{P[i+1].k - P[i-1].k}{f^k_{\max} - f^k_{\min}}$$
    fin Pour
fin Pour

Avec :

- $P\ [i].k$ représente la valeur de l'individu i relative à la $k^{éme}$ fonction objectif dans l'ensemble P.
- $f^k_{\max}$ et $f^k_{\min}$ sont respectivement les valeurs maximales et minimales de la $k^{éme}$ fonction objectif.

L'opérateur crowded-comparison L'opérateur crowded comparison guide le processus de sélection dans l'algorithme vers une répartition plus ou moins uniforme des solutions sur le front Pareto optimal. Le crowding distance n'est introduit que lorsqu'il est nécessaire de sélectionner des individus de même rang de non-domination, c'est-à-dire que le crowding distance est un critère de sélection des individus de même rang.

Pour cette raison, chaque individu $p \in P$ de la population possède deux attributs :

— Classement de non-domination (p_{rank})

— Crowding distance $(p_{distance})$.

— Si x et y sont deux individus de rangs différents $(x_{rank}) \neq (y_{rank})$, alors la solution de rang inférieur est sélectionnée (c'est-à-dire appartenant au meilleur front).

— Si x et y sont deux solutions du même rang, alors la solution qui possède le voisinage le moins dense est sélectionnée, soit l'individu possédant la plus grande valeur de distance de crowding.

Cet opérateur de sélection intensifie donc la recherche des solutions Pareto-optimales mais préserve aussi la diversité parmi des solutions équivalente.

3.4 Algorithme de recherche locale de pareto par voisinage de sélection (SNPLS)

Algorithme 5 Algorithme de recherche locale de pareto par voisinage de sélection (SNPLS)

Input : m, Le nombre de fonction objectif ;
        Q : L'ensemble des enfants obtenu.
        I_{old} : Solution actuelle.
Output : Procédure de calcul d'une solution améliorée I_{new} à partir d'une solution actuelle I_{old}.
Pour $k = 1$ **jusqu'à** m **faire**
    Calculer ω_k relative à I_{old} ;
    Calculer le poids normalisé λ^k relatif à I_{old} ;
fin Pour
Construire la fonction objectif unicritère $\mathbf{J}(x, \lambda)$;
$I_{new} \leftarrow$ Algorithme Gradient $\mathbf{J}(x, \lambda)$ of I_{old} ;
Calculer la valeur de la fonction objectif relative à (I_{new});
Si I_{new} est meilleure que I_{old} **alors**
    I_{new} est retenue ;
Sinon I_{old} est retenue ;
fin Si

Dans le processus de décomposition, le MO-SQKP original est d'abord décomposé en plusieurs sous-problèmes de sac-à-dos quadratiques stochastiques à objectif unique (SO-SQKP). Etant donné le vecteur objectif $F(x) = \left(f^1(x), f^2(x), \ldots, f^m(x)\right)^T$ et le vecteur de pondération $\lambda = (\lambda^1, \ldots, \lambda^m)^T$, où la somme de ces derniers doit être égale à 1.

La fonction objectif d'un sous-problème est définie comme suit :

$$\begin{cases} \mathbf{J}(x,\chi) = \sum_{k=1}^{m} \lambda^{(k)} f^{(k)}(x) & (3.21.1) \\ \sum_{k=1}^{m} \lambda^{(k)} = 1 & (3.21.2) \\ \lambda^{(k)} \geq 0, k = 1,\ldots,m & (3.21.3) \\ x \in \{0,1\}^n & (3.21.4) \end{cases} \tag{3.21}$$

Nous pouvons réécrire la fonction (3.5) comme suit :

$$\mathbf{J}^{(k)}(x,\chi) = \sum_{k=1}^{m} \lambda^{(k)}.\Big[\mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i\chi_i\Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i\chi_i - c\Big]^{+}\Big], \tag{3.22}$$

Les vecteurs de pondération sont calculés en fonction de la valeur de la norme euclidienne et les paramètres ω_i comme suit :

$$\lambda^k = \frac{\omega^k}{\sum_{k=1}^{m} \omega^k},$$

où :

- $\omega^k(x)$ est déterminé comme suit :

$$\omega^k(x) = \frac{f^k(x)}{\|f(x)\|},$$

- m est le nombre de fonctions objectifs.
- $f^k(x)$ est la valeur de la $k^{éme}$ fonction objectif d'une solution x.
- $\|f(x)\|$ est la norme euclidienne, donnée par :

$$\|f(x)\| = \sqrt{(f^1(x))^2 + \ldots + (f^m(x))^2} \tag{3.23}$$

Les fonctions objectifs sont combinées en une seule fonction objectif F, lorsque toutes les pondérations individuelles sont déterminées pour tous les objectifs :

$$F = \lambda^1 f^1 + \lambda^2 f^2 + \ldots + \lambda^m f^m$$

Nous appliquons ensuite l'algorithme du gradient local pour calculer la nouvelle solution locale I_{new} à partir de la solution actuelle I_{old}.
Ensuite, nous calculons la valeur de fitness relative à la nouvelle solution trouvée afin de pouvoir comparer entre la nouvelle et l'ancienne solution car la valeur de la fitness est notre paramètre de comparaison et nous choisissons la solution de meilleure fitness.

Nous répétons le processus ci-dessus pour chaque progéniture.

Algorithme 6 Algorithme de gradient

Choose $x^0 \in \mathbf{X}_{ad} = [0.5, 1]^n$
A l'étape v, $X^v = (X_1^v, \ldots, X_n^v)$ de χ, selon sa distribution normale
Mettre à jour x^v comme suit : $x^{v+1} = x^v + \epsilon^v r^v$ ▷ Où $r^v = \nabla_x J(x^v, \chi^v)$ et $(\epsilon^v)_{v \in \mathbb{N}}$ est le pas.
Pour $i = 1$ **to** n **faire**
    Si $x_i^{v+1} > 1$ **alors**
        L'ensemble $x_i^{v+1} = 1$
        Si $x_i^{v+1} < 0$ **alors**
            L'ensemble $x_i^{v+1} = 0$
        fin Si
    fin Si
fin Pour

Après avoir construit ces deux Algorithmes (algorithme de tri non-dominé, algorithme crowding-distance) incluant l'opérateur de comparaison et après avoir déterminé les différents paramètres (p_{rank}) et $(p_{distance})$, nous sommes maintenant prêts à décrire l'algorithme Memetique.

3.5 Algorithme memetique avec recherche locale de pareto par voisinage de sélection (MASNPL)

Dans cette section, nous détaillons notre algorithme memetique avec recherche locale de pareto par voisinage de sélection MASNPL pour MO-SQKP, y compris l'initialisation, le croisement et la recherche locale.
Initialement, nous construisons notre population initiale de taille N réalisable à l'aide de l'heuristique constructive gloutonne.

Ensuite, nous attribuons une valeur de fitness pour chaque individu p de la population initiale de taille N, puis nous appliquons l'algorithme de tri non-dominé pour déterminer le rang (niveau) de non-domination p_{rank} , ainsi que l'algorithme Crowding-Distance et pour déterminer une valeur de Crowding-Distance $p_{distance}$ pour chaque individu p de la population.

Ces deux paramètres, p_{rank} et $p_{distance}$ sont utilisés pour sélectionner les individus posséd- ant un voisinage le moins dense et afin de maintenir la diversité des solutions sur le front de Pareto. Ensuite, la sélection par tournoi binaire est appliquée pour sélectionner les parents, pour pouvoir appliquer les opérateurs de croisement et de mutation afin de générer de nouvelles solutions (enfants), c'est-à-dire, la population de progéniture de taille N.

Algorithme 7 Algorithme memetique avec recherche locale de pareto par voisinage de sélection (MASNPL)

Input : P : La population actuelle de taille N ;
    N : La taille de la population ;
    Q : L'ensemble des enfants générés ;
    $NG_{\max}$: nombre de génération maximale ;
Trouver une population initiale de taille N en appliquant l'heuristique constructive gloutonne ;
Calculer la valeur de chaque fonction objectif pour chaque individu dans la population de taille N ;
Appliquer l'algorithme de tri non-dominé à (P, N) ;
Appliquer l'algorithme Crowding-Distance à (P, N) ;
$NG = 1$;
Tant que $NG \leq NG_{\max}$ **faire**
    $Q = \emptyset$; $R_t = \emptyset$;
    Pour $k = 1$ **jusqu'à** $\lceil \frac{N}{2} \rceil$ **faire**
        Sélectionner les parents en appliquant la sélection par tournoi binaire ;
        Appliquer l'opération de croisement à deux parents sélectionnés pour générer deux enfants (Q_1, Q_2) ;
        Appliquer l'opération de mutation sur les deux enfants (Q_1, Q_2) avec une probabilité égale à $P_m = \frac{1}{n}$;
        Pour $j = 1$ **jusqu'à** 2 **faire**
            $Q_j \longleftarrow$ Appliquer Algorithme de recherche locale de pareto par voisinage de sélection à (Q_j) ;
            $Q = Q \cup \{Q_j\}$;
        fin Pour
    fin Pour
    $R_t = P \cup Q$;
    $F =$ Appliquer l'algorithme de tri non-dominé à (R_t) ; ▷ $F = (F_1, F_2, ...)$, tous les fonts non-dominés de (R_t).
    $P_{t+1} = \emptyset$ et $i = 1$;
    Tant que $(|P_{t+1}| + |(F_i)|) \leq N$ **faire**
        Appliquer l'algorithme Crowding-Distance à (F_i) ; ▷ Calculer la valeur de crowding distance pour chaque individu de F_i
        $P_{t+1} = P_{t+1} \cup F_i$;
        $i = i + 1$ ▷ Vérification du prochain front non-dominé pour l'inclure.
    fin Tant que
    Sort F_i ; ▷ Trier par ordre décroissant en utilisant l'opérateur crowded-comparaison.
    $P_{t+1} = P_{t+1} \cup F_i[1, N - |P_{t+1}|]$; ▷ Choisir les premiers éléments de $(N - |P_{t+1}|)$ de l'ensemble F_i
    $Q_{t+1} =$ Générer une nouvelle population (P_{t+1}) ; ▷ Appliquer les opérateurs de croisement, mutation pour générer une nouvelle population (Q_{t+1})
    $NG = NG + 1$;
fin Tant que

Crossover operator (croisement) Lors de cette opération, deux parents (chromosomes) sont sélectionnés par le tournoi binaire $P^1 = \{x_1^1, x_2^1, x_3^1, ..., x_n^1\}$ et $P^2 = \{x_1^2, x_2^2, x_3^2, ..., x_n^2\}$ dont ils s'échangent des parties de leurs chaînes, pour donner de nouveaux chromosomes (enfants).

A partir d'un point pivot choisi aléatoirement, les deux chromosomes (parents) s'échangent les informations génétique trouvant sur la sous-chaine de gauche (ou de droite) afin de générer deux enfants c'est-à-dire copier les informations génétique de la première sous chaine de P^1 et de la deuxième sous-chaine de P^2 et les insérées dans l'enfant 1, et de même copier les informations génétique de la deuxième sous-chaine de P^1 et de la première sous-chaine de P^2 et les insérées dans l'enfant 2.

L'étape suivante consiste à appliquer une seule itération de l'algorithme SNPLS sur la population générée (enfants).

Nous obtiendrons une population combinée $(R_t = P_t \cup Q_t)$ de taille $2N$ où :

- P_t : Est la population parent de taille N.
- Q_t : Est la population progéniture (enfant) de taille N.
- R_t : Est la population combinée (les parents avec les enfants générés) de taille $2N$.

Notre but est d'obtenir une solution de taille N à partir de la population combinée, Pour cela, nous appliquons à la population combinée l'algorithme de tri non-dominé ainsi que l'algorihme de crowding-distance, pour obtenir une population triée selon le niveau de non-domination et pouvoir sélectionner les meilleures solutions.
Les solutions appartenant au niveau de non-domination $\mathcal{F}_1$ sont parmi les meilleures solutions de la population combinée et doivent être privilégiées plus que toutes autres solutions de la population combinée.

Si la taille de $\mathcal{F}_1$ est égale à N, alors tous les membres de l'ensemble $\mathcal{F}_1$ seront sélectionnés, par conséquent rajoutés à la nouvelle population P_{t+1} construite de taille N.

Sinon, si la taille de $\mathcal{F}_1$ est inférieure à N, tous les membres de l'ensemble $\mathcal{F}_1$ seront sélectionnés et rajoutés à la nouvelle population P_{t+1}, les membres restants de la population P_{t+1} pour atteindre la taille N sont choisis à partir du front non-dominé suivant $(\mathcal{F}_2)$ en respectant l'ordre de classement des fronts, si nous atteignons la taille N, nous nous arrêtons et nous obtiendrons notre population P_{t+1} de taille N, sinon nous passerons au niveau non-dominé suivant $(\mathcal{F}_3)$ et nous refaisons le même processus jusqu'à ce que nous obtiendrons une population P_{t+1} de taille N. Cette population P_{t+1} est ensuite utilisée pour la sélection, croisement et mutation pour créer une nouvelle population Q_{t+1} de taille également égale à N. Le processus expliqué au-dessus est répété $(NG_{\max})$.

3.6 Résultats numériques

Dans le but d'évaluer les performances de notre méthode de résolution et comme le problème de sac-à-dos multi-objectifs stochastique quadratique n'a pas été traité auparavant par conséquent l'absence de benchmark, nous présentons ci-dessous les résultats relatifs à la comparaison d'un côté entre notre méthode de résolution et une méthode exacte et d'un autre côté entre nos algorithmes et NSGA-II.

L'implémentation des méthodes de résolutions ainsi que les exemples ont été réalisés sur un ordinateur de caractéristiques : processeur Intel core $i5 - 4200U$ de 1.60 Ghz et 4 Go de RAM.

3.6.1 Comparaison entre l'algorithme MASNPL et la méthode exacte

Pour résoudre le problème de sac-à-dos multi-objectifs stochastique quadratique avec recours simple par une méthode exacte et la comparée avec notre méthode de résolution.

Nous implémentons nos algorithmes sur MATLAB, quant à la méthode exacte adoptée pour résoudre notre problème est prédéfini sur le logiciel MATLAB, et considéré comme un algorithme d'optimisation globale, et utilise la fonction objectif déterministe déjà déterminée dans la partie 3.2.1, ces derniers sont exécutés sur des instances créées aléatoirement selon les paramètres présentés dans le paragraphe ci-dessous. Les résultats sont présentés sur les figures qui suivent.

Exemple 3.1.1

- Nombre de fonctions objectifs : 2.
- Poids : Suit la loi normale avec une moyenne de 225 et une variance de 25.
- Profits par unité de poids : générés uniformément entre 1 et 10.
- Nombre d'articles est : 50.
- Capacité du sac-à-dos est : 9000.
- Pénalité d égale à 330.
- Nombre de générations pour l'algorithme MASNPL est : 50.

Le résultat de notre comparaison est présenté graphiquement sur la figure 3.2.
La figure est une représentation graphique des solutions non-dominées obtenues par la méthode exacte ainsi que notre méthode de résolution (algorithme MASNPL).

Le temps total d'exécution de la méthode exacte pour obtenir la solution finale est : 52179,75 secondes (environ 15 heures et 30 mn), quant au temps total d'exécution de l'algorithme MASNPL est : 41,62 secondes (moins d'une minute).

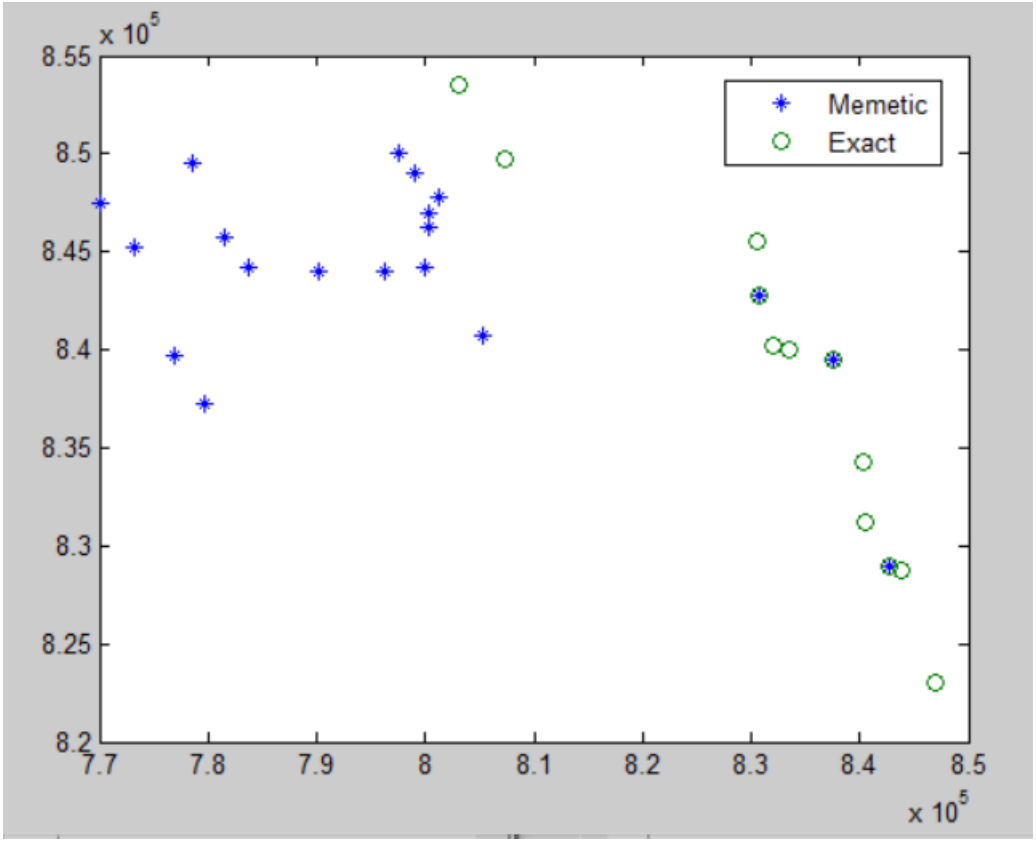

FIGURE 3.2 – Représentation graphique du front de Pareto obtenu par la méthode exacte et l'algorithme *MASNPL* pour 50 d'articles.

En comparant le temps d'exécution pour chaque algorithme appliqué aux mêmes instances créées aléatoirement selon les mêmes paramètres, nous concluons que notre méthode de résolution est plus rapide que la méthode exacte et nous donne des solutions de bonne qualité.

Exemple 3.1.2

- Nombre de fonctions objectifs : 2.
- Poids : Suit la loi normale avec une moyenne de 225 et une variance de 25.
- Profits par unité de poids : générés uniformément entre 1 et 10.
- Nombre d'articles est : 20.
- Capacité du sac-à-dos est : 3600.
- Pénalité d égale à 132.
- Nombre de générations pour l'algorithme MASNPL est : 50.

Le résultat de notre comparaison est présenté graphiquement sur la figure 3.3.
Le temps total d'exécution de la méthode exacte pour obtenir la solution finale est : 2983, 0248789 secondes (environ 49 mn), quant au temps total d'exécution de l'algorithme MASNPL est : 43, 69 secondes (moins d'une minute).

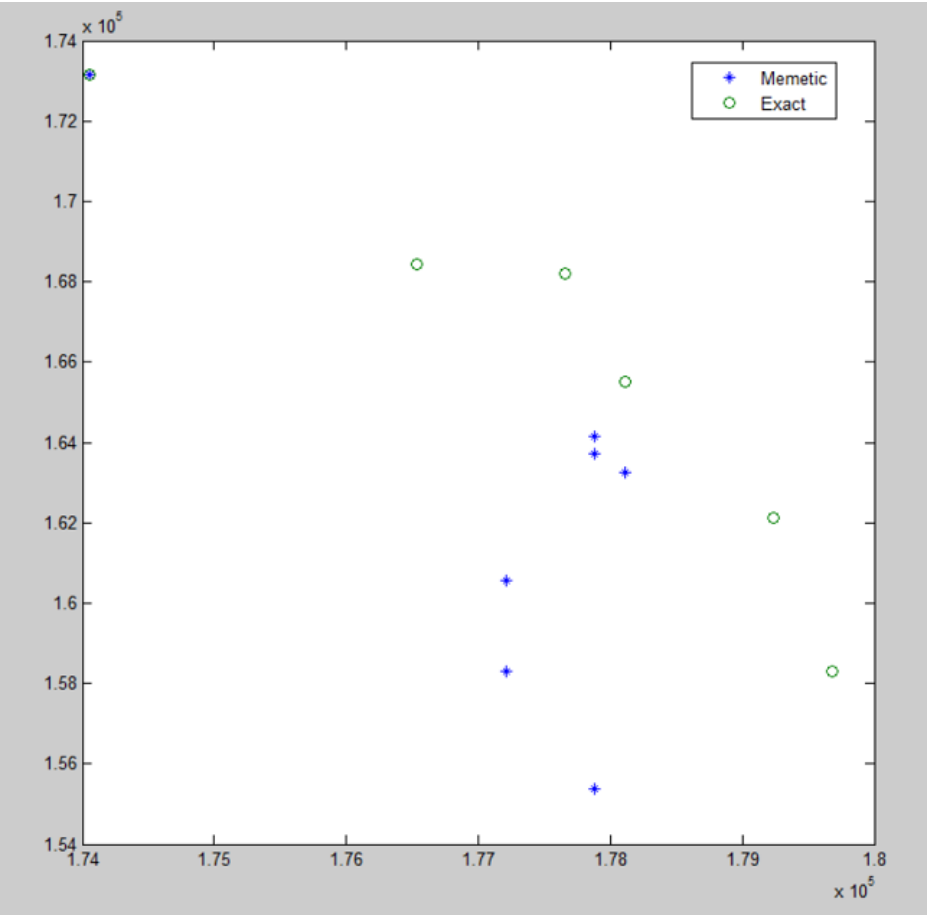

FIGURE 3.3 – Représentation graphique du front de Pareto obtenu par la méthode exacte et l'algorithme *MASNPL* pour 20 d'articles.

En comparant le temps d'exécution pour chaque algorithme appliqué aux mêmes instances créées aléatoirement selon les mêmes paramètres, nous concluons que notre méthode de résolution est plus rapide que la méthode exacte et nous donne des solutions de bonne qualité.

Exemple 3.1.3

- Nombre de fonctions objectifs : 2.
- Poids : Suit la loi normale avec une moyenne de 225 et une variance de 25.
- Profits par unité de poids : générés uniformément entre 1 et 10.
- Nombre d'articles est : 15.
- Capacité du sac-à-dos est : 2700.
- Pénalité d égale à 99.
- Nombre de générations pour l'algorithme MASNPL est : 50.

Le résultat de notre comparaison est présenté graphiquement sur la figure 3.4.

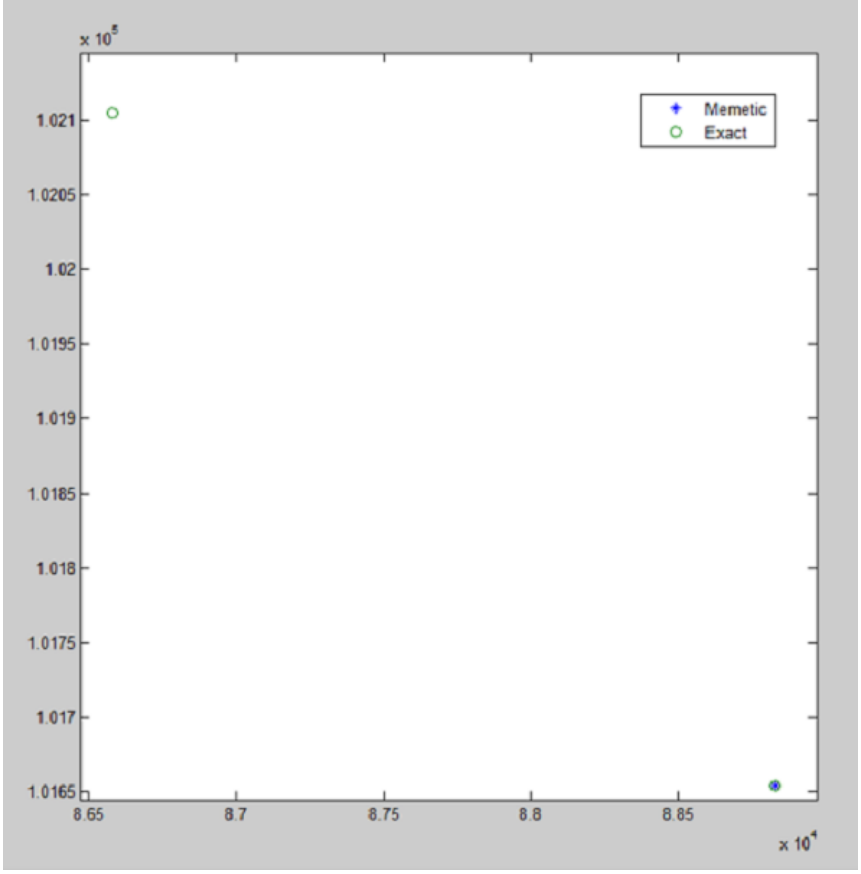

FIGURE 3.4 – Représentation graphique du front de pareto obtenu par la méthode exacte et l'algorithme *MASNPL* pour 15 d'articles.

Le temps total d'exécution de la méthode exacte pour obtenir la solution finale est : 1175,48 secondes (environ 20 mn), quant au temps total d'exécution de l'algorithme MASNPL est : 35,91 secondes (moins d'une minute).

En comparant le temps d'exécution pour chaque algorithme appliqué aux mêmes instances créées aléatoirement selon les mêmes paramètres, nous concluons que notre méthode de résolution est plus rapide que la méthode exacte et nous donne des solutions de bonne qualité.

3.6.2 Comparaison entre l'algorithme MASNPL et NSGA-II

Afin de réaliser une comparaison entre notre algorithme MASNPL et NSGA-II, nous implémentons ces méthodes sur MATLAB en utilisant des instances créées aléatoirement selon les paramètres suivants :

- Poids : Suit la loi normale avec une moyenne de 225 et une variance de 25.
- Profits par unité de poids : générés uniformément entre 1 et 10.
- Nombre de générations pour l'algorithme MASNPL est : 50.
- Nombre de générations pour NSGA-II est : 2500.

Initialement nous créons des instances aléatoires selon les paramètres présentés ci-dessus, par la suite, nous appliquons les algorithmes (MASNPL et NSGA-II) pour les mêmes instances créées 20 fois afin d'obtenir une population finale de taille 50, puis nous calculons la valeur de la fonction objectif f_i pour chaque individu de la population. Ensuite, nous trions la population en différents niveaux de non-domination, et nous attribuons à chaque individu de la population une valeur de crowding distance, en appliquant respectivement l'algorithme de tri non-dominé et l'algorithme de crowding-distance, dont nous sauvegardons les solutions non-dominés (c'est-à-dire de rang égal à 1) obtenues à chaque itération des algorithmes appliquées.

L'étape suivante de notre comparaison consiste à fusionner toutes les solutions non-dominées déjà sauvegardées et obtenues par les algorithmes (MASNPL et NSGA-II). Ensuite, nous trions (classons) les individus de la population fusionnée en différents niveaux de non-domination et nous leurs attribuons une valeur de crowding-distance, en appliquant respectivement l'algorithme de tri de non-dominé et l'algorithme de crowding-distance.

Dans le but d'évaluer la qualité des solutions obtenues en appliquant les algorithmes MASNPL et NSGA-II, nous utilisons le ratio qui détermine le pourcentage relatif au nombre de solutions non-dominées obtenu à partir de chaque algorithme par rapport au nombre total de solutions de rang 1.

Les résultats de nos tests relatifs à l'application des algorithmes (MASNPL, NSGA-II) sont représentés sur des tableaux, dont chaque colonne du tableau est définie comme suit :

La première colonne représente le numéro d'itération et la deuxième colonne représente respectivement le nombre de solutions non-dominées (rang égale à 1) obtenue à partir des algorithmes MASNPL et NSGA-II, la troisième colonne représente respectivement le temps total d'exécution de l'algorithme MASNPL et NSGA-II, finalement, la quatrième colonne représente respectivement le ratio relatif à l'algorithme MASNPL et NSGA-II.

Les exemples ci-dessous présentent les résultats obtenus selon les étapes présentées ci-dessus.

Exemple 3.2.1 Les données utilisées sont :

- Nombre de fonctions objectifs : 2.
- Nombre d'articles est : 20.
- Capacité du sac-à-dos est : 9000.
- Pénalité d égale à 330.

Table 3.1 : représente le nombre de solutions non-dominées obtenues à chaque itération des algorithmes (MASNPL et NSGA-II) pour 20 articles.
Pour les 10 tests effectués nous constatons les résultats suivants :

TABLE 3.1 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 20 articles

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	10	0	1713,51311	3419,76764	100	0
2	6	0	1609,18844	3376,87798	100	0
3	10	0	1516,1261	3250,44791	100	0
4	20	5	1347,10586	2879,00458	80	20
5	2	0	1400,02226	2831,42838	100	0
6	5	0	1319,84907	2909,47836	100	0
7	5	0	1328,41828	2930,77582	100	0
8	6	1	1418,68944	2882,2089	85,71	14,28
9	5	1	1357,27806	2911,37711	83,33	16,67
10	16	2	1422,045	2757,8156	88,89	11,11

Un total de 6 tests parmi les 10 réalisés, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL dominent les solutions obtenues en appliquant l'algorithme NSGA-II (pas de solutions de rang égal à 1) malgré que le nombre de générations pour l'algorithme NSGA-II est 50 fois supérieur au nombre de générations pour l'algorithme MASNPL.

Pour les 04 tests restants, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est plus grand que le nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II, cela signifie que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL sont situées dans une région plus dense.

En comparant les temps d'exécution relatifs à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

Exemple 3.2.2 Les données utilisées sont :

- Nombre de fonctions objectifs : 2.
- Nombre d'articles est : 100.
- Capacité du sac-à-dos est : 21000.
- Pénalité d égale à 752.

Table 3.2 : représente le nombre de solutions non-dominées obtenues à chaque itération des algorithmes (MASNPL et NSGA-II pour 100 articles)

Pour les 10 tests effectués nous constatons les résultats suivants :

Un total de 9 tests parmi les 10 réalisés, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL dominent les solutions obtenues en appliquant l'algorithme NSGA-II (pas de solutions de rang égal à 1) malgré que le

TABLE 3.2 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 100 articles

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	4	0	2555, 20082	4160, 13939	100	0
2	1	1	2485, 20063	4235, 54793	50	50
3	3	0	2393, 36046	4176, 23395	100	0
4	8	0	2481, 62	3965, 26625	100	0
5	1	0	2501, 27197	4039, 40369	100	0
6	1	0	2578, 42903	4144, 85823	100	0
7	8	0	2270, 357	4106, 02298	100	0
8	1	0	2590, 46484	8054, 85057	100	0
9	1	0	2504, 63287	4205, 24089	100	0
10	16	0	2092, 41251	3490, 4851	100	0

nombre de générations pour l'algorithme NSGA-II est 50 fois supérieur au nombre de générations pour l'algorithme MASNPL.

Pour le test restant, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est égal au nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II.

En comparant le temps total d'exécution relatif à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

Exemple 3.2.3 Les données utilisées sont :

- Nombre de fonctions objectifs : 2.
- Nombre d'articles est : 200.
- Capacité du sac-à-dos est : 40000.
- Pénalité d égale à 1424.

Table 3.3 : représente le nombre de solutions non-dominées obtenues à chaque itération par les algorithmes (MASNPL et NSGA-II) pour 200 articles.

Pour les 10 tests effectués nous constatons les résultats suivants :
Au total de 9 tests parmi les 10 réalisés, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL dominent les solutions obtenues en appliquant l'algorithme NSGA-II (pas de solutions de rang égal à 1) malgré que le nombre de générations pour l'algorithme NSGA-II est 50 fois supérieur au nombre de générations pour l'algorithme MASNPL.

Pour le test restant, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est deux fois le nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II.

TABLE 3.3 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 200 articles

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	5	0	4298,38137	4627,86451	100	0
2	5	0	4329,33696	4577,69908	100	0
3	2	1	4286,61502	4587,7257	66,67	33,33
4	1	0	4457,91234	4567,6765	100	0
5	5	0	4040,51267	4342,15335	100	0
6	1	0	4341,55699	4465,75378	100	0
7	3	0	4451,06075	4527,91025	100	0
8	3	0	4537,64983	4554,9791	100	0
9	5	0	4267,92233	4471,04282	100	0
10	1	0	4479,37789	4662,25679	100	0

En comparant le temps d'exécution relatif à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

Exemple 3.2.4 Les données utilisées sont :

- Nombre de fonctions objectifs : 4.
- Nombre d'articles est : 50.
- Capacité du sac-à-dos est : 9000.
- Pénalité d égale à 330.

Table 3.4 : représente le nombre de solutions non-dominées obtenues à chaque itération par les algorithmes (MASNPL et NSGA-II pour 50 articles)

TABLE 3.4 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 50 articles

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	66	26	1851,74066	2445,45638	71,74	28,26
2	31	2	1905,95985	2648,07941	93,94	6,06
3	71	22	1905,95985	2648,07941	76,34	23,65
4	69	19	1930,86798	2560,80004	78,41	21,59
5	29	9	1886,09592	2627,31683	76,31	23,68
6	14	31	1951,97476	2582,64445	36,84	68,69
7	97	10	1946,74633	2408,61901	90,65	9,34
8	13	5	1938,47225	2592,48961	72,22	27,78
9	1	13	2003,46515	2616,81209	7,14	92,86
10	9	11	1965,69301	2734,79297	45	55

Pour les 10 tests effectués nous constatons les résultats suivants :

Au total de 07 tests parmi les 10 réalisés, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est supérieur au nombre

de solutions obtenues en appliquant l'algorithme NSGA-II, malgré que le nombre de générations pour l'algorithme NSGA-II est 50 fois supérieur au nombre de générations pour l'algorithme MASNPL, cela signifie que les solutions non-dominées trouvées en appliquant l'algorithme sont situées dans une région plus dense.

Pour les 03 tests restants, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II est supérieur au nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL.
En comparant le temps d'exécution relatif à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

Exemple 3.2.5 Les données utilisées sont :

- Nombre de fonctions objectifs : 3.
- Nombre d'articles est : 100.
- Capacité du sac-à-dos est : 21000.
- Pénalité d égale à 752.

Table 3.5 : représente le nombre de solutions non-dominées obtenues à chaque itération par les algorithmes (MASNPL et NSGA-II)pour 100 articles et 3 fonctions objectifs.

TABLE 3.5 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 100 articles et 3 fonctions objectifs

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	18	6	3338,65774	4686,83084	75	25
2	1	0	3255,72489	4753,71708	100	0
3	17	1	3316,3073	4848,68752	94,44	5,55
4	10	1	3164,97535	4761,14919	90,91	9,09
5	0	2	3261,58644	4751,8666	0	100
6	18	6	3251,30331	4100,82273	75	25
7	1	0	3204,16986	4137,24102	100	0
8	17	1	3233,53408	4234,42546	94,44	5,55
9	10	1	3138,91231	4260,57573	90,91	9,09
10	1	0	3101,46999	4057,35701	100	0

Pour les 10 tests effectués nous constatons les résultats suivants :

Au total de 03 tests parmi les 10 réalisés, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL dominent les solutions obtenues en appliquant l'algorithme NSGA-II (pas de solution de rang égal à 1) malgré que le nombre de générations pour l'algorithme NSGA-II est 50 fois supérieur au nombre de générations pour l'algorithme MASNPL.

Au total de 06 tests parmi les 10 réalisés, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est supérieur au nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II, cela signifie que les solutions non-dominées trouvées en appliquant l'algorithme sont situées dans une région plus dense.

Pour le seul test restant, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme NSGA-II dominent les solutions non-dominées obtenues en appliquant l'algorithme MASNPL.

En comparant le temps d'exécution relatif à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

Exemple 3.2.6 Les données utilisées sont :

. Nombre de fonctions objectifs : 5.

. Nombre d'articles est : 150.

. Capacité du sac-à-dos est : 28000.

. Pénalité d égale à 1008.

Table 3.6 : représente le nombre de solutions non-dominées obtenues à chaque itération en appliquant les algorithmes (MASNPL et NSGA-II)pour 150 articles et 5 fonctions objectifs.

TABLE 3.6 – Nombre de solutions non-dominées obtenues en appliquant les algorithmes MASNPL et NSGA-II pour 150 articles et 5 fonctions objectifs.

	Nbr_solution Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	14	15	4994,0142	7138,69308	48,27	51,72
2	6	11	5149,28687	7544,84421	35,29	64,7
3	2	3	5273,99488	7636,07286	40	60
4	30	21	5119,07871	7084,38893	58,82	41,18
5	12	2	5262,87698	7011,5114	85,71	14,28
6	3	1	4743,11459	6919,07195	75	25
7	4	0	4885,45103	7210,53789	100	0
8	3	1	4791,12316	7187,80439	75	25
9	5	1	5134,35748	7932,27036	83,33	16,67
10	3	1	5063,31278	7366,53774	75	25

Pour les 10 tests effectués nous constatons les résultats suivants :

Seulement un seul test parmi les 10 réalisés, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme MASNPL dominent les solutions obtenues en appliquant l'algorithme NSGA-II (pas de solution de rang égal à 1) malgré

que le nombre de générations relatif à l'algorithme NSGA-II est 50 fois supérieur au nombre de générations relatif à l'algorithme MASNPL.

Au total de 06 tests parmi les 10 réalisés, nous remarquons que le nombre de solutions non-dominées obtenues en appliquant l'algorithme MASNPL est supérieur au nombre de solutions non-dominées obtenues en appliquant l'algorithme NSGA-II, cela signifie que les solutions non-dominées trouvées en appliquant l'algorithme sont situées dans une région plus dense.

Pour les trois tests restants, nous remarquons que les solutions non-dominées obtenues en appliquant l'algorithme NSGA-II dominent les solutions non-dominées obtenues en appliquant l'algorithme MASNPL.

En comparant le temps d'exécution relatif à chaque algorithme, nous concluons que MASNPL est plus rapide que NSGA-II.

D'après ces résultats, nous pouvons conclure que l'algorithme MASNPL est significativement plus performant que l'algorithme NSGA-II.

3.7 Conclusion

Dans cette section, nous avons détaillé le modèle relatif au problème de sac-à-dos multi-objectifs stochastique quadratique avec recours simple et à poids aléatoires.

Comme les fonctions objectfs ne sont pas différentiables, alors nous avons utilisé la méthode d'approximation par convolution pour estimer leurs gradients.

Dans le but de résoudre le problème MO-SQKP, nous devons passer par plusieurs étapes dont la première est l'élaboration d'une population initiale réalisable en utilisant l'heuristique constructive gloutonne, dont nous lui appliquons l'algorithme de tri non-dominé afin de trier les individus de cette population initiale en différents niveaux de non-domination, ensuite nous lui attribuons une valeur de crowding distance en appliquant l'algorithme de crowding-distance. Au résultat, nous aurons une population initiale réalisable trié selon le niveau de non-domination, avec une valeur de crowding distance attribuée à chaque individu, dans le but d'améliorer la qualité de cette population et avoir des solutions potentiellement efficaces, nous appliquons une série de mutations, de croisements et de recherches locales générer une population d'enfant dont nous appliquons à cette dernière l'algorithme SNPLS basé sur la comparaison entre une solution actuelle (enfant) et une nouvelle solution obtenue par l'algorithme de gradient. Ensuite, l'algorithme de tri non-dominé et l'algorithme de crowding-distance sont appliqués à la progéniture améliorée pour sélectionner nos premiers meilleurs individus finaux de la population.

En ce basant sur les résultats expérimentaux de la comparaison entre le MASNPL et la NSGA-II, nous pouvons conclure que l'utilisation de l'algorithme de gradient avec la NSGA-II donne des résultats nettement meilleurs et plus efficaces que la NSGA-II.

CHAPITRE 4

PROBLÈME DE SAC-À-DOS MULTIPLE STOCHASTIQUE MULTI-OBJECTIFS

4.1 Introduction

Le processus de prise de décision est fondamental dans de nombreuses applications pratiques et réelles. Dans des environnements incertains, les décideurs doivent souvent choisir une solution parmi un ensemble de solutions efficaces, mais cette tâche est difficile à caractériser, car il existe plusieurs définitions de solutions efficaces aléatoires. À notre connaissance, le problème multi-objectifs stochastique n'a pas été étudiés dans la littérature.

Dans cette étude, nous nous concentrons sur le problème de sac-à-dos stochastique, dans lequel les poids des articles sont aléatoires et suivent une loi normale avec une moyenne et une variance connues. Ce problème est formulé comme un problème d'optimisation à deux étapes, également connu sous le nom de modèles d'optimisation avec recours. La première étape consiste à choisir des articles aléatoirement sans connaître leurs poids, tandis que la deuxième étape est une phase de révélation des poids. Cette phase a pour but de corriger la décision prise lors de la première étape en ajoutant des articles si les capacités des sacs-à-dos le permettent ou en retirant des articles dans le cas contraire (surcharge.

Pour résoudre ce problème, nous proposons une méthode de résolution sous forme d'un algorithme memetique qui utilise l'heuristique constructive gloutonne pour trouver une solution initiale réalisable. En outre, nous introduisons une heuristique appelée MTHMn, qui est inspirée de l'heuristique MTHM de Martello et Toth, visant à améliorer la qualité des solutions existantes. Les résultats de calcul sur des instances générées aléatoirement montrent que l'approche proposée aboutit à des solutions de meilleure qualité en un temps d'exécution raisonnable

4.2 Formulation mathématique

Le problème de sac-à-dos multiple stochastique multi-objectifs MO-SMKP est défini comme suit :

Étant donné un ensemble de n articles (ou éléments), soit $S = \{1, \ldots, s\}$ un ensemble de sacs-à-dos, le poids de chaque article n'est pas connu durant la première étape mais il devient connu avant la prise de décision lors de la deuxième étape. Par conséquent, nous traitons les poids comme des variables aléatoires et supposons que le vecteur de poids aléatoire χ_i de l'article i suit la loi normale avec une moyenne $\mu_i > 0$ et un écart-type σ_i. Supposons que tous les poids sont strictement positifs.

Soit $p^1, ..., p^L$ les probabilités non nulles respectives de ces L scénarios.

Chaque sac-à-dos j, $(j \in S)$, a une capacité de poids C_j. L'objectif du problème des sacs-à-dos multiples stochastiques SMKP est d'affecter les n articles aux sacs-à-dos de telle sorte que le profit total des articles affectés soit maximisé tandis que la somme des poids des articles affectés à chaque sac-à-dos ne dépasse pas la capacité de ce sac-à-dos.

En outre, chaque article a un profit fixe par unité de poids de type m-vecteurs $(r_i) = \left(r_i^k\right)_{k=1,\ldots,m}$ telle que $r_i^k \in \mathbb{N}$.

Le choix d'un profit par unité de poids peut être justifié par le fait que la valeur d'un article dépend souvent de son poids que l'on ne connaît pas à l'avance.

La sélection d'un article est définie par une variable de décision binaire x_{ij} qui prend la valeur 1 si l'article i est affecté au sac-à-dos j et prend la valeur 0 dans le cas contraire.

Au cours de la première étape, il est possible de ranger les articles dans les sacs-à-dos. Toutefois, si le poids cumulé de ces articles excède la capacité du sac, il faudra retirer certains d'entre eux dans la deuxième étape. Afin de réduire le pourcentage de cas où les articles placés dépassent la capacité du sac-à-dos lors de la première étape, une contrainte aléatoire est introduite.

Au début de la deuxième étape, les poids de tous les éléments sont révélés. Dans le but de respecter la contrainte de capacité et/ou de maximiser le profit total, des éléments placés lors de la première étape peuvent alors être retirés et des éléments supplémentaires peuvent être placés lors de la deuxième étape. Dans ce cas nous intro- duisons deux variables aléatoires binaires.

Dans le cas d'une surcharge, l'élément i doit être retiré du sac et une pénalité $d_i^{(k)} > 0$ doit être payée et dépend du poids de l'article retiré, qui est strictement supérieure au profit relatif à la première étape. Dans ce cas notre objectif est donc de minimiser la pénalité totale. Le retrait de l'élément i lors de la deuxième étape est modélisé par une variable de décision y_{ij}^- qui prend la valeur 1 si l'élément i est retiré du sac-à-dos j et prend la valeur 0 dans le cas contraire.

Dans le cas où la contrainte de capacité est respectée, $y_{ij}^{+} = 1$ si nous décidons de rajouter l'article i au sac-à-dos j si et seulement si le rajout de cet article ne conduit pas à une surcharge.

Nous supposons que le vecteur profit de l'élément i lors de la deuxième étape $\check{r}_i^k = (\check{r}_i^1, \ldots, \check{r}_i^m), k \in \{1, \ldots, m\}$ est strictement inférieur à celui de la première étape $r_i^k = (r_i^1, \ldots, r_i^m), k \in \{1, \ldots, m\}$.

Le problème de sac-à-dos multiple stochastique multi-objectifs est formulé mathématiquement comme suit :

$$
\begin{aligned}
&\max \sum_{i=1}^{n}\sum_{j=1}^{s} r_i^{(k)} x_{ij} + \mathbf{E}\Big[\max_{y^-,y^+\in\{0,1\}^{n*s}} \sum_{i=1}^{n}\sum_{j=1}^{s} \check{r}_i^{(k)} y_{ij}^{+} - \sum_{i=1}^{n}\sum_{j=1}^{s} d_i^{(k)} y_{ij}^{-}\Big], \forall k \in \{1,\ldots,m\} && (4.1.1)\\
&P\big(\sum_{i=1}^{n} \chi_i x_{ij} \le C_j\big) \;\ge\; p && \forall j \in \{1,\ldots,s\} && (4.1.2)\\
&y_{zj}^{+} \;\le\; 1 - x_{zj}, && \forall z \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,s\} && (4.1.3)\\
&y_{zj}^{-} \;\le\; x_{zj}, && \forall z \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,s\} && (4.1.4)\\
&y_{zj}^{+} \;\le\; 1_{\mathbb{R}^+}\Big(C_j - \sum_{i=1}^{n} \chi_i x_{ij}\Big).y_{zj}^{+}, && \forall z \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,s\} && (4.1.5)\\
&y_{zj}^{-} \;\le\; 1_{\mathbb{R}^+}\Big(\sum_{i=1}^{n} \chi_i x_{ij} - C_j\Big).y_{zj}^{-}, && \forall z \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,s\} && (4.1.6)\\
&\sum_{i=1}^{n}(x_{ij} + y_{ij}^{+} - y_{ij}^{-})\chi_i \;\le\; C_j, && \forall j \in \{1,\ldots,s\} && (4.1.7)\\
&\sum_{j=1}^{s} x_{ij} \;\le\; 1, && \forall i \in \{1,\ldots,n\} && (4.1.8)\\
&\sum_{j=1}^{s} x_{ij}(x_{ij} + y_{ij}^{+}) \;\le\; 1, && \forall i \in \{1,\ldots,n\} && (4.1.9)
\end{aligned}
$$

(4.1)

où :

- $\mathbf{P}[A]$ représente la probabilité d'un événement A.
- $E\,[.]$ représente l'espérance.
- $\mathbf{1}_{\mathbb{R}^+}$ représente la fonction indicatrice.
- $\check{r}_i < r_i$ pout tout $i = 1, \ldots, n,$.
- $p \in [0.5, 1]$ représente la probabilité, sa valeur est fixée par le décideur.

L'équation (4.1.1) a pour objectif de maximiser le gain total estimé.
Contrainte (4.1.2) est une contrainte de probabilité et est introduite dans le but de réduire la pourcentage d'avoir une surcharge lors de la deuxième étape.

Contrainte (4.1.3) garantit que si un élément i est sélectionné au cours de la première étape et n'a pas été retiré, alors il est nécessairement considéré comme sélectionné lors de la deuxième étape.

Contrainte (4.1.4) signifie que si un élément i est sélectionné lors de la première étape peut être retiré.

Contraintes (4.1.5) et (4.2.6) garantissent que les éléments peuvent être rajoutés lors de la deuxième étape si et seulement si les articles choisis lors de la première étape ne conduit pas à une surcharge, et peuvent être retirés dans le cas d'une surcharge.

Contrainte (4.1.7) garantit que le poids total des articles affectés à chaque sac-à-dos ne dépasse pas sa capacité.

Contrainte (4.1.8) stipule que chaque article est affecté à au plus un seul sac-à-dos.

Contrainte (4.1.9) stipule que chaque article rajouté lors de la deuxième étape est affecté à au plus un seul sac-à-dos.

4.3 Méthode de résolution

Lors de la première étape nous devons choisir des articles, dont le poids de chaque article n'est pas connu à l'avance et varie autour d'une valeur moyenne ce qui entraine deux cas possibles dans la deuxième étape.

Après la révélation des poids des articles, cela peut engendrer soit un cas de surcharge, par conséquent nous sommes dans l'obligation de rejeter (retirer) des articles déjà choisis lors de la première étape et qui entraine des coûts supplémentaires (payement d'une pénalité d_i pour chaque article retiré). Afin d'éviter ce cas, nous introduisons la contrainte de probabilité. Par contre le deuxième cas est que la capacité du sac-à-dos n'est pas atteinte par le poids total des articles choisis à l'avance, par conséquent, des articles peuvent être rajoutés lors de la deuxième étape, tout en respectant la contrainte de capacité afin d'éviter des charges supplémentaires.

Dans le processus de décomposition, le MO-SMKP original est d'abord décomposé en plusieurs sous-problèmes de sac-à-dos stochastiques multiples à objectif unique (SO-SMKP). Etant donné le vecteur objectif $F(x) = (f_1(x), f_2(x), \ldots, f_m(x))$ et le vecteur de pondération $\lambda = (\lambda_1, \ldots, \lambda_m)$, où la somme de ces derniers doit être égale à 1.

La fonction objectif d'un sous-problème est définie comme suit :

$$J_k(x, \chi) = \sum_{k=1}^{m} \lambda_k f_k(x) \quad (4.2)$$

$$\sum_{k=1}^{m} \lambda_k = 1 \quad (4.3)$$

Nous pouvons écrire la fonction $J_k(x, \chi)$ comme suit :

$$J_k(x, y) = \sum_{k=1}^{m} \lambda_k \left[\sum_{i=1}^{n} \sum_{j=1}^{s} r_i^{(k)} x_{ij} + \sum_{l=1}^{L} p^l \left[\sum_{i=1}^{n} \sum_{j=1}^{s} \check{r}_i^{(k)} y_{ij}^{+} - \sum_{i=1}^{n} \sum_{j=1}^{s} d_i^{(k)} y_{ij}^{-} \right] \right] \quad (4.4)$$

Les vecteurs de pondération sont calculés en fonction de la valeur de la norme euclidienne et les paramètres ω_i comme suit :

$$\lambda_k = \frac{\omega_k}{\sum_{k=1}^{m} \omega_k} \tag{4.5}$$

Où :

- ω_k est déterminé comme suit :

$$\omega_k(x) = \frac{f_k(x)}{\|f(x)\|} \tag{4.6}$$

- m est le nombre de fonctions objectifs.
- $f_k(x)$ est la valeur de la k^{me} fonction objectif de la solution x,
- $\|f(x)\|$ est la norme euclidienne, donnée par : $\|f(x)\| = \sqrt{f_1^2(x) + \ldots + f_m^2(x)}$
  Les fonctions objectifs sont combinées en une seule fonction objectif F, lorsque toutes les pondérations individuelles sont déterminées pour tous les objectifs, de la façon suivante :

$$F = \lambda_1 f_1 + \lambda_2 f_2 + ... + \lambda_m f_m$$

4.3.1 La population initiale

La méthode utilisée pour créer la population initiale consiste en une heuristique constructive gloutonne. Cette méthode suppose que les sacs-à-dos sont triés initialement par ordre décroissant de capacité $C_1 \leq C_2 \leq \ldots \leq C_s$, tandis que les articles sont triés par ordre croissant en fonction de leur rapport profit/poids : $\frac{r_1}{w_1} \geq \frac{r_2}{w_2} \geq ... \geq \frac{r_n}{w_n}$. Les sacs triés sont ensuite remplis un par un, en commençant par le premier et en utilisant uniquement les éléments restants pour remplir le sac suivant, jusqu'à ce que tous les sacs soient remplis tout en respectant la contrainte de capacité. Si, à la fin du processus, aucun article ne peut être affecté à un sac-à-dos, la capacité actuelle de chaque sac-à-dos est $\check{C}j = Cj$ et la solution actuelle est représentée de cette manière. Cette procédure est répétée s fois jusqu'à ce que tous les sacs soient remplis, la solution actuelle est représentée comme suit :

$$Z_i = \begin{cases} 0 \text{ Si l'élément } i \text{ n'est pas actuellement assigné.} \\ \text{j Indice du sac-à- dos auquel l'article est assigné.} \\ \text{Sinon.} \end{cases}$$

4.3.2 Génération de sous-populations

Notre but est de générer une population de taille N, cet algorithme est détaillé dans le chapitre 3.3.2.

Algorithme 8 Heuristique constructive gloutonne

Input : r_j, w_j,$\check{c}_i$,s,n ;
        N : La taille de population
Output : Obj ,Z_i.
$Obj = 0$;
Pour $j = 1$ **jusqu'à** s **faire**
    $i = 1$;
    Tant que $i \leq n$ **faire**
        Si $Z_i = 0$ et $w_i <= \check{C}_j$ **alors**
            $Z_i := j$
            $\check{C}_j := \check{C}_j - w_i$
            $Obj := Obj + r_i$
            $i = i + 1$;
        fin Si
    fin Tant que
fin Pour

4.3.3 Heuristiques d'amélioration

Heuristique MTHM (Martello et Toth)

Martello & Toth **[40]** ont proposé une heuristique très efficace appelée *MTHM* pour résoudre le problème des sacs-à-dos multiples MKP et obtenir des solutions de bonnes qualitées dans un temps d'exécution raisonnable.

Nous supposons que les sacs-à-dos sont triés dans l'ordre décroissant par rapport aux capacités $C_1 \leq C_2 \leq \ldots \leq C_s$, et que tous les articles sont triés de manière décroissante en fonction de leurs rapports *profit/poids :* $\left(\frac{r_i}{w_i}\right)$, $\frac{r_1}{w_1} \geq \frac{r_2}{w_2} \geq ... \geq \frac{r_n}{w_n}$.

La procédure MTHM est présentée ci-dessous :

Algorithme 9 MTHM Heuristic

Input : n , r_i, w_i ;
        N : La taille de population.
Output : $Obj.Z_i$.
Solutio initiale : Heuristique constructive gloutonne
Rearrangement
Première amélioration
Deuxième amélioration

Phase 1 **: Population Initiale :** en appliquant la procédure constructive gloutonne nous obtenons une population initiale réalisable de taille N.

Phase 2 **: Réarrangement :** le but de cette procédure est d'améliorer les solutions réalisables initialement trouvées en les réorganisant. Initialement tous les articles actue-

llement présents dans la solution sont retirés, puis nous les trions dans l'ordre décroissant relatif au rapport *profit/poids*, ensuite, le premier élément est affecté au premier sac-à-dos, le deuxième élément au deuxième sac-à-dos, et ainsi de suite de manière cyclique, si un élément i ne peut être affecté au sac j, nous testons pour le même élément mais avec le sac suivant, et ainsi de suite, dans le cas où cet élément n'est affecté à aucun sac, nous l'excluons et nous lui attribuons un zéro ($Z_i = 0$)

Phase 3 **:** **Première Amélioration :** le but de cette phase est de maximiser le profit totale, nous considèrons toutes les paires d'éléments assignées à des sacs-à-dos différents, puis nous procédons à la permutations entre deux articles se trouvant dans deux sacs différents entrainant l'insertion d'un nouveau article déjà exclus lors de la phase réarra- ngement, cette permutation est retenue si et seulement si un article déjà exclus est affecté à l'un des sacs-à-dos et que le profit est maximisé, l'exécution de cette phase s'arrête lorsqu'aucun autre remplacement n'est possible.

Phase 4 **:** **Deuxième Amélioration :** dans cette phase, l'algorithme essaie d'exclure chaque élément trouvant actuellement dans la solution, et de le remplacer par un ou plusieurs éléments qui ne sont pas dans la solution dans le but de maximiser le profit totale, l'exécution de cette phase s'arrête lorsqu'aucun autre remplacement n'est possible.

Heuristique MTHMn

Notre méthode appelée *MTHMn* et est inspirée de l'heuristique MTHM, le but de cette méthode est d'améliorer la solution initiale de taille N et d'obtenir des solutions meilleures en un temps d'exécution raisonnable.
Notre méthode MTHMn suit les mêmes phases que la méthode MTHM déjà expliquée, sauf que pour la quatrième phase, nous appliquons le processus suivant :
Pour chaque sac-à-dos, nous trions tous les articles déjà affectés au sac dans l'ordre croissant relatif au rapport $\frac{profit}{poids}$ ($\frac{r_i}{w_i}$), ensuite, nous retirons les articles déjà triés de telle sorte que la somme de leurs capacités est $\left(\frac{1}{12}\right)$ par rapport à la capacité totale du sac, par la suite, nous remplaçons les articles retirés par un ou plusieurs articles qui ne sont pas actuellement dans la solution, ce remplacement est retenu si et seulement s'il entraine une maximisation du profit. l'exécution de cette phase s'arrête lorsqu'aucun autre remplacement n'est possible.

4.3.4 Algorithme NSGA-II

Notre Algorithme Memetique Avec MTHM et MTHMn est inspiré de l'algorithme génétique de tri non-dominé II (NSGA-II).

Le NSGA-II est considéré parmi les algorithmes d'optimisation multi-objectifs les plus populaires et efficaces que sa version précédente NSGA et a tendance à se converger vers le front Pareto rapidement, son principal avantage est basé sur la stratégie de préserver la diversité des solutions. La NSGA-II peut être détaillée comme suit :

Étape 1 **: population initiale :** La population est initialisée en fonction de l'étendue du problème et les contraintes et nous obtenons une population initiale réalisable de taille N en appliquant l'heuristique constructive gloutonne.

Étape 2 **: tri non-dominé :** La population initiale obtenue est triée selon le niveau de non-domination en appliqant L'algorithme de tri non-dominée décrit dans la section 3.3.4.

Étape 3 **: crowding distance :** nous attribuons à chaque indivu de la population initiale une valeur de crowding-distance en appliquant l'algorithme crowding-distance décrit dans la section 3.3.5.

Étape 5 **: opérateurs génétiques :** Le codage GA utilise un croisement binaire simulé et une mutation polynomiale.

Étape 6 **: recombinaison & sélection :** La population de la progéniture et celle de la génération actuelle sont combinées, et les individus de la génération suivante sont choisis par le processus de sélection. Les individus de la nouvelle génération sont ceux appartenant aux meilleurs fronts non-dominé jusqu'à ce que nous auront une population de taille N.

4.4 Algorithme de tri non-dominé

L'objectif principal de l'algorithme de tri non-dominé est de trier les individus de la population en différents niveaux de non-domination. Cet algorithme est détaillé dans le chapitre 3.3.4.

4.5 Algorithme crowding-distance

La valeur de crowding-distance est un critère secondaire pour sélectionner les solutions ayant le même niveau de non-domination. Cet algorithme est détaillé dans le chapitre 3.3.5.

4.6 Algorithme memetique avec MTHM et MTHMn

Dans cette section, nous présentons notre Algorithme Memetique avec MTHM et MTHMn pour MO-SMKP, comprenant l'initialisation, le croisement et la recherche locale.

Algorithme memetique avec MTHM et MTHMn (MAMTHMn)

Input : P : La population actuelle de taille N ;
    N : La taille de population ;
    Q : L'ensemble des enfants générés ;
Trouver une population initiale de taille N en appliquant l'heuristique constructive gloutonne ;
Appliquer la procédure $MTHMn$ à la population initiale de taille N ;
Calculer la valeur de chaque fonction objectif pour chaque individu dans la population de taille N ;
Appliquer l'algorithme de tri non-dominé à (P, N) ;
Appliquer l'algorithme Crowding-Distance (P, N) ;
$NG = 1$;
Tant que $NG \leq NG_{\max}$ **faire**
    $Q = \emptyset$;
    Pour $k = 1$ **jusqu'à** $\frac{N}{2}$ **faire**
        Sélectionner les parents en appliquant la sélection par tournoi binaire ;
        Appliquer l'opération de croisement à deux parents sélectionnés pour générer deux enfants (Q_1, Q_2) ;
        AAppliquer l'opération de mutation sur les deux enfants (Q_1, Q_2) avec une probabilité égale à $(P_m = \frac{1}{n})$;
        Pour $j = 1$ **jusqu'à** 2 **faire**
            $Q_j \longleftarrow$ Appliquer la procédure de $MTHM$ à (Q_j) ;
            $Q = Q \cup \{Q_j\}$;
        fin Pour
    fin Pour
    $R_t = P_t \cup Q_t$;
    Appliquer l'algorithme de tri non-dominé à (R_t) ;
    Réduire la taille de la population à N ;
    Appliquer l'algorithme Crowding-Distance à (R_t, N) ;
    $P_{t+1} = Q_t$;
    $NG = NG + 1$;
fin Tant que

La partie initiale de notre algorithme memetique est une phase primaire et est composée de deux parties : la première consiste à générer une solution réalisable de taille N en appliquant l'heuristique constructive gloutonne, quant à la seconde est une phase d'amélioration dont la procédure MTHMn (resp MTHM) est appliquée à la solution réalisable initialement obtenue.

Ensuite, une valeur de fitness (valeur de la fonction objectif f_i) est calculée pour chaque individu de la population, par la suite, l'algorithme de tri non-dominé est appliqué à la polulation initialement trouvée afin de trier les individus selon le niveau de non-domination. Pour chaque individu, nous calculons une valeur de crowding distance ($(p_{distance})$) en appliquant l'algorithme de crowding-distance. Ces deux paramètres ($(p_{distance})$) et ((p_{rank})) permettent de sélectionner les individus situées dans la région la plus dense et qui ont une meilleur répartition sur le front Pareto optimal. Ensuite, nous effectuons une sélection par tournoi binaire au sein de la population pour choisir les parents, dont nous leurs appliquons une série d'opérations génétiques de croisement et de mutation pour générer de nouvelles solutions progénitures (enfants) de taille N.

L'étape suivante consiste à appliquer la procédure MTHMn (resp MTHM) pour chaque enfant pour obtenir des solutions de meilleures qualités. Nous obtiendrons une population combinée ($(R_t = P_t \cup Q_t)$) qui fusionnent les parents et les enfants.

où :

- P_t : Représente la population parent de taille N.
- Q_t : Représente la population progéniture (enfant) de taille N.
- R_t : Représente la population combinée de taille $2N$.

Dans le but d'obtenir une population de taille N à partir d'une population combinée, nous appliquons à cette dernière les algorithmes de tri non-dominé et Crowding-distance, dont nous obtiendrons une population R_t triée selon le niveau de non-domination, ensuite, nous procédons au processus de sélection afin d'obtenir une population de taille N. le choix (sélection) se fait comme suit :

Choisir les solutions appartenant au font non-dominé $\mathcal{F}_1$ car elles représentent les meilleures solutions de la population combinée et qui ne sont pas dominées par d'autres solutions. Si la taille de $\mathcal{F}_1$ est égale à N, nous rajoutons tous les membres de l'ensemble $\mathcal{F}_1$ à la nouvelle population P_{t+1}. Sinon, si la taille de $\mathcal{F}_1$ est inférieure à N, les membres restants de la population P_{t+1} pour atteindre la taille N sont choisis parmi les fronts non-dominés suivants en respectant l'ordre de leur classement, c'est-à-dire niveau de non-domination $\mathcal{F}_2$, si la taille de la population obtenue et appartenant aux niveaux non-dominés ($\mathcal{F}_1$, $\mathcal{F}_2$) égale à N nous arrêtons l'itération de l'algorithme et nous affectons les solutions appartenant à ($\mathcal{F}_1$, $\mathcal{F}_2$) à la population P_{t+1}, sinon si la taille de la population appartenant aux niveaux non-dominés ($\mathcal{F}_1$, $\mathcal{F}_2$) est inférieur à N, nous appliquons le même processus déjà expliquer pour le front non-dominé suivant ($\mathcal{F}_3$), et nous comparons à chaque fois la taille totale de la solution obtenue par rapport à N, et ainsi de suite jusqu'à avoir une population de taille N qui sera affectée à P_{t+1}, dont nous lui appliquons un opérateur de sélection de tournoi binaire, de croisement et de mutation pour créer une nouvelle population Q_{t+1} de taille N. Le processus expliqué au-dessus est répété $(NG_{\max})$.

4.7 Résultats numériques

Dans cette section nous présentons nos résultats numériques, afin d'évaluer les performances de notre méthode de résolution. Suite absence de benchmark, nous proposons de comparer nos algorithmes avec une méthode exacte.

L'implémentation de toutes les méthodes de résolutions utilisées ainsi que les exemples sont réalisés sur un ordinateur de caractéristiques : processeur Intel core $i5-4200U$ de 1.60 Ghz et 4 Go de RAM.

Pour tous nos résultats, nos algorithmes, la méthode exacte sont testés sur des instances générées aléatoirement et construites selon les paramètres suivants :

- les poids et les profits suivent la loi uniforme sur l'intervalle $[10, 100]$
- La valeur de la Pénalité d utilisée est : 1.5.
- Les poids des articles suivent la loi uniforme sur l'intervalle $\left[0.4 \sum_{i=1}^{n} \frac{w_i}{s}, 0.6 \sum_{i=1}^{n} \frac{w_i}{s}\right]$ avec $j = 1, ..., s-1$..
- La valeur de poids du s^{me} sac-à-dos est calculée comme suit : $C_s = 0.5 \sum_{i=1}^{n} w_i - \sum_{j=1}^{s-1} C_j$

4.7.1 Comparaison entre l'algorithme MAMTHMn et la méthode exacte

Dans le but de résoudre Le problème de sac-à-dos multiple stochastique multi-objectifs MO-SMKP avec une méthode exacte, nous utilisons le solveur ILOG CPLEX pour générer les instances initialement créées. Pour cela, nous devons utiliser deux fonctions objectifs c'est-à-dire transformer notre problème de sac-à-dos multiple stochastique multi-objectifs MO-SMKP en un problème de sac-à-dos multiple stochastique bi-objectifs BO-SMKP comme suit :

$$\begin{aligned} &J_2(x, \chi) = \lambda_1 f_1(x) + \lambda_2 f_2(x) \\ &\lambda_2 = 1 - \lambda_1 \end{aligned} \tag{4.7.1}$$

Afin de représenter graphiquement le front de Pareto du BO- SMKP, nous varions la valeur de λ_1 de $0,1$.

Pour chaque exemple, tous les algorithmes sont exécutés 10 fois, et à chaque exécution les résultats sont sauvegardés et comparés par rapport au front Pareto obtenu ainsi que l'évaluation des solutions relative aux fonctions objectifs.

Les meilleurs résultats de nos instances pour chaque exemple sont représentés graphiquement sur les figures ci-dessous.

Example 4.1.1

- Nombre d'articles est : 20.
- Nombre de sacs-à-dos est : 2.

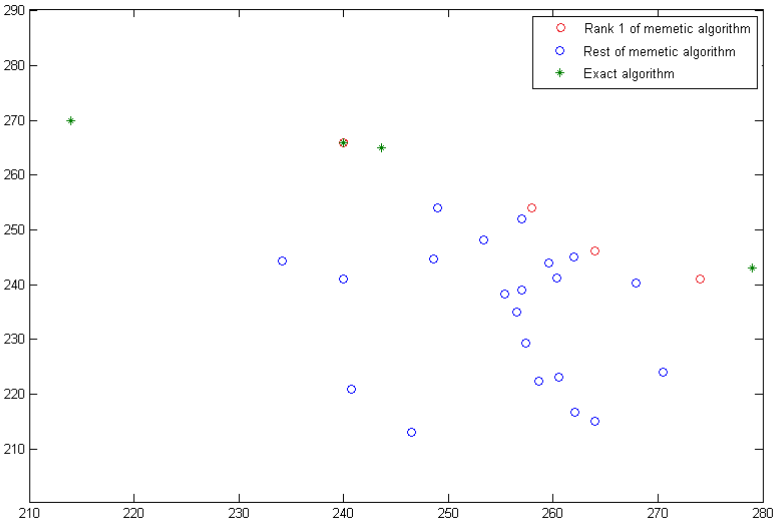

FIGURE 4.1 – Représentation graphique des résultats de ca comparaison du front de pareto obtenu à partir d'un algorithme exact et de l'algorithme MASNPL pour le cas de 20 articles et 2 sacs-à-dos

.

La figure 4.1 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme Memetique Avec MTHMn). Le temps nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 1.4 Secondes quant au temps total d'exécution de l'algorithme Memetique Avec MTHMn est : 2.6749745 Secondes. Selon les résultats graphiques, en comparant le nuage de front Pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité dans un temps d'exécu- tion raisonnable.

Example 4.1.2

- Nombre d'articles est : 25.
- Nombre de sacs-à-dos est : 2.

La figure 4.2 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme memetique avec MTHMn). Le temps nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 2.41 Secondes quant au temps total d'exécution de l'algorithme memetique avec mTHMn est : 2.4735703 secondes. Selon les résultats graphiques, en comparant le nuage de front

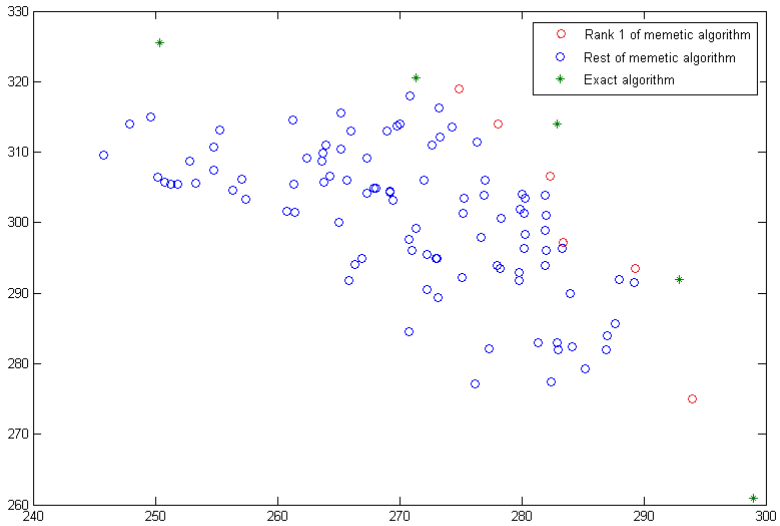

FIGURE 4.2 – Représentation graphique des résultats de la comparaison du front de pareto obtenu à partir d'un algorithme exact et de l'algorithme MASNPL dans le cas de 25 articles et 2 sacs-à-dos

.

pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité dans un temps d'exécution raisonnable.

Example 4.1.3

- Nombre d'articles est : 25.
- Nombre de sacs-à-dos est : 3.

La figure 4.3 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme memetique avec MTHMn). Le temps nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 3.54 Secondes quant au temps total d'exécution de l'algorithme Memetique Avec MTHMn est : 2.6166466 Secondes. Selon les résultats graphiques, en comparant le nuage de front Pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité dans un temps d'exéc- ution raisonnable.

Example 4.1.4

- Nombre d'articles est : 25.
- Nombre de sacs-à-dos est : 4.

La figure 4.4 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme memetique avec MTHMn). Le temps

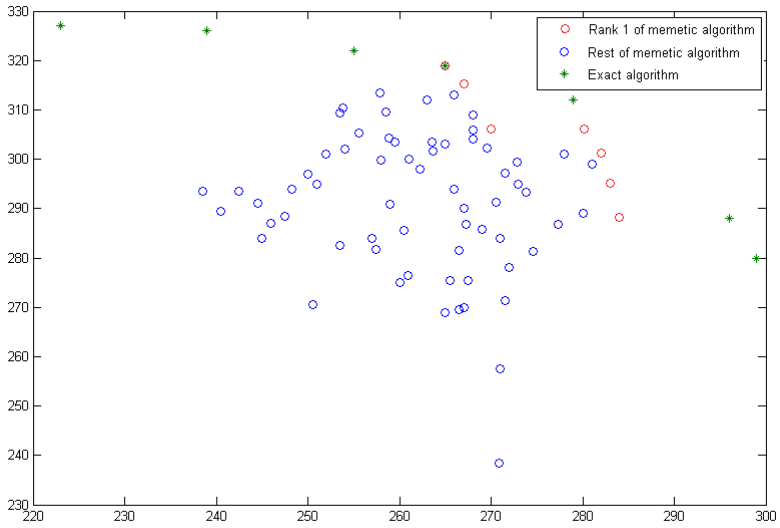

FIGURE 4.3 – Représentation graphique des résultats de la comparaison du front de pareto obtenu à partir d'un algorithme exact et de l'algorithme MASNPL dans le cas de 25 articles et 3 sacs-à-dos

.

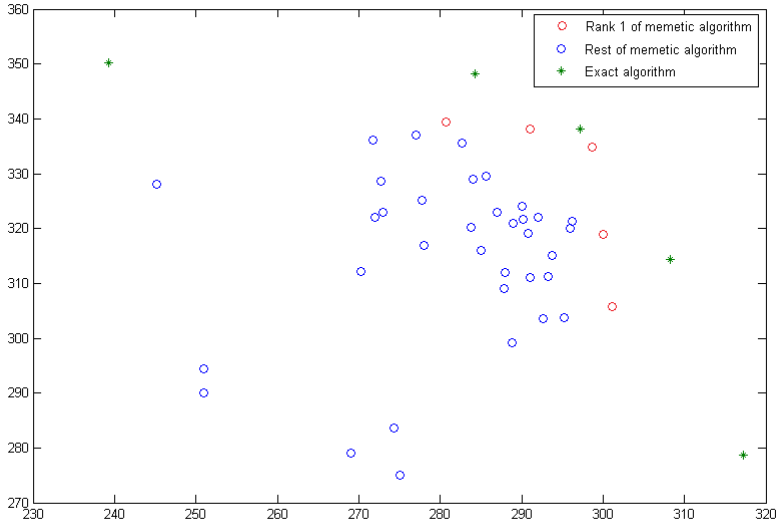

FIGURE 4.4 – Représentation graphique des résultats de la comparaison du front de pareto obtenu à partir d'un algorithme exact et de l'algorithme MASNPL pour le cas de 25 articles et 4 sacs-à-dos

.

nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 117 Secondes quant au temps total d'exécution de l'algorithme memetique avec MTHMn est : 2.8149322 secondes. Selon les résultats graphiques, en comparant le nuage de front

Pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme Memetique Avec MTHMn donne des solutions de haute qualité dans un temps d'exécution raisonnable.

Example 4.1.5

- Nombre d'articles est : 30.
- Nombre de sacs-à-dos est : 2.

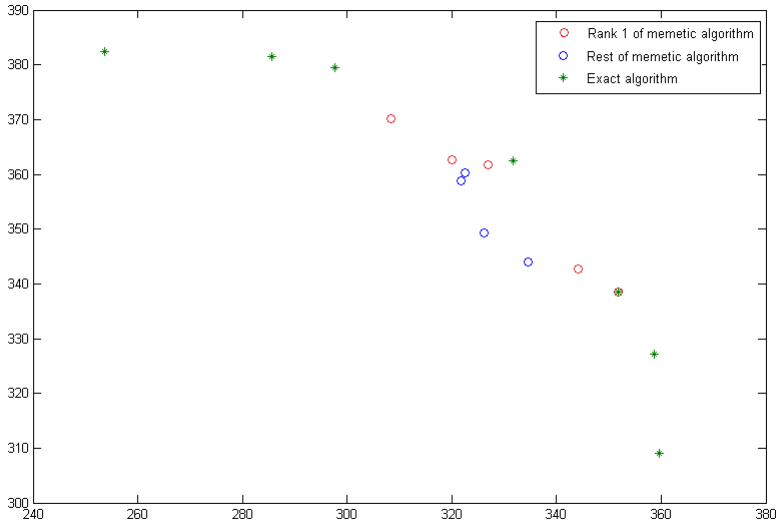

FIGURE 4.5 – Représentation graphique des résultats de la comparaison du front de pareto btenu à partir d'un algorithme exact et de l'algorithme MASNPL pour le cas de 30 articles et 2 sacs-à-dos

.

La figure 4.5 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme memetique avec MTHMn). Le temps nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 2.39 Secondes quant au temps total d'exécution de l'algorithme Memetique Avec MTHMn est : 2.6636418 Secondes. Selon les résultats graphiques, en comparant le nuage de front Pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité dans un temps d'exéc- ution raisonnable.

Example 4.1.6

- Nombre d'articles est : 30.
- Nombre de sacs-à-dos est : 3.

La figure 4.6 montre les solutions non-dominées obtenues en appliquant la méthode exacte et notre méthode de résolution (algorithme memetique avec MTHMn). Le temps

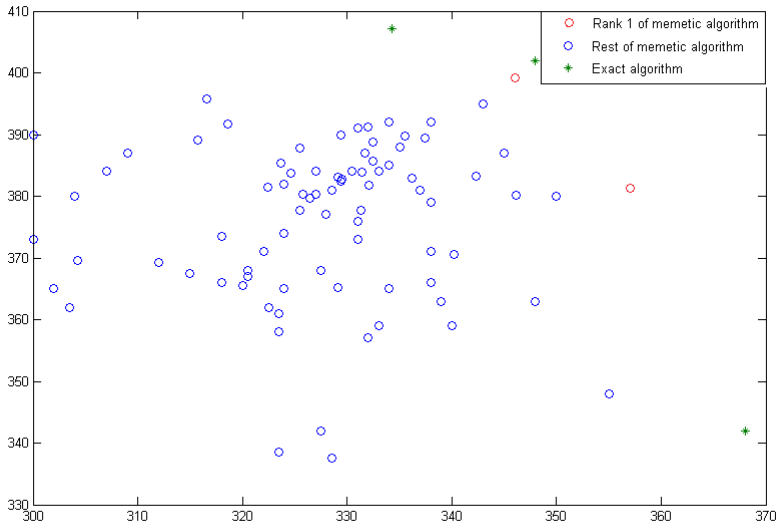

FIGURE 4.6 – Représentation graphique des résultats de la comparaison du front de pareto obtenu à partir d'un algorithme exact et de l'algorithme MASNPL pour le cas de 30 articles et 3 sacs-à-Dds

.

nécessaire d'exécution de la méthode exacte pour obtenir la solution finale est : 1.61 Secondes quant au temps total d'exécution de l'algorithme Memetique Avec MTHMn est : 3.0050965 Secondes. Selon les résultats graphiques, en comparant le nuage de front Pareto relatif à chaque méthode de résolution, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité dans un temps d'exéc- ution raisonnable.

4.7.2 Comparaison entre l'heuristique MAMTHMn et l'heuristique MTHM

Dans le but de comparer entre l'heuristique MTHMn et l'heuristique MTHM, nous appliquons notre algorithme memetique d'une part avec l'heuristique MTHMn et d'autre part avec l'heuristique MTHM sur les mêmes instances initialement créées selon les paramètres présentés ci-dessus. Les résultats de nos tests sont présentés ci-dessous.

Nos algorithmes sont implémentés sur MATLAB

Tableau 4.1 Les résultats de nos tests sont représentés sous forme de tableau, qui représente le nombre total des individus dans chaque niveau de non-domination ($\mathcal{F}_i$), en appliquant l'algorithme Memetique avec MTHMn (NG) fois.
Avec :

- n : Représente le nombre d'articles.
- S : Représente le nombre de sacs-à-dos.

- NG : Représente le nombre de générations.
- N : Représente la taille de la population.
- F_i : Indique le niveau de non-domination.
- CPU : Représente le temps d'exécution total nécessaire pour obtenir la population finale pour chaque exemple.

n	S	penalty (d)	(NG)	(N)	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	CPU (s)
50	3	1,5	50	50	50			6.0538
80	3	1,5	50	50	50			10.845
100	3	1,5	50	50	47	3		14.7158
200	3	1,5	50	50	37	13		45.87
300	3	1,5	50	50	28	22		99.623
500	3	1,5	50	50	14	22	14	272.09
50	5	1,5	50	50	50			7.0605
80	5	1,5	50	50	50			11.115
100	5	1,5	50	50	49	1		14.958
200	5	1,5	50	50	22	27	1	47.497
300	5	1,5	50	50	35	15		106.82
500	5	1,5	50	50	25	24	1	254.07

TABLE 4.1 – Représente le nombre d'individus dans la population finale de taille N pour chaque niveau de non-domination pour chaque nombre d'articles n, nombre de sacs-à-dos S en appliquant l'algorithme memetique avec MTHMn.

Table 4.2 Les résultats de nos tests sont représentés comme le nombre total d'individus pour chaque niveau de non-domination ($\mathcal{F}_i$), en appliquant l'algorithme Memetique avec MTHM (NG) fois.

n	S	penalty (d)	(NG)	(N)	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	CPU (s)
50	3	1.5	50	50	7	30	9	4	7.0532
80	3	1.5	50	50	43	7	0	0	12.644
100	3	1.5	50	50	24	23	3	0	17.839
200	3	1.5	50	50	30	17	3	0	56.828
300	3	1.5	50	50	16	20	14	0	125.01
500	3	1.5	50	50	50	0	0	0	828.3
50	5	1.5	50	50	31	19	0	0	7.4211
80	5	1.5	50	50	11	26	13	0	12.969
100	5	1.5	50	50	22	22	6	0	18.978
200	5	1.5	50	50	8	30	12	0	60.328
300	5	1.5	50	50	12	15	15	8	139.32
500	5	1.5	50	50	5	11	18	16	361.91

TABLE 4.2 – Représente le nombre d'individus dans chaque niveau de non-domination pour chaque nombre d'articles n, nombre de sacs-à-dos S en appliquant l'algorithme memetique avec MTHM.

En comparant les résultats obtenus pour chaque exemple en appliquant l'algorithme memetique avec les heuristiques (MTHMn et MTHM), nous remarquons que :

- Les solutions obtenues sont variées en appliquant l'heuristique MTHMn et de même pour le nombre d'individus appartenant à chaque front non-dominé est meilleurs en appliquant cette heuristique.
- Le nombre d'individu appartenant au front non-dominé 1 est beaucoup plus important en appliquant l'heuristique MTHMn.
- Lorsque nous comparons le temps d'exécution relatif à chaque heuristique et pour chaque exemple, nous concluons que l'algorithme memetique avec MTHMn est plus rapide.

En se basant sur les résultats de comparaison entre les deux heuristiques, nous concluons que l'algorithme memetique avec MTHMn donne des solutions de haute qualité et dans un temps d'exécution plus rapide.

4.8 Conclusion

Nous avons détaillé le modèle du problème de sac-à-dos multiple stochastique multi-objectifs MO-SMKP avec recours (sac-à-dos à deux étapes) à poids aléatoire et profit fixe. Nous introduisons une contrainte aléatoire dans la première étape afin de réduire le pourcentage de cas où les articles placés lors de la première étape ne dépassent pas la capacité du sac-à-dos, pour résoudre ce problème nous appliquons l'algorithme Memetic.

Initialement, nous construisons une population initiale réalisable de taille N en appliquons l'heuristique constructive gloutonne pour MO-SMKP. Ensuite, nous trions les individus en différents niveaux de non-domination et nous leurs attribuons une valeur de crowding-distance en appliquant l'algorithmes de crowding-distance.

Afin d'améliorer la qualité de ces solutions, nous appliquons deux heuristiques différentes : la première est l'heuristique MTHM et la seconde est l'heuristique MTHMn dont nous faisons une comparaison entre ces heuristiques, puis, une série de mutations, de croise- ments et de recherche locale sont appliquées à cette population pour générer des enfants.

Dans le but d'améliorer la qualité des progénitures, nous appliquons à ces dernières les heuristiques MTHM et MTHMn ainsi que l'algorithme de tri non-dominé et l'algorithme calculant la valeur de crowding-distance qui sont de même appliqués à la progéniture améliorée afin de sélectionner les individus situés dans la région la plus dense et qui ont une meilleur répartition sur le front Pareto optimal.

Enfin, nos résultats numériques relatifs à la comparaison entre l'algorithme Memetic en utilisant l'Heuristique MTHMn est significativement plus performant et efficace qu'en utilisant l'Heuristique MTHM.

CONCLUSION ET PERSPECTIVES

Dans cette thèse, nous avons étudié une classe des problèmes d'optimisation combinatoire, dont nous nous concentrons principalement sur les différentes variantes du problème de sac-à-dos qui fait l'objet de nombreuses études dans la littérature.

Le problème de sac-à-dos appartenant à la classe de problèmes NP-difficiles, par conséquent, les heuristiques et les métaheuristiques sont les moyens les plus efficaces pour résoudre de tels problèmes dans les plus courts délais.

Ces problèmes possèdent de nombreuses applications dans l'industrie, la planification financière, la gestion de portefeuille, le transport ainsi que d'autres problématiques liées au quotidien, ce qui pousse les chercheurs à développer de manière continue de nouvelles méthodes qui donnent des solutions de bonnes qualités en un temps raisonnable.

Notre contribution est présentée dans les chapitres 3 et 4.

Dans le troisième chapitre nous avons initialement défini le problème de sac-à-dos Quadratique Stochastique Multi-objectifs avec recours et à poids aléatoires, puis nous l'avons formulé mathématiquement, nous avons déterminé le problème déterministe équivalent afin de pouvoir résoudre notre problème en utilisant une méthode exacte car il n'a pas été traité auparavant par conséquent l'absence de benchmarks.

Pour résoudre ce problème, nous avons utilisé un algorithme hybride basé sur l'algorithme génétique combiné avec l'algorithme de gradient qui a pour but d'améliorer la qualité des solutions trouvées.

Initialement, nous appliquons l'heuristique constructive gloutonne afin de déterminer une population initiale réalisable de taille N, dont nous lui appliquons respectivement les algorithmes de tri non-dominé ainsi que crowding-distance afin d'attribuer des valeurs (i_{rank}), $(i_{distance})$ pour chaque individu de la population. Au résultat, nous aurons une population initiale réalisable trié selon le niveau de non-domination, avec une valeur de crowding distance à chaque individu.

Dans le but d'améliorer la qualité de cette population et avoir des solutions efficaces, nous appliquons une série de mutations, de croisements et de recherches locales à cette population pour générer une population d'enfant dont nous appliquons à cette dernière l'algorithme SNPLS basé sur la comparaison entre une solution actuelle (enfant) et une nouvelle solution obtenue par l'algorithme du gradient. Par la suite, l'algorithme de tri non-dominé et l'algorithme de crowding-distance sont appliqués à la

progéniture améliorée pour sélectionner nos premiers meilleurs individus finaux de la population.

Un panel d'expériences numériques est présenté et analysé, où nous comparons les performances de notre algorithme hybride avec une méthode exacte ainsi qu'à NSGA-II. En se basant sur les résultats numériques, nous pouvons conclure que l'utilisation de l'algorithme de gradient avec la NSGA-II donne des résultats nettement meilleurs et plus efficaces que la NSGA-II.

Par contre, dans le quatrième chapitre, nous avons défini le problème de sac-à-dos multiple stochastique multi-objectifs à deux étapes et à poids aléatoires, puis nous l'avons formulé mathématiquement, dont nous avons introduit une contrainte de prob- abilité afin de réduire le pourcentage d'avoir une surcharge lors de la deuxième étape et après avoir révélé les poids exacts des articles, et de minimiser les coûts supplémentaires (payement d'une pénalité d_i pour chaque article retiré).

Pour résoudre ce problème, nous avons utilisé un algorithme memetique basé sur l'algorithme génétique combiné avec les heuristiques MTHM et MTHMn qui a pour but d'améliorer la qualité des solutions trouvées.

Initialement, nous appliquons l'heuristique constructive gloutonne afin de construire une population initiale réalisable de taille N, dont nous trions les individus de cette population en différents niveaux de non-domination en appliquant l'algorithme de tri non-dominé, et nous leur attribuons une valeur de crowding-distance en appliquant l'algorithme de crowding-distance.

Dans le but d'améliorer la qualité des solutions trouvées, nous leurs appliquons d'un coté, l'heuristique MTHMn, et d'un autre, l'heuristique MTHM, dont nous comparons les résultats obtenus en appliquant chaque heuristique. Puis une série de sélection, de mutations, de croisements et de recherche locale sont appliquées à cette population pour générer des enfants.

De même, nous appliquons les heuristiques MTHM et MTHMn afin d'améliorer la qualité des enfants, nous attribuons des valeurs (i_{rank}) et $(i_{distance})$ à chaque individu de la population améliorée (enfant) afin de sélectionner les individus qui ont une meilleur répartition sur le front Pareto optimal.

Enfin, nos résultats numériques relatifs à la comparaison entre l'algorithme Memetique en utilisant l'Heuristique MTHMn est significativement plus performant et efficace qu'en utilisant l'Heuristique MTHM.

Cette étude se concentre sur le problème de sac-à-dos, en particulier le problème de sac-à-dos quadratique stochastique multi-objectifs et le problème de sac-à-dos multiple stochastique multi-objectifs. Les méthodes de résolution présentées dans cette thèse ont été rigoureusement testées et ont démontré leur efficacité et leur rapidité, offrant

ainsi de nouvelles perspectives pour l'utilisation d'algorithmes hybrides et Memetiques, avec une adaptation adéquate, afin de résoudre d'autres problèmes d'optimisation combinatoire, tels que le Problème de Bin Packing et le Problème d'Ordonnancement.

Les résultats obtenus avec les algorithmes hybrides et Memetiques utilisant d'autres heuristiques et algorithmes, dont le but est d'améliorer la qualité des solutions existantes, nous encouragent à approfondir davantage nos recherches et à explorer d'autres algori- thmes et approches pour résoudre de tels problèmes, ainsi que d'autres problèmes nécessitant une adaptation adéquate. À titre d'exemple, l'algorithme stochastique Arrow-Hurwicz peut donner de bons résultats pour les problèmes stochastiques, tout comme l'approche de recherche itérative à seuil réactif (IRTS) qui a donné de bons résultats pour résoudre le Problème de Sac-à-dos Quadratique Multiple.

Bibliographie

[1] Aïder, M., Gacem, O., & Hifi, M. (2022a). Branch and solve strategies-based algorithm for the quadratic multiple knapsack problem. Journal of the Operational Research Society, 1–18. doi :10.1080/01605682.2020.184398

[2] Aïder, M., GACEM, O. & HIFI, M. (2022b). A hybrid population-based algorithm for the bi-objective quadratic multiple knapsack problem. Expert Systems With Applications, 191 :116238.

[3] Andrieu, L., Cohen, G. & V´azquez-Abad, F. 2007. Stochastic programming with probability constraints. http ://fr.arxiv.org/abs/0708.0281.

[4] Arshad, S., Yang, S. & Li, C. 2009. A sequence based genetic algorithm with local search for the travelling salesman problem. *Proceedings of the 2009 UK Workshop on Computational Intelligence*, 98–105.

[5] Balas, E. & Zemel, E. (1980). An Algorithm for Large Zero-One Knapsack Problems. Operations Research, 28(5), 1130–1154. doi :10.1287/opre.28.5.1130 .

[6] Beale, E. M. L. (1955). On minimizing a convex function subject to linear inequalities. J. Roy. Statist. Soc., 17.

[7] Bhuvana, J. & Aravindan, C. (2015). Memetic algorithm with Preferential Local Search using adaptive weights for multi-objective optimization problems. Soft Computing, 20(4), 1365 1388. doi :10.1007/s00500-015-1593-9.

[8] Blado, D. & Toriello, A. (2020). A column and constraint generation algorithm for the dynamic knapsack problem with stochastic item sizes. Mathematical Programming Computation. doi :10.1007/s12532-020-00189-0.

[9] Cacciani, V., Idri, M. & Martello, A. L. S. (2022a). Knapsack problems - An overview of recent advances. Part I : Single knapsack problems. Computers & Operations Research, 143 : 1056922

[10] Cacciani, V., Idri, M. & Martello, A. L. S. (2022b). Knapsack problems - An overview of recent advances. Part II : Multiple, multidimensional, and quadratic knapsack problem. Computers & Operations Research, 143 : 1056923.

[11] Charnes, A., Cooper, W. W. & Symonds, G. H. (1958). Cost Horizons and Certainty Equivalents : An Approach to Stochastic Programming of Heating Oil. Management Science, 4(3), 235–263. doi :10.2307/2627328

[12] Chen, Y., & Hao, J.-K. (2014). Iterated responsive threshold search for the quadratic multiple knapsack problem. Annals of Operations Research, 226(1), 101–131. doi :10.1007/s10479-014-1720-5.

[13] Chen, Y., & Hao, J.-K. (2016). The bi-objective quadratic multiple knapsack problem : Model and heuristics. Knowledge-Based Systems, 97, 89–100. doi :10.1016/j.knosys.2016.01.014§§.

[14] Chu X & Yu X. 2018. Improved Crowding Distance for NSGA-II. https ://arxiv.org/abs/1811.12667v1.

[15] Cohn, A. & Barnhart, C. (1998). The stochastic knapsack problem with random weights : A heuristic approach to robust transportation planning. In Proceedings of the Triennial Symposium on Transportation Analysis (TRISTAN III).

[16] Dantzig, G. B. (1957). Discrete-Variable Extremum Problems. Operations Research, 5(2), 266–288. doi :10.1287/opre.5.2.266.

[17] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182–197. doi :10.1109/4235.996017

[18] Elkihel, M., Authiè, G. & Viader, F. (2002). An efficient hybrid dynamic algorithm to solve 0-1 knapsack problems, International Symposium on Combinatorial Optimization (CO'2002), Paris (France).

[19] Fayard, D., & Plateau, G. (1982). An algorithm for the solution of the 0–1 knapsack problem. Computing, 28(3), 269–287. doi :10.1007/bf02241754

[20] Gallo, G., Hammer, P. L., & Simeone, B. (1980). Quadratic knapsack problems. Combinatorial Optimization, 132–149. doi :10.1007/bfb0120892

[21] Gilmore, P. C., & Gomory, R. E. (1961). A Linear Programming Approach to the Cutting-Stock Problem. Operations Research, 9(6), 849–859. doi :10.1287/opre.9.6.849

[22] Guerrouma, A. and Aïder, M. (2022). A hybridized multi-objective memetic algorithm for the multi-objective stochastic quadratic knapsack problem. https ://doi.org/10.1590/0101-7438.2022.042.00257386 (doi : 10.1590/0101-7438.2022.042.00257386)

[23] Guerrouma, A. and Aïder, M. An Evolutionary Algorithm For The Multi-Objective Stochastic Multiple Knapsack Problem. (En preparation)

[24] Hifi, M., & Roucairol, C. (2001). Approximate and Exact Algorithms for Constrained (Un) Weighted Two-dimensional Two-staged Cutting Stock Problems. Journal of Combinatorial Optimization, 5(4), 465–494. doi :10.1023/a :1011628809603.

[25] Hiley, A., & Julstrom, B. A. (2006). The quadratic multiple knapsack problem and three heuristic approaches to it. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO '06. doi :10.1145/1143997.1144096.

[26] Horowitz, E., & Sahni, S. (1974). Computing Partitions with Applications to the Knapsack Problem. Journal of the ACM, 21(2), 277–292. doi :10.1145/321812.321823

[27] Kellerer, H., Pferschy, U. & Pisinger, D. (2003). Knapsack Problems. Springer.

[28] Kim, H. & Liou, M.-S. (2012). New fitness sharing approach for multi-objective genetic algorithms. Journal of Global Optimization, 55(3), 579-595. doi :10.1007/s10898-012-9966-4

[29] Klopfenstein.O. & Nace, D. (2008). A robust approach to the chance-constrained knapsack problem. Operations Research Letters, 36(5), 628–632. doi :10.1016/j.orl.2008.03.006.

[30] Kosuch, S., Letournel, M., & Lisser, A. (2017). STOCHASTIC KNAPSACK PROBLEM : APPLICATION TO TRANSPORTATION PROBLEMS. Pesquisa Operacional, 37(3), 597–613. doi :10.1590/0101-7438.2017.037.03.0597.

[31] Kosuch, S., & Lisser, A. (2009). Upper bounds for the 0-1 stochastic knapsack problem and a B&B algorithm. Annals of Operations Research, 176(1), 77–93. doi :10.1007/s10479-009-0577-5.

[32] Kosuch, S., & Lisser, A. (2011). On two-stage stochastic knapsack problems. Discrete Applied Mathematics, 159(16), 1827–1841. doi :10.1016/j.dam.2010.04.006.

[33] Kosuch, S. (2014). Approximability of the two-stage stochastic knapsack problem with discretely distributed weights. Discrete Applied Mathematics, 165(), 192–204. doi :10.1016/j.dam.2013.02.015.

[34] Kosuch, S., Letournel, M., & Lisser, A. (2017). STOCHASTIC KNAPSACK PROBLEM : APPLICATION TO TRANSPORTATION PROBLEMS. Pesquisa Operacional, 37(3), 597–613. doi :10.1590/0101-7438.2017.037.03.0597.

[35] Laalaoui, Y., & Mhalla, H. (2015). Efficient heuristic for very large 0/1 multiple knapsack problems. International Journal of Computational Systems Engineering, 2(2), 112. doi :10.1504/ijcsyse.2015.077065

[36] Létocart, L., Plateau, M. C. & Plateau, G. (2014). An efficient hybrid heuristic method for the 0-1 exact k-item quadratic knapsack problem. Pesquisa Operacional, 34(1) : 49–72.

[37] Lisser, A., & Lopez, R. (2010). Stochastic Quadratic Knapsack with Recourse. Electronic Notes in Discrete Mathematics, 36, 97–104. doi :10.1016/j.endm.2010.05.013.

[38] Martello, S., & Toth, P. (1981). Heuristic algorithms for the multiple knapsack problem. Computing, 27(2), 93–112. doi :10.1007/bf02243544.

[39] Martello, S., & Toth, P. (1988). A New Algorithm for the 0-1 Knapsack Problem. Management Science, 34(5), 633–644. doi :10.1287/mnsc.34.5.633.

[40] Martello, S., & Toth, P. (1990). Knapsack Problems : Algorithms and Computer Implementations. John Wiley & Sons Ltd.

[41] Mei, Y., Tang, K. & Yao, X. (2011). Decomposition-Based Memetic Algorithm for Multiobjective Capacitated Arc Routing Problem. IEEE Transactions on Evolutionary Computation, 15(2), 151–165. doi :10.1109/tevc.2010.2051446.

[42] Pisinger, D. (2007). The quadratic knapsack problem—a survey. Discrete Applied Mathematics, 155(5), 623 648. doi :10.1016/j.dam.2006.08.007.

[43] Range, T. M., Kozlowski, D., & Petersen, N. C. (2018). A shortest-path-based approach for the stochastic knapsack problem with non-decreasing expected overfilling costs. Computers & Operations Research, 97, 111–124. doi :10.1016/j.cor.2018.04.013

[44] Shang, R., Wang, J., Jiao, L., & Wang, Y. (2014). An Improved Decomposition-Based Memetic Algorithm for Multi-Objective Capacitated Arc Routing Problem. Applied Soft Computing, 19, 343–361. doi :10.1016/j.asoc.2014.03.005.

[45] Song, B., Li, Y., Chen, Y., Yao, F., & Chen, Y. (2018). A Repair-based approach for stochastic quadratic multiple knapsack problem. Knowledge-Based Systems, 145, 145–155. doi :10.1016/j.knosys.2018.01.012.

[46] Tang, K., Mei, Y. & Yao, X. (2009). Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. IEEE Transactions on Evolutionary Computation, 13(5), 1151–1166. doi :10.1109/tevc.2009.2023449

[47] Tönissen, D. D., van den Akker, J. M., & Hoogeveen, J. A. (2017). Column generation strategies and decomposition approaches for the two-stage stochastic multiple knapsack problem. Computers & Operations Research, 83, 125–139. doi :10.1016/j.cor.2017.02.009

[48] Tönissen, D. & Schlicher, L. (2021). Using 3D-printing in disaster response : The two-stage stochastic 3D-printing knapsack problem. Computers & Operations Research, 133, 105356. doi :10.1016/j.cor.2021.105356.

[49] Zhang, Q. & Li, H. (2007). MOEA/D : A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation, 11(6), 712–731. doi :10.1109/tevc.2007.892759.

[50] Zhou, Q., Hao, J.-K., & Wu, Q. (2022). A hybrid evolutionary search for the generalized quadratic multiple knapsack problem. European Journal of Operational Research. doi :10.1016/j.ejor.2021.04.001