N^o d'ordre : **207/2024-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 des mathématiques

Thèse de Doctorat en Mathématiques
Présentée pour l'obtention du grade de Docteur

Spécialité : Recherche opérationnelle et Management (ROM)

Par :
LACHEMI Nadia

Thème

L'optimisation sur l'ensemble de solutions Pareto-Optimal pour les problèmes MOCO

Soutenue publiquement le 09/07/2024, devant le jury composé de :

M. AIDER Méziane	Professeur	à l'USTHB	Président.
M. CHAABANE Djamal	Professeur	à l'USTHB	Directeur de thèse.
M. BOUROUBI Sadek	Professeur	à l'USTHB	Examinateur.
Mme. DAHMANI Isma	Maître de conférence/A	à l'USTHB	Examinatrice.
Mme. FRIKHA Hela Moalla	Professeur	à l'HBS. Sfax, Tunisie	Examinatrice.

Remerciements

Je remercie avant tout le bon Dieu pour m'avoir donné la force d'accomplir ce travail et de le mener à bien.

Je tiens particulièrement à exprimer ma sincère reconnaissance et mes chaleureux remerciements à mon directeur de thèse, Monsieur CHAABANE Djamal, professeur à l'USTHB. Je lui suis reconnaissante pour le sujet passionnant qu'il m'a proposé, ainsi que pour son encadrement et ses conseils précieux tout au long de ce parcours. Qu'il trouve ici l'expression de mes plus profonds respects.

Mes vifs remerciements sont également adressés à Monsieur AIDER Meziane, professeur à l'USTHB, pour avoir accepté de présider le jury de ce travail, un honneur pour lequel je lui suis profondément reconnaissante.

Je souhaite exprimer ma profonde gratitude à Monsieur BOUROUBI Sadek, professeur à l'USTHB, à Madame DAHMANI Isma, maître de conférences de classe A à l'USTHB, ainsi qu'à Madame FRIKHA Hela Moalla, professeur à l'HSB de Sfax, Tunisie, pour avoir accepté de consacrer leur temps et leur expertise à l'évaluation de ce travail. Leur contribution est inestimable, et je leur exprime ici ma profonde reconnaissance.

Mes remerciements les plus sincères sont adressés à ma famille : en premier lieu, à mes parents et à ma belle-mère, pour leur soutien constant et leurs encouragements. A mon époux et à mes filles, Ikram et Cirine, pour leur soutien indéfectible au quotidien et leur enthousiasme inspirant. Je remercie aussi mes frères, mes sœurs, mes belles-sœurs et tous les membres de ma famille pour leur soutien précieux tout au long de ce parcours.

Enfin, je remercie chaleureusement mes collègues et mes amies, particulièrement Meriem et Hayet. Leur soutien inébranlable a été une force transcendante à travers les moments de doute et de difficulté, je leur suis infiniment redevable.

Dédicaces

Je dédie ce travail :

À mes chers parents, frères et sœurs.

À ma chère belle-mère, mes belles-sœurs et mes beaux-frères.

À mon cher époux ainsi qu'à mes adorables filles.

A toute ma famille.

À mes collègues et à mes chères amies.

À tous mes enseignants du cycle universitaire.

À tous ceux qui aiment Nadia et à tous ceux que Nadia aime.

Nadia

مُلَخَّص

تتطلب معظم المشكلات التي تتم مواجهتها في الواقع اتخاذ قرارات بناءً على عدة أهداف، غالبًا ما تكون متناقضة. والواقع أن الهدف هو إيجاد أفضل الحلول التي تسمى الحلول الفعالة. تم نمذجة هذه المشكلات على شكل مشكلات متعددة الأهداف MOP. في هذه الأطروحة نهتم بمشكلة التحسين الخطي متعدد الأهداف MOLP. يمكن أن تكون مجموعة الحلول الفعالة لـ MOLP كبيرة جدًا بشكل عام، لا نهائية بشكل خاص في الحالة المستمرة، بينما في بعض المواقف لا يحتاج متخذ القرار إلى جميع الحلول الفعالة، ولكن فقط تلك التي تحقق المستوى الأمثل لهدف آخر يسمى الهدف الرئيسي. وهذا يقودنا إلى التحسين على مجموعة الحلول الفعالة لـ MOLP. قد تم اقتراح عدة طرق لحل المشكلات لكل من المتغيرات المستمرة والصحيحة. في هذه الأطروحة، نتعامل مع مشكلة تحسين الدالة الخطية على المجموعة الفعالة لمشكلة الحقيبة ثنائية الأهداف BOKP. تعتمد عملية الحل بشكل أساسي على البرمجة الديناميكية. توفر الطريقة المقترحة مجموعة فرعية من الحلول الفعالة، بما في ذلك الحل الذي يحقق المستوى الأمثل للهدف الرئيسي دون الحاجة إلى حساب جميع الحلول الفعالة للمشكلة. تم انجاز دراسة تجريبية، حيث تم النظر في أمثلة مختلفة ذات مجموعات فعالة كبيرة الحجم لإظهار فعالية الخوارزمية المقترحة مقارنة بخوارزمية مقترحة سابقا.

الكلمات المفتاحية : البرمجة متعددة الأهداف، مسألة الحقيبة ثنائية الأهداف، البرمجة الديناميكية، مجموعة الحلول الفعالة، الحل الأمثل.

Abstract

Most real-life problems require decisions involving several, often conflicting objectives. Indeed, the aim is to find the best compromise solutions, known as efficient solutions. These problems are modeled as multiple objective problems MOP. In this thesis, we focus on multiple objective linear optimization problems MOLP. The efficient set of a MOLP can generally be very large, notably infinite in the continuous case, while in some situations the decision-maker does not need all the efficient solutions, but only those that reach the optimum of another objective called the main criterion. This leads to optimizing over the efficient set of a MOLP. Several methods have been proposed for both continuous and discrete cases. In this thesis, we address the problem of optimizing a linear function over the efficient set of a bi-objective knapsack problem in binary variables BOKP. The solution process is essentially based on dynamic programming. The proposed method provides a subset of efficient solutions, including a solution that optimizes the main criterion without having to enumerate all the efficient solutions of the problem. An experimental study is reported, different instances with large sizes of the efficient sets are considered to show the effectiveness of our algorithm compared with an approach proposed in the literature.

Keywords : Multiple objective programming, Bi-objective knapsack problem, Dynamic programming, Efficient set, Optimal solution.

Résumé

La plupart des problèmes rencontrés dans la vie réelle imposent des prises de décisions en fonction de plusieurs objectifs, souvent contradictoires. En effet, le but est de trouver des solutions de meilleur compromis dites solutions efficaces. Ces problèmes se modélisent sous forme de problèmes multi-objectif MOP. Dans cette thèse on s'intéresse au problèmes d'optimisation linéaire multi-objectif MOLP. L'ensemble efficace d'un MOLP peut généralement être très vaste, à savoir infini dans le cas continu, alors que dans certaines situations, le décideur n'a pas besoin de toutes les solutions efficaces, mais seulement de celles qui atteignent l'optimum d'un autre objectif appelé objectif principal. Cela nous conduit à l'optimisation sur l'ensemble des solutions efficaces d'un MOLP. Plusieurs méthodes ont été proposées tant pour le cas continu que pour le cas discret. Dans cette thèse, nous traitons le problème de l'optimisation d'une fonction linéaire sur l'ensemble efficace d'un problème de sac à dos bi-objectif en variables binaires BOKP. Le processus de résolution est essentiellement basé sur la programmation dynamique. La méthode proposée fournit un sous-ensemble de solutions efficaces, y compris une solution qui optimise le critère principal sans avoir à énumérer toutes les solutions efficaces. Une étude expérimentale est menée sur différentes instances ayant des ensembles efficaces de tailles considérables, afin de démontrer l'efficacité de notre méthode par rapport à une approche proposée dans la littérature.

Mots clés : Programmation Multi-objectif, Problème de sac à dos bi-objectif, Programmation dynamique, Ensemble efficace, Solution optimale.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction générale

De nombreuses situations rencontrées dans le monde réel nécessitent des prises de décisions par un ou plusieurs décideurs. Ces décisions impliquent généralement plusieurs objectifs conflictuels. Le constat que les problèmes du monde réel doivent être résolus de manière optimale selon des critères qui excluent la possibilité d'obtenir une solution idéale "optimale pour chacun des critères considérés" a conduit au développement de l'optimisation multi-objectif MOP. Depuis ses premières racines, posées par Pareto à la fin du $19^{ème}$ siècle [124], cette discipline a prospéré et s'est développée, en particulier au cours des cinq dernières décennies. Aujourd'hui, de nombreuses méthodes sont proposées pour traiter les problèmes à objectifs multiples dans le but d'obtenir un ensemble ou sous-ensemble de solutions appelé ensemble non-dominé, ces solutions sont celles pour lesquelles l'amélioration d'un des objectifs entraîne systématiquement la détérioration de la qualité d'au moins un autre objectif.

La plupart des problèmes d'optimisation multi-ojectif peuvent être modéliser sous forme de problèmes d'optimisation multi-objectif en variables binaires [49], on parle alors de l'optimisation combinatoire multi-objectif. De manière formelle, un problème d'optimisation multi-objectif en variables binaires est un problème de la forme "opt" : $F(x) = (f_1(x), f_2(x), \ldots, f_p(x))$ tel que $x \in \{0,1\}^n$ soit une solution réalisable. Pour le cas mono-objectif, un grand nombre d'études ont été menées depuis plus d'un siècle, cependant, l'intérêt pour le cas multi-objectif est relativement récent (principalement depuis les années 1990). Les problèmes d'optimisation combinatoire multi-ojectif (également appelés MOCO) sont connus pour être particulièrement difficiles, tant en théorie qu'en pratique. En fait, la plupart de ces problèmes sont $\mathcal{NP}$-difficile, même lorsque la version mono-objectif du problème appartient à la classe de complexité $\mathcal{P}$ [49]. Il existe plusieurs problèmes de type MOCO, chacun ayant ses propres particularités. Parmi lesquels nous distinguons : le problème d'affectation, plus court chemin, tournée de véhicule, sac à dos, voyageurs de commerce, etc.

Notre attention s'est focalisée sur le problème de sac à dos multi-objectif. Ce problème est réputé pour être l'un des problèmes MOCO les plus connus et les plus étudiés dans la littera-

ture, en effet, il intervient comme un sous-problème dans de nombreuses applications comme la logistique, l'industrie, l'économie, le transport, etc. Dans ce travail, nous nous somme interessé particulièrement au problème du sac à dos bi-objectif en variables binaires (désigné par BOKP). Ce problème a reçu une attention significative dans la littérature, il a été largement étudié par de nombreux chercheurs. Plusieurs approches ont été proposées pour sa résolution : des algorithmes exacts qui fournissent l'ensemble (ou un sous-ensemble) de solutions efficaces et des algorithmes approximatifs qui produisent une bonne approximation de l'ensemble efficace. L'une des méthodes bien connues dans la littérature est la méthode en deux phases introduite par Ulungu et Teghem [132, 133] où dans la première phase, l'ensemble des solutions efficaces dites supportées est calculé en résolvant un problème de somme pondérée des fonctions d'objectif. Tandis que dans la deuxième phase, les solutions efficaces dites non supportés sont trouvées en utilisant soit une méthode exacte, soit une méthode approximative. Toutefois, Captivo et *al.* [27] ont modélisé le BOKP par un problème de plus court chemin bi-objectif, puis ont utilisé un algorithme de marquage pour determiner les solutions efficaces et en comparant avec la méthode en deux phases [132, 133], de meilleurs résultats ont été obtenus. Bazgan et *al.* [10] ont développé trois relations de dominance complémentaires dans un algorithme de programmation dynamique pour calculer les solutions efficaces dans l'espace des critères. Ces relations ont été appliquées pour eliminer des états ne peuvant pas conduire à des solutions efficaces. Cette méthode a été améliorée dans [57] par Figueira et *al.* qui ont proposé de nouvelles techniques d'élimination des solutions pour accélérer le processus de résolution. Ensuite, Correia et *al.* ont présenté dans [36] des techniques algorithmiques qui étendent et améliorent empiriquement l'utilisation de la mémoire d'un algorithme de programmation dynamique pour calculer l'ensemble des solutions efficaces à la fois dans l'espace des critères et dans l'espace des décisions. En plus des méthodes exactes, quelques bonnes approximations des solutions non dominées d'un BOKP ont également été proposées (voir [8, 56]). La plupart des instances du problème BOKP dans la littérature ont des ensembles de solutions efficaces de cardinalités très considérables et certainement les décideurs n'ont pas besoin de toutes ses solutions, mais uniquement de celles qui réalisent l'optimum d'un autre objectif déja fixé (différent des deux objectifs du problème). Ceci mène vers la recherche d'une solution optimale d'un critère sur l'ensemble des solutions efficaces d'un BOKP.

Le problème de l'optimisation d'un critère sur l'ensemble des solutions efficaces d'un problème MOP est un cas particulier des problèmes d'optimisation non convexe, cela est dû principalement à la non convexité du domaine admissible constitué des solutions efficaces du problème MOP. Ce problème a été largement étudié dans le cas de variables continues (voir,

à titre d'exemple : [12, 13, 14, 48, 116]). Quant au cas discret, au cours de ces trois dernières decennies, ce problème est devenu l'un des domaines les plus importants et les plus intéressants de la programmation multi-objectif. Plusieurs travaux ont été proposés, spécifiquement dédiés au cas d'objectifs linéaires (problèmes appelés MOILP). Parmi ceux-ci, les références suivantes sont citées dans l'ordre chronologique : [30, 29, 75, 33, 31, 22, 89, 107].

Dans cette thèse, nous nous sommes particulièrement intéressés à l'optimisation sur l'ensemble des solutions efficaces d'un problème de sac à dos bi-objectif en variable binaires BOKP. Bien que certains algorithmes existants développés pour les problèmes MOILP peuvent être appliqués au cas de variables binaires, à notre connaissance aucune étude n'a été spécialement développée pour ce type de problèmes. Dans ce contexte, nous proposons une méthode basée essentiellement sur la programmation dynamique pour optimiser sur l'ensemble des solutions efficaces d'un BOKP. Cette contribution a fait l'objet d'une publication dans la revue internationale "Yugoslav Journal of Operations Research", intitulée "Optimizing over the efficient set of the binary bi-objective knapsack problem" [32] et présentée oralement dans une conférence internationale "The 2018 International Conference of the African Federation of Operational Research Societies (AFROS 2018)".

Cette thèse est répartie en cinq chapitres. Dans le premier chapitre, nous définissons les concepts de base et les principaux résultats relatifs à la programmation multi-objectifs, en se concentrant sur le cas des problèmes en nombres entiers MOILP et en décrivant brièvement quelques méthodes exactes qui leur sont dédiées. Le chapitre deux est consacré à l'optimisation combinatoire multi-objectif MOCO, nous présentons quelques problèmes de type MOCO et nous citons également quelques méthodes de résolution exactes et non exactes conçues pour ces problèmes. Dans le chapitre trois, nous abordons la problématique de l'optimisation sur l'ensemble des solutions efficaces des problèmes MOILP. Nous commençons par définir le problème, puis nous exposons quelques méthodes de résolution. Nous explorons dans le quatrième chapitre le problème de sac à dos multi-objectif en rappelant ces différentes variantes, nous discutons ensuite quelques méthodes de résolution pour le cas de variables binaires. Notre contribution est exposée au cinquième et dernier chapitre. Nous présentons les définitions et résultats de base relative à la programmation dynamique, la description générale de notre méthode et l'algorithme développé en détails. Pour bien comprendre le processus de la méthode, un exemple illustratif est déroulé suivi des résultats expérimentaux menés sur des instances du problème BOKP, ainsi qu'une discussion des résultats obtenus. Le travail est enfin achevé par une conclusion et quelques perspectives de recherche.

Chapitre 1

Optimisation multi-objectif

1.1 Introduction

L'optimisation multi-objectif (MO) consiste à optimiser simultanément plusieurs objectifs (critères), le plus souvent conflictuels soumis à un ensemble de contraintes. Un problème multi-objectif peut être formulé comme suit :

$$(MOP)\begin{cases} \text{“optimiser”} & (f_1(x), f_2(x), \ldots, f_p(x)) \\ \text{s.c.} & x \in \mathcal{X}, \end{cases} \tag{1.1}$$

où $f : \mathcal{X} \to \mathbb{R}^p$ est une fonction vectorielle composée de p fonctions à valeurs réelles $f_k : \mathcal{X} \to \mathbb{R}, k = 1, \ldots, p$. L'ensemble $\mathcal{X}$ est un ensemble réalisable, supposé être un ensemble non vide et compact tel que $\mathcal{X} = \{x \in \mathbb{R}^n \mid g_j(x) \leq 0,\ j = 1, 2, \ldots, m\}$.

1.2 Programmation linéaire multi-objectif

Si les objectifs $f_k,\ k = 1, 2, \ldots, p$ et les fonctions $g_j,\ j = 1, 2, \ldots, m$ dans le problème (1.1) sont linéaires, le problème devient un problème de programmation linéaire multi-objectif noté (MOLP), il s'écrit mathématiquement comme suit :

$$(MOLP)\begin{cases} \text{“optimiser”} & Z_k(x) = c^k x, k = 1, 2, \ldots, p \\ \text{s.c.} & x \in S, \end{cases}$$

où : $c^k \in \mathbb{R}^{1\times n}, k = 1, 2, \ldots, p$, $S = \{x \in \mathbb{R}^n | Ax \leq b,\ x \geq 0\}$, $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m, n \in \mathbb{N}$.

Lors de la résolution d'un problème d'optimisation multi-objectif, on obtient généralement un grand nombre de solutions qui ne peuvent pas toutes être optimales étant donné que certains objectifs sont contradictoires. Un concept important qui nous permet de choisir les bonnes solutions est le compromis. En effet, l'amélioration de la performance sur un objectif implique une dégradation de performance sur au moins un autre objectif. Par conséquent, la notion d'optimalité disparaît pour ce type de problèmes au profit de la notion d'efficacité.

Sans perte de généralité, on considère tout au long de cette thèse que les problèmes d'optimisation multi-objectif sont des problèmes à maximiser.

1.2.1 Définitions et concepts de base

Soit le problème MOLP :

$$(MOLP)\begin{cases} \text{“max”} & Z_k(x) = c^k x, \quad k = 1, 2, \ldots, p \\ \text{s.c.} & x \in S. \end{cases} \tag{1.2}$$

On considère l'application f qui associe à chaque vecteur de décision $x \in S$ son image $Z = f(x) = (c^1 x, \ldots, c^p x)$.

Définition 1.1

- *L'espace $\mathbb{R}^n$ dans lequel se situe l'ensemble des décisions S ($S \subseteq \mathbb{R}^n$) est appelé* ***espace des décisions****.*
- *L'espace $\mathbb{R}^p$ dans lequel se situe $f(S)$ est appelé* ***espaces des critères****.*

1.2.1.1 Dominance et efficacité

— Soit “$>$” la relation définie sur $\mathbb{R}^p$ par :
$\forall Z, Z' \in \mathbb{R}^p$, $Z > Z'$ si et seulement si : $Z_k \geq Z'_k$, $\forall k \in \{1, \ldots, p\}$ et $Z_k > Z'_k$ pour au moins un k.

— Soit “$\gg$” la relation définie sur $\mathbb{R}^p$ par :
$\forall Z, Z' \in \mathbb{R}^p$, $Z \gg Z'$ si : $Z_k > Z'_k$, $\forall k \in \{1, \ldots, p\}$.

La notion de la dominance est définie comme suit.

Définition 1.2

- *Soient deux vecteurs $Z, Z' \in f(S)$, on dit que Z* ***domine*** *Z' si : $Z > Z'$.*
- *Un vecteur $Z \in f(S)$ est dit* ***non dominé****, s'il n'existe pas un autre vecteur $Z' \in f(S)$ tel que $Z' > Z$.*
- *Une solution $x^* \in S$ est dite* ***efficace*** *si : $\nexists x \in S \ : \ Z(x) > Z(x^*)$. x^* est aussi appelée solution* ***Pareto optimale****.*
- *L'image de l'ensemble des solutions efficaces (frontière efficace) dans l'espace des critères est appelée* ***front de Pareto****.*

Définition 1.3

- *Le vecteur critère $Z \in f(S)$* ***domine fortement*** *un autre vecteur $Z' \in f(S)$ si : $Z \gg Z'$.*
- *Un vecteur $Z \in f(S)$ est dit* ***faiblement non dominé*** *s'il n'existe aucun autre vecteur $Z' \in f(S)$ qui le domine fortement.*
- *Une solution $x^* \in S$ est dite* ***fortement efficace*** *s'il n'existe pas de $x \in S$ tel que $Z(x) > Z(x^*)$.*
- *Une solution $x^* \in S$ est dite* ***faiblement efficace*** *s'il n'existe pas de $x \in S$ tel que $Z(x) \gg Z(x^*)$.*

Remarque 1.1
La dominance forte implique la dominance implique la dominance faible.

Définition 1.4

- *Deux vecteurs critères $Z, Z' \in f(S)$ sont dits* ***équivalents*** *si :*
  $Z_k = Z'_k, \quad \forall k \in \{1, \ldots, p\}$.
- *Deux solutions réalisables $x, x' \in S$ sont dites* ***équivalentes*** *si : $Z_k(x) = Z_k(x'), \quad \forall k \in \{1, \ldots, p\}$.*

Remarque 1.2

— *Une solution $x^* \in S$ qui est fortement efficace est une solution efficace qui n'a pas de solutions équivalentes.*
— *L'efficacité forte implique l'efficacité qui implique l'efficacité faible, et la réciproque n'est pas vraie.*

Dans ce qui suit, l'ensemble des solutions efficaces sera noté E. Son image dans l'espace des objectifs sera noté ND.

1.2.1.2 Optimalité lexicographique

Nous introduisons dans cette section une autre définition de l'optimalité pour les problèmes MOP, l'optimalité lexicographique. Cette notion d'optimalité est utilisée lorsqu'il y a un ordre de préférence entre les objectifs. Par exemple, si $p = 2$, on commence par optimiser le premier objectif sans tenir compte du deuxième objectif, sil existe plusieurs solutions optimales, on optimise le deuxième objectif tout en gardant la valeur optimale obtenue pour le premier objectif.

Nous présentons maintenant une définition formelle de l'optimalité lexicographique. Soient $y^1, y^2 \in \mathbb{R}^p$, $y^1 \neq y^2$, soit $k^* := \min\{k : y^1_k \neq y^2_k\}$.

- **Ordre lexicographique :** $y^1 \succeq_{\text{lex}} y^2 \Leftrightarrow y^1_{k^*} > y^2_{k^*}$ ou $y^1 = y^2$.
- **Ordre lexicographique strict :** $y^1 \succ\succ_{\text{lex}} y^2 \Leftrightarrow y^1_{k^*} > y^2_{k^*}$.

Le problème d'optimisation lexicographique peut être écrit comme suit :

$$\underset{x\in\mathcal{X}}{\text{lexmax}}\,(f_1(x), \ldots, f_p(x))\,.$$

Définition 1.5 *(Solution lexicographiquement optimale)*
Une solution réalisable $x^* \in \mathcal{X}$ est appelée solution lexicographiquement optimale s'il n'existe aucune autre solution réalisable $x \in \mathcal{X}$ telle que $f(x) \succ\succ_{lex} f(x^*)$.

Remarque 1.3
Nous pouvons affirmer que $x^* \in \mathcal{X}$ est une solution lexicographiquement optimale si $f(x^*) \succeq_{lex} f(x)$ pour tout $x \in \mathcal{X}$.

Théorème 1.1 *(Ehrgott[50]) Soit $x^* \in \mathcal{X}$ tel que pour tout $x \in \mathcal{X}$, $f(x^*) \succeq_{lex} f(x)$, alors $x^* \in E$.*

La preuve de ce théorème peut être consultée dans [50].

1.2.1.3 Points particuliers

Dans le cadre de l'optimisation multi-objectif, il existe certains points de référence qui ont été définis dans l'espace des critères. Ces points peuvent représenter des solutions réalisables ou non et définissent des bornes inférieures et supérieures sur les points non dominés. Ils sont données par : le point idéal, le point anti-idéal, le point nadir et le point utopique, un exemple illustratif de ces points est donné dans la figure (Fig. 1.1).

- **Point idéal :** que l'on note $\bar{Z}$ est le point dans l'espace des critères qui a comme coordonnées la valeur maximale de chaque objectif en les maximisant séparément.

$$\bar{Z}_k = \max_{x \in S}(Z_k(x)), \quad k = 1, \ldots, p.$$

  Ce point ne correspond pas à une solution réalisable.
- **Point anti-idéal :** que l'on note $\underline{Z}$ est le point dans l'espace des critères qui a comme coordonnées la valeur minimale de chacune des fonctions objectif.

$$\underline{Z}_k = min_{x \in S}(Z_k(x)), \quad k = 1, \ldots, p.$$

- **Point nadir :** que l'on note N est un point de l'espace des critères ayant pour coordonnées les pires valeurs obtenues par chaque fonction objectif lorsque l'on restreint la recherche au front de Pareto.
  Il existe plusieurs approches pour trouver le point nadir, la plus simple consiste à utiliser une matrice carrés de dimension p (le nombre d'objectifs à maximiser), cette matrice est appelée ***matrice des gains*** (*Payoff table*), elle est donnée par :

$$G = \begin{pmatrix} \bar{Z}_1 & \ldots & Z_{1j} & \ldots & Z_{1p} \\ \vdots & & \vdots & & \vdots \\ Z_{j1} & \ldots & \bar{Z}_j & \ldots & Z_{jp} \\ \vdots & & \vdots & & \vdots \\ Z_{p1} & \ldots & Z_{pj} & \ldots & \bar{Z}_p \end{pmatrix}$$

  Où : la diagonale principale de cette matrice représente les coordonnées du point idéal.
  A partir de cette matrice le point nadir est définir par :

$$N_k = \min_{j=1,\ldots,p}(Z_{kj}), \quad k = 1, \ldots, p.$$

- **Point utopique :** à partir du point idéal, il est possible de définir un point utopique noté $\bar{Z}^U$ comme suit : $\bar{Z}^U = \bar{Z} + \epsilon I$, où $\epsilon > 0$ est un petit nombre positif, et $I \in \mathbb{R}^p$ est le vecteur unitaire. À noter que ce point n'est pas réalisable.

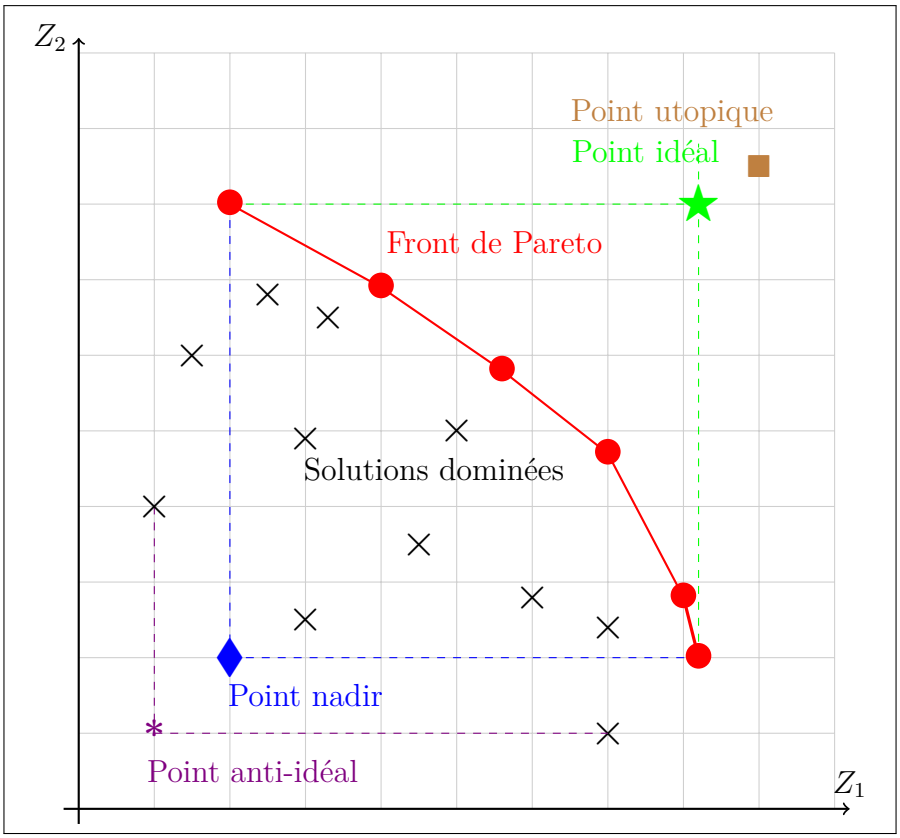


Fig. 1.1 : Exemple illustratif des points particuliers

1.2.2 Caractérisation des solutions efficaces

Nous présentons quelques caractérisations qui permettent de tester l'efficacité d'une solution réalisable d'un problème d'optimisation linéaire multi-objectif.
Soit x^* une solution réalisable arbitraire d'un problème d'optimisation multi-objectif (MOP) et soit le problème uni-critère suivant :

$$\begin{cases} \max & \Theta = \sum_{i=1}^{p} \psi_i \\ \text{s.c.} & f_i(x) + \psi_i = f_i(x^*), \ i = 1, \ldots, p, \\ & x \in \mathcal{X}; \quad \psi_i \geq 0, \quad \forall i = 1, \ldots, p. \end{cases} \tag{1.3}$$

Théorème 1.2 *(Benson [12]) x^* est une solution efficace pour (MOP) si et seulement si la valeur optimale de la fonction objectif Θ est nulle dans le problème (1.3).*
Si pour une solution réalisable $\hat{x}$, la valeur de Θ est différente de zéro alors $\hat{x}$ est efficace pour (MOP).

Dans le cas d'un problème MOLP, nous démontrons ce théorème comme suit :

Preuve

1. x^* est une solution efficace pour (MOLP) si et seulement si la valeur optimale de la fonction objectif Θ est nulle dans le programme linéaire (1.3).
   Soit (x, ψ) une solution réalisable du problème (1.3), alors $\psi_i \geq 0$, $i = 1, \ldots, p$, et $\psi_i = c^i x^* - c^i x$, $i = 1, \ldots, p$, nous aurons : $c^i x \leq c^i x^*$, $i = 1, \ldots, p$.
   De plus, la valeur optimale de Θ est nulle, alors $\psi_i = 0$, $i = 1, \ldots, p$, ainsi $c^i x = c^i x^*$, $i = 1, \ldots, p$. Donc, x^* est efficace. Comme (x, ψ) est une solution réalisable du problème (1.3), alors $c^i x \leq c^i x^*$, $i = 1, \ldots, p$. Si x^* est efficace, alors nécessairement $c^i x = c^i x^*$, $i = 1, \ldots, p$, et donc, $\psi_i = 0$, $i = 1, \ldots, p$. Par conséquent, la valeur optimale de Θ est nulle.

2. Si pour un point $\hat{x}$, la valeur de Θ est différente de zéro alors $\hat{x}$ est efficace pour (MOLP).
   Supposons que la solution $\hat{x}$ n'est pas efficace. Alors il existe une solution réalisable x' telle que $c^i \hat{x} \geq c^i x'$, $i = 1, \ldots, p$. Posons $\psi'_i = c^i x^* - c^i x'$, $i = 1, \ldots, p$ où x^* est une solution réalisable de (MOLP). Alors (x', ψ') est une solution réalisable du problème (1.3) et car pour tout i fixé, on a : $\psi'_i = c^i x^* - c^i x' \geq c^i x^* - c^i \hat{x} = \hat{\psi}_i$ avec au moins une inégalité stricte. Ainsi, $\sum_{i=1}^{p} \psi'_i > \sum_{i=1}^{p} \hat{\psi}_i$ alors $(\hat{x}, \hat{\psi})$ n'est pas une solution optimale pour (1.3).
   □

Considérons le problème (MOLP). Soit le problème paramétrique suivant :

$$(P_\lambda) \begin{cases} \max & \displaystyle\sum_{i=1}^{p} \lambda_i Z_i(x) \\ \text{s.c.} & x \in \mathcal{X}, \end{cases} \tag{1.4}$$

où : $\lambda \in \Lambda \left\{ \lambda \in \mathbb{R}^p \mid \displaystyle\sum_{i=1}^{p} \lambda_i = 1, \lambda_i > 0, i = 1, \ldots, p \right\}$.
Le théorème suivant s'applique à un programme avec objectifs linéaires et domaine convexe.

Théorème 1.3 (Geoffrion [63]) *La solution x est une solution efficace pour le problème (MOLP) si et seulement si x est une solution optimale du problème paramétrique (P_λ).*

1.3 Programmation linéaire multi-objectif discrète

La programmation linéaire multi-objectif en nombres entiers fait partie du cadre d'aide à la décision discrète. Au cours des dernières décennies, plusieurs méthodes ont été développées pour résoudre les problèmes MOILP, des méthodes qui renvoient un ensemble de toutes les solutions efficaces et d'autres méthodes qui ne produisent qu'un sous-ensemble de solutions efficaces, selon les préférences du décideur. Dans ce qui suit, on présente d'abord la structure générale d'un problème MOILP et quelques résultats relatifs, puis on décrit quelques méthodes exactes dédiées à la détermination de l'ensemble des solutions efficaces du problème MOILP basant sur la recherche dans l'espace des critères et d'autres qui accomplissent une recherche dans l'espace des décisions.

1.3.1 Structure générale d'un problème MOILP

Un problème d'optimisation linéaire multi-objectif en nombres entiers s'écrit sous la forme suivante :

$$(MOILP)\begin{cases} \text{“max”} & Z_k = c^k x, k = 1, 2, \ldots, p \\ \text{s.c.} & x \in D. \end{cases}$$

Où $D = S \cap \mathbb{Z}^n$, $\mathbb{Z}$ étant l'ensemble des nombres entiers relatifs et $S = \{x \in \mathbb{R} | Ax \leq b,\ x \geq 0\}$, $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m, n \in \mathbb{N}$.

1.3.2 Classification des solutions efficaces

D'après le Théorème (1.3), toutes les solutions efficaces du MOLP sont des solutions efficaces du problème (P_λ). Cependant, Bowman [78] a démontré par un contre-exemple que dans le cas du MOILP, il existe des solutions efficaces qui ne sont pas des solutions optimales pour le problème (P_λ). Ceci est dû à la non-convexité du domaine des solutions réalisables. Deux types de solutions efficaces peuvent alors être distinguées [78] (voir la figure (Fig. 1.2)).

- **Solutions supportées** notées SE, l'image de cet ensemble est appelée l'ensemble des points non dominés supportés, les points se situent sur la frontière de l'enveloppe convexe l'espace des critères (le plus petit ensemble convexe [1] contenant l'ensemble des points non dominés).

1. Un ensemble $\mathcal{X}$ de $\mathbb{R}^n$ est dit convexe si tout segment d'extrémités prises dans $\mathcal{X}$ est inclus dans $\mathcal{X}$, c-à-d $\lambda x + (1-\lambda)y \in \mathcal{X},\ \forall x, y \in \mathcal{X},\ \forall \lambda \in [0, 1]$.

Le problème multi-objectif est transformé en un problème mono-objectif paramétré :

$$(P_\lambda) : \max\{\lambda^t Cx \mid x \in D\}.$$

$$\text{Avec } \lambda \in \Lambda \left\{ \lambda \in \mathbb{R}^p \mid \sum_{i=1}^{p} \lambda_i = 1, \lambda_i > 0, i = 1, \ldots, p \right\}.$$

Nous pouvons distinguer deux types de solutions efficaces supportées :

- Solutions efficaces supportées extrêmes : l'ensemble de ces solutions est noté SE_E. Son image dans l'espace des objectifs est l'ensemble des points non dominés supportés extrêmes qui se situent sur les sommets de l'enveloppe convexe de l'espace des critères.
- Solutions efficaces supportées non extrêmes : l'ensemble de ces solutions est noté par SE_{NE}. Cet ensemble est le complément de SE_E par rapport à l'ensemble des solutions efficaces supportées SE, $SE_{NE} = SE \setminus SE_E$. Son image dans l'espace des objectifs est appelée l'ensemble des points non dominés supportés non extrêmes.

- **Solutions non supportées** notées NSE, elles ne se situent pas sur l'enveloppe convexe de D. Aucune de ces solutions ne peut être trouvée en optimisant une agrégation linéaire des objectifs.

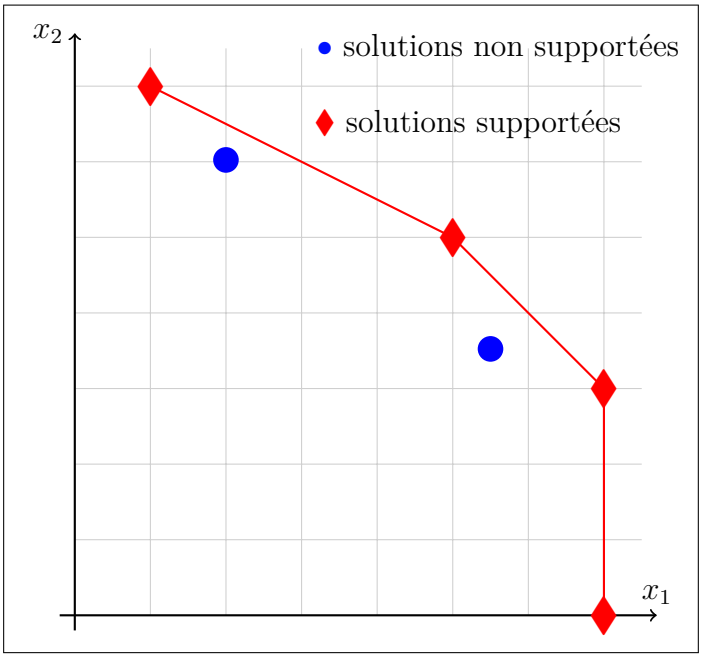


Fig. 1.2 : Solutions supportées et non supportées

1.3.3 Quelques méthodes de résolution d'un problème MOILP

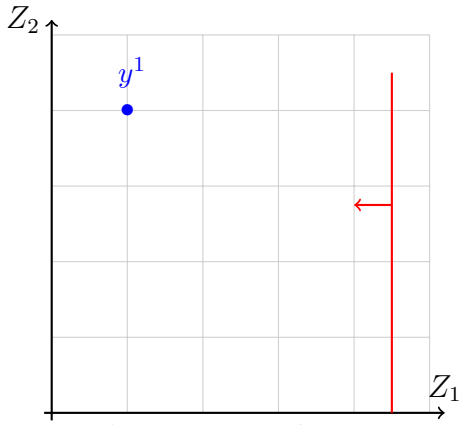
De nombreuses méthodes ont été proposées, qui consistent à caractériser totalement ou partiellement l'ensemble des solutions efficaces du problème de programmation linéaire multi-objectif en nombres entiers. Puisque nous nous intéressons au cas linéaire, nous décrirons dans ce qui suit quelques méthodes dédiées à ce type de problèmes dans l'ordre chronologique.

1.3.3.1 La méthode ϵ-contrainte

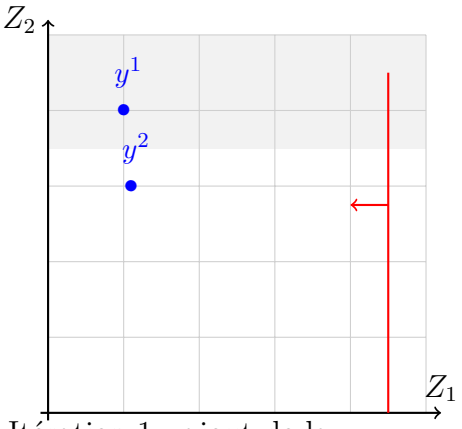
Cette méthode introduite la première fois par Haimes et *al.* [69] est conçu pour la résolution des problèmes MOP et permet de générer toutes les solutions non dominées (elle peut générer aussi des solutions faiblement non dominées). Dans cette méthode, il n'y a pas d'agrégation des objectifs, une fonction objectif est optimisée tandis que les autres fonctions objectifs sont transformées en contraintes. Le problème d'optimisation ϵ-contrainte est défini comme suit :

$$P(\epsilon)\begin{cases} \min_{x\in\mathcal{X}} Z_i(x) \\ Z_k(x) \leq \epsilon_k,\ k = 1,\ldots,p,\ k \neq i, \end{cases}$$

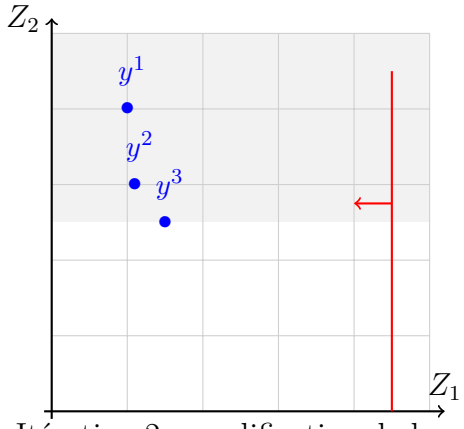
où $\epsilon \in \mathbb{R}^p$.
La figure (Fig. 1.3) illustre le fonctionnement de la méthode. L'inconvénient majeur de cette méthode est qu'elle nécessite une résolution d'un problème mono-objectif pour chacune des solutions non dominées ce qui rend la résolution très onéreuse surtout si la méthode de résolution mono-objectif utilisée est coûteuse. A cet effet, plusieurs pistes d'amélioration de la méthode ont été proposées. D'une part, certains auteurs s'intéressent à la technique d'exploration de l'espace des objectifs lorsque $p > 2$ et aux valeurs que prennent les constantes "ϵ". En effet, lorsqu'une unique fonction objectif Z_2 à valeurs dans $\mathbb{N}$ est contrainte et qu'une nouvelle solution efficace x est trouvée, la constante "ϵ" prend la valeur $Z_2(x)+1$. Ce changement n'est pas aussi direct lorsque la fonction objectif contrainte est à valeur dans $\mathbb{R}$. De plus, le problème se complexifie lorsque plusieurs fonctions objectif sont contraintes. Laumanns et *al.* [86] proposent de séparer l'espace des objectifs en une hyper-grille de taille $p-1$, où chaque case est résolue par une méthode ϵ-contrainte. Pour les problèmes en nombres entiers multi-objectifs, Özlen et Azizoglu [142] et Özlen et *al.* [143] génèrent Y^N en choisissant des constantes "ϵ" les plus précises possibles pour chaque contrainte. Kirlik et Sayın [81] séparent également l'espace des objectifs en une hyper-grille de dimension $p-1$. La case de plus grand volume est alors parcourue, et les nouveaux points non dominés trouvés forment une nouvelle grille. Leur méthode est plus efficace que celle de Laumanns et *al.* [86] et Özlen et *al.* [143].



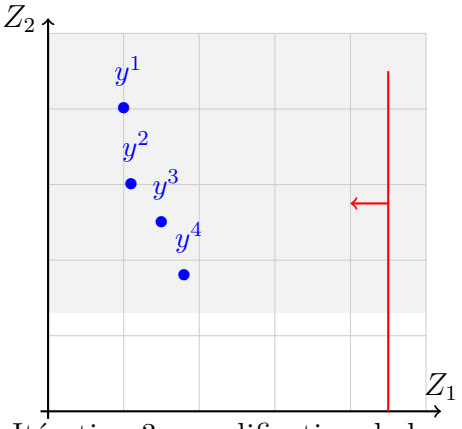
Initialisation : on résout en considérant l'objectif 1, on obtient y^1.



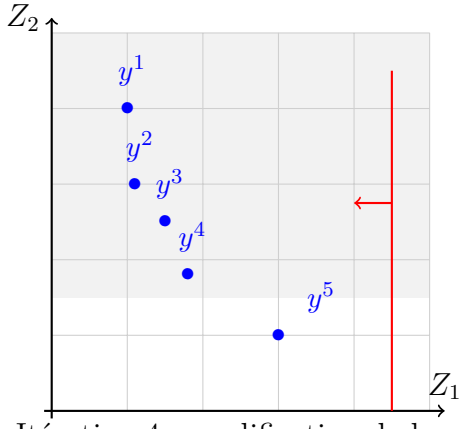
Itération 1 : ajout de la contrainte ϵ. et obtention de y^2.



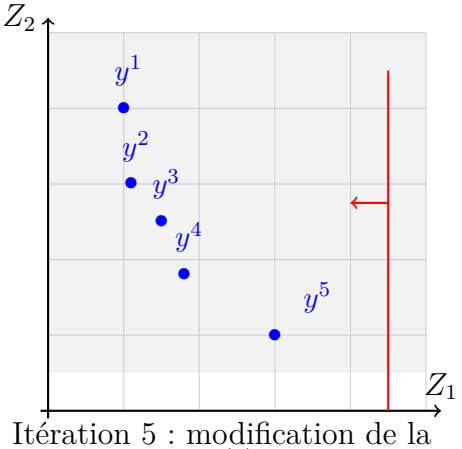
Itération 2 : modification de la contrainte ϵ et obtention de y^3.



Itération 3 : modification de la contrainte ϵ et obtention de y^4.



Itération 4 : modification de la contrainte ϵ et obtention de y^4.



Itération 5 : modification de la contrainte ϵ, $P(\epsilon)$ est irréalisable.

Fig. 1.3 : Illustration de la méthode ϵ-contrainte

1.3.3.2 Méthode de Klein & Hannan

Le principe de cette méthode [84] est de résoudre progressivement une séquence de programmes linéaires mono-objectif en nombres entiers avec des contraintes ajoutées à chaque étape. Ces contraintes ajoutées éliminent les solutions efficaces déjà trouvées, et font en sorte que les nouvelles solutions générées soient efficaces. Elle permet d'obtenir l'ensemble des solutions efficaces en appliquant les étapes suivantes :

Etape 1

Résoudre le problème (P_1) défini comme suit :

$$(P_1): \quad \max\{Z^s = c^s x \mid x \in D\}.$$

Cas 1. Si la solution optimale de (P_1), soit x_1, est unique alors elle est efficace pour (P).

Cas 2. Sinon, déterminer toutes les solutions alternatives à x_1 et par comparaison deux à deux des vecteurs critères associés, garder uniquement celles qui sont efficaces pour construire l'ensemble $E(P_1)$ des solutions efficaces générées à l'étape 1.

Etape générale j

A l'étape j, on résout le problème (P_j) défini comme suit :

$$(P_j)\begin{cases} \max & \{Z^s = c^s x\} \\ \text{s.c.} & x \in D, \\ & \bigcap\limits_{i=1}^{q}\left(\bigcup\limits_{k=1,k\neq s}^{p} c^k x \geq c^k y^i + f_k\right), \\ & k \neq s. \end{cases} \tag{1.5}$$

avec :

- L'indice s est pris arbitrairement dans $1, \ldots, p$.
- $f_k \geq 1$ et il est entier pour $k \in \{1, \ldots, p\}, k \neq s$.
- y^i, $i \in \{1, \ldots, q\}$ sont les solutions efficaces obtenues aux étapes $1, \ldots, j-1$.

Si $E(P_j)$ est l'ensemble des solutions efficaces obtenues à l'étape j et Y^j l'ensemble des points efficaces cumulés à la fin de l'étape j, alors $Y^j = \{Y^{j-1} \cup E(P_j)\}$ pour $j \geq 2$ avec $Y^1 = E(P_1)$.

Etape finale n

La procédure s'arrête lorsque le problème (P_n) est irréalisable.

1.3.3.3 Méthode de Teghem and Kunsch

Cette méthode nommée "MOMIX" a été developpée dans [129] pour résoudre des problèmes en variables entières mixtes. C'est une version interactive de la méthode Branch and Bound qui se déroule en deux phases :

Phase 1. Initialisation

Déterminer le premier point efficace x^0 en minimisant la norme de Tchebytcheff. Le problème suivant est donc résolu pour $m = 0$.

$$\left(P^{(m)}\right) \begin{cases} \min & \delta \\ \text{s.c.} & \beta_{m,k}\left(M_{m,k} - c^k x\right) \leq \delta, \quad k = 1, 2, \ldots, p, \\ & x \in S. \end{cases}$$

Où $S_0 = S$, $M_{m,k}$ la valeur optimale de la $k^{\text{ème}}$ fonction objectif sur S_m et $\beta_{m,k} = \frac{\alpha_{m,k}}{\sum_{i=1}^{p} \alpha_{m,i}}$, le poids de la $k^{\text{ème}}$ fonction objectif ; par exemple,

$$\alpha_{m,k} = \frac{M_{m,k} - n_{m,k}}{\max\left(|n_{m,k}|, |M_{m,k}|\right)} \frac{1}{\|c^k\|}.$$

Où $n_{m,k}$ est la k ème composante du point nadir sur S_m.

Phase 2.

La méthode Branch and Bound interactive est utilisée en deux étapes :

Première étape (Procédure descendante "depth first") :

Soit la $m^{ème}$ itération.

- Soit x^{m-1} le $m^{ème}$ compromis et $z_k^{(m-1)} = c^k x^{m-1}$ la valeur correspondante pour la $i^{ème}$ fonction objectif.

- Le DM indique le critère $l_m(1) \in \{1, 2, \ldots, p\}$ à améliorer en priorité.

- Un nouveau compromis est obtenu en résolvant le problème $\left(P^{(m)}\right)$ avec : $S_m = S_{m-1} \cap \left\{x \mid z_{l_m(1)}(x) > z_{l_m(1)}^{(m-1)}\right\}$; le critère $l_m(1)$ sera ainsi amélioré.

Tests d'arrêt

Des tests d'arrêt ont été définis ; on cite quelques-uns :

- $S_m = \emptyset$.
- $M_{m,k} - n_{m,k} \leq \varepsilon_k$ pour toutes les valeurs de k ($\varepsilon_k > 0$, fixé).

- $\widehat{z}$, le vecteur des meilleures valeurs trouvées précédemment, est préféré au vecteur idéal M_m relatif à D_m.
- Un nombre Q d'itérations a été effectué.
- Aucune amélioration n'a été apportée à $\widehat{z}$ après un nombre d'itérations fixé par le décideur DM.

Deuxième étape (Procédure remontante "backtracking") :

La région d'admissibilité négligée à chaque itération de la procédure descendante est scannée pour un éventuel meilleur compromis que $\widehat{z}$ dans cette région. Les différentes opérations sont :

- Le noeud correspondant au compromis x^{m-1} est séparé en p branches.

- Soit $l_m(k); k = 1, 2, \ldots, p$ l'ordre de priorité dans lequel le décideur veut améliorer les p critères par rapport à ce compromis ;

- Les p sous-noeuds sont obtenus par adjonction respective des contraintes suivantes :

$$z_{l_m(1)} > z^{m-1}_{l_m(1)}, \quad \left\{ \begin{array}{l} z_{l_m(2)} > z^{m-1}_{l_m(2)} \\ \text{et} \\ z_{l_m(1)} \leq z^{m-1}_{l_m(1)} \end{array} \right. , \quad \cdots \quad \left\{ \begin{array}{l} z_{l_m(p)} > z^{m-1}_{l_m(p)} \\ \text{et} \\ z_{l_m(k)} \leq z^{m-1}_{l_m(k)}, \quad k = 1, 2, \ldots, p. \end{array} \right.$$

L'ensemble de solutions admissibles est partitionné et à chaque sous-noeud on résout le problème $\left(P^{(m)}\right)$. Les mêmes tests d'arrêt sont utilisables.

1.3.3.4 La méthode en deux phases

La méthode en deux phases constitue une approche générique conçue pour résoudre les problèmes MOILP. Elle a été initialement développée par Ulungu et Teghem [132, 133] pour la résolution d'un problème d'affectation bi-objectif. Le principe générale est de construire l'ensemble des solutions efficaces en deux étapes, la recherche des solutions supportées puis la recherche des solutions non supportées.

Phase I : Détermination des solutions supportées

Dans cette phase, on résout une succession de problèmes mono-objectif obtenus en effectuant une agrégation linéaire des deux objectifs. Ainsi la méthode commence par la recherche des deux solutions extrêmes (Fig. 1.4 (a)), puis fixe les poids pour définir une direction de recherche par rapport au plan formé par ces deux points (Fig. 1.4 (b)), ensuite d'une manière

itérative la recherche dichotomique est effectuée (Fig. 1.4 (c)) jusqu'à ce qu'aucun nouveau point ne soit trouvé. A la fin de cette phase on obtient toutes les solutions Pareto optimales supportées (Fig. 1.4 (d)).

Phase II : Détermination des solutions non supportées

La deuxième phase consiste à la recherche des solutions non supportées appartenant au front Pareto qui ne peuvent pas être obtenues par régression linéaire. La recherche se fait en utilisant les solutions supportées trouvées lors de la première phase pour définir des "zones de recherche" en considérant que les solutions non supportées restantes sont forcément dans les triangles rectangles engendrés par deux solutions supportées consécutives. Entre chaque couple de solutions supportées adjacentes, trois zones de recherches se définissent comme illustré dans la figure (Fig. 1.4 (e)). La zone rose est éliminée car elle ne peut pas contenir de solutions réalisables, la zone rouge est dominée par les deux solutions efficaces et peut donc être évitée aussi, la recherche se restreint alors à la zone verte (Fig. 1.4 (f)). L'exploration de cette zone de recherche doit utiliser les spécificités du problème étudié pour être efficace. Elle peut être une méthode de type branch and bound (Visée et *al.* [138] pour le problème de sac à dos bi-objectif, Chaabane et Hamidi [71] pour le problème de recouvrement maximal bi-objectif et Vincent,[137] pour les problèmes MOILP). Une méthode de de programmation dynamique (Delort et Spanjaard,[43] pour le problème de sac à dos bi-objectif). Une procédure de ranking (Przybylski et *al.* [109] pour le problème d'affectation bi-objectif) et (Jorge [76] pour le problème de sac à dos bi-objectif).
A la fin de la seconde phase, l'ensemble des points non supportés est obtenu (Fig. 1.4 (g)).

Cette méthode est très intéressante car elle présente un processus de résolution très général qui ne dépend pas de la nature du problème. Son principe repose sur la décomposition de l'espace de recherche en faisant appel à des méthodes mono-objectif pour les différentes résolutions successives (recherche solutions extrêmes, résolution des agrégations linéaires,...), par conséquent, il est important de bien choisir la méthode mono-objectif qui doit être aussi efficace que possible. Des améliorations de cette méthode et des extensions au cas de trois objectifs ont été proposées par la suite (à savoir les références [55, 111, 110]).

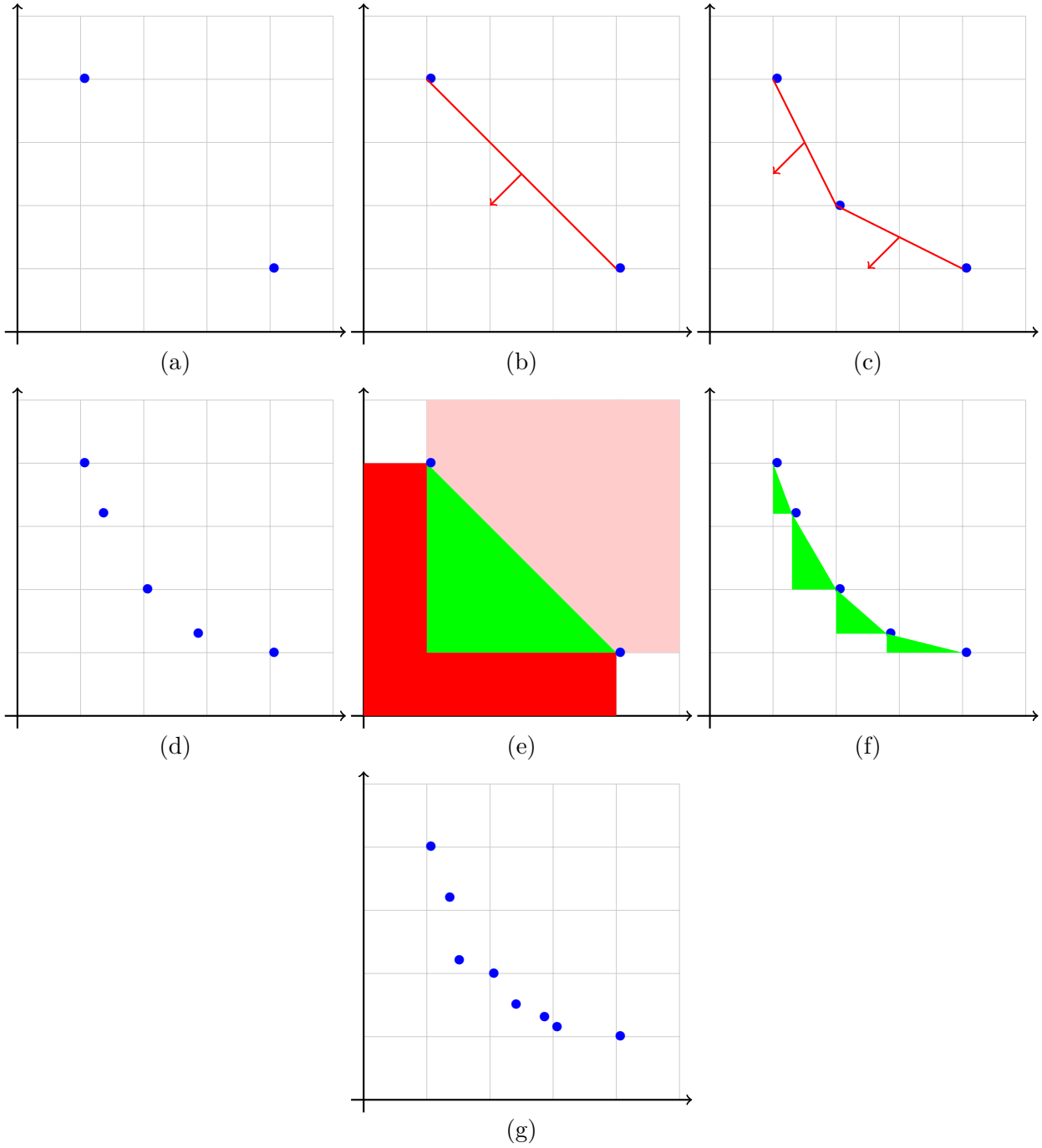


Fig. 1.4 : Illustration de la méthode en deux phases

1.3.3.5 Méthode de Sylva et Crema

Cette méthode développée par Sylva et Crema [125] est une variante de celle de Klein et Hannan étudiée précédemment. Son principe repose sur la résolution d'une succession de

programmes linéaires en nombres entiers optimisant à chaque étape une combinaison positive des critères. Un ensemble de contraintes est rajouté à chaque fois assurant la détection d'une nouvelle solution efficace. A la fin, la méthode fournit l'ensemble de tous les points non dominées du problème de programmation linéaire discrète à objectifs multiples. La méthode procède comme suit :

Etape 1

Fixer le vecteur des poids $\lambda = (\lambda_1, \lambda_2, ..., \lambda_p)$ à des valeurs strictement positives telles que : $\sum_{i=1}^{p} \lambda_i = 1$.

Résoudre le problème pondéré :

$$(P_1) : \quad \max\{\sum_{i=1}^{p} \lambda_i c^i x \mid x \in D\}.$$

Deux cas peuvent se présenter :

Cas 1. Si (P_1) n'admet pas de solutions, alors (P) l'est aussi.

Cas 2. Sinon, une solution x^1 est trouvée et elle est efficace.

Ensuite, une suite de programmes linéaires en nombres entiers augmentés par certaines contraintes sont résolus progressivement. Après k étapes du processus, deux cas figurent :

Cas 1. Si (P_k) est irréalisable, alors l'algorithme prend fin.

Cas 2. Sinon, une nouvelle solution efficace, soit x_k, est trouvée et le nouveau problème (P_{k+1}) est définit à partir de (P_k) en lui éliminant toutes les solutions vérifiant $c^i x \leq c^i x^k$, $\forall i \in \{1, \ldots, p\}$.

Ceci peut être traduit par l'ajout de contraintes suivantes :

$$\begin{cases} c^i x \geq (c^i x^k + 1) y_i^k - M_i(1 - y_i^k), & i \in \{1, \ldots, p\}, \\ \sum_{i=1}^{p} y_i^k \geq 1, \quad y_i^k \in \{0, 1\}, & i \in \{1, \ldots, p\}, \end{cases}$$

où $-M_i$ est un minorant pour toute valeur réalisable de la i^{eme} fonction objectif.

Etape générale $k+1$

Résoudre le problème (P_{k+1}) :

$$(P_{k+1})\begin{cases} \max & \sum_{i=1}^{p} \lambda_i c^i x \\ \text{s.c.} & x \in D, \\ & c^i x \geq (c^i x^j + 1) y_i^j - M_i(1 - y_i^j), \quad i \in \{1,\ldots,p\}, \;\; j \in \{1,\ldots,k\}, \\ & \sum_{i=1}^{p} y_i^j \geq 1, \quad y_i^j \in \{0,1\}, \quad i \in \{1,\ldots,p\}, \;\; j \in \{1,\ldots,k\}. \end{cases}$$

Etape finale N La procédure prend fin dans le cas où (P_N) est irréalisable.

1.3.3.6 Méthode de Boland et *al.*

Cette méthode dite "Balanced Box Method" [20, 21] est conçue pour résoudre les problèmes d'optimisation mixtes en nombres entiers bi-objectif. Initialement, les deux points lexicographiquement optimaux y_1 et y_2 sont calculés (étapes (a) et (b)) puis définir à partir de ces deux point le rectangle qui constitue la première zone de recherche. Ensuite, le processus suivant est itéré sur les régions de recherche suffisamment grandes (pouvant contenir des points non dominés). Le rectangle défini par (y_1, y_2) est divisé en deux parties égales selon le second objectif Z_2 (étape (c)). Soit R_I (respectivement R_S) le rectangle inférieur (respectivement supérieur) comprenant le point y_2 minimisant Z_2 (respectivement y_1 minimisant Z_1). Dans le rectangle R_I, le point y_3 minimisant Z_1 est obtenu (étape (d)) et dans le rectangle R_S, le point y_4 minimisant Z_2 est défini (étape (e)). De nouveaux rectangles sont constitués en tant que nouvelles régions de recherche : les rectangles définis par (y_1, y_4) et (y_2, y_3) (étape (f)). La même procédure est répétée pour tous les rectangles nouvellement créés. La première itération de cette méthode est illustrée dans la figure (Fig. 1.5) .

Des extensions de cette méthode sont ensuite développées dans [22, 23].

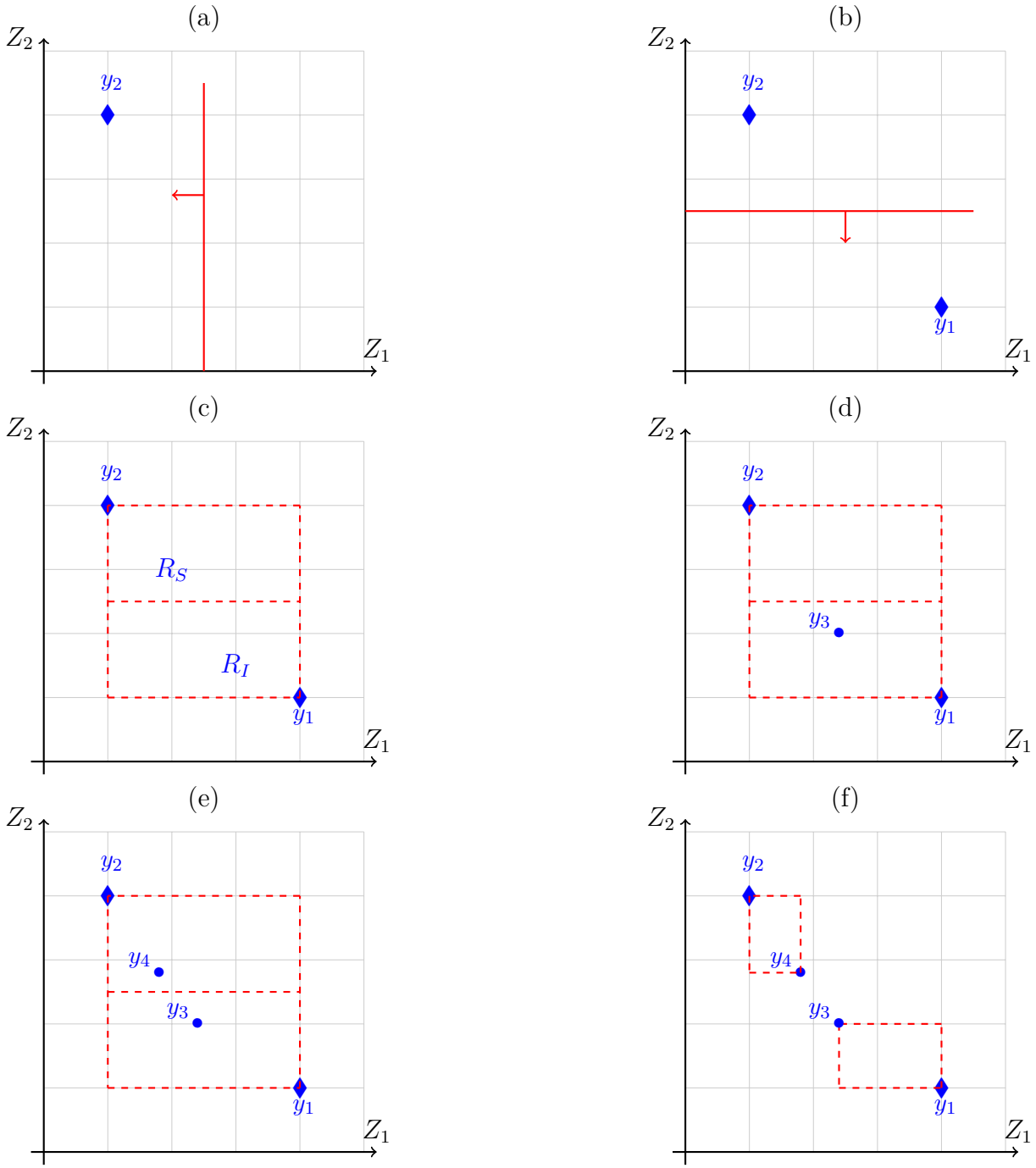


Fig. 1.5 : Les différentes étapes de l'itération initiale de Balanced Box Method

Nous donnons dans la table (Tab. 1.1) quelques méthodes de résolution des problèmes MOILP catégorisées selon le processus de résolution adopté. Le lecteur peut consulter le survey [70], dans lequel Halfmann et *al.* catégorisent et présentent des méthodes exactes pour

les problèmes linéaires multi-objectif en variables entières mixtes.

Branch-and-Bound, Branch-and-Cut ou exploration de l'espace des decisions	Bitran et Rivera [18], Abbas et Chaabane [1], Abbas et *al.* [2], Boland et *al.* ([24], [25]), Deckro et Winkofsky [41] , Ehrgott et Gandibleux [54], Gadegaard et *al.* [59], Jozefowiez et *al.* [77], Kiziltan et Yucaoglu [82], Parragh et Tricoire [103], Przybylski et Gandibleux [108], Sergienko et Perepelitsa [118], Simopoulos [119], Sourd et Spanjaard [121], Sourd et *al.* [122], White [139], Zionts [141].
Programmation disjonctive	Bektas [11], Klein et Hannan [84], Lokman et Köksalan [91], Sylva et Crema ([125],[126])
Programmation dynamique	Bergman et Cire [17], Bergman, Bodur et *al.* [16], Klötzler [85], Trzaskalik [130], Villarreal et Karwan([135], [136]), Bazgan et *al.* [10]
Méthodes epsilon-contrainte	Al-Rabeeah et *al.* [3], Bérubé et *al.* [26], Chalmet et *al.* [34], De Santis et *al.* [115], Kirlik et Sayın [81], Lokman et *al.* [90], Mavrotas ([96], [97]), Mavrotas et Florios ([98], [100], [99]), Özlen et Azizoglu [142], Özlen et *al.* [143], Pettersson et Özlen ([104], [105]), Sáez-Aguado et Trandafir [127], Zhang et Reimann [140].
Méthodes de décomposition de l'espace des critères	Boland, Charkhgard, et Savelsbergh ([19], [20], [21], [24], [23]), Dächert et Klamroth [45], Dächert et *al.* [46], Dhaenens et *al.* [44], Klamroth et *al.* [44], Leitner et *al.* [87], Lemesre et *al.* [88], Tamby et Vanderpooten [128].
Méthodes en deux phases	Ulungu et Teghem [133], Visée et *al.* [138], Dai et Charkhgard [37], Pal et Charkhgard [102], Przybylski et *al.* [111]

Tab. 1.1 : Classification de quelques méthodes de résolution d'un problème MOILP.

1.4 Conclusion

Dans ce chapitre, nous avons défini quelques concepts de base ainsi que les principaux résultats nécessaires liés à l'optimisation linéaire multi-objectif, en nous concentrant sur le cas des problèmes en nombre entiers MOILP. Nous avons par la suite mentionné et référé quelques méthodes de résolution des problèmes MOILP. Le chapitre suivant est consacré aux problèmes d'optimisation linéaire combinatoire multi-objectif MOCO.

Chapitre 2

Optimisation combinatoire multi-objectif

2.1 Introduction

L'optimisation combinatoire est un domaine largement étudié par de nombreux chercheurs, vu le nombre important d'applications réelles pouvant être modélisées sous forme de problèmes d'optimisation en variables qui ne peuvent prendre que deux valeurs 0 ou 1. Elle a prospéré au cours des dernières décennies, un bon aperçu de l'état de l'art peut être trouvé dans [42]. Dans ce chapitre, nous nous intéressons au cas où plusieurs objectifs sont pris en compte, ce que l'on appelle l'optimisation combinatoire multi-objectif (MOCO Multi-Objective Combinatorial Optimisation). Les problèmes MOCO sont le plus souvent faciles à définir mais généralement difficiles à résoudre. En fait, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles [117]. Dans ce qui suit, nous présentons la formulation mathématique des problèmes MOCO, quelques problèmes de type MOCO parmi les nombreux types existants et quelques méthodes de résolutions exactes et approchées dédiées à ce type de problèmes.

2.2 Formulation mathématique d'un problème MOCO

Un problème d'optimisation combinatoire multi-objectif s'écrit sous la forme suivante :

$$(MOCO)\begin{cases} \text{“max”} & Z_k = c^k x, \quad k = 1, 2, \ldots, p \\ \text{s.c.} & Ax \leq b, \\ & x \in \{0,1\}^n. \end{cases}$$

Où : A, b et c sont des matrices d'entiers ce qui n'est pas restrictif en pratique. Si, au départ, A, b et c sont formées de nombres rationnels on les ramène à des matrices d'entiers en multipliant tous les coefficients par le plus petit commun multiple de leurs dénominateurs.

2.3 Quelques problèmes MOCO classiques

Il existe de nombreux problèmes de type MOCO, dans cette section nous définirons brièvement quelques exemples (d'autres exemples peuvent être trouvés dans [52]).

2.3.1 Problème d'affectation multi-objectif MOAP

Le problème d'affectation (Assignment Problem AP) est un problème particulier de programmation linéaire en variables binaires. Il consiste à affecter n agents (personnes, équipements, lieux,...) à n taches (travaux, activités, services,...). Par exemple : affecter des ouvriers à des emplois, des enseignants à des classes, des produits à des dépôts, des équipages à des vols, etc.

Il s'agit donc de définir une bijection de $\{1, \ldots, n\}$ sur $\{1, \ldots, n\}$ ou encore une permutation de n objets.

On définit par $i = 1, \ldots, n$ les agents et par $i = 1, \ldots, n$, on définit la variable x_{ij} comme suit :

$$x_{ij} = \begin{cases} 1 & \text{si l'agent } i \text{ est affecté à la tâche } j, \\ 0 & \text{sinon.} \end{cases}$$

Les contraintes du problème d'affectation s'écrivent donc comme suit :

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n.$$

(L'agent i doit être affectée à une et une seule tâche).

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n.$$

(La tâche j ne peut être affectée qu'à un et un seul agent).

Dans le cas multi-objectif, l'affectation de l'individu i à l'activité j occasionne des coûts d'affectation $c_{ij}^k, k = 1, \ldots, p,\ i = 1, \ldots, n,\ j = 1, \ldots, n$. Ce coefficient peut être un coût associé à l'affectation d'un individu i à une tâche j, une durée d'affectation, un facteur de fiabilité, etc.

Ainsi, le problème d'affectation multi-objectif (Multiple Objective Assignment Problem MOAP) peut s'écrire comme suit :

$$(MOAP) = \begin{cases} \text{“min”} & Z_k = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^k x_{ij}, \quad k = 1, \ldots, p \\ \text{s.c.} & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n, \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n, j = 1, \ldots, n. \end{cases}$$

2.3.2 Problème de voyageur de commerce multi-objectif MOTSP

Le problème du voyageur de commerce (Traveling Salesman Problem TSP) est certainement le problème le plus connu et le plus étudié des problèmes combinatoire NP-difficile [68]. Sa définition est simple, étant donné n villes , un voyageur de commerce doit réaliser une tournée en visitant une et une seule fois toutes les villes tout en minimisant le coût du déplacement. Ce problème se modélise sous forme d'un graphe non-orienté $\mathcal{G}(\mathcal{S}, \mathcal{A})$, où $\mathcal{S} = 1, \ldots, n$, est l'ensemble des sommets du graphe qui représentent les villes et $\mathcal{A} = (i, j),\ i, j \in \mathcal{S},\ i \neq j$ l'ensemble des arrêtes qui représentent les routes entre deux villes i et j.

La version multi-objectif de ce problème (Multiple Objective Traveling Salesman Problem MOTSP) considère que le déplacement entre les n villes engendre plusieurs frais (distance, coût, risque, etc.). Alors pour chaque paire de sommets $(i, j),\ i, j \in \mathcal{S},\ i \neq j$, on associe un vecteur coût de p valeurs $c_{ij}^k, \quad k = 1, \ldots, p$. L'objectif de ce problème est de rechercher un circuit hamiltonien qui soit de coût minimum. Soit la variable x_{ij} telle que :

$$x_{ij} = \begin{cases} 1 & \text{si le trajet est affecté de la ville } i \text{ vers la ville } j, \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, le problème MOTSP se formule mathématiquement comme suit :

$$(MOTSP)\begin{cases} \text{“min”} & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^{k}x_{ij}, \quad k=1,\ldots,p \\ \text{s.c.} & \sum_{i=1}^{n} x_{ij}=1, \quad j=1,\ldots,n, \\ & \sum_{j=1}^{n} x_{ij}=1, \quad i=1,\ldots,n, \\ & \sum_{i\in V}\sum_{j\in\mathcal{S}\setminus V} x_{ij}\geq 1, \quad \forall V\subset\mathcal{S}, \text{ avec } 2\leq|V|<|\mathcal{S}|-2, \\ & x_{ij}\in\{0,1\}, \quad \forall i,j. \end{cases}$$

Même si ce problème est l'un des plus connus en optimisation combinatoire multi-objectif, facile à modéliser, mais n'a pas encore reçu l'attention qu'il mérite vu sa difficulté de résolution qui réside dans le fait qu'il ne peut y avoir de sous-tours dans la permutation, vu son caractère cyclique [132].

2.3.3 Problème du plus court chemin multi-objectif MOSPP

Le problème du plus court chemin multi-objectif (Multiple Objective Shortest Path Problem MOSPP) est défini par un graphe orienté acyclique $\mathcal{G}=(\mathcal{S},\mathcal{A})$, où $\mathcal{S}=1,\ldots,n$, est un ensemble de sommets et $\mathcal{A}$ est un ensemble d'arcs reliant des couples de sommets. A chaque arc (i,j), on affecte un vecteur de p valeurs positives $c^k(i,j)$, $k=1,\ldots,p$, pouvant exprimer un coût, une durée, une distances, etc. allant du sommet i vers le sommet j. Soient t et s deux sommets fixés de $\mathcal{S}$, appelés respectivement sommet de départ et sommet d'arrivée. l'ensemble de tous les chemins reliant t et s est noté $\mathcal{C}$. A chaque chemin $K\in\mathcal{C}$ correspond donc les valeurs $Z_k(K)=\sum_{(i,j)\in K} c^k(i,j)$, $k=1,\ldots,p$. Le problème du plus court chemin multi-objectif consiste à minimiser ces p valeurs.

Soit la variable x_{ij} telle que :

$$x_{ij}=\begin{cases} 1 & \text{si l'arc } (i,j)\in K, \\ 0 & \text{sinon.} \end{cases}$$

La formulation mathématique du MOSPP peut s'écrire comme suit :

$$(MOSPP)=\begin{cases} \text{“min”} & Z_k(\mathcal{X})=\sum_{(i,j)\in A} c^k(i,j)x_{ij}, \quad k=1,\ldots,p \\ \text{s.c.} & \sum_{j\in N} x_{ij}-\sum_{j\in N} x_{ji}=\begin{cases} 0 & \text{si } i\neq s,t, \\ 1 & \text{si } i=s, \\ -1 & \text{si } i=t. \end{cases} \end{cases}$$

2.3.4 Problème de couverture multi-objectif MSCP

Le problème de couverture multi-objectif (Multiobjective Set Covering Problem) peut être défini comme suit : on considère un ensemble de localités qu'on doit atteindre et qu'il existe n routes ayant chacune p coûts c_j^k, $j = 1, \ldots, n$, $k = 1, \ldots, p$ (p coûts associés à chaque route) à parcourir pour y arriver. L'objectif est donc de trouver les "meilleures" routes possibles pour atteindre toutes les localités, c'est à dire déterminer l'ensemble de routes des coûts minimaux couvrant toutes les localités.

La notion de couverture est définie mathématiquement comme suit : Soit $I = \{1, 2, \ldots, m\}$, un ensemble de m éléments, $i \in I$. Considérons n sous-ensembles H_j de I,
$j \in J = \{1, 2, \ldots, n\}$. Une couverture de I est un ensemble de H_j tel que $j \in J'$ vérifiant la condition $\bigcup_{j \in J'} H_j = I$.

Soit c_j^k, $k = 1, \ldots, p$, le coût associé à H_j pour l'objectif k, le problème multi-objectif de couverture peut s'écrire de la manière suivante :

$$(MOSCP) \begin{cases} \text{"min"} & Z_k = \sum_{j=1}^{n} c_j^k x_j, \quad k = 1, \ldots, p \\ \text{s.c.} & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in I, \\ & x_j \in \{0, 1\}, \quad \forall j \in J, \end{cases}$$

où : $a_{ij} = \begin{cases} 1, & \text{si } i \in H_j, \\ 0, & \text{sinon.} \end{cases}$, $x_j = \begin{cases} 1, & \text{si } H_j \text{ appartient à la couverture,} \\ 0, & \text{sinon.} \end{cases}$

et $c_j^k > 0$, $\forall (k, j)$.

2.3.5 Problème du sac à dos multi-objectif MOKP

Le problème du sac à dos (Knapsack Problem KP) est l'un des problèmes d'optimisation combinatoire les plus connus. Il apparaît comme un sous-problème dans de nombreuses applications du monde réel comme la logistique, l'industrie, l'économie, le transport et bien d'autres domaines. Ce problème a reçu une attention particulière dans la littérature pour les raisons suivantes : sa formulation simple, où il ne possède qu'une seule contrainte et que plusieurs interprétations en sont possibles (problème de chargement, d'investissement, etc.). Une étude large de ce problème est donnée dans [93, 79].

Sa version multi-objectif (Multiple Objective Knapsack problem MOKP) sera étudiée explicitement dans le chapitre suivant.

2.4 Complexité des problèmes MOCO

Les problèmes d'optimisation combinatoire multi-objectifs (MOCO) sont particulièrement complexes. En effet, la complexité des problèmes d'optimisation, due au phénomène d'explosion combinatoire, est combinée à de nouvelles difficultés spécifiques aux problèmes d'optimisation multi-objectifs. Dans de nombreux MOCO, il est très difficile de déterminer un ensemble complet de solutions efficaces. Malgré le fait que les solutions supportées sont obtenues par la résolution d'un programme linéaire pondéré si l'on dispose d'un algorithme efficace pour le résoudre. Mais aussi, le calcul des solutions non supportées est également très onéreux. Cependant, la détermination de ces solutions nécessite généralement la résolution de problèmes NP-difficile, même si le problème mono-objectif considéré peut être résolu en temps polynomial. Par conséquent, l'une des difficultés pratiques de la résolution des MOCO réside dans la détermination de l'ensemble des solutions non supportées. En outre, des expériences ont montré que les solutions non supportées, non seulement sont plus difficiles à obtenir, mais qu'elles sont plus nombreuses que les solutions supportées [138]. Pour conclure, tant sur le plan théorique qu'expérimental, la majorité des problèmes MOCO sont reconnus comme étant NP-difficiles [49].

2.5 Quelques méthodes de résolution des problèmes MOCO

La plupart des problèmes MOCO sont NP-difficiles [117]. Pour leur résolution, des méthodes exactes, ainsi que des méthodes approchées ont été proposées. Dans cette section, nous exposerons une revue rapide de quelques méthodes exactes et approchées, le but ici n'est pas de présenter des méthodes spécifiques à des problèmes donnés, ni de faire une liste exhaustive des méthodes existantes. Pour avoir une vue plus globale, le lecteur peut se référer à [52].

2.5.1 Méthodes exactes

Peu de méthodes de résolution exactes ont été développées pour les problèmes MOCO. Cependant, des méthodes spécifiques aux problèmes qui tiennent compte de leur particularités ont été proposées. Cette section présente quelques algorithmes génériques classiques utilisés pour résoudre les problèmes MOCO.

2.5.1.1 La recherche dichotomique

La recherche dichotomique, proposée la première fois par Aneja et Nair [6] dédiée à la résolution exacte du problème de transport bi-objectif (à minimiser) qui peut être appliquée à n'importe quel autre problème MOCO bi-objectif. Elle offre un schéma d'application de l'agrégation linéaire permettant d'obtenir que les solutions supportées en ignorant, consciemment ou non, les solutions non supportées. Cette méthode consiste à explorer de façon dichotomique des intervalles de front de plus en plus petits.

Soit SE l'ensemble de solutions efficaces supportées. La méthode est initialisée par la détermination de deux solutions X^1 et X^2 lexicographiquement optimales par rapport aux objectifs $\text{lexmax}_{x\in\mathcal{X}}\,(Z^1(x), Z^2(x))$ et $\text{lexmax}_{x\in\mathcal{X}}\,(Z^2(x), Z^1(x))$ respectivement. Durant la recherche dichotomique, des solutions de l'ensemble SE sont triées par ordre croissant des valeurs Z_1, une itération de cet algorithme consiste à résoudre un problème paramétrique (P_λ) défini par $\lambda_1 = Z_2^r - Z_2^s$ et $\lambda_2 = Z_1^s - Z_1^r$ où Z^r et Z^s sont deux points consécutifs avec $Z_1^r < Z_1^s$ et donc $Z_2^r > Z_2^s$. Le vecteur λ correspond à la normale de la droite joignant les points Z^r et Z^s. Cet algorithme est initialisé par $X^r = X^1$ et $X^s = X^2$. Toutes les solutions obtenues du problème paramétrique (P_λ) sont énumérées, ces dernières peuvent être extrêmes ou non-extrêmes. Dans une situation particulière où le problème paramétrique admet une unique solution, on a $X^1 = X^2$.

Soit $\{X^t, \quad t \in T\}$ l'ensemble des solutions optimales du problème (P_λ), où T est l'ensemble d'indices tel que $|T|$ est le nombre de solutions optimales. Deux cas sont possibles :

(a.) Si $\{X^r, X^s\} \cap \{X^t, \quad t \in T\} = \emptyset$.
Dès lors, les solutions X^t sont des nouvelles solutions supportées et elles sont inclues dans l'ensemble des solutions supportées extrêmes SE_E.

(b.) Si $\{X^r, X^s\} \subset \{X^t, \quad t \in T\}$.
Dans ce cas, si X^r et X^s sont les deux seules solutions X^t, il n'y a pas d'autres solutions supportées entre X^r et X^s ; dans le cas contraire, les autres solutions X^t sont des nouvelles solutions supportées non extrêmes, elles sont ajoutées à SE_{NE}.

Cette méthode prend fin lorsque toutes les paires (X^r, X^s) de l'ensemble SE_E sont examinées. A la fin on obtient $SE = SE_E \cup SE_{NE}$.

La méthode dichotomique peut donc être appliquée à n'importe quel problème d'optimisation car elle n'utilise aucune spécificité du problème de transport. Elle est souvent utilisée pour la détermination d'un ensemble de solutions efficace supportées. La figure (Fig. 2.1) illustre le fonctionnement de la recherche dichotomique.

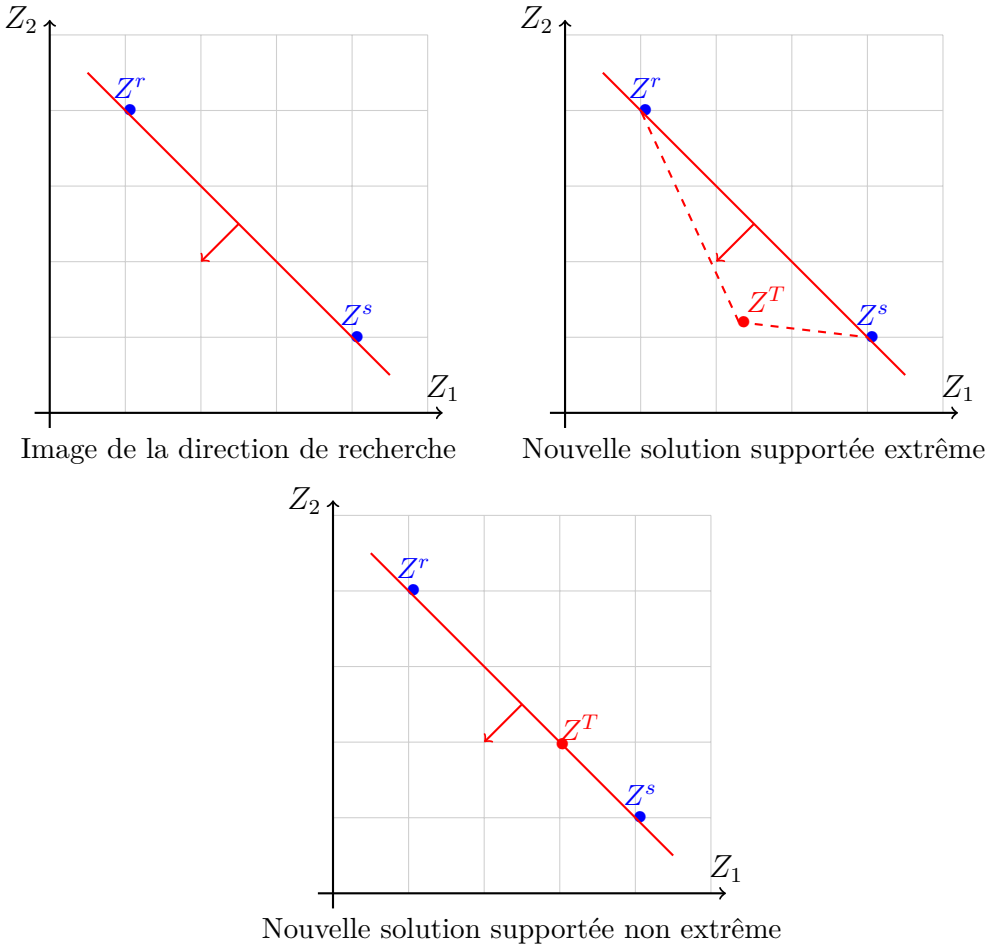


Fig. 2.1 : Illustration de la recherche dichotomique

2.5.1.2 Procédure de séparation et évaluation

Les procédures de séparation et d'évaluation sont des méthodes classiques dans le domaine de l'optimisation mono-objectif. L'idée générale de ces procédures est de construire implicitement l'ensemble des solutions réalisables pour en obtenir celles qui sont efficaces. Dans le contexte de l'optimisation combinatoire, la séparation et l'évaluation se déroulent ainsi :

- La séparation : elle consiste à sélectionner une variable x_k dont la valeur est inconnue puis à partager l'espace des solutions réalisables en deux sous-espaces dans lesquels x_k prend la valeur 1 ou 0, respectivement. Le choix de la variable à fixer lors de la procédure de partitionnement est appelé stratégie de branchement. Elle peut être statique (fixation des variables dans un ordre pré-établi) ou dynamique (utilisation des informations de l'énumération pour faire le choix).

- L'évaluation : l'objectif de cette étape est de mesurer la pertinence d'un sous-espace avant de l'explorer, afin de déterminer s'il peut potentiellement contenir des solutions efficaces ou non. Si ce n'est pas le cas, la poursuite de la séparation dans ce sous-espace est arrêtée. Ce processus est exécuté de manière répétitive pour chaque sous-espace, jusqu'à ce que chacun d'entre eux ne contienne qu'une seule et unique solution.

Les techniques de séparation et évaluation sont fréquemment intégrées dans d'autres méthodes exactes, telles que la méthode "MOMIX" [129], les méthodes en deux phases [138, 71], etc.

D'autres méthodes ont été proposées telles que la méthode ϵ-contrainte et la méthode en deux phases, que nous avons déjà discutées dans le chapitre précédent (voir la sous-section (1.3.3)). Plusieurs variantes de la méthode ϵ-contrainte ont été également développées [44, 88]. Des méthodes qui utilisent des techniques de scalarisation en reformulant le problème MOCO en un problème mono-objectif, tels que l'algorithme de Benson [15] et la méthode des contraintes élastiques [51]. Des méthodes de programmation dynamique spécifiques à certains problèmes ont néanmoins été adaptées avec succès, telles que le problème du plus court chemin multi-objectif [94] et de sac à dos multi-objectif [83, 27, 10].

2.5.2 Méthodes non exactes

Pour la résolution des problèmes MOCO de grande taille, il est parfois inévitable de faire recours aux méthodes de résolution non exactes telles que les heuristiques ou les méta-heuristiques. Contrairement aux méthodes exactes, les méthodes non exactes aussi appelées approchées n'offrent pas une garantie absolue de trouver toutes les solutions Pareto optimales. Elles visent à fournir une approximation aussi précise que possible de cet ensemble de solutions potentiellement efficaces et cela pendant un temps de calcul acceptable. Le choix dépend souvent de la nature spécifique du problème MOCO à résoudre et des préférences de l'utilisateur en termes de trade-offs entre la qualité de la solution et le temps de calcul. Dans ce qui suit, nous présentons quelques heuristiques et méta-heuristiques les plus adaptées au problèmes MOCO.

2.5.2.1 Les heuristiques

Une heuristique est définie comme une technique qui cherche à obtenir des bonnes solutions (c'est-à-dire proches de l'optimal), tout en maintenant un coût computationnel raisonnable. Toutefois, elle ne peut garantir formellement l'optimalité, fournir une indication précise de la

proximité d'une solution réalisable avec l'optimalité ou dans certains cas, assurer la faisabilité. Souvent, les heuristiques sont spécifiques au problème, de sorte qu'une méthode qui fonctionne pour un problème ne peut pas être utilisée pour résoudre un problème différent [53].

Les heuristiques pour les problèmes MOCO peuvent souvent être dérivées des heuristiques pour la version mono-objectif d'un problème d'optimisation combinatoire [53]. Parmi ces méthodes, on décrit deux d'entre elles qui sont particulièrement célèbres.

Méthodes gloutonnes : Ces méthodes consistent à créer des solutions pas à pas d'une manière constructive, en exploitant les particularités du problème. Dans le contexte multi-objectif, l'algorithme glouton vise à optimiser simultanément plusieurs objectifs conflictuels. A chaque étape, il sélectionne la solution qui offre le meilleur compromis entre les objectifs. Son efficacité dépend fortement du problème spécifique et de la nature des objectifs impliqués, en particulier dans des espaces de problèmes complexes et de grande dimension. Considérons $\mathcal{X} \subseteq 2^A$ pour un ensemble fini A et supposons, pour simplifier l'exposition, que si $x \in \mathcal{X}$ et $x' \subseteq x$, alors $x' \in \mathcal{X}$. L'algorithme glouton pour les problèmes multi-objectif est résumé dans (Algo. 2.1).

Algo. 2.1 : Algorithme glouton

Input ↓ : A : ensemble fini.

Output ↑ : $\tilde{E}$, un ensemble des solutions potentiellement efficaces.

Initialisation :
$\tilde{E} \leftarrow nondom(A)$; % $nondom(A)$: les solution dans A non dominées.

Tant que $\exists x \in \tilde{E}, a \in A \setminus x$ *tel que* $x \cup a \in \mathcal{X}$ **faire**
  Pour *tout* $x \in \tilde{E}, a \in nondom(A \setminus x)$ **faire**
    Si $x \cup a \in \mathcal{X}$ **alors**
      $\tilde{E} \leftarrow \tilde{E} \cup \{x \cup a\}$;
    Fsi
  Fpour
Ftq
$\tilde{E} \leftarrow nondom(\tilde{E})$;

Cet algorithme a été utilisé pour résoudre beaucoup de problème MOCO, à savoir, le problème de l'arbre couvrant de poids minimum [35] et le prolème de capital budgeting [112].

La recherche locale : Le principe fondamental d'une recherche locale consiste à partir d'une configuration initiale (ou d'une affectation complète) et, à travers un processus itératif, à remplacer cette configuration par une meilleure solution tirée de son voisinage défini. L'objectif est d'améliorer progressivement la configuration actuelle dans l'espoir d'atteindre une solution optimale. Toutes les approches de recherche locale utilisent la notion de voisinage. Un aspect fondamental de ces approches est donc la détermination de ce voisinage. Le voisinage d'une solution courante x, que l'on note $V(x)$, est un sous-ensemble de solutions qu'il est possible d'atteindre par une série de transformations données. Il existe une infinité de manières de choisir V, il faut adapter ce choix au problème, c'est-à-dire choisir la meilleure fonction V selon le problème considéré. Le principe de la recherche locale pour les problèmes multi-objectif est décrit dans l'algorithme (Algo. 2.2).

Algo. 2.2 : La recherche locale

Input ↓ : x, solution réalisable.
Output ↑ : $\tilde{E}$, l'ensemble des solutions potentiellement efficaces locales.
Initialisation :
$\tilde{E} \leftarrow \mathcal{X}$, $S \leftarrow x$;
Répéter
  Pour *tout* $x' \in S$ **faire**
    Pour *tout* $x \subseteq V(x')$ **faire**
      Si $(x \not\succ x'$ *et* $x \not\succ x')$ **alors**
        $S \leftarrow S \cup \{x\}$;
      Fsi
      Si $(x > x')$ **alors**
        $S \leftarrow S \cup \{x\} \setminus \{x'\}$;
      Fsi
    Fpour
  Fpour
  $S \leftarrow \text{nondom}(S)$;
Jusqu'à $S = \tilde{E}$;

Parmi les applications de cette méthode dans la résolution des problèmes MOCO, on cite les références [113, 5]. Il est possible de combiner l'heuristique de recherche locale et l'heuristique gloutonne. L'idée est de construire d'abord des solutions à l'aide de l'algorithme glouton et d'utiliser la recherche locale pour améliorer les solutions ou générer des solutions potentiellement plus efficaces. L'algorithme glouton multi-objectif et l'algorithme de recherche locale multi-objectif sont bien sûr des des modèles généraux qui peuvent être appliqués à une grande variété de problèmes et plus particulièrement, aux problèmes de type MOCO.

L'algorithme glouton et l'algorithme de recherche locale peuvent être combinés. L'idée est de construire d'abord des solutions à l'aide de l'approche gloutonne et d'utiliser la recherche locale pour améliorer les solutions ou générer des solutions potentiellement plus efficaces. Ces deux algorithmes sont des modèles généraux qui peuvent être appliqués à une grande variété de problèmes.

2.5.2.2 Les méta-heuristiques

Les méta-heuristiques sont des techniques puissantes applicables généralement à un grand nombre de problèmes. Une métaheuristique fait référence à une stratégie maîtresse itérative qui guide et modifie les opérations des heuristiques subordonnées en combinant intelligemment différents concepts pour explorer et exploiter l'espace de recherche [53]. Différentes méta-heuristiques ont été développées exploitant la stratégie de recherche locale dans leur fonctionnement, parmi ces méthodes, on peut citer les méthodes suivantes.

Le recuit simulé : Le recuit simulé proposé dans [80] est une méthode d'optimisation inspirée des simulations de la mécanique statistique. Cette méthode commence avec une solution initiale et explore aléatoirement son voisinage pour trouver une autre solution. Sa particularité réside dans sa capacité à explorer des solutions voisines de qualité inférieure avec une probabilité non nulle, permettant ainsi d'éviter les optima locaux. Le processus du recuit simulé consiste en une série d'itérations visant à trouver des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui peuvent détériorer la fonction de coût. Cette méthode a été appliquée pour la première fois dans le cadre multi-objectif par Serafini en 1992. Toutes les méthodes basées sur le recuit simulé pour l'optimisation multi-objectif développées depuis lors sont toujours étroitement liées à la méthode originale à objectif unique. Elles étendent l'algorithme à objectif unique pour traiter la notion d'efficacité. La méthode du recuit simulé est donnée par l'algorithme (Algo. 2.3).

Algo. 2.3 : Recuit simulé

Input ↓ :

$\downarrow T_0, \alpha, Seuil, it_{palier}, V, f, x_0$.

Output ↑ : x une solution optimale.

Initialisation :

$x \leftarrow x_0$;

$T \leftarrow T_0$;

Tant que $T > Seuil$ **faire**

  Pour *nombre-itérations* ***de*** 1 ***à*** it_{palier} **faire**

    Choisir un point x' de $V(x)$;

    $\Delta(f) \leftarrow f(x') - f(x)$;

    Si $\Delta(f) < 0$ **alors**

      $x \leftarrow x'$;

    Sinon

      Choisir r de manière aléatoire entre 0 et 1 ;

      Si $r < e^{-\Delta(f)/T}$ **alors**

        $x \leftarrow x'$;

      Fsi

    Fsi

  Fpour

  $T \leftarrow \alpha T$;

Ftq

$x^* \leftarrow x$;

Avec :

— T_0 : la température initiale,

— $Seuil$: le seuil minimal que la température peut atteindre,

— α : le coefficient de décroissance de la température (paramètre de refroidissement),

— it_{palier} : le nombre d'itérations à effectuer dans un palier,

— V : la fonction de voisinage,

— f : la fonction d'évaluation,

— x_0 : la configuration initiale servant de point de départ à l'algorithme.

Dans cet algorithme, le nombre d'itérations (it_{palier}) devant être atteint pour effectuer un changement de palier est fixe. Dans certaines versions de cet algorithme, it_{palier} peut varier en fonction de la température. Dans le cadre des problèmes MOCO, différentes méthodes de recuit simulé ont été développées (voir [120, 132, 134, 61]).

La recherche tabou : La recherche tabou est introduite dans [65] en 1986. L'idée est d'explorer l'espace des solutions en interdisant certains mouvements afin d'éviter de rester coincé dans des solutions sous-optimales. La recherche tabou examine toutes les configurations appartenant à un voisinage $V(x)$ de x et retient la meilleure x_0 même si x_0 est plus mauvaise que x. Cependant, cette stratégie peut entraîner des cycles. Pour pallier ce problème, on mémorise les k dernières configurations visitées dans une mémoire à court terme et on interdit de retenir toute configuration qui en fait déjà partie. Cette mémoire, appelée la liste tabou, est une des composantes essentielles de la méthode. En effet, elle permet d'éviter tous les cycles de longueurs inférieures à k, k étant un paramètre déterminé en fonction du problème. Plusieurs méthodes de résolution dans le cadre multi-objectif, notamment pour les problèmes MOCO sont basées sur la méthode de recherche tabou telles que [60, 62, 4]. L'algorithme (Algo. 2.4) résume le fonctionnement de cette méthode.

Algo. 2.4 : Recherche tabou

Input ↓ :
↓ x_0 : Solution initiale.
↓ L : Liste Tabou vide de taille k.
Output ↑ :
Solution optimale.
Tant que *Condition d'arrêt n'est pas satisfaite* **faire**
  x : est la solution courante ;
  Trouver le meilleur voisin $x' \in V(x)$;
  Mettre à jour L ;
  Au besoin, appliquer l'intensification et/ou la diversification ;
Ftq

Algorithmes évolutionnaires (AE) Les algorithmes évolutionnaires (AE) sont généralement composés de trois éléments fondamentaux :

- Une population constituée d'individus (également appelés chromosomes) qui représentent, via un codage, les solutions réalisables.
- Une fonction de fitness qui évalue l'adaptation des individus à leur environnement.
- Un processus d'évolution qui fait évoluer la population en générant et éliminant des individus.

Parmi les AE, nous nous intéresserons uniquement aux algorithmes génétiques (AG) [66, 64]. Ces algorithmes introduits par Holland [73] en 1975, sont des techniques d'optimisation basées sur le concept de sélection naturelle. Pour un problème mono-objectif, un codage

naturel des solutions est considéré et un opérateur de croisement qui n'a besoin que de deux parents pour créer un seul enfant (nom standard donné au résultat du croisement) est utilisé. Dans les AG, une population est conçue selon les principes de la sélection, de la reproduction et de la mutation afin d'obtenir une population bien diversifiée et contenant au moins une solution de bonne qualité. Les individus de la population sont évalués à l'aide d'une fonction de fitness qui est souvent basée sur l'objectif à optimiser. Les algorithmes génétiques peuvent être facilement adaptés aux problèmes multi-objectif. En effet, puisqu'une population de solutions est déjà gérée, il suffirait de remplacer la population P par un ensemble non-dominé, c'est-à-dire un ensemble ne contenant que des solutions potentiellement efficaces. Cependant, cette option pourrait être trop élitiste, et en général, deux populations sont gérées : la population P et l'ensemble élitiste, c'est-à-dire l'ensemble d'approximation $\tilde{E}$ contenant les meilleures solutions potentiellement efficaces trouvées. Cela donne l'algorithme MOGA présenté dans (Algo. 2.5) en utilisant les notations suivantes :

- P_0 : une population initiale de taille nP, composée de solutions x dans $\mathcal{X}$ (supposant un codage naturel des solutions).
- $F(x)$: une fonction de fitness.
- $SP(P)$: une procédure de sélection.
- $SP1(P)$: une procédure de sélection du premier parent.
- $SP2(P)$: une procédure de sélection du deuxième parent.
- $CO(x^1, x^2)$: un opérateur de croisement.
- $M(x)$: un opérateur de mutation.
- nP' : le nombre de reproductions.

Algo. 2.5 : MOGA (Algorithme Génétique Multi-Objectif)

Input ↓ : P_0,$F(x)$, $CO(x^1, x^2)$, nP', $M(x)$.

Output ↑ : Une approximation $\tilde{E}$ de l'ensemble efficace E.

Initialisation de la population P :

$P \leftarrow P_0$;

Initialisation de $\tilde{E}$:

pour chaque *chaque* $x^i \in P$ **faire**

  AjouterSolution($\tilde{E}$ ↕, x_i ↓, $f(xi)$ ↓) ;

fin

Répéter

  Génération d'une nouvelle population P' à partir de P et $\tilde{E}$;

  $P' \leftarrow \emptyset$;

  pour $i = 1$ *à* nP' **faire**

    Sélection du premier parent

    $x^1 \leftarrow SP1(\{P \cup \tilde{E}\})$;

    Sélection du deuxième parent

    $x^2 \leftarrow SP2(\{P \cup \tilde{E}\})$;

    Reproduction entre les deux parents à travers l'opérateur de croisement

    $x^3 \leftarrow CO(x^1, x^2)$;

    AjouterSolution($\tilde{E}$ ↕, x^3 ↓, $f(x^3)$ ↓) ;

    Éventuellement, l'enfant x^3 est muté

    $x^4 \leftarrow M(x^3)$;

    AjouterSolution($\tilde{E}$ ↕, x^4 ↓, $f(x^4)$ ↓) ;

    $P' \leftarrow P' \cup \{x^4\}$;

  fin

  Sélection entre $\{P \cup P' \cup \tilde{E}\}$ pour mettre à jour la population P

  $P \leftarrow S(\{P \cup P' \cup \tilde{E}\})$;

Jusqu'à *ce que le critère d'arrêt soit atteint*;

Parmi les nombreuses adaptations des algorithmes génétiques (AG) aux problèmes multi-objectifs (MO), nous présentons deux méthodes populaires : NSGA et NSGA-II.

NSGA (Non-dominated Sorting Genetic Algorithm) : Cet algorithme a été proposé par Srinivas et Deb [123] en se basant sur l'idée proposée par Goldberg [67] sur l'utilisation du concept de tri par dominance de Pareto. Son principe consiste à diviser la population

actuelle en plusieurs groupes, où les individus du $l^{ème}$ groupe constituent le $l^{ème}$ front de Pareto, puis on attribue une valeur unique d'efficacité F_l à tous les individus d'un même groupe. Cela signifie que les individus d'un même groupe ont les mêmes chances de se reproduire. La division de la population s'effectue, tout d'abord, en identifiant les individus non dominés pour former le premier groupe, et leur attribuer la valeur F_1. Ces individus sont ensuite retirés de la population. Dans la population restante, les individus non dominés sont identifiés et la valeur F_2 leur est attribuée, et ainsi de suite jusqu'à ce que tous les individus de la population aient une valeur. Pour toutes les procédures restantes (sélection, reproduction (croisement et mutation)), la méthode NSGA se comporte comme n'importe quel autre algorithme génétique. Les étapes de cette méthode sont décrites dans l'algorithme (Algo. 2.6).

Algo. 2.6 : NSGA (Non-dominated Sorting Genetic Algorithm)

Input ↓ : N : Taille de la population.
Output ↑ : Une approximation $\tilde{E}$ de l'ensemble efficace E.
Initialisation :
Créer une population initiale P_0 de taille N ;
Mettre dans $\tilde{E}$ les solutions efficaces ;
$P_t \leftarrow P_0$;
Tant que *Condition d'arrêt n'est pas validée* **faire**
  Partager la population P_t en différents fronts de Pareto ;
  Attribuer des valeurs d'efficacités F_l aux différents fronts de Pareto ;
  Appliquer la fonction de partage à tous les individus de la population P_t ;
  Appliquer l'opérateur de sélection ;
  Appliquer l'opérateur de croisement ;
  Appliquer l'opérateur de mutation ;
  $P_t \leftarrow$ la population actuelle ;
Ftq
Mettre les solutions efficaces dans $\tilde{E}$;

NSGA-II (Non-dominated Sorting Genetic Algorithm II) : La méthode NSGA-II, développée par Deb et *al.* en 2002 [40], est une version améliorée de NSGA. Cet algorithme utilise des techniques avancées d'élitisme et de diversification en intégrant un opérateur de sélection, basé sur un calcul de la distance de crowding pour la préservation de la diversité. L'algorithme assure qu'à chaque nouvelle génération, les meilleurs solutions rencontrées soient conservées.

Le principe du NSGA-II est très simple. En effet, à l'itération t, une population parents P_t et une population enfants Q_t de même taille N, P_t et Q_t sont combinées pour former une

seule population $P_t \cup Q_t$ de taille $2N$. Les individus de $P_t \cup Q_t$ sont triés suivant la relation de dominance de Pareto afin de déterminer les différents fronts F_j. Les meilleurs individus se trouvent donc toujours sur les premiers fronts. La nouvelle population parents P_{t+1} est formée par l'ajout des fronts $F_1, \ldots, F_j$ au fur et à mesure de telle sorte que la taille de leur union ne dépasse pas N. Si le nombre d'individus dans P_{t+1} est inférieur à N, alors la distance de crowding est appliquée au front suivant F_{j+1} pour compléter la population P_{t+1} avec les $(N - |P_{t+1}|)$ individus de F_{j+1}. Une fois que la population P_{t+1} a été formée, une nouvelle population enfant Q_{t+1} est générée en appliquant les opérateurs de sélection, de croisement et de mutation. Ce processus est répété jusqu'à ce que le test d'arrêt soit vérifié.
Les étapes de la méthode NSGA-II sont résumées dans l'algorithme (Algo. 2.7).

Algo. 2.7 : NSGA-II(Non-dominated Sorting Genetic Algorithm II)

Input ↓ : N : Taille de la population.
Output ↑ : Une approximation $\tilde{E}$ de l'ensemble efficace E.
Initialisation : Créer les populations initiales P_0 et Q_0 de taille N ;
$P_t \leftarrow P_0$, $Q_t \leftarrow Q_0$;
Tant que *Condition d'arrêt n'est pas vérifiée* **faire**
  Créer la population $R_t = P_t \cup Q_t$;
  Calculer les différents fronts de Pareto F_i de la population R_t ;
  $P_{t+1} \leftarrow \emptyset$;
  $i \leftarrow 1$;
  Tant que $|P_{t+1}| + |F_i| < N$ **faire**
    $P_{t+1} \leftarrow P_{t+1} \cup F_i$;
    $i \leftarrow i + 1$;
  Ftq
  Ajouter à P_{t+1} les $(N - |P_{t+1}|)$ meilleurs éléments de F_i selon la distance de crowding ;
  Appliquer des opérateurs de sélection, de croisement, et de mutation pour générer une nouvelle population Q_{t+1} ;
  $P_t \leftarrow P_{t+1}$;
  $Q_t \leftarrow Q_{t+1}$;
Ftq
Mettre dans $\tilde{E}$ les solutions efficaces ;

Il existent d'autre méthodes non exactes pour la résolutions des problèmes MOCO telles que les schémas d'approximation en temps entièrement polynomial FPTAS (Fully Polynomial

Time Approximation Schemes) à savoir les références [114, 7, 9, 131]. Les méthodes hybridant les algorithmes précédents comme Multiple Objective Genetic Tabu Search (MOGTS), Multiple Objective Genetic Local Search (MOGLS) et autres (voir [53]). Il est recommandé aux lecteurs de de se référer aux références bibliographiques [52, 53, 92, 72] pour obtenir des informations plus détaillées sur les méthodes approchées pour les problèmes MOCO.

2.6 Conclusion

Dans ce chapitre, nous avons rapporté dans un premier temps la définition de certains problèmes d'optimisation combinatoire multi-objectif MOCO, puis nous avons brièvement décrit quelques méthodes de résolution exactes et approchées les plus populaires dédiées à ce type de problèmes. Dans le chapitre suivant, nous allons aborder la problématique de l'optimisation d'un critère linéaire sur l'ensemble des solutions efficace d'un problème MOILP.

Chapitre 3

Optimisation sur l'ensemble des solutions efficaces

3.1 Introduction

La plupart des problèmes MOP contiennent des ensembles efficaces de cardinalités très importantes, ce qui peut embrouiller le décideur dans sa prise de décision. En outre, la génération de toutes les solutions efficaces du MOP est une tâche informatique difficile notamment pour les grandes tailles. Par conséquent, l'énumération de toutes les solutions efficaces du MOP n'est pas toujours recommandée. Pour faire face à cette situation, l'optimisation d'une fonction dite "fonction objectif principale" sur un ensemble efficace de MOP joue un rôle très important et constitue, dès lors, un sujet essentiel dans le cadre de l'optimisation multi-objectif. Ce problème est particulièrement difficile à résoudre et cela est dû à la nature non convexe de l'ensemble admissible (ensemble des solutions efficaces).

L'optimisation sur l'ensemble des solutions efficaces a été abordée la première fois en 1972 par Philip [106] et depuis, plusieurs chercheurs ont suivi cette voie en s'interessant à l'optimisation dans le domaine efficace d'un problème multi-objectif linéaire en variables continues MOLP où l'objectif principal était une fonction linéaire, nous citons en particulier les références : [12, 74, 39, 13, 14, 48, 116].

En 1992 Nguyen [101] a proposé pour la première fois une méthode pour optimiser sur l'ensemble des solutions efficace d'un problème multi-objectif en variables discrètes, où seulement une borne supérieure de la valeur optimale de la fonction objectif est fournie par cette

méthode. En 2004, Chaabane [28] a présenté une méthode qui consiste à la recherche dans l'espace des critères en optimisant une somme pondérée des fonctions objectifs initiales. Une autre méthode a été développée en 2006 par Abbas et chaabane [30] basée sur l'exploration de l'espace de decisions. Différents types de coupes sont utilisées successivement à chaque itération pour réduire le domaine de recherche tout en améliorant la valeur de la fonction objectif jusqu'à ce que le problème soit irréalisable. Jorge [75] a développé un algorithme itératif où à chaque étape de résolution, le critère principal est optimisé sur le domaine restreint en introduisant progressivement des contraintes pour éliminer les solutions dominées par la solution courante, ce processus se répète jusqu'à ce qu'une solution optimale non dominée soit finalement trouvée. Deux variantes de cette méthode ont depuis été proposées [22, 89]. Ces deux méthodes traitent le cas d'une fonction objectif principale exprimée sous forme d'une combinaison linéaire des objectifs. En 2010, Chaabane et Pirlot [33] ont proposé une méthode qui procède dans l'espace des critère. Puis en 2012, Chaabane et *al.* [31] ont développé une méthode où ils ont utilisé la norme de Tchebechev pour tester l'efficacité des solutions et pour éviter de passer par des sous-programmes utilisés dans les méthodes précédentes. Plus récemment, Prerna et Sharma [107] ont développé deux algorithmes dans lesquels une série de programmes linéaires de même dimension sont résolus à chaque étape. Le but de toutes ces méthodes est de produire une solution optimale pour l'objectif principal sans avoir à énumérer explicitement toutes les solutions efficaces.

Dans la suite de ce chapitre, nous exposons la formulation mathématique du problème ainsi que quelques résultats fondamentaux concernant l'optimisation d'un critère linéaire sur l'ensemble des solutions efficaces d'un problème linéaire multi-objectif en variables discrètes. Par la suite, nous présentons deux algorithmes de référence dans ce domaine : celui de Jorge [75] et celui de Chaabane et Pirlot [33]. Afin de tenir le lecteur informé des dernières avancées dans ce domaine, nous présentons également quelques-unes des méthodes les plus récentes, notamment celles de Boland et *al.* [22], de Lokman [89] et de Prerna et Sharma [107].

3.2 Formulation du problème

Le problème d'optimisation d'un critère linéaire sur l'ensemble des solutions efficaces d'un problème MOILP peut se formuler comme suit :

$$(P_E)\begin{cases} \max & \phi(x) = dx \\ \text{s.c.} & x \in E, \end{cases} \tag{3.1}$$

où d est un vecteur ligne de dimension n qui a pour $j^{ième}$ composante le nombre entier d_j et E représente l'ensemble des solutions efficaces du problème multi-objectif linéaire à variables entières suivant :

$$(MOILP)\begin{cases} \text{“ max ”} & Z_i = c^i x, \quad i = 1, 2, \ldots, p \\ \text{s.c.} & x \in D, \end{cases} \tag{3.2}$$

où $D = S \cap \mathbb{Z}^n$, $\mathbb{Z}$ étant l'ensemble des nombres entiers relatifs et $S = \{x \in \mathbb{R} | Ax \leq b,\ x \geq 0\}$, $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m, n \in \mathbb{N}$. Soit (P_R) le problème relaxé :

$$(P_R)\begin{cases} \max & \phi(x) = dx \\ \text{s.c.} & x \in D, \end{cases} \tag{3.3}$$

avec $D = S \cap \mathbb{Z}^n, S = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$.

Théorème 3.1 (Caractérisation d'une solution efficace [47]) *Soit x^* une solution quelconque de D. x^* est efficace pour le problème $(MOILP)$ si et seulement si la*

valeur optimale de la fonction objectif Θ est nulle dans le programme de programmation linéaire mixte suivant :

$$(P_\Theta(x^*))\begin{cases} \max & \Theta = \sum_{i=1}^{p} \psi_i \\ s.c. & Cx - I\Psi = Cx^*, \\ & x \in D,\ \psi_i \in \mathbb{R}^+,\ \forall i \in \{1, \ldots, p\}, \end{cases} \tag{3.4}$$

où C est la matrice d'ordre $p \times n$ dont la $i^{ème}$ ligne correspond à $c^i, i \in \{1, \ldots, p\}$, I est la matrice identité d'ordre p et $\Psi = (\psi_i)_{i \in \{1,\ldots,p\}}$.

Ce théorème est utilisé comme un test d'efficacité des solutions dans les algorithmes suivants.

3.3 Quelques méthodes d'optimisation sur l'ensemble efficace d'un MOILP

3.3.1 Méthode de Jorge

Cette méthode est introduite en 2009 par Jorge [75]. Elle est itérative, l'objectif est de produire une solution optimale au problème (P_E) sans avoir à rechercher toutes les solutions efficaces du problème $(MOILP)$. Pour ce faire, le problème relaxé (P_R) est d'abord résolu, ce n'est que dans quelques cas spécifiques qu'une solution optimale de (P_R) donne une solution optimale de (P_E). Alors, si ce n'était pas le cas, une solution efficace est obtenue en renvoyé

par le test d'efficacité. Ensuite, dans chaque itération, l'objectif principal est optimisé dans un domaine plus réduit en ajoutant les coupes proposées par Sylva et Crema [125] au problème (P_R) pour fournir de nouvelles solutions qui ne sont pas dominées par les points précédemment testés, jusqu'à ce que la solution optimale soit finalement trouvée. La méthode de Jorge est présentée dans l'algorithme (Algo. 3.1).

Algo. 3.1 : Algorithme de Jorge

Input ↓ :
$\downarrow A, \downarrow b$ et $\downarrow C$; % les paramètres du MOILP
$\downarrow d$; % le paramètre du critère principal
Output ↑ :
$\uparrow x_{opt}$; % solution optimale de (P_E)
Initialisation : $\phi_{inf} \leftarrow -\infty$, $x_{opt} \leftarrow \emptyset$ $D_0 = \{Ax <= b, x \in \mathbb{N}\}$, $\phi_{sup} \leftarrow +\infty$, $l \leftarrow 0$, $(R_0) : \{\max_{x \in D_0} \quad \phi(x)\}$;
Tant que R_l *a une solution optimale* x^l **faire**
  Si *(*x^l*) est efficace* **alors**
    Si $\phi(x^l) > \phi_{inf}$ **alors**
      $\phi_{inf} \leftarrow \phi(x^l)$;
      $x_{opt} \leftarrow x^l$;
    Stop. x_{opt} est la solution optimale.
  Sinon
    Résoudre $(T_l) : \max\{\phi(x) | Cx = Cx^l, x \in D\}$; Soit $\hat{x}^l$ la solution optimale de T_l.
    Si $\phi(\hat{x}^l) > \phi_{inf}$ **alors**
      $\phi_{inf} \leftarrow \phi(\hat{x}^l)$ et $x_{opt} \leftarrow \hat{x}^l$;
    Si $\phi_{inf} = \phi_{sup}$ **alors**
      Stop. x_{opt} est la solution optimale.
    Construire $(D_{l+1}) : D_{l+1} = D_l \cup \{Cx > C\hat{x}^l\}$; $l \leftarrow l + 1$;

Plusieurs variantes de la méthode de Jorge ont été proposées pour optimiser une fonction sur l'ensemble efficace d'un MOP à savoir celles de Boland et *al.* [22] et de Lokman [89] qu'on va décrire par la suite.

3.3.2 Méthode de Chaabane et Pirlot

En 2010, Chaabane et Pirlot ont développé dans [33] une technique implicite qui évite de rechercher toutes les solutions efficaces mais garantit de trouver une solution qui optimise la fonction ϕ. La méthode commence par une solution efficace initiale x^0 obtenue en résolvant le problème $(P_\Theta(x^*))$ défini dans le Théorème (3.1), où x^* est la solution optimale du problème

relaxé (P_R). Ensuite, une recherche d'une solution alternative à la solution x^0 qui soit meilleure sur le critère ϕ est effectuée en résolvant le problème suivant :

$$(T_l)\begin{cases} \max & dx \\ \text{s.c.} & Cx = Cx^0, \\ & x \in D. \end{cases} \tag{3.5}$$

Soit $\bar{x}^l$ la solution optimale obtenue pour (T_l) ($\bar{x}^t$, elle est considérée comme une première solution efficace, poser $x_{\text{opt}} = \bar{x}^l$ et $\phi_{\text{opt}} = d\bar{x}^l$. Par la suite, à chaque itération l, le domaine de recherche des solutions efficaces est réduit progressivement en éliminant toutes les solutions dominées par $\bar{x}^l$ en utilisant l'idée de Sylva et Crema [125]. La résolution du problème suivant permet d'effectuer cette élimination (réduction).

$$(P_l)\begin{cases} \max & dx \\ \text{s.c.} & x \in D \backslash \bigcup_{s=1}^{l} D_l, \end{cases} \tag{3.6}$$

où $D_s = \{x \in \mathbb{Z}^n \mid Cx \leq C\bar{x}^s, x \geq 0\}$. Le domaine admissible du problème (3.6) est donc défini par l'ensemble des contraintes suivantes :

$$\begin{cases} H^0 = D, \\ H^k = H^{k-1} \cup \Delta_k, \quad \forall k \in \{1, \ldots, l-1\}, \end{cases}$$

où :

$$\Delta_k = \left\{ x \in D \;\middle|\; \begin{array}{l} Z_i(x) \geq (Z_i(x_{opt}) + 1)\, y_i^k - M_i \left(1 - y_i^k\right) \forall i \in \{1, \ldots, p\}, \\ \sum_{i=1}^{p} y_i^k \geq 1, \\ y_i^k \in \{0, 1\} \quad \forall i \in \{1, \cdots, p\}. \end{array} \right\}$$

$-M_i$ est une borne inférieure de la $i^{ème}$ fonction objectif dans D. y_i^k est une variable associée à chaque critère $Z_i = c^i x$, $i = 1, \ldots, p$, définie par :

$$y_i^k = \begin{cases} 1 \text{ si le critère } Z_i \text{ est strictement amélioré par rapport à } Z_i(x_{opt}), \\ 0 \text{ sinon.} \end{cases}$$

La contrainte $\sum_{i=1}^{p} y_i^k \geq 1$ indique qu'au moins un des critères est amélioré.
Si une solution optimale x^l du problème (P_l) est efficace, l'algorithme prend fin avec x^l comme solution optimale du problème principale (P_E). Sinon, les arêtes incidentes à x^l sont explorées afin de chercher une éventuelle solution efficace alternative. Si aucune alternative efficace n'est

Algo. 3.2 : Algorithme de Chaabane et Pirlot

Input $\downarrow$ **:** $\downarrow A, \downarrow b, \downarrow C$ et $\downarrow d$.
Output $\uparrow$ **:** $\uparrow x_{opt}$ et $\uparrow \phi_{opt}$.
Initialisation : $\phi_{opt} \leftarrow -\infty$, $l \leftarrow 0$, $recherche \leftarrow vrai$;
$[\uparrow x_0, \uparrow Z_0] = Pb\_relaxed(\downarrow d, \downarrow A, \downarrow b)$;
Si (P_R) *est irréalisable* **alors**
  (P_E) est irréalisable;
Sinon
  $[\uparrow x_t, \uparrow Z_t] = test\_eff(\downarrow C, \downarrow A, \downarrow b, \downarrow x_0)$;
  Si $Z_t = 0$ **alors**
    $x_{opt} \leftarrow x_0$, $\phi_{opt} \leftarrow Z_0$;
  Sinon
    $x_{ef} \leftarrow x_t$, $(P_0) \leftarrow (P_R)$;
    Tant que *recherche=vrai* **faire**
      $[\uparrow x_{eq}, \uparrow Z_{eq}] = opt(\downarrow C, \downarrow A, \downarrow b, \downarrow c, \downarrow x_{ef})$;
      Si $Z_{eq} > \phi_{opt}$ **alors**
        $x_{opt} \leftarrow x_{eq}$, $\phi_{opt} \leftarrow Z_{eq}$, $l \leftarrow l + 1$;
      Fsi
      $[\uparrow x_l, \uparrow Z_l, \uparrow Dom_l, \uparrow table_{opt}] = solve\_P_l(\downarrow A, \downarrow b, \downarrow C, \Delta_l, \downarrow d, \downarrow x_{ef})$; % $table_{opt}$: tableau final optimal
      Si $Dom_l = \emptyset$ **ou** $Z_l < \phi_{opt}$ **alors**
        $recherche \leftarrow faux$;
      Sinon
        $[\uparrow x_t^l, \uparrow Z_t^l] = test\_eff(\downarrow C, \downarrow A, \downarrow b, \downarrow x_l)$;
        Si $Z_t^l = 0$ **alors**
          $x_{opt} \leftarrow x_l$, $\phi_{opt} \leftarrow Z_l$, $recherche \leftarrow faux$;
        Sinon
          Construire l'ensemble $J_l = \{j \in N_l | \phi_j - c_j = 0\}$;
          Si $J_l = \emptyset$ **alors**
            $[\uparrow x_{exp}, \uparrow Z_{exp}, exist] = explorer(\downarrow C, \downarrow A, \downarrow b, \downarrow c, \downarrow table_{opt}, J_l)$;
          Sinon
            Si $exist = vrai$ **alors**
              $x_{opt} \leftarrow x_{exp}$, $\phi_{opt} \leftarrow Z_{exp}$, $recherche \leftarrow vrai$;
            Fsi
          Fsi
        Fsi
      Fsi
    Ftq
  Fsi
Fsi

trouvée, la région admissible est réduite et le processus se poursuit en améliorant la valeur de $\phi(x)$ et en réduisant le domaine d'admissibilité jusqu'à ce qu'il sera vide. Une description technique de la méthode est présentée dans l'algorithme (Algo. 3.2).
L'algorithme contient les procédures suivantes :
- $Pb\_relaxed$: résoudre (P_R).
- $test\_eff$: tester l'efficacité de x_0.
- opt : résoudre le problème (T_l).
- N_l : l'ensemble des indices des variables hors base de x_l.
- $solve\_P_l$: résoudre le problème (P_l).
- explorer : recherche d'éventuelles solutions efficaces sur les arrêtes incidentes à une solution efficace existante.

3.3.3 Méthode de Boland et *al.*

En 2017, Boland et *al.* [22] ont modifié l'algorithme de Jorge [75] en employant un nouveau mécanisme de décomposition du domaine de recherche (l'espace des critères) qui d'une part limite le nombre de sous-espaces à explorer et d'autre part limite le nombre de contraintes disjonctives nécessaires pour définir le programme linéaire qui recherche un point non dominé dans un sous-espace.
Les auteurs de cette méthode ont considéré le problème MOILP suivant :

$$\min_{x \in D} \{Z_1(x), \ldots, Z_p(x)\}, \tag{3.7}$$

où : $D \subseteq \mathbb{Z}^n$ et $Z_j(x) \in \mathbb{Q}$ pour tout $j \in \{1, \ldots, p\}$, soient : $\mathcal{Y}$ l'image de D dans l'espace des critères, E l'ensemble des solutions efficaces et $\mathcal{Y}_N$ leurres images dans l'espace des critères.
Le problème d'optimisation sur l'ensemble des solution efficace est définie comme suit :

$$\min_{x \in E} \phi(x),$$

où $\phi(x)$ est une combinaison linéaire des critères (pas strictement positive).

La méthode est basée sur la décomposition de l'espace des critères en sous-espaces définis par des ensembles de contraintes disjonctives. Chaque sous-espace est désigné par $O(\tilde{\mathcal{Y}}, y, y^L)$, où $\tilde{\mathcal{Y}} = \{y^1, \ldots, y^t\} \subseteq \mathcal{Y}$ et $y, y^L \in \mathcal{Y}$. $\tilde{\mathcal{Y}}, y, y^L$ représentent l'enveloppe de la borne supérieure, le point repère de la recherche et la borne inférieure respectivement. Chaque sous-espace $O(\tilde{\mathcal{Y}}, y, y^L)$ est définie de la manière suivante :

$$O\left(\tilde{\mathcal{Y}}, y, y^L\right) = z^a \in \mathbb{Q}^p : \begin{cases} z_j^a \geq y_j^L, & \forall j \in \{1,\ldots,p\}, \\ z_j^a \leq \left(y_j^i - \epsilon - M_j\right) b_{ij} + M_j, & \forall i \in \{1,\ldots,t\}, \forall j \in \{1,\ldots,p\}, \\ \sum_{j=1}^{p} b_{ij} = 1, & \forall i \in \{1,\ldots,t\}, \\ z_j^a \leq (y_j - \epsilon - M_j)\, b'_j + M_j, & \forall j \in \{1,\ldots,p\}, \\ \sum_{j=1}^{p} b'_j = 1, & \\ b_{ij} \in \{0,1\}, \quad b'_j \in \{0,1\}, & \forall i \in \{1,\ldots,t\}, \forall j \in \{1,\ldots,p\}, \end{cases}$$

où ϵ est une petite constante positive, et M_j est une grande constante choisie de manière appropriée (on peut prendre la borne supérieure de la $j^{ème}$ fonction objectif dans D).
Pour effectuer la décomposition de l'espace de recherche, quatre points $y^L, y', y'', \hat{y}$ sont déterminés pour chaque critère j, $(j = 1,\ldots,p)$ comme suit :

$* \ \hat{y}_j^L = \max\left(y_j^L, \min\left(y_j, Z_j\left(x^n\right)\right)\right)$. $\quad * \ y'_j = \begin{cases} Z_j\left(x^n\right) & \text{if } Z_j\left(x^n\right) > y_j \text{ et } Z_j\left(x^n\right) > \hat{y}_j^L, \\ -\infty & \text{sinon.} \end{cases}$

$* \ y''_j = \begin{cases} y_j & \text{si } Z_j\left(x^n\right) < y_j \text{ et } \hat{y}_j^L < y_j, \\ -\infty & \text{sinon.} \end{cases} \quad * \ \hat{y}_j = \begin{cases} \hat{y}_j^L & \text{et } \hat{y}_j^L > y_j^L, \\ -\infty & \text{sinon.} \end{cases}$

Les différentes procédures utilisées dans cet algorithme sont décrites comme suit :

- Find-NDP (y) : Trouver une solution efficace x^n telle que $Z(x^n)$ domine $y, y \in \mathbb{Q}^p$ un point arbitraire dans l'espace des critères, cela en résolvant le problème suivant :

$$\begin{cases} \min & \sum_{i=1}^{p} z_j^a \\ \text{s.c.} & x \in D, \\ & z_j^a = Z_j(x), \quad j = 1,\ldots,p, \\ & z_j^a \leq y_j, \quad j = 1,\ldots,p, \\ & z_j^a \in \mathbb{Q}, \quad j = 1,\ldots,p, \end{cases}$$

  où $z_j^a \in \mathbb{Q}$ sont des variables auxiliaires.
- Find-Best(x^n) : Trouver une solution alternative à la solution efficace x^n qui donne la valeur minimale de $\phi(x)$ en résolvant le problème suivant :

$$\begin{cases} \min & \phi(x) \\ \text{s.c.} & x \in D, \\ & Z_i(x) \leq Z_i(x^n), \quad \forall i \in \{1,\ldots,p\}. \end{cases}$$

- Find-LB-Obj$(\tilde{\mathcal{Y}}, y, y^L)$: Calculer une borne inférieure de $\phi(x)$ sur l'ensemble des solutions efficaces ayant leurs images dans le sous-espace $O(\tilde{\mathcal{Y}}, y, y^L)$ en résolvant le

Algo. 3.3 : Algorithme de Boland et *al.*

Initialisation : $\phi_{opt} \leftarrow \max_{x \in D} \phi(x)$;
$x_o^l \leftarrow$ Find-LB-Obj$(\emptyset, null, null)$;
Si $x_o^l \neq null$ **alors**
  $\phi_{inf} \leftarrow \phi(x_o^l)$;
  Initialiser la file de priorité $F = \Big((\emptyset, null, null), (x_o^l, \phi(x_o^l))\Big)$;
  $SearchDone \leftarrow Faux$;
  Tant que $F \neq \emptyset$ *et* $SearchDone = Faux$ **faire**
    Retirez un élément de F et désigner-le par $((\tilde{\mathcal{Y}}, y, y^L), (x_o^l, \phi(x_o^l)))$;
    $\phi_{inf} \leftarrow \phi(x_o^l)$;
    Si $\frac{\phi_{opt} - \phi_{inf}}{\phi_{opt} + \epsilon_1} > \epsilon_2$ **alors**
      $x^n \leftarrow$ Find-NDP$(Z(x_o^l))$;
      Si $Z(x_o^l) = Z(x^n)$ **alors**
        $x^f \leftarrow x_o^l$;
      Sinon
        $x^f \leftarrow$ Find-Best(x^n) ;
      Si $\phi_{opt} > \phi(x^f)$ **alors**
        $\phi_{opt} \leftarrow \phi(x^f)$;
        $x_{opt} \leftarrow x^f$;
      Si $\phi_{inf} < \phi_{opt}$ **alors**
        Calculer $\hat{y}^L$;
        Si $y \neq null$ **alors**
          Calculer y' et y''
          Si $y' \neq (-\infty, \ldots, -\infty)$ *et* $y'' \neq (-\infty, \ldots, -\infty)$ **alors**
            $x_o^l \leftarrow$ Find-LB-Obj$(\tilde{\mathcal{Y}} \cup \{y', y''\}, null, \hat{y}^L)$;
            Si $x_o^l \neq null$ *et* $\phi(x_o^l) < \phi_{opt}$ **alors**
              Ajouter $((\tilde{\mathcal{Y}} \cup \{y', y''\}, null, \hat{y}^L), (x_o^l, \phi(x_o^l)))$ à la file de priorité F ;
        Calculer $\hat{y}$
        Si $\hat{y} \neq (-\infty, \ldots, -\infty)$ **alors**
          $x_o^l \leftarrow$ Find-LB-Obj$(\tilde{\mathcal{Y}}, \hat{y}, y^L)$;
          Si $x_o^l \neq null$ *et* $\phi(x_o^l) < \phi_{opt}$ **alors**
            Ajouter $((\tilde{\mathcal{Y}}, \hat{y}, \hat{y}^L), (x_o^l, \phi(x_o^l)))$ à la file de priorité F ;
    Sinon
      $SearchDone \leftarrow vrai$;
Retourner ϕ_{opt} et x_{opt}.

problème d'optimisation suivant :

$$\begin{cases} \min & \phi(x) & \\ \text{s.c.} & x \in D, & \\ & z_j^a = z_j(x), & \forall j \in \{1,\dots,p\}, \\ & z_j^a \geq y_j^L, & \forall j \in \{1,\dots,p\}, \\ & z_j^a \leq \left(y_j^i - \epsilon - M_j\right) b_{ij} + M_j, & \forall i \in \{1,\dots,t\}, \forall j \in \theta\left(y^i\right), \\ & \sum_{j \in \theta(y^i)} b_{ij} = 1, & \forall i \in \{1,\dots,t\}, \\ & z_j^a \leq \left(y_j - \epsilon - M_j\right) b'_j + M_j, & \forall j \in \theta(y), \\ & \sum_{j \in \boldsymbol{\theta}(y)} b'_j = 1, & \\ & z_j^a \in \mathbb{Q}, & \forall j \in \{1,\dots,p\}, \\ & b_{ij} \in \{0,1\}, & \forall i \in \{1,\dots,t\}, \forall j \in \theta\left(y^i\right), \\ & b'_j \in \{0,1\}, & \forall j \in \theta(y), \end{cases}$$

où $\theta(y^i) = \{j \in \{1,\dots,p\} \ : y_j^i > -\infty\}$ et $\theta(y) = \{j \in \{1,\dots,p\} \ : y_j > -\infty\}$.

Les deux ensembles $\theta(y^i)$ pour tout $y^i \in \tilde{\mathcal{Y}}$ et $\theta(y)$ sont introduits uniquement pour éviter de générer des variables et des contraintes redondantes.
La description technique de cette méthode est donnée par l'algorithme (Algo. 3.3).

3.3.4 Méthodes de Lokman

Lokman a proposé en 2021 dans [89] deux algorithmes appelés DSA et DSA$_m$ pour optimiser une fonction linéaire sur l'ensemble efficace d'un MOILP.

3.3.4.1 Première méthode DSA

La variante DSA est une extension de la méthode de Jorge, il génère itérativement des points non dominés et réduit l'ensemble réalisable en excluant non seulement les régions dominées, mais aussi les régions inférieures par rapport à la fonction objectif principale.
Dans cette méthode, le problème d'optimisation sur l'ensemble efficace considéré s'écrit comme suit :

$$(P_E(\nu)) \begin{cases} \max & \phi(x) = \sum_{i=1}^{p} \nu_i Z_i(x) \\ \text{s.c.} & x \in E, \end{cases} \tag{3.8}$$

où E est l'ensemble des solutions efficaces du problème (3.2), $\phi(x)$ est une combinaison linéaire des objectifs qui n'est pas strictement positive, ç-a-d que $\nu_i \leq 0$ pour quelques $i \in \{1,\dots,p\}$. Soit (P^l) le problème résolut par la méthode de Jorge [75] à une itération $l > 1$, où les points

non dominés générés précédemment, $ND^l = \{Z^1, Z^2, ..., Z^{l-1}\}$, sont exclus de l'ensemble réalisable afin de générer un nouveau point tel que :

$$(P^l)\begin{cases} \max & \phi(x) = \sum_{i=1}^{p} \nu_i Z_i(x) & & \\ \text{s.c.} & x \in D, & & \\ & Z_i(x) \geq (Z_i^t + \epsilon) y_i^t + M_i(1 - y_i^t), & i = 1, \ldots, p, \quad t = 1, \ldots, l-1, \\ & \sum_{i=1}^{p} y_i^t = 1, & t = 1, \ldots, l-1, \\ & y_i^t \in \{0, 1\}, & i = 1, \ldots, p, \quad t = 1, \ldots, l-1. \end{cases}$$

Dans la méthode DSA, à l'itération $l > 1$ l'ensemble réalisable (dans l'espace des critères) est partitionné en sous-ensembles en imposant des bornes sur chaque critère. Dans chaque sous-ensemble, le point non dominé (le cas échéant) qui maximise la fonction ϕ est recherché. Après avoir considéré tous les sous-ensembles, le point ayant la valeur maximale de la fonction ϕ est enfin choisi, celui-ci correspond au point optimal du problème (P^l), il est noté x^{inc}.
Les sous ensembles sont crées en identifiants un ensemble de k vecteurs noté K^l et à chaque vecteur $k = (k_1, \ldots, k_p)$, $0 \leq k_i \leq l-1$ pour tout $i \in \{1, \ldots, p\}$, on lui associe un vecteur $b^k = (b_1^k, \ldots, b_p^k)$ qui représente des bornes inférieures sur les p critères. Si $k_i = 0$, alors il n'y a pas de borne inférieure supplémentaire pour le critère i et elle est définie donc par M_i, autrement, la valeur du $i^{\text{ème}}$ critère de $Z^{k_i} \in ND^l$ est utilisée pour imposer une borne inférieure sur le critère i. Les bornes b_i^k sont définies comme suit :

$$b_i^k = \begin{cases} M_i & k_i = 0, \\ Z_i^{k_i} + \epsilon & k_i \neq 0. \end{cases}$$

Après avoir définie les sous ensembles associés à une itération l, on résout les problèmes (S^k), $k \in K^l$ tel que :

$$(S^k)\begin{cases} \max & \phi(x) = \sum_{i=1}^{p} \nu_i Z_i(x) \\ \text{s.c.} & x \in D, \\ & Z_i(x) \geq b_i^k, \quad i = 1, \ldots, p, \\ & \sum_{i=1}^{p} \nu_i Z_i(x) \geq \phi_{inf} + g^* |\phi_{inf}|, \\ & \sum_{i=1}^{p} \nu_i Z_i(x) \leq \phi^{sup}. \end{cases}$$

Parmi les solutions optimales correspondantes, la meilleure par rapport à la fonction ϕ est retenue et est notée par x'^l. Cette solution est tester pour son efficacité en résolvant le problème

suivant :

$$(P_\lambda^l)\begin{cases} \max & \sum_{i=1}^{p} \lambda_i Z_i(x) \\ \text{s.c.} & x \in D, \\ & Z_i(x) \geq Z_i(x'^l) \quad i = 1, \ldots, p, \end{cases}$$

où $\lambda = (\lambda_1, \ldots, \lambda_p)$ est un vecteur des poids tel que $\lambda_i = \nu_i - \min_{i=1,\ldots,p} \nu_i + 1$ pour tout $i = 1, \ldots, p$ (pour obtenir des poids positifs).
Si x'^l n'est pas efficace, (P_λ^l) retourne un une solution efficace x^l tel que $Z(x^l)$ domine $Z(x'^l)$ et si $\phi(x^l) > \phi^{inf}$, on met à jour x^{inc} et ϕ_{inf} pour l'étape suivante. Une borne inférieure ϕ_{inf} et une borne supérieure ϕ^{sup} de la fonction ϕ sont mise à jour à chaque itération. En fixant au départ une certaine précision $g^* \geq 0$, l'algorithme s'arrête lorsque $\frac{|\phi^{sup} - \phi_{inf}|}{|\phi_{inf}|} \leq g^*$, ou bien tous les problèmes (S^k) d'une itération l soient irréalisables.
La description technique de la méthode est donnée dans l'algorithme (Algo. 3.4).

3.3.4.2 Deuxième méthode DSA$_m$

La méthode DSA$_m$ maximise l'un des critères noté m, tout au long de l'algorithme dans le but de réduire le nombre de points dominés générés. Le critère m est choisi en fonction des coefficients de la fonction ϕ : $m = \arg\max_{i=1,\ldots,p} \nu_i$, et l'amélioration de la fonction ϕ est imposée en contraintes. La décomposition de l'espace réalisable se fait de la même manière que dans la méthode DSA. A chaque itération $l > 0$, l'algorithme résout les problèmes suivants :

$$(P_m^k)\begin{cases} \max & Z_m(x) \\ \text{s.c.} & x \in D, \\ & Z_i(x) \geq b_i^k, \quad i = 1, \ldots, p, \quad i \neq m, \\ & \sum_{i=1}^{p} \nu_i Z_i(x) \geq \phi_{inf} + g^* |\phi_{inf}|, \\ & \sum_{i=1}^{p} \nu_i Z_i(x) \leq \phi^{sup}, \end{cases}$$

où $k \in K_m^l$, le vecteur K_m^l est définit de la même manière que le vecteur K^l dans DSA sauf qu'aucune borne n'est imposée dans cette variante sur le critère m ç-à-d que $k_m^l = 0$ pour tout $k \in K_m^l$. Le test d'efficacité d'une solution et la mise à jour de la valeur de la fonction ϕ se font comme décrit dans la méthode précédente. L'algorithme (Algo. 3.5) donne la description technique de DSA$_m$.

Algo. 3.4 : Algorithme de Lokman (DSA)

Etape 1 : Initialisation
$l = 1$, $\lambda_i = \nu_i - \min_{i=1,\dots,p} \nu_i + 1$, $i = 1, \dots, p$;
$x^{'1} = \arg\max_{x \in D} \phi(x)$;
Etape 2 :
Résoudre (P^l_λ) ; soit x^l la solution optimale.
Si $Z(x^l) = Z(x^{'l})$ **alors**
  $\phi^{sup} = \phi_{inf} = \phi(x^{'l})$;
  $x^{inc} = x^{'l}$;
  aller à l'étape 5 ;
Sinon
  aller à l'étape 3 ;
Etape 3 :
$\phi^{sup} \leftarrow \phi(x^{'l})$;
Si $\phi(x^l) \geq \phi_{inf}$ **alors**
  $\phi_{inf} \leftarrow \phi(x^l)$;
  $x^{inc} \leftarrow x^l$;
Si $\frac{|\phi^{sup} - \phi_{inf}|}{|\phi_{inf}|} \leq g^*$ **alors**
  aller à l'étape 5 ;
Sinon
  aller à l'étape 4 ;
Etape 4 :
$l \leftarrow l + 1$;
mettre à jour K^l ;
Résoudre (S^k) pour chaque $k \in K^l$; soient $x^k,\ k \in K^l$ les solutions optimales correspondantes.
Si (S^k) *est irréalisable pour tout* $k \in K^l$ **alors**
  aller à l'étape 5 ;
Sinon
  Trouver $k^* = \arg\max_{k \in K^l, (S^k) réalisable} \phi(x^k)$;
  $x^{'l} \leftarrow x^{k^*}$;
  aller à l'étape 2 ;
Etape 5 :
$x_{opt} \leftarrow x^{inc}$;
$\phi_{opt} \leftarrow \phi(x_{opt})$ (avec un certain niveau de précision g^*).

Algo. 3.5 : Algorithme de Lokman (DSA_m)

Etape 1 : Initialisation

$m = \arg\max_{i=1,\dots,p} \nu_i$, $\lambda_i = \nu_i - \min_{i=1,\dots,p} \nu_i + 1$, $i = 1, \dots, p$, $x^{'0} = \arg\max_{x \in D} \phi(x)$ et $\phi^{sup} = \phi(x^{'0})$;

Résoudre (P^0_λ), soit x^0 la solution optimale, $x^{inc} \leftarrow x^0$ et $\phi_{inf} = \phi(x^{inc})$;

$x^{'1} = \arg\max Z_m(x)$, $l \leftarrow 1$;

Etape 2 :

Résoudre (P^l_λ), soit x^l la solution optimale. **Si** $\phi(x^l) \geq \phi(x^{inc})$ **alors**

  $x^{inc} \leftarrow x^l$, $\phi_{inf} \leftarrow \phi(x^{inc})$;

Si $\frac{|\phi^{sup} - \phi(x^{inc})|}{|\phi(z^{inc})|} \leq g^*$ **alors**

  aller à l'étape 5 ;

Sinon

  aller à l'étape 4 ;

Etape 4 :

$l \leftarrow l + 1$, mettre à jour K^l_m ;

Résoudre (P^k_m) pour chaque $k \in K^l_m$, soient x^k, $k \in K^l$ les solutions optimales correspondantes ;

Si (S^k) *est irréalisable pour tout* $k \in K^l_m$ **alors**

  aller à l'étape 5 ;

Sinon

  Trouver $k^* = \arg\max_{k \in K^l_m, (P^k_m) réalisable} Z_m(x^k)$;

  $x^{'l} \leftarrow x^{k^*}$; aller à l'étape 2 ;

Etape 5 :

$x_{opt} \leftarrow x^{inc}$;

$\phi_{opt} \leftarrow \phi(x_{opt})$ (avec un certain niveau de précision g^*).

Plus de détails sur ces deux variantes peuvents être consulter dans [89].

3.3.5 Méthodes de Prerna et Sharma

Récemment (2023), Prerna et Sharma ont proposé dans leur article [107] deux algorithmes pour optimiser une fonction linéaire sur l'ensemble efficace d'un MOILP. Les auteurs ont considéré le problème (3.1), où la fonction $\phi(x) = dx$ est linéaire sans aucune condition sur le paramètre d.

3.3.5.1 Première méthode

L'objectif de cet algorithme est de déterminer l'ensemble de toutes les solutions optimales du problème (P_E) noté G_E en résolvant successivement les problèmes (P_k) qu'on va définir par la suite.

Initialement, on fixe $k = 1$ et l'algorithme commence par résoudre le problème suivant :

$$(P_1)\begin{cases} \max & \phi(x) = dx \\ \text{s.c.} & x \in D. \end{cases} \tag{3.9}$$

L'ensemble de toutes les solutions optimales de ce problème noté G^1 sont déterminées en utilisant la technique de branch and bound ou bien la technique de coupe de Gomory. La valeur optimale de la fonction ϕ notée ϕ^1 constitue donc une borne supérieure du problème principale (P_E).
Toutes les solutions de l'ensemble G^1 sont tester pour leur efficacité en résolvant le problème $(P_\Theta(x_i^1))$ décrit dans l'équation (3.4) pour chaque solution $x_i^1 \in G^1, \quad i = 1, \ldots, l_1$, tel que l_1 est le nombre de solutions dans G^1. Si $\Theta = 0$ pour certains $i \in \{1, \ldots, l_1\}$ dans $(P_\Theta(x_i^1))$, alors $x_i^1 \in G_E$, $\phi_{opt} = \phi^1$ est la valeur optimale de la fonction objectif ϕ et l'algorithme se termine. Si aucune solution de G^1 n'est efficace, on procède à l'étape suivante $(k = 2)$.
A chaque itération k, On résout le problème (P_k), l'ensemble de toutes les solutions optimales noté G^k est construit tel que :

$$(P_k)\begin{cases} \max & \phi(x) = dx \\ \text{s.c.} & x \in D_k, \end{cases} \tag{3.10}$$

où : $D_k : \{x \in D | \phi(x) \leq \phi^{k-1} - 1\}$, tel que ϕ^{k-1} représente la meilleure valeur possible pour ϕ dans le problème (P_{k-1}). Cette valeur est utilisée comme une borne supérieure pour la fonction ϕ dans le problème (P_k). Ensuite, on teste l'efficacité des solutions de G^k, pour ce faire, $(P_\Theta(x_i^k))$ est résolu pour tout x_i^k appartenant à $G^k, \quad i \in \{1, 2, \ldots, l_k\}$. Si le test d'efficacité est satisfait pour certains x_i^k, alors, x_i^k est ajoutée à G_E^k et $\phi_{opt} = \phi(x_i^k)$. Si aucune solution n'est efficace, on passe à l'itération suivante $(k = k + 1)$.
Ce processus est répété jusqu'à ce que l'ensemble G_E^k soit non vide. On donne ci-après, la description algorithmique de la méthode (Algo. 3.6).

3.3.5.2 Deuxième méthode

Contrairement à la première méthode, conçue pour définir toutes les solutions optimales de (P_E), cette méthode a été élaborée dans le but de trouver uniquement une seule solution optimale au problème (P_E). La deuxième méthode procède comme suit :
Tout d'abord, le problème (3.9) est résolu pour trouver l'ensemble G^1 et une borne supérieure de ϕ notée ϕ^1, de la même manière que dans le premier algorithme. Ensuite, pour chaque x_i^1 dans l'ensemble G^1, le problème $(P_\Theta(x_i^1))$ est résolu, et l'ensemble E^1 (un sous-ensemble de E) est construit. Pour chaque solution dans l'ensemble E^1, la valeur de la fonction objectif ϕ est calculée, et ϕ_{inf}^1 est déterminée, représentant la borne inférieure actuelle de la valeur de la fonction objectif dans le problème (P_E). Deux cas peuvent se présenter : soit $\phi_{inf}^1 \geq \phi^1 - 1$,

soit $\phi^1_{inf} < \phi^1 - 1$.

- Si pour un certain $\hat{x}^*$ dans E^1, $\phi^1_{inf} \geq \phi^1 - 1$, alors $\hat{x}^*$ est considérée comme une solution optimale du problème (P_E) et l'algorithme se termine.
- Si $\phi^1_{inf} < \phi^1 - 1$, alors fixer $k = k + 1$ et répéter la procédure ci-dessus. Résoudre (P_k), construire les ensembles G^k et E^k, et trouver la valeur de ϕ^k_{inf}. Si $\phi(\hat{x}) = \phi^k_{inf}$ est supérieure ou égale à $\phi^k - 1$ pour certain $\hat{x} \in \bigcup_{l=1}^{k} E^l$, alors $\hat{x}$ est considérée comme une solution optimale du problème (P_E) et l'algorithme se termine. sinon, répéter la procédure pour la valeur suivante de k, jusqu'à ce que $\phi^k_{inf} \geq \phi^k - 1$.

La description technique de cette méthode est donnée par l'algorithme (Algo. 3.7).

Algo. 3.6 : Prerna et Sharma (Premier algorithme)

Initialisation :

$k \leftarrow 1$, $G^1 \leftarrow \{\hat{x} \in D | \phi(\hat{x}) = \max_{x \in D} \phi(x)\}$;

$\phi^1 \leftarrow \phi(x^1_i)$ pour $x^1_i \in G^1$, $G^1_E \leftarrow \emptyset$;

Pour chaque $x^1_i \in G^1$ **faire**

  résoudre $(P_\Theta(x^1_i))$;

  Si $\Theta = 0$ **alors**

    $G^1_E \leftarrow \{G^1_E, x^1_i\}$;

Tant que $G^k_E = \emptyset$ **faire**

  $k \leftarrow k + 1$;

  $D_k \leftarrow \{x \in D | \phi(x) \leq \phi^{k-1} - 1\}$;

  $G^k \leftarrow \{x \in D_k | \phi(x) = \max_{x \in D_k} \phi(x)\}$;

  $\phi^k \leftarrow \phi(x^k_1)$ pour $x^1_k \in G^k$;

  $G^k_E \leftarrow \emptyset$;

  Pour chaque $x^k_i \in G^k$ **faire**

    résoudre $(P_\Theta(x^k_i))$;

    Si $\Theta = 0$ **alors**

      $G^k_E \leftarrow \{G^k_E, x^k_i\}$;

Si $G^k_E \neq \emptyset$ **alors**

  $G_E \leftarrow G^k_E$;

Algo. 3.7 : Prerna et Sharma (Deuxième algorithme)

Initialisation :

$k \leftarrow 1$, $G^1 \leftarrow \{\hat{x} \in D | \phi(\hat{x}) = \max_{x \in D} \phi(x)\}$, $\phi^1 \leftarrow \phi(x_i^1)$ pour $x_i^1 \in G^1$, $E^1 = \emptyset$;

Pour $x_i^1 \in G^1$ **faire**

  résoudre $(P_\Theta(x_i^k))$;

  Si x^* *est une solution optimale de* $(P_\Theta(x_i^1))$ **alors**

    $E^1 \leftarrow \{E^1, \ x^*\}$

  ;

$\phi_{inf}^1 = \max\{\phi(x^*) | x^* \in E^1\}$;

Tant que $\phi_{inf}^k < \phi^k - 1$ **faire**

  $k \leftarrow k + 1$;

  $D_k \leftarrow \{x \in D | \phi(x) \leq \phi^{k-1} - 1\}$;

  $G^k \leftarrow \{\hat{x} \in D_k | \phi(\hat{x}) = \max_{x \in D_k} \phi(x)\}$;

  $\phi^k \leftarrow \phi(x_i^k)$ pour $x_i^k \in G^k$;

  $E^k \leftarrow \emptyset$;

  Pour $x_i^k \in G^k$ **faire**

    résoudre $(P_\Theta(x_i^k))$

    Si x^* *est une solution optimale de* $(P_\Theta(x_i^k))$ **alors**

      $E^k \leftarrow \{E^k, \ x^*\}$

  $\phi_{inf}^k = \max\{\phi(x^*) | x^* \in \cup_{l=1}^k E^l\}$;

Si $\phi(\hat{x}^*) = \phi_{inf}^k$ *pour quelques* $\hat{x}^* \in \cup_{l=1}^k E^l$ **alors**

  $\hat{x}^* \in G_E$;

3.4 Conclusion

Dans ce chapitre, nous avons abordé la problématique de l'optimisation d'un critère linéaire sur l'ensemble des solutions efficace d'un problème MOP noté (P_E), nous avons donc décrit quelques méthodes exactes pour résoudre (P_E) dans le cas où les variables sont discrètes. A notre connaissance, aucune recherche n'a été effectuée spécifiquement pour le cas des variables binaires, ceci nous a davantage motivé à explorer cette question dans le dernier chapitre. Le chapitre suivant est dédié aux problèmes d'optimisation combinatoire multi-objectif (MOCO), notre contribution s'inscrit dans ce domaine.

Chapitre 4

Problème de sac à dos multi-objectif MOKP

4.1 Introduction

Le problème du sac à dos est l'un des problèmes classiques d'optimisation combinatoire les plus étudiés au cours des cinq dernières décennies, en raison de ses nombreuses applications dans le monde réel. En effet, ce problème figure comme sous-problème à résoudre dans différents domaines : la logistique comme le chargement d'avions ou de navires, l'économie comme la gestion de portefeuilles ou dans l'industrie comme la découpe de matériaux. Ce problème fait partie des problèmes d'optimisation NP-complets, son énoncé est très simple, cela explique le nombre important d'ouvrages qui lui sont consacrés parmi lesquels on cite le livre de Mertello et Toth [93] et le livre de Kellerer et Pisinger [79].

Ce chapitre a pour objectif de définir le problème de sac à dos multi-objectif et discuter quelques méthodes de résolution qui lui sont dédiées.

4.2 Généralités sur le problème de sac à dos mono-objectif

4.2.1 Formulation du problème

Le problème de sac à dos consiste à sélectionner un sous-ensemble d'objets pour remplir un sac à dos dont on dispose. En outre, cette sélection doit être faite de manière à maximiser une fonction exprimée en fonction des profits associés aux objets, tout en respectant la contrainte

relative à la capacité du sac. Formellement, si n est le nombre d'objets, chaque objet i, $i = 1 \ldots, n$ est caractérisé par un poids w_i et un profit c_i, on cherche le sous-ensemble d'objets à charger dans un sac de capacité ω afin de maximiser la somme des profits. Soit la variable de décision x_i telle que :

$$x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est chargé dans le sac,} \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, le problème de sac à dos uni-dimensionnel en variables binaires KP peut être formulé de la manière suivante :

$$(KP) \begin{cases} \max & Z(x) = \sum_{i=1}^{n} c_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq \omega, \\ & x_i \in \{0, 1\}. \end{cases} \tag{4.1}$$

- Les poids w_i et les profits c_i ainsi que la capacité ω sont des entiers positifs $i \in \{1, \ldots, n\}$.
- Pour s'assurer due chaque article peut être chargé dans le sac on suppose que : ${}_i \leq \omega, \quad foralli \in \{1, ..., n\}$.
- Pour éviter éviter le cas triviale où le sac contient tous les objets, on suppose que : $\sum_{i=1}^{n} w_i \geq \omega$.

4.2.2 Quelques variantes du problème de sac à dos

Il existe plusieurs variantes du problème de sac à dos, catégorisées selon la nature des variables (entières, bornées, binaires ou réelles), le nombre de sacs à remplir (sac à dos multiple), le nombre de contraintes (uni-dimensionnel, bi-dimensionnel ou multi-dimensionnel), la structure des objets (à choix multiple), le nombre de critères à maximiser (mono-objectif, bi-objectif ou multi-objectif), etc. On présente dans cette section quelques unes d'entre ces variantes.

4.2.2.1 Sac à dos à variables continues

Ce problème (Continuous Knapsack Problem) dénoté LKP est dit problème relaxé du problème KP, il peut être obtenu en remplaçant $x_i \in \{0, 1\}$ par $x \in [0, 1]$, le LKP s'écrit donc

comme suit :

$$(LKP)\begin{cases} \max & Z(x) = \sum_{i=1}^{n} c_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq \omega, \\ & x_i \in [0,1]. \end{cases}$$

Dantzig [38] a montré que l'obtention de la solution optimale de ce problème notée $Z^*(LKP)$ est très facile, en s'appuyant sur les concepts d'efficacité d'un objet et d'élément critique, définis ci-dessous.

Définition 4.1 ***(efficacité d'un objet)*** *On appelle efficacité d'un objet i le rapport de son coût sur son poids, noté $e_i = \frac{c_i}{w_i}$.*

Définition 4.2 ***(élément critique (split item))***

On appelle élément critique et on note s le premier objet ne pouvant pas entièrement entrer dans le sac lorsque les objets sont ajoutés par ordre décroissant de l'efficacité des objets : $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \ldots \geq \frac{c_n}{w_n}$.

$$s = \min\left\{k : \sum_{i=1}^{k} w_i > \omega\right\}.$$

La valeur optimale $Z^*(LKP)$ est obtenue en prenant les objets dans l'ordre décroissant de leur efficacité, jusqu'à l'élément critique puis en ajoutant la fraction de cet élément permettant de saturer le sac.

$$Z^*(LKP) = \sum_{i=1}^{s-1} c_i + \left(\omega - \sum_{i=1}^{s-1} w_i\right) \frac{c_s}{w_s}.$$

4.2.2.2 Sac à dos multi-dimensionnel

Le problème (Multi-dimensionnel Knapsack Problem MKP) est une variante du problème du sac à dos, dans lequel plusieurs contraintes de capacité (poids, volume, temps, etc.) doivent être simultanément respectées. On dispose d'un ensemble d'objets, chacun étant caractérisé par un profit c_i, pour $i = 1, \ldots, n$, ainsi que par plusieurs poids w_{ij}, pour $j = 1, \ldots, m$ chaque poids étant associé à une contrainte spécifique. L'objectif est de sélectionner un sous-ensemble d'objets qui maximise le profit total, tout en respectant les m contraintes de capacité : $\sum_{i=1}^{n} w_{ij} x_i \leq \omega_j$, $j = 1, \ldots, m$, où ω_j, $j = 1, \ldots, m$ sont les capacités correspondantes au sac.

Le problème MKP peut être formulé comme suit :

$$(MKP)\begin{cases} \max & Z(x) = \sum_{i=1}^{n} c_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_{ij} x_i \leq \omega_j, \quad j = 1, \ldots, m, \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n, \end{cases}$$

où :

$$x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est chargé dans le sac,} \\ 0 & \text{sinon.} \end{cases}$$

Les paramètres c_i, w_{ij} et $\omega_j,\ i = 1, \ldots, n,\ j = 1, \ldots, m$ sont des entiers positifs.

4.2.2.3 Sac à dos multiple

Le problème du sac à dos multiple (Multiple Knapsack Problem m-KP), se caractérise par m sac de capacité respective ω_j, $j = 1, \ldots, m$, pouvant contenir les différents objets. Soit la variable de décision binaire x_{ij}, pour $j = 1, \ldots, m$, et $i = 1, \ldots, n$:

$$x_{ij} = \begin{cases} 1 & \text{si l'objet } i \text{ est chargé dans le sac } j, \\ 0 & \text{sinon .} \end{cases}$$

Ainsi, le problème peut s'écrire comme suit :

$$(m\text{-}KP)\begin{cases} \max & Z(x) = \sum_{j=1}^{m} \sum_{i=1}^{n} c_i x_{ij} \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_{ij} \leq \omega_j, \quad j = 1, \ldots, m, & (*) \\ & \sum_{j=1}^{m} x_{ij} \leq 1, \quad i = 1, \ldots, n, & (**) \\ & x_{ij} \in \{0,1\}, \quad j = 1, \ldots, m, \quad i = 1, \ldots, n, \end{cases}$$

où c_i, w_i et $\omega_j,\ i = 1, \ldots, n,\ j = 1, \ldots, m$ sont des entiers positifs et les contraintes (*) et (**) assurent respectivement, que le remplissage du sac à dos j ne dépasse pas sa capacité correspondante ω_j et que chaque objet sélectionné est attribué, au plus, à un seul sac à dos.

4.2.2.4 Sac à dos à choix multiple

Le problème du sac à dos à choix multiples (Multiple Choice Knapsack Problem m-CKP) est une extension du problème du sac à dos, dans lequel l'ensemble des objets est divisé en m classes ($i = 1, \ldots, m$), la classe i contenant N_i objets. L'objet $j \in \{1, \ldots, N_i\}$ de la classe i est caractérisé par son poids w_{ij} et sa valeur c_{ij}. Le sac à dos doit contenir exactement un objet unique de chaque classe, les m objets sélectionnés doivent maximiser la valeur du chargement à condition de ne pas dépasser la capacité ω du sac à dos. Soit la variable de décision binaire x_{ij}, pour $i = 1, \ldots, m$, et $j = 1, \ldots, N_i$:

$$x_{ij} = \begin{cases} 1 & \text{si l'objet } j \text{ de la classe } i \text{ est sélectionné ,} \\ 0 & \text{sinon .} \end{cases}$$

Le problème m-CKP se formule comme suit :

$$(m-KCP) \begin{cases} \max & Z(x) = \sum_{i=1}^{m} \sum_{j \in N_i} c_{ij} x_{ij} \\ \text{s.c.} & \sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq \omega, & (*) \\ & \sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \ldots, m, & (**) \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j \in N_i, \end{cases}$$

où c_{ij}, w_{ij} et $\omega,\ i = 1, \ldots, n,\ j \in N_i$ sont des entiers positifs. La contrainte (*) assure que la somme des poids des objets sélectionnés ne doit pas dépasser la capacité du sac et la contrainte (**) impose qu'un seul objet doit être sélectionné dans chaque classe.

Dans la section suivante, nous nous intéresserons au problème du sac à dos binaire dans sa version multi-objectif. D'autres variantes du problème de sac à dos sont décrites dans [79, 93].

4.3 Problème de sac à dos multi-objectif MOKP

Le problème de sac multi-objectif (Multiple objective Knapsack Problem) est une variante du problème de sac à dos dans laquelle p coûts sont associés à chaque objet. Cette variante est applicable à toutes celles présentées précédemment. Néanmoins, on ne s'intéresse dans la suite qu'au problème de sac à dos multi-objectif unidimensionnel en variables binaire MOKP.

4.3.1 Formulation du problème

Étant donné un ensemble de n objets, où chaque objet i est caractérisé par un poids w_i et p coûts c_j^i pour $j = 1, \ldots, p$, correspondant à p objectifs $Z_1, \ldots, Z_p$, le but est de sélectionner

un sous-ensemble d'objets à charger dans un sac tout en respectant sa capacité ω, afin de maximiser simultanément ces p objectifs. Dans sa forme la plus simple, appelée sac à dos unidimensionnel avec variables binaires, le problème MOKP se formule comme suit :

$$(MOKP)\begin{cases} \text{“ max ”} & Z_i(x) = \sum_{j=1}^{n} c_j^i x_j, \quad i = 1, \ldots, p \\ \text{s.c.} & \sum_{j=1}^{n} w_j x_j \leq \omega, \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, n. \end{cases} \tag{4.2}$$

où les coefficients c_j^i, w_j et ω sont des entiers positifs.
La variable x_i est la variable de décision telle que :

$$x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est chargé dans le sac,} \\ 0 & \text{sinon.} \end{cases}$$

4.3.2 Méthodes de résolution exactes pour MOKP

Les méthodes de résolution des problèmes MOKP sont décomposées en deux catégories : les méthodes exactes et les méthodes approximatives. Dans cette section, nous nous intéressons aux méthodes de résolution exacte, au regard de la littérature, il existe trois grandes techniques pour résoudre de manière exacte des problèmes MOKP : la méthode en deux phases, l'algorithme du plus court chemin et la programmation dynamique.

4.3.2.1 Méthode en deux phases

Visée et *al.* [138] ont proposé un algorithme en deux phases pour déterminer l'ensemble des solutions efficaces pour le problème MOKP dans le cas de deux objectifs.

La première phase est similaire à celle présentée dans la section (1.3.3.4), où l'ensemble des solutions supportées SE est généré en utilisant la recherche dichotomique de Aneja et Nair [6]. Dans la deuxième phase, les solutions efficaces non supportées sont recherchées en utilisant une procédure de séparation et évaluation dérivée de Martello et Toth [93].

Pour la seconde phase, à chaque triangle $\Delta(Z^r, Z^s)$ est associé un problème mono-objectif P_λ avec $\lambda = (Z_2^r - Z_2^s, Z_1^r - Z_1^s)$, représentant une direction de recherche orthogonale à l'hypoténuse. Ces problèmes sont ensuite résolus à l'aide d'une méthode PSE de Martello et Toth [93] pour le sac à dos mono-objectif, dont la partie évaluation est adaptée pour explorer chaque triangle. Dans le cadre bi-objectif, la zone pertinente à explorer illustrée par la figure (Fig. 4.1) est délimitée par trois bornes :

- $\underline{Z}_1 = Z_1^r$ est une bonne borne inférieure des solutions non supportées dont les images sont incluses dans le triangle pour le premier objectif.

- $\underline{Z}_2 = Z_2^s$ pour le second objectif.
- Une borne inférieure des solutions du problème P_λ est donnée par $\underline{Z}_\lambda = \lambda Z^N$, où Z^N est le point nadir local défini par $Z^N = (Z_1^r, Z_2^s)$.

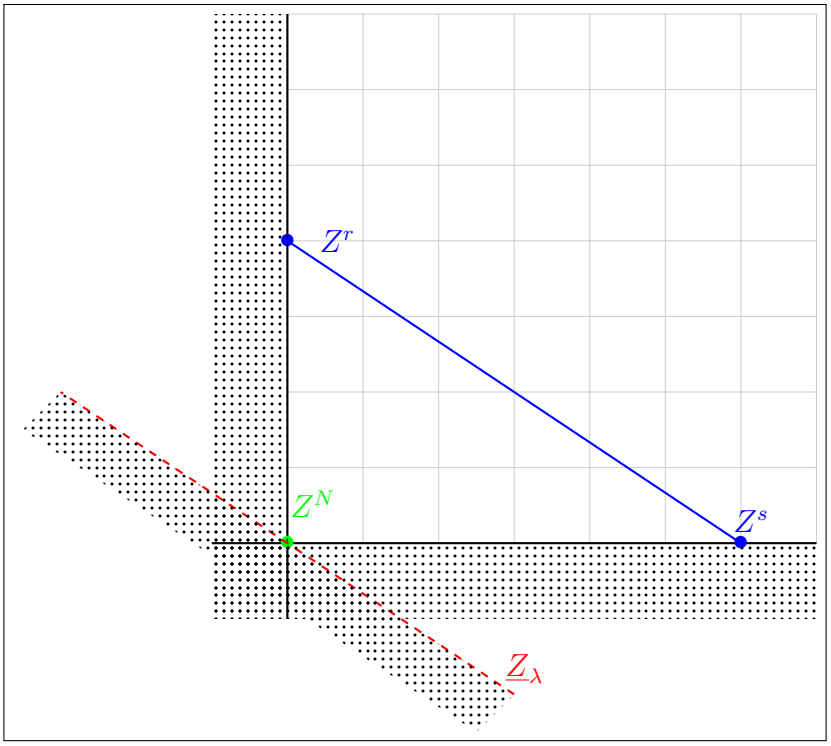


Fig. 4.1 : Illustration des bornes lors de l'exploration d'un triangle dans la seconde phase

Cette dernière borne est mise à jour au fur et à mesure que de nouvelles solutions potentiellement efficaces sont trouvées.

L'évaluation des nœuds pendant la PSE se fait en calculant séparément une borne supérieure pour chaque objectif Z_1, Z_2 et Z_λ. Si l'une de ces bornes est inférieure ou égale à la borne inférieure correspondante, le nœud est sondé.

Les auteurs appliquent un algorithme de réduction des variables à chaque problème résolu, pour chaque phase. L'exploration se concentre ensuite sur un ensemble réduit d'objets, avec des capacités de sac à dos ajustées en fonction des variables fixées.

4.3.2.2 Méthode du plus court chemin multi-objectif

Captivo et *al.* [27] ont proposé en 2003 une méthode exacte basée sur une transformation du MOKP en un problème de plus court chemin multi-objectif. Ce problème est ensuite résolu à l'aide d'une version adaptée de l'algorithme de marquage de Martins et *al.*[95].

Formulation du MOKP en un problème de plus court chemin multi-objectif

Le problème de plus court chemin multi-objectif est défini comme suit :
Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe orienté et connecté, où $\mathcal{S}$ est l'ensemble des sommets et $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ est l'ensemble des arcs. L'arc reliant les sommets i et j est noté (i, j). Soient $c^1(i, j), \ldots, c^p(i, j)$ les p critères associés à l'arc (i, j). Un chemin $\mathcal{P}$ d'un sommet de départ s à un sommet d'arrivée t dans $\mathcal{G}$ est une séquence d'arcs et de sommets de s à t, où le sommet initial d'un arc donné coïncide avec le sommet d'arrivée de l'arc précédent dans le chemin. Soit $f^k(\mathcal{P})$ la valeur d'un chemin $\mathcal{P}$ par rapport au critère k, pour tout $k = 1, \ldots, p$. L'objectif est de minimiser chaque critère k. Un chemin $\mathcal{P}$ est dit efficace si et seulement s'il n'existe pas de chemin $\mathcal{P}'$ dans $\mathcal{G}$ tel que $f^k(\mathcal{P}') \leq f^k(\mathcal{P})$, pour tout $k = 1, \ldots, p$, avec au moins une inégalité stricte. Le problème de plus court chemin multi-objectif consiste à rechercher tous les chemins efficaces d'un sommet de départ s à un nœud d'arrivée t dans $\mathcal{G}$.

Il existe une correspondance bijective entre l'ensemble des solutions réalisables du problème du sac à dos et l'ensemble des chemins de s à t dans le graphe $\mathcal{G}$. les auteurs pondèrent les arcs avec l'opposé des profits associés aux variables ; de manière à pouvoir utiliser directement un algorithme de plus courts chemins. Ainsi, le chemin dans $\mathcal{G}$ et une solution réalisable du problème de sac à dos ont le même profit. Les idées fondamentales de cette technique sont illustrées dans les figures (Fig. 4.2) et (Fig. 4.3), elles peuvent être formulées comme suit :

1. Déterminer l'ensemble des sommets en utilisant une technique de niveaux. Chaque niveau intermédiaire a plusieurs sommets. Le premièr niveau (niveau 0) comprend un seul sommet s (le sommet de départ). Ensuite, le niveau j peut être directement obtenue à partir du niveau $j-1$, pour tous $j = 1, \ldots, n$, où chaque niveau, de $j = 1$ à n, a au plus $\omega + 1$ sommets, $j^0, \ldots, j^\omega$. Enfin, le dernier niveau (niveau $n+1$) a un seul sommet t (le sommet d'arrivée). Ainsi, $|S| \leq (\omega + 1)n + 2$.

2. Définir l'ensemble des arcs et les coûts des arcs. Le sommet s a toujours deux arcs sortants : $(s, 1^0)$ avec $c(s, 1^0) = (0, \ldots, 0)$, et $(s, 1^{w_1})$ avec $c(s, 1^{w_1}) = -(c_1{}^1, \ldots, c_1{}^p)$. Le premier arc représente la décision de ne pas inclure l'objet 1 dans le sac à dos, et le deuxième arc représente la décision de l'inclure. En ce qui concerne les niveaux de $j = 1$ à $j = n-1$, chaque sommet j^a, pour $a = 0, \ldots, \omega$, a au plus deux arcs sortants :

   - L'arc $(j^a, (j+1)^a)$ avec $c(j^a, (j+1)^a) = (0, \ldots, 0)$, ce qui signifie que l'objet $(j+1)$ n'est pas inclus dans le sac à dos.
   - L'arc $(j^a, (j+1)^a + w_{j+1})$ avec $c(j^a, (j+1)^a + w_{j+1}) = (c^1_{j+1}, \ldots, c^p_{j+1})$, ce qui signifie que l'objet $(j+1)$ n'est pas inclus dans le sac à dos.

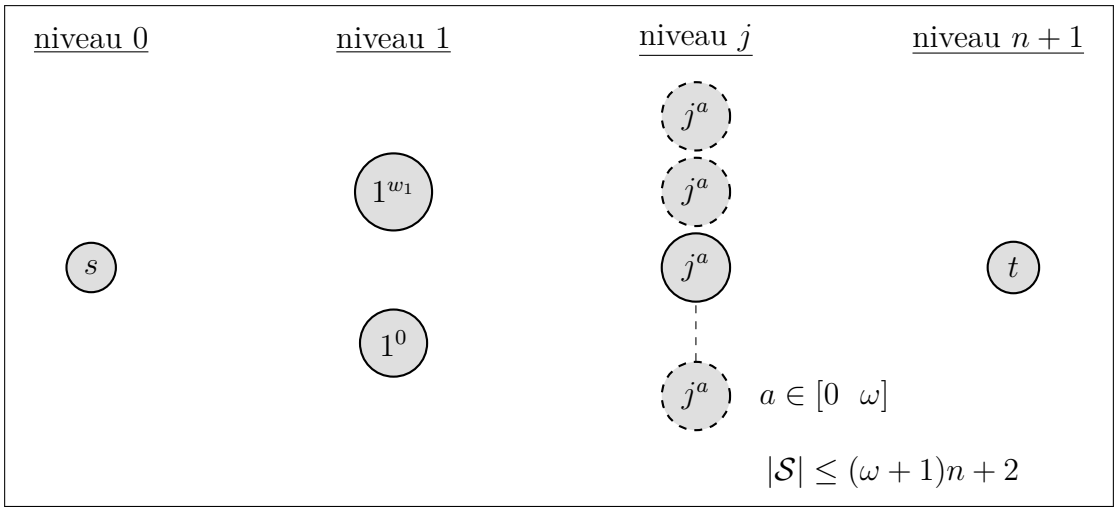


Fig. 4.2 : Création de sommets

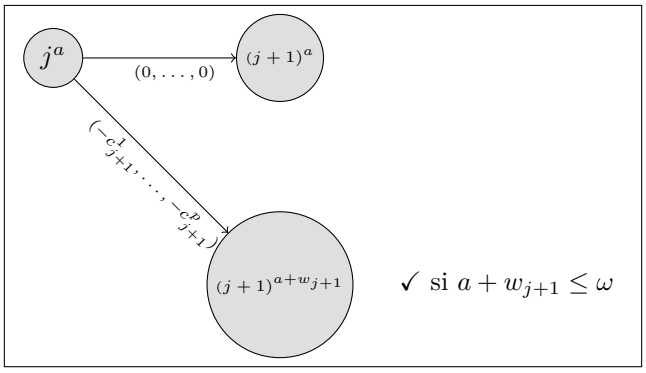


Fig. 4.3 : Création d'arcs et leurs coûts correspondants

Algorithme du plus court chemin pour MOKP

Après la conversion du MOKP en un problème de plus court chemin, les auteurs adaptent la procédure de marquage proposée dans [95]. Les différentes étapes de la méthode sont résumées dans l'algorithme (Algo. 4.1).

A la fin de cette algorithme, l'ensemble des chemins non dominés est déterminés, se qui constituent l'ensemble des solutions non dominées du problème MOKP avec un signe opposé.

Algo. 4.1 : Algorithme du plus court chemin pour MOKP

Input ↓ : $\mathcal{G}$: Le graphe construit depuis l'instance du problème.
Output ↑ : S : L'ensemble des solutions non dominées.
Initialisation :
$S(1^0) \leftarrow \{(0,\ldots,0)\}$ et $S(1^{w_1}) \leftarrow \{(-c_1^1,\ldots,-c_1^p)\}$;
$T \leftarrow \{0, w_1\}$ et $V \leftarrow \{\}$; `% T et V sont deux listes des cumuls des poids associés aux noeuds au niveau` $j-1$ `et` j `respectivement.`
Pour $j = 2$ *à* n **faire**
  Pour $a = 0$ *à* ω **faire**
    Si $a \in T$ **alors**
      $V \leftarrow V \cup \{a\}$;
      Si $a - w_j \in T$ **alors**
        $S(j^a) \leftarrow \big((S(j-1)^a) \cup \{(-c_j^1,\ldots,-c_j^p)\} + S((j-1)^{a-w_j})\big)$;
        $S(j^a) \leftarrow$ les points non dominés dans $S(j^a)$;
      Sinon
        $S(j^a) \leftarrow S((j-1)^a)$;
      Fsi
    Sinon
      Si $a - w_j \in T$ **alors**
        $S(j^a) \leftarrow \{(-c_j^1,\ldots,-c_j^p)\} + S((j-1)^{a-w_j})$;
        $V \leftarrow V \cup \{a\}$;
      Fsi
    Fsi
  Fpour
  $T \leftarrow V$ et $V \leftarrow \{\}$;
Fpour
$S(t) \leftarrow$ les points non dominés dans $\bigcup_{a=0}^{\omega} S(n^a)$;
$S \leftarrow S(t)$;

Soulignons que cette méthode est valable quel que soit le nombre d'objectifs à optimiser. Cependant, les auteurs ont initialement mis en œuvre l'algorithme uniquement pour le cas bi-objectif. Figueira et *al.* [58] ont présenté par la suite les résultats d'une approche similaire, basée sur les résultats de Klamroth et Wiecek [83] concernant la structure des graphes. Leur travail présente plusieurs types de graphes construits à partir des instances de différents problèmes de sac à dos. Figueira et *al.* [58] ont utilisé ces structures pour le problème de sac à dos multi-objectif non borné, à variables entières, avant d'appliquer une procédure de calcul des plus longs chemins multi-objectifs.

4.3.2.3 Programmation dynamique

Le processus de la programmation dynamique proposé par Bazgan et *al.* [10] consiste en n-phases (n est le nombre d'objets du problème MOKP). Cette méthode est une énumération implicite qui se base en particulier sur le filtrage des états. A chaque étape de résolution k, un ensemble d'états A^k est généré tel que A^k représente un sous ensemble de solutions réalisables du problème (4.2) engendré par les cas premiers objets. La génération des états de A^k se fait à partir de l'ensemble A^{k-1} des états générés à l'étape $k-1$, cela en appliquant trois relations de dominance pour éliminer les états ne pouvant pas être efficaces pour le problème (4.2).

- La première de ces relations est très facile à appliquer, elle sert à déterminer les états de l'ensemble A^{k-1} qui peuvent être contenues dans A^k (l'objet k n'est pas sélectionné). Si la capacité résiduelle associée à un état $s^{k-1} \in A^{k-1}$ est supérieure ou égale à la somme des poids des objets restants $k, \ldots, n$, alors la seule façon d'obtenir une solution potentiellement efficace à partir de s^{k-1} est d'y ajouter tous ces objets, donc inutile de générer des états sans l'objet k.

- La deuxième relation proposée est basée sur la dominance de Pareto (voir sous section (1.2.1.1)). Soient $s^k, \tilde{s}^k \in A^k$ et soit R et $\tilde{R}$ les capacités résiduelles du sac à l'état s^k et $\tilde{s}^k$ respectivement. Si $R \geq \tilde{R}$, alors tous les objets pouvant être ajoutés à l'état $\tilde{s}^k$ peuvent également l'être à l'état s^k. En outre, si $s^k \geqq \tilde{s}^k$ alors l'état s^k permettra d'aboutir à des solutions au moins aussi bonnes qu'à partir de $\tilde{s}^k$, ce dernier peut donc être éliminé pour la suite.

- La troisième relation est utilisée pour écarter des états dont la borne supérieure est dominée par une solution potentiellement non dominée. Les auteurs ont utilisé pour calculer la borne supérieure d'un état la formule indroduite dans [93]. Cette relation est plus difficile à vérifier que les deux premières relations, car elle nécessite beaucoup plus de tests et d'informations sur les états. Elle nécessite de calculer, pour chaque état auquel elle est appliquée, une borne supérieure sur chaque objectif selon deux ordres différents d'objets.

Notons qu'on utilisant l'ordre "$\geqq$" dans cette méthode nous permet d'obtenir uniquement l'ensemble complet minimal des solutions efficaces.

Les auteurs ont implémenté leur méthode sur des instances du problème MOKP avec deux et trois objectifs et ont montré qu'elle était extrêmement efficace par rapport à la méthode de Captiva et *al.* [27] sur les instances du problème MOKP en terme du temps d'exécution et du taille des instances résolues.
Le principal inconvénient de cette méthode est la consommation excessive de la mémoire de temps processeur, ce qui nécessite de très performants outils de programmation. Pour cette raison, Figueira et *al.* [57] ont proposé de nouvelles techniques de filtrage des états afin d'accélérer le processus de résolution. Récemment, Correia et *al.* [36] ont développé un algorithme de programmation dynamique pour améliorer l'utilisation de la mémoire dans le but de calculer l'ensemble des solutions efficaces, tant dans l'espace des critères que dans l'espace des décisions.

4.4 Conclusion

Nous avons présenté dans ce chapitre, la définition du problème de sac à dos multi-objectif en passant en revue sur ses variantes et quelques méthodes de résolution exactes dans le cas de variables binaires. Le chapitre suivant rapporte notre contribution dans le domaine de l'optimisation d'une fonction linéaire sur l'ensemble des solutions efficaces d'un problème de sac à dos bi-objectif en variables binaires.

Chapitre 5

Optimisation sur l'ensemble des solutions efficaces d'un BOKP

5.1 Introduction

La plupart des travaux de recherche dans le domaine de l'optimisation efficace des ensembles considèrent le cas de variables continues ou discrètes. Cependant, à notre connaissance, aucune étude n'a été menée spécifiquement dans le cas des variables binaires. Comme la plupart des problèmes d'optimisation combinatoire peuvent être modélisés sous la forme d'un problème de sac à dos, nous avons choisi de nous focaliser sur ce problème, en particulier sur sa version bi-objectif. Dans ce chapitre, nous proposons un algorithme pour optimiser une fonction linéaire sur l'ensemble efficace d'un problème de sac à doc bi-objectif en variables binaires désigné dans ce qui suit par BOKP. Cette méthode est basée sur la programmation dynamique au lieu de la programmation linéaire ou des techniques de coupes qui sont les plus utilisées dans les études précédentes. Elle s'inspire de l'algorithme de programmation dynamique développé par Bazgan et *al.* [10] pour la résolution d'un BOKP. Cependant, nous utilisons certaines des relations de dominance proposées dans [10] afin d'éliminer certaines solutions partielles qui ne peuvent pas fournir de solutions efficaces. Une autre relation que nous proposons est utilisée pour écarter certaines solutions partielles qui ne peuvent pas améliorer le critère principal, cette relation permet d'éliminer précocement plus de solutions qu'elles soient efficaces ou pas. Afin d'obtenir une solution optimale, il est necessaire de garder les solutions dans l'espace de décision en mémoire lors de l'exécution de l'algorithme, ce qui nécessite plus d'espace mémoire et un temps de calcul plus long avec la programmation dynamique. Pour faire face à ce problème, notre algorithme utilise un programme de test d'efficacité pour d'une part garantir l'efficacité de la solution et d'autre part renvoyer la solution

efficace correspondante dans l'espace de décision sans enregistrer les variables de décision en mémoire. Nous évaluons la performance de l'algorithme proposé sur des instances du BOKP bien connues. Nous comparons nos résultats avec ceux fournis par l'algorithme de Jorge [75] en termes de pourcentage de solutions efficaces calculées et de temps d'exécution.

5.2 Concepts basiques et définitions

Soit un sac d'une capacité ω et un ensemble de n objets caractérisés par leur poids w_j et leur valeur (coût) c_j^1, c_j^2 selon deux critères Z_1 et Z_2 respectivement, $j \in \{1, \ldots, n\}$. Un problème BOKP consiste à sélectionner un sous ensembles d'objets de manière à maximiser le profit apporté par les objets choisis sur les deux critères Z_1 et Z_2 tout en respectant la contrainte de capacité du sac. Le problème de sac à dos bi-objectif peut s'écrire mathématiquement comme suit :

$$(BOKP) \begin{cases} \text{“ max ”} Z_i(x) & = & \displaystyle\sum_{j=1}^{n} c_j^i x_j, \ i = 1, 2 \\ \text{s.c.} & & \displaystyle\sum_{j=1}^{n} w_j x_j \leq \omega, \\ & & x_j \in \{0, 1\}, \ \forall j \in \{1, \ldots, n\}, \end{cases} \tag{5.1}$$

où x_j est une variable de décision telle que :

$$x_j = \begin{cases} 1 \text{ si l'objet } j \text{ est sélectionné dans le sac,} \\ 0 \text{ sinon.} \end{cases}$$

Nous désignons par $c^{(i)}$, $i = 1, 2$, le vecteur des coûts associés au $i^{\text{ème}}$ critère tel que :
$c^i = (c_j^i), j \in \{1, \ldots, n\}, i \in \{1, 2\}$.
Tous les paramètres ω, c_j^i, w_j, pour $i = 1, 2$, $j = 1, \ldots, n$ sont des entiers positifs.
Pour éviter les solutions triviales on suppose que :

- $w_j \leq \omega, \quad \forall j = 1, \ldots, n.$
- $\displaystyle\sum_{j=1}^{n} w_j > \omega.$

On note X l'ensemble des solutions réalisables et E l'ensemble des solutions efficaces du problème $(BOKP)$. Soit $\phi(x) = \displaystyle\sum_{j=1}^{n} d_j x_j$ une fonction linéaire appelée “objectif principal”, où $d_j, j \in \{1, \ldots, n\}$ est supposé être un entier positif.

Le problème d'optimisation sur l'ensemble des solutions efficaces d'un BOKP peut être formulé comme suit :

$$(P_E) \begin{cases} \max & \phi(x) = \displaystyle\sum_{j=1}^{n} d_j x_j \\ \text{s.c.} & x = (x_1 \ \ldots \ x_n)' \in E. \end{cases} \tag{5.2}$$

5.2.1 Notions liées à la programmation dynamique

Le processus séquentiel utilisé par la programmation dynamique consiste en n phases. À chaque phase k, l'ensemble de toutes les solutions réalisables composées des objets appartenant exclusivement aux k premiers objets ($k = 1, \ldots, n$). Ces solutions sont appelés "états".

Considérons le problème suivant :

$$(BOKP_k)\begin{cases} \text{“ max ”} & Z_i(x) = \sum_{j=1}^{k} c_j^i x_j, \quad i = 1, 2 \\ \text{s.c.} & \sum_{j=1}^{k} w_j x_j \leq \omega, \\ & x_j \in \{0, 1\}, \quad \forall j \in \{1, \ldots, k\}. \end{cases} \tag{5.3}$$

Le problème $(BOKP_k)$ est le BOKP engendré par les k premiers objets.

Définition 5.1 *(Etat)*
Un état noté par s^k, $s^k = (s_1^k, s_2^k, s_3^k, s_4^k)$ représente une solution réalisable du problème $(BOKP_k)$, où s_i^k est la valeur du critère i, $(i = 1, 2)$ qui lui correspond, s_3^k et s_4^k le poids et la valeur de l'objectif principal qui lui sont associés respectivement.

L'ensemble de toutes les solutions réalisables du problème $(BOKP_k)$, $k \in \{1, \ldots, n\}$ est définie par S^k :

$$\begin{aligned} S^k = S^{k-1} \cup \; & \Big\{ \Big(s_1^{k-1} + c_k^1, s_2^{k-1} + c_k^2, s_3^{k-1} + w_k, s_4^{k-1} + d_k \Big) \mid \\ & s_3^{k-1} + w_k \leq \omega, \; s^{k-1} \in S^{k-1} \Big\}. \end{aligned} \tag{5.4}$$

Notons que :

- L'ensemble initial des états S^0, ne contient que l'état $s^0 = (0, 0, 0, 0)$ qui correspond au sac à dos vide.
- L'ensemble final des états S^n représente l'ensemble de toutes les solutions réalisables du $(BOKP)$.
- Les notions de l'efficacité et la dominance définies sur l'ensemble réalisable X (voir section (1.2.1)) sont aussi définies de la même manière sur S^k. Ainsi, pour $s^k, \tilde{s}^k \in S^k$, on dit que s^k domine $\tilde{s}^k$ et on note par $s^k \underline{\Delta} \tilde{s}^k$ si et seulement si $s_i^k \geq \tilde{s}_i^k, \forall i \in \{1, 2\}$, avec au moins une inégalité stricte.

Définition 5.2 *(Extension)*
Soit un sous-ensemble $J \subseteq \{k+1, \ldots, n\}$ *tel que :* $s_3 + \sum_{j \in J} w_j \leq \omega$*, un état* $s^n \in S^n$ *est dit une extension d'un état* $s^k \in S^k, (k \leq n)$ *si et seulement si*

$$s_i^n = s_i^k + \sum_{j \in J} c_j^i, i = 1, 2;\ s_3^n = s_3^k + \sum_{j \in J} w_j;\ s_4^n = s_4^k + \sum_{j \in J} d_j. \tag{5.5}$$

Soient $s^k, \tilde{s}^k \in S^k$, nous définissons :

— La relation lexicographique notée par $\underline{\Delta}_{lex}$ dans l'ensemble S^k comme suit :

$$s^k \underline{\Delta}_{lex} \tilde{s}^k \Leftrightarrow s_t^k > \tilde{s}_t^k, \tag{5.6}$$

où $t = \min\{i \in \{1, 2\} : s_i^k \neq \tilde{s}_i^k\}$ ou bien $s_i^k = \tilde{s}_i^k, \forall i \in \{1, 2\}$.

— La relation lexicographique notée par $\geq_{lex}$ dans l'ensemble S^k comme suit :

$$s^k \geq_{lex} \tilde{s}^k \Leftrightarrow s_3^k < \tilde{s}_3^k \text{ ou bien } (s_3^k = \tilde{s}_3^k \text{ et } s^k \underline{\Delta}_{lex} \tilde{s}^k). \tag{5.7}$$

5.2.2 Quelques notations

Nous adoptons dans les sections qui suivent, les notations suivantes :

- E_f : le sous-ensemble des solutions efficaces de $(BOKP)$ calculé par la méthode proposée.
- ND_f : le sous-ensemble des solutions non dominées correspondant à E_f.
- ϕ_{inf} : une borne inférieure de l'objectif principal ϕ.
- x_{opt} : une solution optimale de (P_E).
- ϕ_{opt} : la valeur optimale de l'objectif principal ϕ qui correspond à x_{opt}.

5.3 Description générale de la méthode

Dans cette section, nous décrivons le principe général de la méthode en mettant en évidence chaque procédure utilisée. La méthode consiste en m étapes $(m \leq n)$. Tout d'abord, nous commençons par une borne inférieure ϕ_{inf} de la fonction ϕ, puis à chaque étape $k,\ k = 1, \ldots, m$, nous générons et nous trions progressivement un sous-ensemble d'états $A^k \subseteq S^k$ à partir de A^{k-1}, avec $A^0 = S^0$. Nous utilisons des relations de comparaison pour écarter certains états de A^k qui ne peuvent pas conduire à des solutions efficaces qui améliorent ϕ_{inf}. Nous calculons un sous-ensemble de solutions efficaces améliorant la valeur du critère principal, puis nous mettons à jour ϕ_{inf}.
Afin d'éviter de générer des états inutiles dans A^k, nous utilisons les deux relations D_r^k et $D_{\underline{\Delta}}^k$ proposées dans [10]. Une autre relation, que nous appelons Z_b^k, est utilisée pour exclure de A^k

les états dont la borne supérieure est dominée par des solutions non dominées déjà trouvées. Pour écarter les états qui ne peuvent pas conduire à une amélioration de la fonction ϕ nous proposons une nouvelle relation appelée Φ_b^k.

5.3.1 Calcul d'une borne inférieure de l'objectif principal

Pour déterminer une borne inférieure de l'objectif principal ϕ dans E, nous devons trouver au moins une solution efficace de $(BOKP)$.

Deux solutions efficaces extrêmes x^1_{opt} et x^2_{opt} peuvent toutefois être trouvées lexicographiquement comme suit :

1. Résoudre le problème à objectif unique (P_1) induit par le premier objectif Z_1, $(P_1) : \Big\{\max \quad Z_1(x) = c^{(1)}x, x \in X\Big\}$, soit x^1 une solution optimale de (P_1). Si elle est unique, alors $x^1_{opt} = x^1$.

   Sinon, si la solution optimale n'est pas unique, nous résolvons le problème : $(P_2) : \Big\{\max \quad Z_2(x) = c^{(2)}x, x \in X, c^{(1)}x = c^{(1)}x^1\Big\}$, parmi les solutions optimales trouvées, nous sélectionnons x^1_{opt} qui donne la meilleure valeur du critère principal ϕ.
2. De la même manière, nous obtenons la deuxième solution efficace x^2_{opt}.

Ainsi, nous obtenons ϕ_{inf} une borne inférieure de la fonction ϕ, telle que $\phi_{inf} = \max\{\phi(x^1_{opt}), \phi(x^2_{opt})\}$.

5.3.2 Test d'efficacité

Pour connaître la nature d'un nouveau état généré $s^* \in S^n$, qui correspond à une solution réalisable x^* tel que : $s^* = (s^*_1, s^*_2, s^*_3, s^*_4) \in S^n$, et $c^{(1)}x^* = s^*_1$, $c^{(2)}x^* = s^*_2$, nous utilisons le test d'efficacité d'Ecker et Kouada (voir [47]) comme suit :

Soit $(BOKP) : \{\text{“}max\text{”}\ (c^{(1)}\ c^{(2)})x,\ x \in X\}$ le problème définie dans (5.1).

Théorème 5.1 *$x^* \in X$ est une solution efficace du BOKP si et seulement si la valeur optimale de la fonction objectif Θ est nulle dans le problème de programmation linéaire en nombres entiers mixtes suivant :*

$$(P_\Theta(x^*)) \begin{cases} \max & \Theta = \psi_1 + \psi_2 \\ s.c. & c^{(1)}x = \psi_1 + c^{(1)}x^*, \\ & c^{(2)}x = \psi_2 + c^{(2)}x^*, \\ & x \in X; \quad \psi_i \ \ \textit{sont des réels non négatifs pour } i = 1, 2. \end{cases} \tag{5.8}$$

5.3.3 Comparaison entre les états

Les relations suivantes sont utilisées pour comparer entre les états.

Définition 5.3 (Relation D_r^k [10])
Soit $s^k \in S^k$ et $\tilde{s}^{k-1} \in S^{k-1}, k = 1, \ldots, n$. La relation D_r^k est définit comme suit :

$$s^k D_r^k \tilde{s}^{k-1} \Leftrightarrow \begin{cases} s^k & = & (\tilde{s}_1^{k-1} + c_k^1, \tilde{s}_2^{k-1} + c_k^2, \tilde{s}_3^{k-1} + w_k, \tilde{s}_4^{k-1} + d_k), \\ & & et \\ \tilde{s}_3^{k-1} & \leq & \omega - \sum_{j=k}^{n} w_j. \end{cases} \tag{5.9}$$

L'équation $\tilde{s}_3^{k-1} \leq \omega - \sum_{j=k}^{n} w_j$ signifie que tous les objets ($k, \ldots, n$) peuvent être ajoutés au sac à dos, de sorte que la seule extension de s^k qui peut représenter une solution efficace est celle qui contient les objets $k, \ldots, n$, il n'est donc pas nécessaire de calculer les extensions sans l'objet k. Cette relation est utilisée pour déterminer les états de A^{k-1} qui peuvent appartenir à A^k à l'étape k. Les états de $A^k, k = 1, \ldots, n$, sont triés selon l'ordre décroissant par rapport à la relation $\geq_{lex}$ (équation (5.7)). Cet ordre permet de déterminer facilement j le premier indice de l'état dans A^{k-1} qui n'est pas dominé par rapport à la relation D_r^k, il n'est donc pas nécessaire d'inclure les états $s^{k-1(1)}, \ldots, s^{k-1(j-1)}$ dans A^k.

Définition 5.4 (Relation $D_{\underline{\Delta}}^k$ [10])
Soient deux états $s^k, \tilde{s}^k \in S^k, k = 1, \ldots, n$. La relation $D_{\underline{\Delta}}^k$ est définit comme suit :

$$s^k D_{\underline{\Delta}}^k \tilde{s}^k \Leftrightarrow s^k \underline{\Delta} \tilde{s}^k \quad et \quad s_3^k \leq \tilde{s}_3^k, \ k < n. \tag{5.10}$$

Puisque $s_3^k \leq \tilde{s}_3^k$, alors tous les objets qui peuvent être générés à partir de $\tilde{s}^k$ peuvent également l'être à partir de s^k. Par conséquent, si $s^k \underline{\Delta} \tilde{s}^k$, l'état s^k fournira des solutions au moins aussi bonnes que celles fournies par $\tilde{s}^k$. Ainsi, ce dernier peut être omis.

Soit $u = (u_1, u_2, u_\phi)$ une borne supérieure d'une solution réalisable représentée par un état s^k, u_1, u_2 sont les bornes supérieures des objectifs Z_1 et Z_2 respectivement et u_ϕ une borne supérieure de ϕ. Nous utilisons la borne supérieure définie par Martello et Toth dans [93].

Soit $O_i, i = 1, 2$, l'ordre décroissant des ratios coût/poids (l'efficacité de l'objet par rapport au critère i) c_j^i/w_j, $j \in \{k+1, \ldots, n\}$, $i = 1, 2$. t_i est la position du premier objet dans $\{k+1, \ldots, n\}$, qui ne peut être ajouté à s^k lorsque les objets sont réordonnés en fonction de l'ordre O_i. Soit $\overline{\omega}(s^k) = \omega - s_3^k$ la capacité résiduelle associée à l'état s^k. Une borne supérieure

u_i de l'objectif i est calculée comme suit :

$$u_i = s_i^k + \sum_{j=k+1}^{t_i-1} c_j^i + \max\left\{ \lfloor \overline{\omega}(s^k) \frac{c_{t_i+1}^i}{w_{t_i+1}} \rfloor, \lfloor c_{t_i}^i - (w_{t_i} - \overline{\omega}(s^k)) \frac{c_{t_i-1}^i}{w_{t_i-1}} \rfloor \right\}. \tag{5.11}$$

De même, pour l'objectif principal ϕ :

$$u_\phi = s_4^k + \sum_{j=k+1}^{t_\phi-1} d_j + \max\left\{ \lfloor \overline{\omega}(s^k) \frac{d_{t_\phi+1}}{w_{t_\phi+1}} \rfloor, \lfloor d_{t_\phi} - (w_{t_\phi} - \overline{\omega}(s^k)) \frac{d_{t_\phi-1}}{w_{t_\phi-1}} \rfloor \right\}, \tag{5.12}$$

où t_ϕ est la position du premier objet dans $\{k+1, \ldots, n\}$ qui ne peut pas être ajouté à l'état s^k lorsque les objets sont réordonnés selon l'ordre O_ϕ. O_ϕ est l'ordre décroissant des rapports d_j/w_j, $j \in \{k+1, \ldots, n\}$.

En plus des relations précédentes, nous introduisons une nouvelle relation (définition 5.5) afin de réduire efficacement l'ensemble A^k et d'éliminer autant d'états inutiles que possible.

Définition 5.5 (Relation Z_b^k)
Soit $s^k \in S^k, k = 1, \ldots, n$ et $u = (u_1, u_2, u_\phi)$ sa borne supérieure associée. S'il existe $s = (s_1, s_2) \in ND_f$ tel que $(s_1, s_2) \underline{\Delta} (u_1, u_2)$, s^k ne peut pas produire une solution efficace, il peut donc être éliminé.

Dans la proposition suivante, nous montrons comment exclure un état s^k qui ne peut pas être optimale pour le problème principal (P_E) en comparant sa borne supérieure par rapport à l'objectif principal avec la borne inférieure actuelle de l'objectif principal.

Proposition 5.1 (Relation Φ_b^k)
Soit $s^k \in S^k, k = 1, \ldots, n$ et $u = (u_1, u_2, u_\phi)$ sa borne supérieure associée. Soit ϕ_{inf} la borne inférieure de l'objectif ϕ mis à jour à l'étape k. Si $\phi_{inf} \geq u_\phi$, alors s^k ne peut pas aboutir à une amélioration de la valeur de l'objectif principal, et s^k peut donc être omis.
On dit que s^k est dominé au sens de la relation Φ_b^k.

Preuve
Soit ϕ_{inf} la borne inférieure de l'objectif ϕ mise à jour à l'étape k. Considérons que s^k est dominé au sens de Φ_b^k. Cela implique qu'il existe un autre état $\tilde{s}^k$ dans S^k conduisant à un état efficace s^*, $s^* = (s_1^*, s_2^*, s_3^*, s_4^*)$ tel que $s_4^* = \phi_{inf}$. Soit u_ϕ une borne supérieure de s^k sur l'objectif principal, ce qui signifie que $u_\phi \geq s_4^n$, pour toute extension $s^n = (s_1^n, s_2^n, s_3^n, s_4^n)$ de l'état s^k. Puisque $\phi_{inf} \geq u_\phi$, alors s^k est dominé par $\tilde{s}^k$ et ne peut pas fournir un état efficace avec une meilleure valeur de l'objectif principal ϕ.
□

5.3.4 Etape générale k

Soit $A^{k-1} = \{s^{k-1(1)}, \ldots, s^{k-1|A^{k-1}|}\}$ l'ensemble des états retenus à l'étape $k-1$, un sous-ensemble d'états A^k est généré et réduit progressivement comme suit :
Nous commençons tout d'abord par l'identification des états de A^{k-1} qui peuvent appartenir à A^k. Pour ce faire, la relation de dominance D_k^r est appliquée. L'ordre des états dans A^{k-1} permet de définir l'indice j tel que les états $s^{k-1(j+1)}, s^{k-1(j+2)}, \ldots, s^{k-1(|A^{k-1}|)}$ puissent être ajoutés dans A^k. Nous générons ensuite de nouveaux états à partir de A^{k-1} en appliquant la relation de dominance $D_{\underline{\Delta}}^k$. Cette procédure appelée *generate&trim* est décrite dans l'algorithme (Algo. 5.1).

Algo. 5.1 : Procédure $generate\&trim(A^{k-1})$

$\%A^{k-1} = \{s^{k-1(1)}, \ldots, s^{k-1(i)}, \ldots, s^{k-1(|A^{k-1}|)}\}$ dans l'ordre décroissant de $\geq_{lex}$.
$A^k \leftarrow \emptyset$, $M^k \leftarrow \emptyset$, $i \leftarrow 1$, $j \leftarrow 1$;
% j, est l'indice du premier état dans A^{k-1} qui n'est pas dominé au sens de D_r^k.
Tant que $j \leq |A^{k-1}|$ *et* $s_3^{k-1(j)} + \sum_{l=k}^{n} w_l \leq \omega$ **faire**
  $j \leftarrow j + 1$;
Ftq
Tant que $i \leq |A^{k-1}|$ *et* $s_3^{k-1(i)} + w_k \leq \omega$ **faire**
  $s^k \leftarrow (s_1^{k-1(i)} + c_k^1, s_2^{k-1(i)} + c_k^2, s_3^{k-1(i)} + w_k, s_4^{k-1(i)} + d_k)$;
  Tant que $j \leq |A^{k-1}|$ *et* $s^{k-1(j)} \geq_{lex} s^k$ **faire**
    $Maintain\_NDS(s^{k-1(j)}, M^k, A^k)$;
    $j \leftarrow j + 1$;
  Ftq
  $Maintain\_NDS(s^k, M^k, A^k)$;
  $i \leftarrow i + 1$;
Ftq
Tant que $j \leq |A^{k-1}|$ **faire**
  $Maintain\_NDS(s^{k-1(j)}, M^k, A^k)$;
  $j \leftarrow j + 1$;
Ftq

Nous maintenons un sous-ensemble $M^k \subseteq A^k$ qui retient les états de A^k non dominés . Un nouvel état généré s^k est inséré dans A^k si et seulement s'il n'existe aucun état m^k dans M^k tel que $m^k \underline{\Delta} s^k$. Les sous-ensembles M^k et A^k sont mis à jour à chaque insertion d'un nouvel état. M^k est trié par ordre décroissant en fonction de la relation $\underline{\Delta}_{lex}$ (voir Algo. 5.2).

Algo. 5.2 : Procédure $Maintain\_NDS(s^k, M^k, A^k)$

$\%M^k = \{m^{k(1)}, \ldots, m^{k(|M^k|)}\}$ dans l'ordre décroissant de $\underline{\Delta}_{lex}$.
$dom \leftarrow faux, \ell \leftarrow 1$;
Tant que $\ell \leq |M^k|$ *et* $m_1^{k(\ell)} \geq s_1^k$ **faire**
  $\ell \leftarrow \ell + 1$;
Ftq
Si $\ell > 1$ *et* $m_2^{k(\ell-1)} > s_2^k$ **alors**
  $dom \leftarrow vrai$;
Fsi
Si $dom = faux$ **alors**
  $M^k \leftarrow M^k \cup \{s^k\}$ dans la $\ell^{ème}$ position dans M^k ;
  $A^k \leftarrow A^k \cup \{s^k\}$;
  Si $\ell > 1$ *et* $m_1^{k(\ell-1)} = s_1^k$ *et* $m_2^{k(\ell-1)} < s_2^k$ **alors**
    $M^k \leftarrow M^k \setminus \{m^{k(\ell-1)}\}$;
    $\ell \leftarrow \ell - 1$;
  Fsi
  $\ell \leftarrow \ell + 1$;
  Tant que $\ell \leq |M^k|$ *et* $s_1^k \geq m_1^{k(\ell)}$ **faire**
    $M^k \leftarrow M^k \setminus \{m^{k(\ell)}\}$;
    $\ell \leftarrow \ell + 1$;
  Ftq
Fsi

Une fois que la génération et le tri des états sont effectués, nous recherchons des états efficaces qui améliorent ϕ_{inf} de la manière suivante :
Pour chaque état de M^k, une extension est générée selon l'ordre O_1. Seules les extensions améliorant ϕ et n'étant dominées par aucune solution dans ND_f sont listées. Ces dernières sont testées pour leur efficacité, puis on mis à jour E_f, ND_f et ϕ_{opt}. Cette étape est répétée selon O_2. Cette procédure appelée *test_nondominated* est donnée par l'algorithme (Algo. 5.3).

Algo. 5.3 : Procédure $test\_nondominated(ND_f, E_f, \phi_{inf}, s^n)$

$dom \leftarrow faux$;
Si $s_4^n \leq \phi_{inf}$ **alors**
  $dom \leftarrow vrai$;
Sinon
  $\ell \leftarrow 1$;
  Tant que $\ell \leq |ND_f|$ *et* $dom = faux$ **faire**
    Si $s^\ell \underline{\Delta} s^n$ **alors**
      $dom \leftarrow vrai$;
    Fsi
    $\ell \leftarrow \ell + 1$;
  Ftq
  Si $dom = faux$ **alors**
    résoudre (P_Θ) ; soit x^* une solution optimale.
    Si $\Theta = 0$ **alors**
      $ND_f \leftarrow ND_f \cup \{(s_1^n, s_2^n)\}$, $E_f \leftarrow E_f \cup \{x^*\}$, $\phi_{inf} \leftarrow s_4^n$;
    Sinon
      Si $\phi(x^*) > \phi_{inf}$ **alors**
        $ND_f \leftarrow ND_f \cup \{(Z_1(x^*)\ Z_2(x^*))\}$, $E_f \leftarrow E_f \cup \{x^*\}$;
        $\phi_{inf} \leftarrow \phi(x^*)$;
      Fsi
    Fsi
  Fsi
Fsi

Finalement, nous appliquons les relations Z_b^k et Φ_b^k de la manière suivante :
Pour chaque état s^k dans A^k, nous calculons sa borne supérieure associée sur les deux critères $u = (u_1, u_2)$. S'il existe une solution non dominée s dans ND_f telle que $(s_1, s_2) > (u_1, u_2)$, s^k est éliminé, sinon nous calculons u_ϕ sa borne supérieure sur l'objectif principal, s^k est éliminé si $\phi_{inf} \geq u_\phi$ (voir Algo. 5.4).

Algo. 5.4 : Procédure $removing\_states(A^k, ND_f, \phi_{inf})$

$i \leftarrow 1$;
Tant que $i \leq |A^k|$ **faire**
  Calculer (u_1, u_2) une borne supérieure de $(s_1^{k(i)}, s_2^{k(i)})$.
  $j \leftarrow 1$, $remove \leftarrow faux$;
  Tant que $j \leq |ND_f|$ *et* $remove = faux$ **faire**
    Si $(s_1^{(j)}, s_2^{(j)}) \underline{\Delta} (u_1, u_2)$ **alors**
      $remove \leftarrow vrai$, $A^k \leftarrow A^k \setminus \{s^{k(i)}\}$;
    Sinon
      $j \leftarrow j + 1$;
    Fsi
  Ftq
  Si $remove = faux$ **alors**
    Calculer u_ϕ une borne supérieure de $s_4^{k(i)}$.
    Si $\phi_{inf} \geq u_\phi$ **alors**
      $A^k \leftarrow A^k \setminus \{s^{k(i)}\}$;
    Fsi
  Fsi
  $i \leftarrow i + 1$;
Ftq

L'algorithme se termine à l'étape n lorsque tous les objets sont pris en compte ou à une étape $m \leq n$, lorsque tous les états de l'étape $m - 1$ sont omis (A^{m-1} est vide et nous ne pouvons plus générer d'états).

5.4 Présentation technique de l'algorithme

La description technique de la méthode proposée est donnée par l'algorithme (Algo. 5.5) qui inclut toutes les procédures discutées dans la section précédente.

Algo. 5.5 : Optimizing over the (BOKP) efficient set

Input ↓ :
$\downarrow p^1_{(1\times n)}, \downarrow p^2_{(1\times n)}, \downarrow w_{(1\times n)}, \downarrow \omega, \downarrow d_{(1\times n)}$; % Les paramètres du problème (P_E)

Output ↑ :
$\uparrow x_{opt}, \uparrow \phi_{opt}$; % La solution optimale de (P_E) et la valeur optimale de ϕ

Initialisation :
$A^0 \leftarrow \{s^0\} = (0,0,0,0)$, $k \leftarrow 1$;
Calculer ϕ_{inf} une borne inférieure de ϕ ; soient x^1, x^2 les deux solutions efficaces extrêmes.
$E_f \leftarrow \{x^1, x^2\}$, $ND_f \leftarrow \{Z(x^1), Z(x^2)\}$;
Tant que $k \leq n$ *and* $|A^{k-1}| > 0$ **faire**
  $generate\&trim(A^{k-1})$;
  Si $k = n$ **alors**
    Pour $j = 1$ *à* $|M^k|$ **faire**
      $test\_nondominated(ND_f, E_f, \phi_{inf}, s^{k(j)})$;
    Fpour
  Sinon
    Pour $i = 1$ *à* 2 **faire**
      Ordonner les objets $k+1, \ldots, n$ selon O_i ;
      Pour $j = 1$ *à* $|M^k|$ **faire**
        $s^n \leftarrow m^{k(j)}$;
        Pour $\ell = k+1$ *à* n **faire**
          Si $s^n_3 + w_j \leq \omega$ **alors**
            $s^n \leftarrow (s^n_1 + p^1_\ell, s^n_2 + p^2_\ell, s^n_3 + w_\ell, s^n_4 + d_\ell)$;
          Fsi
        Fpour
        $test\_nondominated(ND_f, E_f, \phi_{inf}, s^n)$;
      Fpour
    Fpour
    $removing\_states(A^k, ND_f, \phi_{inf})$;
  Fsi
  $k \leftarrow k + 1$;
Ftq
$\phi_{opt} \leftarrow \phi_{inf}$;

5.5 Illustration numérique

Considérons l'exemple suivant :

$$(BOKP)\begin{cases} \max & Z_1(x) = 15x_1 + 16x_2 + 8x_3 + 12x_4 + 6x_5 + 14x_6 \\ \max & Z_2(x) = 3x_1 + 10x_2 + 12x_3 + 6x_4 + 11x_5 + 7x_6 \\ \text{s.c.} & 8x_1 + 3x_2 + 9x_3 + 5x_4 + 6x_5 + 8x_6 \leq 19, \\ & x_j \in \{0,1\}, j \in \{1,2,\ldots,6\}. \end{cases}$$

Soit le problème principal :

$$(P_E)\begin{cases} \max & \phi(x) = 2x_1 + 5x_2 + 9x_3 + 6x_4 + 4x_5 + 7x_6 \\ \text{s.c.} & x \in E. \end{cases}$$

- Initialisation :
  - $A^0 \leftarrow \{s^0\} = \{(0,0,0,0)\}$.
  - Calcul d'une borne inférieure de l'objectif principal ϕ :
    Les deux solutions efficaces extrêmes sont les suivantes :$x^1 = (1\,1\,0\,0\,0\,1)'$,
    $x^2 = (0\,1\,1\,0\,1\,0)'$,
    ϕ_{inf}=max$\{\phi(x^1), \phi(x^2)\}$=max$\{14, 18\} = 18$, $x_{opt} = x^2 = (0\,1\,1\,0\,1\,0)'$,
    $Z(x^1) = (45, 20)$, $Z(x^2) = (30, 33)$, $E_f = \{(1\,1\,0\,0\,0\,1)', (0\,1\,1\,0\,1\,0)'\}$,
    $ND_f = \{(45, 20), (30, 33)\}$.
- Etape 1.
  $k = 1$, $A^1 = \emptyset$, $M^1 = \emptyset$.
  1. Génération et tri des états : Nous commençons par examiner si les états de A^0 peuvent être inclus dans A^1.
     $s^0 = (0,0,0,0)$, $\sum_{j=1}^{6} w_j = 39 > \omega$, alors s^0 est inclue dans A^1.
     On génère s^1 à partir de s^0, $s^1 = (0,0,0,0) + (15,3,8,2) = (15,3,8,2)$.
     $A^1 \leftarrow \{(0,0,0,0), (15,3,8,2)\}$, $M^1 \leftarrow \{(15,3,8,2)\}$.
  2. Génération des extensions : l'ordre des objets $2,3,\ldots,6$ selon O_1 est : $2,4,6,5,3$.
     Ainsi $s^n = (15,3,8,2) + (16+12, 10+6, 3+5, 5+6) = (43,19,16,13)$, $s_4^n < \phi_{inf}$.
     s^n n'améliore pas ϕ_{inf}.
     L'ordre des objets $2,3,\ldots,6$ suivant O_2 est : $2,5,3,4,6$.
     $s^n = (15,3,8,2) + (16+6, 10+11, 3+6, 5+4) = (37,24,17,11)$.
     $s_4^n < \phi_{inf}$.

3. Elimination des états :

   - Calcul des bornes supérieures des états de A^1 :
     Etat s^0 : $(u_1, u_2) = (49, 34)$, n'est dominée par aucune solution dans ND_f, $u_\phi = 22 > \phi_{inf}$, donc s^0 ne peut pas être omis de A^1.
     Etat s^1 : $(u_1, u_2) = (46, 30)$, n'est dominée par aucune solution dans ND_f, $u_\phi = 15 < \phi_{inf}$, s^1 est retiré de A^1,
     $A^1 \leftarrow A^1 \setminus \{s^1\} = \{(0, 0, 0, 0)\}$.

- Etape 2.
  $k = 2$, $A^2 = \emptyset$, $M^2 = \emptyset$.

  1. Génération et tri des états :
     $A^1 = \{s^0\} = \{(0, 0, 0, 0)\}$, $\sum_{j=2}^{6} w_j = 31 > \omega$ alors s^0 est ajouté à A^2.
     On génère s^2 à partir de s^0, $s^2 = (0, 0, 0, 0) + (16, 10, 3, 5) = (16, 10, 3, 5)$.
     $A^2 \leftarrow \{s^0, s^2\} = \{(0, 0, 0, 0), (16, 10, 3, 5)\}$, $M^2 \leftarrow \{s^2\} = \{(16, 10, 3, 5)\}$.
  2. Génération des extensions : l'ordre des objets $3, 4, \ldots, 6$ selon O_1 est : $4, 6, 5, 3$. Ainsi $s^n = (16, 10, 3, 5) + (12 + 14, 6 + 7, 5 + 8, 6 + 7) = (42, 23, 16, 18)$. s^n n'est dominé par aucune solution dans ND_f mais $s^n_4 = 18 = \phi_{inf}$, s^n n'améliore pas ϕ_{inf}, il est donc inutile de tester son efficacité.
     L'ordre des objets $3, 4, \ldots, 6$ selon O_2 est : $5, 3, 4, 6$.
     $s^n = (16, 10, 3, 5) + (6 + 8, 11 + 12, 6 + 9, 4 + 9) = (30, 33, 18, 18)$.
     $s^n_4 = 18 = \phi_{inf}$.
  3. Elimination des états :

     - Calcul des bornes supérieures des états de A^2 :
       Etat s^0 : $(u_1, u_2) = (32, 29)$ n'est pas dominé. $u_\phi = 19 > \phi_{inf}$, alors s^0 ne peut être écarté de A^2.
       Etat s^2 : $(u_1, u_2) = (44, 34)$ n'est dominée par aucune solution de ND_f, $u_\phi = 22 > \phi_{inf}$, s^2 est donc maintenu dans A^2.

- Etape 3.
  $k = 3$, $A^3 = \emptyset$, $M^3 = \emptyset$.

  1. Génération et tri des états :
     $A^2 = \{s^0, s^2\} = \{(0, 0, 0, 0), (16, 10, 3, 5)\}$. s^0 et s^2 peuvent être inclus dans A^3 car $s^0_3 + \sum_{j=3}^{6} w_j = 28 > \omega$ et $s^2_3 + \sum_{j=3}^{6} w_j = 31 > \omega$.

On génère $s^{3(1)}$ à partir de s^0, $s^{3(1)} = (0,0,0,0) + (8,12,9,9) = (8,12,9,9)$.
$A^3 \leftarrow \{(0,0,0,0),(8,12,9,9)\}$, $M^3 \leftarrow \{(8,12,9,9)\}$.
On génère $s^{3(2)}$ à partir de s^2, $s^{3(2)} = (16,10,3,5) + (8,12,9,9) = (24,22,12,14)$.
$A^3 \leftarrow A^3 \cup \{s^2, s^{3(2)}\} = \{(0,0,0,0),(16,10,3,5),(8,12,9,9),(24,22,12,14)\}$.
$M^3 \leftarrow M^3 \setminus \{(s^{(1)}\} \cup \{s^{3(2)}\} = \{(24,22,12,14)\}$.

2. Génération des extensions :
   Etat $s^{3(2)}$: l'ordre des objets $4,5,6$ selon O_1 est : $4,6,5$.
   $s^n = (24,22,12,14) + (12,6,5,6) = (38,28,17,20)$. $s_4^n > \phi_{inf}$, alors nous testons l'efficacité de s^n en résolvant le problème :

$$(P_\Psi)\begin{cases} \max & \Theta = \psi_1 + \psi_2 \\ \text{s.c.} & 15x_1 + 16x_2 + 8x_3 + 12x_4 + 6x_5 + 14x_6 = \psi_1 + 38 \\ & 3x_1 + 10x_2 + 12x_3 + 6x_4 + 11x_5 + 7x_6 = \psi_2 + 28 \\ & 8x_1 + 3x_2 + 9x_3 + 5x_4 + 6x_5 + 8x_6 \leq 19 \\ & x_j \in \{0,1\},\ j = 1,2,\ldots,6; \quad \psi_1, \psi_2 \text{ sont des réels non négatifs.} \end{cases}$$

   Soit $x^* = (0\,1\,1\,1\,0\,0)'$ la solution optimale de (P_ψ) obtenue avec $\Theta \neq 0$, alors s^n n'est pas efficace, ainsi x^* est efficace, et $x_{opt} \leftarrow x^* = (0\,1\,1\,1\,0\,0)'$, $\phi_{inf} = s_4^n = 20$.
   $E_f \leftarrow E_f \cup \{x^*\} = \{(1\,1\,0\,0\,0\,1)', (0\,1\,1\,0\,1\,0)', (0\,1\,1\,1\,0\,0)'\}$, $ND_f \leftarrow ND_f \cup \{(s_1^n, s_2^n)\} = \{(45,20),(30,33),(36,28)\}$.
   L'ordre des objets $4,5,6$ suivant O_2 est : $5,4,6$. Ainsi : $s^n = (24,22,12,14) + (6,11,6,4) = (30,33,18,18)$, $s_4^n < \phi_{inf}$.

3. Elimination des états :

   - Calcul des bornes supérieures des états de A^3 :
     Etat s^0 : $(u_1,u_2) = (32,24)$ dominée par $(36,28)$, alors $A^3 \leftarrow A^3 \setminus \{s^0\}$.
     Etat s^1 : $(u_1,u_2) = (48,34)$ n'est pas dominée donc on calcule u_ϕ, $u_\phi = 22 > \phi_{inf}$, s^1 est maintenu dans A^3.
     Etat $s^{3(1)}$: $(u_1,u_2) = (26,29)$ dominée par $(30,33)$ alors $s^{3(1)}$ est omis.
     $A^3 \leftarrow A^3 \setminus \{s^{3(1)}\}$.
     Etat $s^{3(2)}$: $(u_1,u_2) = (38,33)$ n'est pas dominée, $u_\phi = 21 < \phi_{inf}$, l'état $s^{3(2)}$ est maintenu dans A^3.

Toutes les étapes de résolution sont résumées dans le tableau (Tab. 5.1).

Etape	A^k	M^k	Etats éliminés	x_{opt}	ϕ_{opt}
1	$\{(0,0,0,0),(15,3,8,2)\}$	$\{(15,3,8,2)\}$	$\{(15,3,8,2)\}$	$(0\,1\,1\,0\,1\,0)'$	18
2	$\{(0,0,0,0),(16,10,3,5)\}$	$\{(16,10,3,5)\}$	$\{\}$	$(0\,1\,1\,0\,1\,0)'$	18
3	$\{(0,0,0,0),(16,10,3,5),$ $(8,12,9,9),(24,22,12,14)\}$	$\{(24,22,12,14)\}$	$\{(0,0,0,0),(8,12,9,9)\}$	$(0\,1\,1\,1\,0\,0)'$	20
4	$\{(16,10,3,5),(28,16,8,11),$ $(24,22,12,14),(36,28,17,20)\}$	$\{(36,28,17,20)\}$	$\{(16,10,3,5),(24,22,12,14)\}$	$(0\,1\,1\,1\,0\,0)'$	20
5	$\{(28,16,8,11),(34,27,14,15),$ $(36,28,17,20)\}$	$\{(36,28,17,20)\}$	$\{(28,16,8,11),(34,27,14,15)\}$	$(0\,1\,1\,1\,0\,0)'$	20
6	$\{(36,28,17,20)\}$	$\{(36,28,17,20)\}$	$\{\}$	$(0\,1\,1\,1\,0\,0)'$	20

Tab. 5.1 : Les résultats obtenus à chaque étape

L'algorithme s'achève avec :

$x_{opt} = (0\,1\,1\,1\,0\,0)'$, $\phi_{opt} = 20$, $E_f = \{(1\,1\,0\,0\,0\,1)', (0\,1\,1\,0\,1\,0)', (0\,1\,1\,1\,0\,0)'\}$,

$ND_f = \{(45,20),(30,33),(36,28)\}$.

L'ensemble des points non dominés de ce problème, calculé à l'aide de la méthode de Bazgan et al. [10], est donné par $ND = \{(45,20),(42,23),(37,24),(36,28),(30,33)\}$.

Pour obtenir l'ensemble des solutions efficaces correspondantes, nous avons inclus la variable x dans le processus de résolution de Bazgan et *al.* afin de récupérer les solutions dans l'espace de décision. Nous avons ainsi obtenu :

$E = \{(1\,1\,0\,0\,0\,1)', (0\,1\,0\,1\,0\,1)', (1\,1\,0\,0\,1\,0)', (0\,1\,0\,0\,1\,1)', (0\,1\,1\,0\,1\,0)'\}$.

On remarque que la solution $(0\,1\,1\,1\,0\,0)'$, bien qu'optimale pour le problème principal, ne figure pas dans l'ensemble E, car elle est une solution alternative à la solution $(0\,1\,0\,0\,1\,1)'$, qui correspond au même point non dominé $(36,28)$. Cette observation souligne l'inconvénient de la méthode naïve, qui consiste à calculer d'abord les solutions efficaces, puis à sélectionner la meilleure selon le critère principal. Toutefois, le calcul de toutes les solutions efficaces (y compris les solutions alternatives) est nécessaire, mais s'avère coûteux en termes de temps

d'exécution et d'espace mémoire utilisé.

Dans cet exemple, notre méthode a atteint la solution optimale en examinant uniquement 3 solutions efficaces sur un ensemble d'au moins 6 solutions efficaces, sans avoir besoin de parcourir l'ensemble complet des solutions efficaces.

5.6 Etude expérimentale et résultats

Dans cette section, nous décrivons les expériences mises en œuvre pour évaluer les performances de l'algorithme décrit ci-dessus. Cependant, pour ce faire, nous comparons cet algorithme à celui proposé par Jorge [75]. Ce dernier a été développé pour optimiser une fonction linéaire sur l'ensemble efficace d'un programme en nombres entiers multi-objectif qui peut être adapté au problème BOKP. Dans cette étude, nous nous intéressons principalement au pourcentage de solutions non dominées calculées avant d'atteindre une solution optimale par rapport au nombre total de solutions non dominées d'une instance et au temps de calcul de chaque algorithme.

5.6.1 Résultats expérimentaux

Les expérimentations ont été réalisées sur un Intel(R) Core I5, 2,30 GHz avec 4Go de RAM. Notre algorithme ainsi que celui de Jorge ont été implémentés sous MATLAB R2017a, Windows 10 (64 bits). Nous avons utilisé Cplex 12.9 (version académique) pour résoudre des problèmes de programmation en nombres entiers. Un ensemble d'instances nommées 1B avec des tailles importantes des ensembles de solutions non dominées est considéré. Ces instance sont disponibles dans la bibliothèque MOCOlib http ://xgandibleux.free.fr/MOCOlib/MOKP.html.

L'ensemble 1B contient 40 instances réparties en quatre classes $1B/A$, $1B/B$, $1B/C$ et $1B/D$. Chaque classe contient entre 50 et 500 objets.

- ***Classe 1B/A*** Les vecteurs profits et vecteurs poids sont générés uniformément dans $[1, 100]$.

- ***Classe 1B/B*** Créée à partir de 1B/A en remplaçant le deuxième vecteur de profits par le premier dans l'ordre inverse ($c_i^2 = c_{n-i+1}^1$).

- ***Classe 1B/C*** Le vecteur des profits est généré uniformément dans $[1, 100]$, par plateaux de longueurs générées aléatoirement dans $[1, 0.1 \times n]$. Les poids sont générés

indépendamment suivant une distribution uniforme dans $[1, 100]$, il s'agit d'une instance avec des profits répétés.

- ***Classe 1B/D*** Créée à partir de la classe précédente en remplaçant le deuxième vecteur des profits par le premier dans l'ordre inverse.
- Pour toutes les instances, la capacité du sac est définie par $\omega = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} w_i \right\rfloor$, où n représente le nombre d'objets.

Le critère principal ϕ est généré aléatoirement selon une distribution uniforme dans $[1, 100]$ pour chaque instance.

Les résultats de l'étude expérimentale sont résumés dans les tables (Tab. 5.2) et (Tab. 5.3) où les notations suivantes sont adoptées :

- $|ND|$: nombre de solutions non dominées de chaque instance (disponible dans la bibliothèque MOCOLib).
- $|ND_f|$: nombre de solutions non dominées parcourues.
- $Temps$: temps de calcul.
- $\%ND_f$: pourcentage des solutions non dominées parcourues.
  $\%ND_f = \frac{|ND_f|}{|ND|} \times 100$.

Les résultats du tableau (Tab. 5.2) montrent que notre algorithme ne génère qu'entre 0.8% et 14.71% de solutions non dominées avant d'atteindre une solution efficace qui optimise le critère principal. L'algorithme de Jorge génère entre 23.19% et 66.68%. Pour toutes les classes d'instances, notre algorithme calcule moins de solutions non dominées, comme le montrent les figures (Fig. 5.5), (Fig. 5.6), (Fig. 5.7) et (Fig. 5.8).
Concernant le temps de calcul, nous pouvons observer que, sur les 40 instances testées, l'algorithme de Jorge est plus rapide pour seulement 8 d'entre elles. Cependant, notre approche est meilleure sur 32 instances. Nous pouvons également constater d'après les figures (Fig. 5.1), (Fig. 5.2), (Fig. 5.3) et (Fig. 5.4), que notre algorithme est toujours plus performant que celui de Jorge pour les instances ayant des tailles importantes des ensembles non dominés, comme en témoignent les instances ayant jusqu'à 1000 solutions non dominées.

Classe	Instance	$\|ND\|$	Algorithme de Jorge			algorithme proposé		
			$\|ND_f\|$	$Temps(s)$	$\%ND_f(\%)$	$\|ND_f\|$	$Temps(s)$	$\%ND_f(\%)$
A	$2kp50$	34	15	3.21	44.12	5	0.69	14.71
	$2kp100$	172	52	27.12	30.23	10	19.20	5.82
	$2kp150$	244	116	128.40	47.54	7	91.33	2.87
	$2kp200$	439	165	401.82	37.59	10	229.01	2.28
	$2kp250$	629	224	571.94	35.61	12	687.11	1.91
	$2kp300$	713	247	999.87	34.64	13	998.52	1.83
	$2kp350$	871	332	3100.78	38.12	11	2294.24	1.26
	$2kp400$	1000	421	4555.64	42.10	8	4280.01	0.80
	$2kp450$	1450	596	5984.29	41.10	14	5907.08	0.97
	$2kp500$	1451	630	9990.68	43.42	13	9368.45	0.90
B	$2kp50$	52	17	2.29	38.60	3	0.08	5.77
	$2kp100$	174	50	29.87	28.74	10	18.04	5.75
	$2kp150$	348	129	180.00	37.07	11	105.88	3.16
	$2kp200$	398	164	599.22	41.21	9	391.80	2.27
	$2kp250$	629	271	1111.58	43.08	14	761.01	2.23
	$2kp300$	617	233	1205.30	37.76	11	1047.98	1.79
	$2kp350$	955	414	1999.54	43.35	10	2398.01	1.05
	$2kp400$	1109	506	5094.68	45.63	12	4918.33	1.09
	$2kp450$	1626	759	11029.83	46.68	15	8446.18	0.93
	$2kp500$	1669	800	13036.48	47.93	20	9987.39	1.20
C	$2kp50$	66	20	2.98	30.30	3	0.67	4.55
	$2kp100$	64	23	17.55	35.94	8	11.81	12.50
	$2kp150$	166	67	91.81	40.36	7	60.18	4.22
	$2kp200$	328	134	403.47	40.85	11	248.68	3.36
	$2kp250$	528	247	800.39	46.78	16	542.98	3.03
	$2kp300$	400	246	960.11	61.50	12	733.39	3.00
	$2kp350$	845	375	2924.58	44.38	10	1299.30	1.19
	$2kp400$	946	436	3900.51	46.02	12	3125.97	1.27
	$2kp450$	655	276	3694.58	42.14	12	5210.00	1.83
	$2kp500$	522	349	6986.35	66.68	10	8889.16	1.92
D	$2kp50$	69	16	2.50	23.19	6	0.99	8.70
	$2kp100$	76	26	11.09	34.21	4	9.07	5.26
	$2kp150$	132	41	17.00	31.06	5	16.98	3.79
	$2kp200$	361	152	370.91	42.11	14	244.80	3.88
	$2kp250$	424	187	673.35	44.10	13	671.08	3.07
	$2kp300$	214	114	501.37	53.27	9	793.55	4.21
	$2kp350$	472	223	930.22	47.25	18	1005.28	3.82
	$2kp400$	1670	661	9205.70	39.58	21	8947.24	1.26
	$2kp450$	398	215	3651.25	54.02	7	4877.36	1.76
	$2kp500$	654	330	7000.27	50.46	9	9262.01	1.38

Tab. 5.2 : Performance de notre algorithme et l'algorithme de Jorge sur les instances 1B.

n	50	100	150	200	250	300	350	400	450	500
$Temps\ moyen(s)$	0.60	14.53	68.59	278.57	665.54	893.36	1749.20	5317.88	6110.15	9376.75

Tab. 5.3 : Temps d'exécution de notre algorithme sur les instance 1B

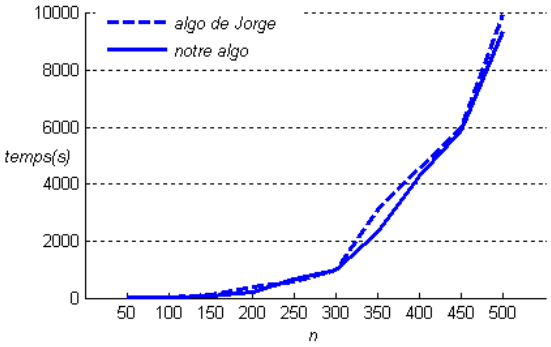


Fig. 5.1 : Temps d'exécution des deux algorithmes sur les instances (Classe A)

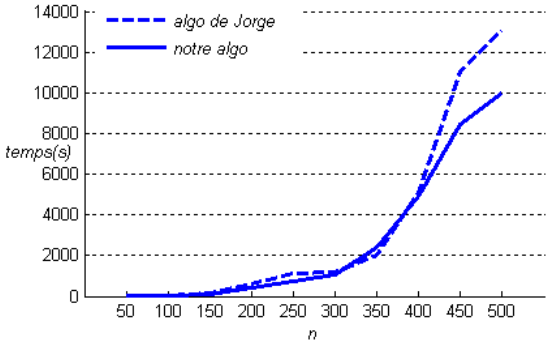


Fig. 5.2 : Temps d'exécution des deux algorithmes sur les instances (Classe B)

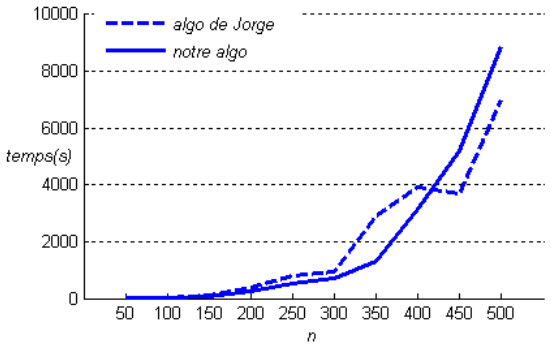


Fig. 5.3 : Temps d'exécution des deux algorithmes sur les instances (Classe C)

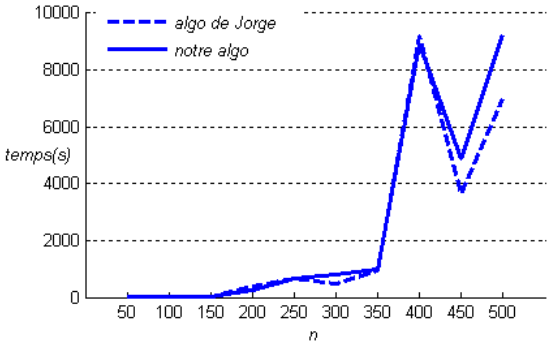


Fig. 5.4 : Temps d'exécution des deux algorithmes sur les instances (Classe D)

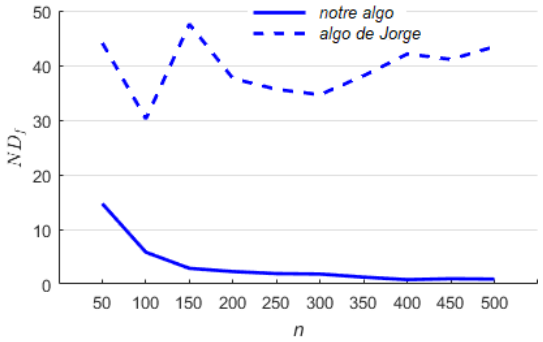


Fig. 5.5 : Pourcentage des solutions non dominées calculées (Classe A)

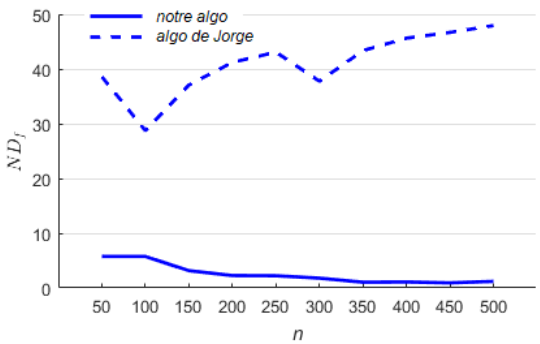


Fig. 5.6 : Pourcentage des solutions non dominées calculées (Classe B)

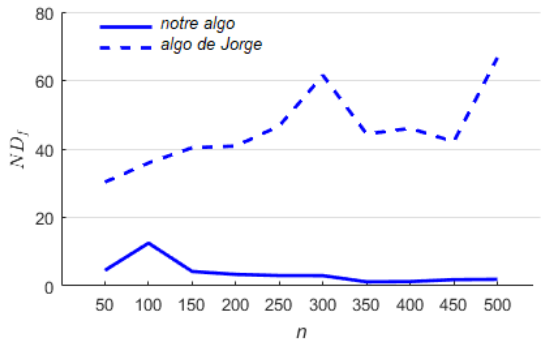


Fig. 5.7 : Pourcentage des solutions non dominées calculées (Classe C)

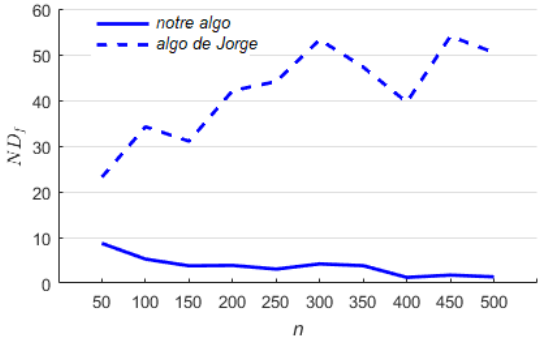


Fig. 5.8 : Pourcentage des solutions non dominées calculées (Classe D)

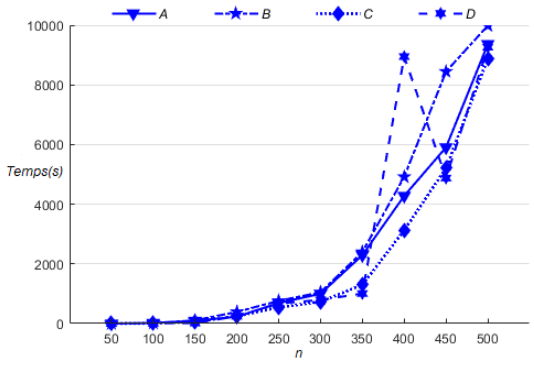


Fig. 5.9 : Temps d'exécution moyen de notre algorithme sur les instance 1B

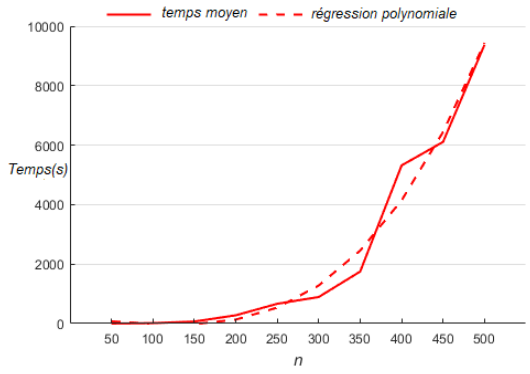


Fig. 5.10 : Estimation de la complexité de notre algorithme

Nous examinons dans la figure (Fig. 5.9), le temps de calcul de notre méthode, il croît avec la dimension de l'instance pour les quatre classes. De plus, lorsque le nombre de solutions efficaces devient beaucoup plus grand, le temps d'exécution augmente considérablement

(instance 1B/D 2kp400).

5.6.2 Complexité de l'algorithme

Le temps d'exécution de la méthode proposée en fonction de la taille des instances étudiées peut être estimé comme une fonction polynomiale $\mathcal{C}(n) = \mathcal{P}(n)$. En effectuant une régression polynomiale sous MATLAB entre la taille du problème ($n = 50, 100, \ldots, 500$) et le temps de calcul moyen de chaque taille sur les quatre classes (voir tableau Tab. 5.3), nous obtenons $\mathcal{P}(n) = 0.0001n^3 - 0.0209n^2 - 0.6050n + 143.3234$. Le temps moyen de notre méthode et la fonction $\mathcal{C}(n)$ sont représentés dans la figure (Fig. 5.10). Comme ils semblent assez similaires, nous pouvons conclure que la complexité de notre méthode sur les instances considérées est polynomiale.

5.7 Conclusion

Dans ce chapitre, nous avons développé un nouvel algorithme exact spécialement conçu pour optimiser une fonction linéaire sur l'ensemble efficace d'un problème BOKP. Il est basé sur la programmation dynamique, l'utilisation de différentes relations pour comparer les états permet d'éliminer au préalable des états ne pouvant pas être efficaces ou qui ne peuvent pas produire des solutions efficaces améliorant le critère principal. Une étude expérimentale a été menée sur plusieurs instances de grandes dimensions, avec un cardinal considérable de l'ensemble non dominé. La lecture des résultats de notre expérimentation montre que la méthode proposée surpasse en termes de temps cpu 80 % l'algorithme de Jorge sur les instances étudiées.

Conclusion générale et perspectives

Dans ce travail, nous nous somme intéressés à l'optimisation sur l'ensemble des solutions efficaces des problèmes MOCO qui constitue le sujet central de notre étude. C'est ainsi que nous nous sommes focalisés sur la mise en oeuvre d'une méthode exacte pour résoudre le problème d'optimisation sur l'ensemble des solutions efficaces d'un problème de sac à dos bi-objectif BOKP.

Nous avons passé en revue dans les premiers chapitres, les concepts fondamentaux de la programmation linéaire multi-objectif, particulièrement en nombre entiers MOILP et combinatoire MOCO. Nous avons exposé quelques méthodes de résolution exactes conçues aux problèmes MOILP et aux problèmes MOCO. Nous avons également introduit le problème d'optimisation sur l'ensemble des solutions efficaces d'un problème MOILP, en exposant ensuite quelques méthodes de résolution.

Notre contribution a été exposée au dernier chapitre, elle a fait l'objet d'une communication dans une conférence internationale (AFROS'18) et d'une publication dans la revue YUJOR en 2022. Cette méthode repose sur la programmation dynamique, l'application de différentes relations de dominance contribue à une énumération implicite des solutions avant d'obtenir la solution optimale du problème. A fin d'évaluer la performance de notre méthode, une étude expérimentale comparative a été réalisée sur plusieurs instances de grandes dimensions, et de cardinalités considérables des ensembles non dominés.

Les outils mathématiques utilisés dans le développement de cette méthode, ouvrent des perspectives de recherche nombreuses et intéressantes, parmi lesquelles nous envisageons les suivantes :

1. Effectuer une mise en ordre d'objets avant de résoudre le problème BOKP et inclure d'autres relations de comparaison pour améliorer le temps de calcul.
2. Implémentation de la méthode en utilisant le parallélisme pour accélérer le processus de résolution.
3. La projection de la méthode sur les problèmes de sac à dos ayant plus de deux objectifs et d'autres problèmes de nature MOCO.
4. Réaliser un survey sur les méthodes d'optimisation sur l'ensemble efficace, établir une étude comparative et examiner les avantages et inconvénients de chaque méthode.

Bibliographie

[1] M. Abbas and D. Chaabane. An algorithm for solving multiple objective integer linear programming problem. *RAIRO- Operations Research*, 36 :351–364, 2002.

[2] M. Abbas, M.E-A. Chergui, and M. Ait Mehdi. Efficient cuts for generating the non-dominated vectors for multiple objective integer linear programming. *International Journal of Mathematics in Operational Research*, 4(3) :302–316, 2012.

[3] M. Al-Rabeeah, A. Al-Hasani, S. Kumar, and A. Eberhard. Enhancement of the improved recursive method for multi-objective integer programming problem. *Journal of Physics : Conference Series*, 1490 :012061, 2020.

[4] A. Alfieri, C. Castiglione, and E. Pastore. A multi-objective tabu search algorithm for product portfolio selection : A case study in the automotive industry. *Computers & Industrial Engineering*, 142(106382), 2020.

[5] K.A. Andersen, K. Jörnsten, and M.Lind. On bicriterion minimal spanning trees : An approximation. *Computers & Operations Research*, 23(12) :1171–1182, 1996.

[6] Y.P. Aneja and K.P. K. Nair. Bicriteria transportation problem. *Management Science*, 25 :73–78, 1979.

[7] C. Bazgan, H. Hugot, and D. Vanderpooten. A practical efficient fptas for the 0-1 multi-objective knapsack problem. In *Algorithms–ESA 2007 : 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007. Proceedings 15*, pages 717–728. Springer, 2007.

[8] C. Bazgan, H. Hugot, and D. Vanderpooten. Implementing an efficient fptas for the 0- 1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1) :47–56, 2009.

[9] C. Bazgan, H. Hugot, and D. Vanderpooten. Implementing an efficient fptas for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1) :47–56, 2009.

[10] C. Bazgan, H. Hugot, and D. Vanderpooten. Solving efficiently the {0-1} multiobjective knapsack problem. *Computers & Operations Research*, 36(1) :260–279, 2009.

[11] T. Bektas. Disjunctive programming for multiobjective discrete optimisation. *INFORMS Journal on Computing*, 30(4) :625–633, 2018.

[12] H.P. Bensen. Optimization over the efficient set. *Journal of Mathematical Analysis and Applications*, 98 :562–580, 1984.

[13] H.P. Bensen. A finite nonadjacent extreme point search algorithm over the efficient set. *Journal of Optimization theory and applications*, 73 :47–64, 1992.

[14] H.P. Bensen and S. Sayin. Optimizing over the efficient set : Four special cases. *Journal of Optimization theory and applications*, 80 :3–18, 1994.

[15] H.P. Benson. Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Appl*, 26 :569–580, 1978.

[16] D. Bergman, M. Bodur, C. Cardonha, and A.A. Cire. Network models for multiobjective discrete optimization. arXiv e-prints. arXiv : 1802.08637[math.OC], 2018.

[17] D. Bergman and A.A. Cire. Multiobjective optimization by decision diagrams. In M. Rueher, editor, *Principles and practice of constraint programming*, pages 86–95. Springer International Publishing, 2016.

[18] G.R. Bitran and J.M. Rivera. A combined approach to solve binary multicriteria problems. *Naval Research Logistics Quarterly*, 29 :181–201, 1982.

[19] N. Boland, H. Charkhgard, and M. Savelsbergh. A simple and efficient algorithm for solving three objective integer programs. Optimization Online, 2014.

[20] N. Boland, H. Charkhgard, and M. Savelsbergh. A criterion space search algorithm for biobjective integer programming : The balanced box method. *INFORMS Journal on Computing*, 27(4) :735–754, 2015.

[21] N. Boland, H. Charkhgard, and M. Savelsbergh. A criterion space search algorithm for biobjective mixed integer programming : The rectangle splitting method. Optimization Online, 2015.

[22] N. Boland, H. Charkhgard, and M. Savelsbergh. A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European Journal of Operational Research*, 260 :904–919, 2017.

[23] N. Boland, H. Charkhgard, and M. Savelsbergh. The quadrant shrinking method : A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research*, 260(3) :873–885, 2017.

[24] N. Boland, H.Charkhgard, and M. Savelsbergh. On the existence of ideal solutions in multi-objective 0-1 integer programs. 2016.

[25] N. Boland, H.Charkhgard, and M. Savelsbergh. Preprocessing and cut generation techniques for multi-objective binary programming. *European Journal of Operational Research*, 274 :858–875, 2019.

[26] J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. An exact ϵ-constraint method for bi-objective combinatorial optimization problems : Application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194(1) :39–50, 2009.

[27] M.E. Captivo, J.C.N. Clìmaco, J.R. Figueira, E.Q.V. Martins, , and J.L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, 30(12) :1865–1886, 2003.

[28] D. Chaabane. A method for optimizing over the integer efficient set in the criteria space. In *Second International Engineering Conference Proceeding*, pages 19–21, Ryadh, December 2004.

[29] D. Chaabane. *Contribution á l'Optimisation Multicritére en Variables Discrètes*. Dissertation soumise à la faculté polytechnque de mons en vue l'obtention du titre de docteur en sciences appliquées, 2007.

[30] D. Chaabane and M. Abbas. Optimizing a linear function over an integer efficient set. *European Journal of Operational Research*, 174(2) :1140–1161, 2006.

[31] D. Chaabane, B. Brahmi, and Z. Ramdani. The augmented weighted tchebychev norm for optimizing a linear function over an integer efficient set of a multicriteria linear program. *International Transactions in Operational Research*, 19 :531–545, 2012.

[32] D. Chaabane and N. Lachemi. Optimizing over the efficient set of the binary bi-objective knapsack problem. *Yugoslav Journal of Operations Research*, 33(1) :91–110, 2022.

[33] D. Chaabane and M. Pirlot. A method for optimizing over the integer efficient set. *Journal of Industrial and Management Optimization*, 6(4) :811–823, 2010.

[34] L.G. Chalmet, L. Lemonidis, and D.J. Elzinga. An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research*, 25(2) :292–300, 1986.

[35] HW. Corley. Efficient spanning trees. *Journal of optimization theory and applications*, 45 :481–485, 1985.

[36] P. Correia, L. Paquete, and J. Figueira. Compressed data structures for bi-objective {0,1}-knapsack problems. *Computers and Operations Research*, 89 :82–93, 2018.

[37] R. Dai and H.A. Charkhgard. two-stage approach for bi-objective integer linear programming. *Operations Research Letters*, 46(1) :81–87, 2018.

[38] G. Dantzig. Discrete variable extremum problems. *Operations Research*, 5 :226–227, 1957.

[39] J.P. Dauer. Optimization over the efficient set using an active constraint approach. *ZOR-Methods and Models of Operations Research*, 35 :185–195, 1991.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyariva. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197, 2002.

[41] R. Deckro and E. Winkofsky. Solving zero-one multiple objective programs through implicit enumeration. *European Journal of Operational Research*, 12 :362–374, 1983.

[42] M. Dell'Amico, F. Maffioli, and S. Martello. *Annotated bibliographies in combinatorial optimization*. Wiley, 1997.

[43] C. Delort and O. Spanjaard. Using bound sets in multiobjective optimization : Application to the biobjective binary knapsack problem. In *Experimental Algorithms : 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*, pages 253–265. Springer, 2010.

[44] C. Dhaenens, J. Lemesre, and E.G. Talbi. K-ppm : A new exact method to solve multi-objective combinatorial optimization problems. *European Journal of Operational Research*, 200(1) :45–53, 2010.

[45] K. Dächert and K. Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, 4(643-676) :2015, 61.

[46] K. Dächert, K. Klamroth, R. Lacour, and D. Vanderpooten. Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 260(3) :841–855, 2017.

[47] J.G. Ecker and I.A. Kouada. Finding efficient points for multi-objective linear programs. *Mathematical Programming*, 8 :375–377, 1975.

[48] J.G. Ecker and J.H. Song. Optimizing a linear function over an efficient set. *Journal of Optimization theory and applications*, 83 :541–563, 1994.

[49] M. Ehrgott. *Multiple Criteria Optimization - Classification and Methodology*. Shaker Verlag, Aachen, 1997.

[50] M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[51] M. Ehrgott. A discussion of scalarization techniques for multiple objective integer programming. *Annals of Operations Research*, 147(1) :343–360, 2006.

[52] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22 :425–460, 2000.

[53] M. Ehrgott and X. Gandibleux. Approximative solution methods for multiobjective combinatorial optimization. *Top*, 12(1) :1–63, 2004.

[54] M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34 :2674–2694, 2007.

[55] M. Ehrgott, D. Tenfelde-Podehl, and T. Stephan. A level set method for multiobjective combinatorial optimization : Application to the quadratic assignment problem. *Pacific Journal on Optimization*, 2 :521–544, 2006.

[56] T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multi-objective knapsack problems. *Management Sciences*, 48(12) :1603–1612, 2002.

[57] J.R. Figueira, L. Paquete, M. Simões, and D. Vanderpooten. Algorithmic improvements on dynamic programming for the bi-objective {0,1} knapsack problem. *Computational Optimization and Applications*, 56 :97–111, 2013.

[58] J.R. Figueira, G. Tavares, and M.M. Wiecek. Labeling algorithms for multiple objective integer knapsack problems. *Computers & operations research*, 37(4) :700–711, 2010.

[59] S.L. Gadegaard, L.R. Nielsen, and M. Ehrgott. Bi-objective branch and-cut algorithms based on lp-relaxation and bound sets. *INFORMS Journal on Computing*, 31 :790–804, 2019.

[60] X. Gandibleux and A. Fréville. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem : The two objectives case. *Journal of Heuristics*, 6 :361–383, 2000.

[61] X. Gandibleux, J. Jorge, S. Martínez, and N. Sauer. Premier retour d'experience sur le flow-shop biobjectif et hybride a deux etages avec une contrainte de blocage particuliere. In *Avril. 6ème Conférence Francophone de Modélisation et Simulation MOSIM*, volume 6, 2006.

[62] X. Gandibleux, N. Mezdaoui, and A.Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In Francisco Ruiz Rafael Caballero and Ralph Steuer, editors, *Advances in Multiple Objective and Goal Programming*, page 291–300. Springer Berlin Heidelberg, 1996.

[63] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22 :618–630, 1968.

[64] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Boston, 2003.

[65] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5) :533–549, 1986.

[66] D.E. Goldberg. Cenetic algorithms in search. *Optimization, Machine Learning*, 1989.

[67] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms : Motivation, analysis, and first results. *Complex systems*, 3(5) :493–530, 1989.

[68] G. Gutin and A.P. Punnen. *The travelling salesman problem and its variation*. Kluwer Academic Publishers, 2002.

[69] Y.Y. Haimes, L.S. Lasdon, and D.A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1 :296–297, 1971.

[70] P. Halffmann, L.E. Schäfer, K. Dächert, K. Klamroth, and S. Ruzika. Exact algorithms for multiobjective linear optimization problems with integer variables : A state of the art survey. *Journal of Multi-Criteria Decision Analysis*, 2022. *doi.org*/10.1002/*mcda*.1780.

[71] I. Hamidi and D. Chaabane. A two-phase method based on the branch and bound algorithm for solving the bi-objective set covering problem. *International Journal of Multicriteria Decision Making*, 9(4) :281–321, 2023.

[72] A. Herzel, S. Ruzika, and C. Thielen. Approximation methods for multiobjective optimization problems : A survey. *INFORMS Journal on Computing*, 33(4) :1284–1299, 2021.

[73] J.H. Holland. *Adaptation in natural and artificial systems*. Phd thesis, University of Michigan Press, 1975.

[74] H. Isermann and E. Steuer. Computational experience concerning payoff tables and minimum values over the efficient set. *European Journal of Operational Research*, 33 :91–97, 1987.

[75] J. Jorge. An algorithm for optimizing a linear function over an integer efficient set. *European Journal of Operational Research*, 195(1) :98–103, 2009.

[76] J. Jorge. *Nouvelles propositions pour la résolution exacte du sac à dos multiobjectif unidimensionnel en variables binaires*. Thèse de doctorat, Université de Nantes, 2010.

[77] N. Jozefowiez, G. Laporte, and F. Semet. A generic branch-and-cut algorithm for multiobjective optimization problems. *INFORMS Journal on Computing*, 24 :554–564, 2012.

[78] V. Joseph Bowman Jr. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple Criteria Decision Making : Proceedings of a Conference Jouy-en-Josas, France*, pages 76–86. Springer, 1976.

[79] H. Kellerer and D.Pisinger. *Knapsack Problems*. Springer Verlag, Berlin, 2004.

[80] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[81] G. Kirlik and S. Sayın. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3) :479–488, 2014.

[82] G. Kiziltan and E. Yucaoglu. An algorithm for multiobjective zero one linear programming. *Management Science*, 29(12) :1444–1453, 1983.

[83] K. Klamroth and M.M. Wiecek. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics (NRL)*, 47(1) :57–76, 2000.

[84] D. Klein and E. Hannan. An algorithm for multiple objective integer linear programming problem. *European Journal of Operational Research*, 9 :378–385, 1982.

[85] R. Klötzler. Multiobjective dynamic programming. *Mathematische Operationsforschung und Statistik. Series Optimization*, 9(3) :423–426, 1978.

[86] M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3) :932–942, 2006.

[87] M. Leitner, I. Ljubic, M. Sinnl, and A. Werner. Ilp heuristics and a new exact method for bi-objective 0/1 ilps : Application to fttx-network design. *Computers & Operations Research*, 72 :128–146, 2016.

[88] J. Lemesre, C. Dhaenens, and E. Talbi. Parallel partitioning method (ppm) : A new exact method to solve bi-objective problems. *Computers & Operations Research*, 34(8) :2450–2462, 2007.

[89] B. Lokman. Optimizing a linear function over the nondominated set of multiobjective integer programs. *International Transactions in Operational Research*, 28(4) :2248–2267, 2021.

[90] B. Lokman, G. Ceyhan, and M. Köksalan. A web-based solution platform for multi-objective integer programs. `http://www.onlinemoco.com/MOIP/jsp/guide/MCDM2017-BanuLokman-LV.pdf`, 2017.

[91] B. Lokman and M. Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2) :347–365, 2013.

[92] T. Lust. New metaheuristics for solving moco problems : application to the knapsack problem, the traveling salesman problem and imrt optimization. *Faculté Polytechnique de Mons*, 2010.

[93] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations*. Wiley, New York, 1990.

[94] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16 :236–245, 1984.

[95] E.Q.V. Martins and J.L.E. Dos Santos. The labeling algorithm for the multiobjective shortest path problem. Technical Report 99/005 CISUC, Departamento de Matemática, Universidade de Coimbra, November 1999.

[96] G. Mavrotas. Generation of efficient solutions in multiobjective mathematical programming problems using gams. Effective implementation of the ϵ-constraint method. Technical report, National Technical University of Athens, 2008.

[97] G. Mavrotas. Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2) :455–465, 2009.

[98] G. Mavrotas and K. Florios. Finding the exact pareto set in multiple objective integer programming problems using an improved version of the augmented epsilon constraint method. 2012.

[99] G. Mavrotas and K. Florios. Augmecon 2 : A novel version of the ϵ-constraint method for finding the exact pareto set in multi-objective integer programming problems. Technical report, National Technical University of Athens, 2013.

[100] G. Mavrotas and K. Florios. An improved version of the augmented $\epsilon-$constraint method (augmecon2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18) :9652–9669, 2013.

[101] N.C. Nguyen. An algorithm for optimizing a linear function over the integer efficient set. *Konrad-Zuse-Zentrum fur Informations technik Berlin*, Nov 1992.

[102] A. Pal and H. Charkhgard. MSEA. jl : A multi-stage exact algorithm for biobjective pure integer linear programming in julia. *Optimization Online*, 2018.

[103] S.N. Parragh and F. Tricoire. Branch-and-bound for bi-objective integer programming. *INFORMS Journal on Computing*, 31(4) :805–822, 2019.

[104] W. Pettersson and M. Özlen. A parallel approach to bi-objective integer programming. *ANZIAM Journal*, 58 :69–81, 2016.

[105] W. Pettersson and M. Özlen. Multiobjective integer programming : Synergistic parallel approaches. *INFORMS Journal on Computing*, 32(2) :461–472, 2019.

[106] J. Philip. Algorithms for the vector maximization problem. *Mathematical Programming*, 2 :207–229, 1972.

[107] Prerna and V. Sharma. Optimization of a linear function over an integer efficient set. *Journal of Industrial and Management Optimization*, 19(12) :8633–8656, 2023.

[108] A. Przybylski and X. Gandibleux. Multi-objective branch and bound. *European Journal of Operational Research*, 260(3) :865–872, 2017.

[109] A. Przybylski, X. Gandibleux, and M. Ehrgott. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2) :509–533, 2008.

[110] A. Przybylski, X. Gandibleux, and M. Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer program. *INFORMS Journal on Computing*, 22 :371–386, 2010.

[111] A. Przybylski, X. Gandibleux, and M. Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3) :149–165, 2010.

[112] M.J. Rosenblatt and Z. Sinuany-Stern. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research*, 37(3) :384–394, 1989.

[113] H.W. Hamacherand G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52(4) :209–230, 1994.

[114] H.M. Safer and J.B. Orlin. Fast approximation schemes for multi-criteria combinatorial optimization. 1995.

[115] M. De Santis, G. Grani, and L. Palagi. Branching with hyperplanes in the criterion space : The frontier partitioner algorithm for biobjective integer programming. *European Journal of Operational Research*, 283(1) :57–69, 2020.

[116] S. Sayin. Optimizing over the efficient set using a top-down search of faces. *Operations Research*, 48 :65–72, 2000.

[117] P. Serafinin. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent Advances and Historical Development of Vector Optimization : Proceedings of an International Conference on Vector Optimization*, pages 222–232. The Technical University of Darmstadt, FRG, Springer Berlin Heidelberg, August 4–7 1987.

[118] I.V. Sergienko and V.A. Perepelitsa. Finding the set of alternatives in discrete multicriterion problems. *Cybernetics*, 23(5) :673–683, 1987.

[119] A.K. Simopoulos. Multicriteria integer zero-one programming : A tree-search type algorithm. Naval Postgraduate School Monterey Californian, 1977.

[120] C.C. Skiscim and B.L Golden. Optimization by simulated annealing : A preliminary computational study for the tsp. Technical report, Institute of Electrical and Electronics Engineers (IEEE), 1983.

[121] F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework : Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3) :472–484, 2008.

[122] F. Sourd, O. Spanjaard, and P. Perny. Multi-objective branch and bound. application to the bi-objective spanning tree problem. In *7th International Conference in Multi-Objective Programming and Goal Programming*, Tours, France, 2006.

[123] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3) :221–248, 1994.

[124] W. Stadler. A survey of multicriteria optimization or the vector maximum problem, part i : 1776–1960. *Journal of Optimization Theory and Applications*, 29(1) :1–52, 1979.

[125] J. Sylva and A. Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European journal of operational Research*, 158(1) :46–55, 2004.

[126] J. Sylva and A. Crema. Enumerating the set of non-dominated vectors in multiple objective integer linear programming. *RAIRO–operations Research*, 42(3) :371–387, 2008.

[127] J. Sáez-Aguado and P.C. Trandafir. Variants of the ϵ-constraint method for biobjective integer programming problems : Application to p-median-cover problems. *Mathematical Methods of Operations Research*, 87(2) :251–283, 2018.

[128] S. Tamby and D. Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 33(1) :72–85, 2020.

[129] J. Teghem and P.L. Kunsch. Interactive method for multiobjective integer linear programming. *G. Fandel et al. (Eds.), Large Scale Modelling and Interactive Decision Analysis, Springer Verlag*, (75-87), 1986.

[130] T. Trzaskalik. Multiple criteria discrete dynamic programming. In G. Fandel & T. Gal, editor, *Multiple criteria decision making*, pages 202–211. Springer, 1997.

[131] G. Tsaggouris and C. Zaroliagis. Multiobjective optimization : Improved fptas for shortest paths and non-linear objectives with applications. *Theory of Computing Systems*, 45(1) :162–186, 2009.

[132] E.L. Ulungu. *Optimisation Combinatoire multicritère : Détermination de l'ensemble des solutions efficaces et méthodes interactives*. Thèse de doctorat, Université de Mons-Hainaut, Faculté des Sciences, Mons, Belgique, 1993.

[133] E.L. Ulungu and J. Teghem. The two phases method : An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of computing and decision sciences*, 20(2) :149–165, 1995.

[134] E.L. Ulungu, J. Teghem, PH. Fortemps, and D. Tuyttens. Mosa method : a tool for solving multiobjective combinatorial optimization problems. *Journal of multicriteria decision analysis*, 8(4) :221, 1999.

[135] B. Villarreal and M.H. Karwan. Multicriteria integer programming : A (hybrid) dynamic programming recursive approach. *Mathematical Programming*, 21(1) :204–223, 1981.

[136] B. Villarreal and M.H. Karwan. Multicriteria dynamic programming with an application to the integer case. *Journal of Optimization Theory and Applications*, 38(1) :43–69, 1982.

[137] T. Vincent. *Caractérisation des solutions efficaces et algorithmes d'énumération exacts pour l'optimisation multiobjectif en variables mixtes binaires*. PhD thesis, Nantes, 2013.

[138] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12(2) :139–155, 1998.

[139] D.J. White. A branch and bound method for multi-objective boolean problems. *European Journal of Operational Research*, 15 :126–130, 1984.

[140] W. Zhang and M. Reimann. A simple augmented ϵ-constraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research*, 234(1) :15–24, 2014.

[141] S. Zionts. Integer linear programming with multiple objectives. In B. Korte G. Nemhauser P. Hammer, E. Johnson, editor, *Studies in integer programming*, volume 1 of *Annals of Discrete Mathematics*, page 551–562. Elsevier, 1977.

[142] M. Özlen and M. Azizoglu. Multi-objective integer programming : A general approach for generating all non-dominated solutions. *European Journal of Operational Research*, 199(1) :25–35, 2009.

[143] M. Özlen, B.A. Burton, and C.A.G. MacRae. Multi-objective integer programming : An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 160(2) :470–482, 2014.