THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'Université Paris-Dauphine

Approches génériques pour la résolution de problèmes d'optimisation discrète multiobjectif

École doctorale de Dauphine – ED 543

Spécialité INFORMATIQUE

Soutenue par **Satya TAMBY**
le 7 décembre 2018

Dirigée par **Daniel VANDERPOOTEN**

COMPOSITION DU JURY :

Directeur de thèse

Daniel VANDERPOOTEN
Professeur
Université Paris-Dauphine

Rapporteurs

José Rui FIGUEIRA
Professeur
Technical University of Lisbon

Kathrin KLAMROTH
Professeur
Bergische Universität Wuppertal

Examinateurs

Laetitia JOURDAN
Professeur
Université de Lille
Présidente du jury

Anthony PRZYBYLSKI
Maître de conférences
Université de Nantes

Remerciements

Avec le doctorat, je me doutais bien qu'une aventure débutait. Une aventure scientifique, certes, mais également sociale. Que de sujets à étudier, que de personnes à rencontrer ! Mais j'étais loin d'imaginer à quel point...

En premier lieu, je vous remercie, Daniel, d'avoir accepté d'être mon directeur de thèse. Je vous remercie non seulement de m'avoir octroyé votre confiance pendant ces cinq années, mais également pour le soutien que vous m'avez apporté au quotidien.

Bien évidemment, je remercie le jury devant lequel j'ai défendu mes travaux. Merci à vous Laetita Jourdan et Anthony Przybylski d'avoir accepté d'examiner la thèse. Merci à vous, Kathrin Klamroth et José Figueira d'avoir accepté d'être rapporteurs. Merci à vous tous pour vos remarques, et en particulier à Kathrin pour les nombreuses discussions enrichissantes que nous avons eues.

En cinq ans, j'ai eu l'occasion de voir les thésards se succéder. Je vous remercie tous, pour ces années inoubliables que nous avons partagé. Je n'oublie pas non plus les maîtres de conférences avec qui j'ai lié une amitié un peu tardive, certes, mais tout aussi solide.

Enfin, je remercie mes parents pour m'avoir donné le goût d'apprendre et pour m'en avoir toujours donné les moyens. Sans votre soutien sans failles, jamais je ne serai arrivé jusque là.

In memoriam
Maxence PONT

Table des matières

Introduction

Dans un certain nombre de processus de décision, les alternatives considérées sont évaluées par un ensemble de critères le plus souvent conflictuels. Dès lors, les alternatives possibles peuvent être assimilées à des compromis entre les critères. Puisque tous les compromis ne sont pas nécessairement comparables entre eux, la recherche d'un optimum, problématique habituelle en optimisation classique, n'a plus de sens. L'optimisation multiobjectif s'intéresse donc à déterminer les compromis pertinents, appelés *points non dominés*, tels que toute amélioration d'un critère entraine la dégradation d'un autre, ainsi qu'aux solutions associées : les *solutions efficaces*. Nous nous intéresserons dans ce document aux problèmes d'optimisation discrète où l'ensemble des points non dominés est potentiellement de très grande taille.

D'une manière générale, deux stratégies de résolution de problèmes multiobjectif peuvent être considérées. En présence d'information préférentielle sur les compromis, en s'appuyant par exemple sur l'avis d'un décideur, il est possible de sélectionner un sous ensemble restreint de compromis, voire un seul d'entre eux. Ces préférences peuvent prendre différentes formes : nous pouvons par exemple introduire des poids modélisant l'importance de chaque critère, définir un ordre total entre eux (*optimisation lexicographique*) ou même trouver le compromis minimisant une distance à un *point de référence*. En revanche, en l'absence de telles informations préalables, il n'est pas possible de définir un ordre sur les points non dominés, et il est donc nécessaire de tous les énumérer. Cette problématique, dans laquelle nous nous plaçons, est difficile à résoudre. En effet, en raison du grand nombre de points non dominés, même si le problème d'optimisation classique est simple, sa version multiobjectif est complexe.

Les algorithmes visant à générer les points non dominés se divisent en deux catégories : les algorithmes exacts et les métaheuristiques. Les algorithmes exacts ont pour objectif de générer précisément l'ensemble des points non dominés, ou de l'approximer avec une garantie *a priori* sur le sous-ensemble de points énumérés. À l'inverse, les métaheuristiques, bien qu'en général plus rapides, ne fournissent aucune garantie sur la qualité des points retournés qui peuvent donc être dominés. Ces approches sont généralement des adaptations d'algorithmes évolutionnaires ou de recherche locale permettant de s'intéresser à de très larges instances de problèmes réputés complexes (Coello et al., 2010, Cornu, 2017, Liefooghe et al., 2012, Talbi, 2009). L'intérêt de chacune de ces deux catégories dépend donc des exigences en terme de qualité des solutions ou de rapidité d'exécution. Une synthèse des algorithmes multiobjectifs a été proposée par Ehrgott et Gandibleux (2003).

Deux catégories d'approches exactes générant l'ensemble des points non dominés peuvent être distinguées : celles construisant progressivement des solutions en explorant l'espace des décisions, s'appuyant en général sur des méthodes par évaluation et séparation (Przybylski et Gan-

dibleux, 2017, Sourd et Spanjaard, 2008) ou sur la programmation dynamique (Abo-Sinna et Hussein, 1995), et celles explorant essentiellement l'espace des critères en résolvant des programmes mathématiques. La première catégorie d'approches peut exploiter des propriétés spécifiques liées à la structure des problèmes résolus (détermination de bornes sur les performances des solutions dans le cas des méthodes d'évaluation et séparation, principe de Bellman dans le cadre de la programmation dynamique), voire même être dédiées à un certain type de problèmes comme par exemple au *plus court chemin multiobjectif* (Martins, 1984), l'*allocation multiobjectif* (Stiglmayr et al., 2014) ou encore différentes variantes du *sac à dos multiobjectif* (Bazgan et al., 2009, Figueira et al., 2010, Rong et al., 2013). Ce n'est pas le cas des approches reposant sur la programmation mathématique, si ce n'est que l'ensemble des solutions réalisables doit pouvoir se formaliser sous la forme d'un ensemble de variables discrètes contraintes linéairement (ce qui est le cas de la majeure partie des problèmes d'optimisation combinatoire). Nous nous intéresserons ici à cette seconde catégorie d'approche, que nous qualifierons de générique car étant adaptée à la résolution d'un grand nombre de problèmes.

Ces méthodes résolvent itérativement des programmes mathématiques en nombres entiers, chacun d'entre eux pouvant potentiellement retourner un nouveau point, le défi étant de réduire le nombre et la difficulté de résolution des programmes considérés. Dans le cas biobjectif, l'approche communément utilisée est la méthode ε-contrainte proposée par Chankong et Haimes (1983), bien que différentes variantes ont été étudiées, comme par exemple par Chalmet et al. (1986), Ralphs et al. (2006) ou Boland et al. (2015). Cet algorithme exhibe itérativement les points non dominés en optimisant le premier objectif tout en contraignant de plus en plus le second. Cette approche est très efficace car chaque chaque itération - à l'exception de la dernière - permet de générer un nouveau point. Cependant, il n'est pas trivial de la généraliser pour résoudre des problèmes faisant intervenir plus de 2 objectifs comme le montre la grande variété de publications à ce sujet. En effet, celles-ci sont confrontées à un grand nombre de difficultés, les rendant peu satisfaisantes pour répondre à cette problématique. S'inspirant de la méthode ε-contrainte, certaines méthodes ajoutent successivement des contraintes disjonctives au programme mathématique résolu, retirant ainsi du domaine de réalisabilité l'ensemble des points dominés par ceux qui ont déjà été énumérés, idée proposée initialement par Klein et Hannan (1982), reprise par Sylva et Crema (2004), et dans une moindre mesure par Sylva et Crema (2008) et Boland et al. (2017). De telles approches exhibent à chaque itération un nouveau point non dominé, mais les programmes résolus se complexifient au fur et à mesure des itérations. À l'opposé, certaines approches ne considèrent que des programmes mathématiques de taille constante (il s'agit en général du domaine de réalisabilité initial augmenté d'une contrainte par objectif). Cependant, le nombre d'itérations augmente significativement car certains programmes se révèlent infaisables et, pour certaines approches, le même point non dominé peut être énuméré plusieurs fois (Kirlik et Sayin, 2014, Laumanns et al., 2005, Lokman et Köksalan, 2013, Mavrotas, 2009, Özlen et Azizoğlu, 2009, Zhang et Reimann, 2014). Bien que l'algorithme que nous proposons entre dans cette seconde catégorie, nous montrerons qu'il est possible d'en limiter les aspects négatifs en s'appuyant sur un certain nombre de résultats, permettant ainsi d'aboutir à une méthode générant l'intégralité des points non dominés en ne résolvant qu'un nombre restreint de programmes mathématiques. Nos expérimentations numériques montrent que les performances de notre algorithme sont significativement meilleures que

celles des algorithmes de l'état de l'art.

L'ensemble des points non dominés pouvant être considérable, il peut être délicat de l'exploiter tel quel pour prendre une décision. Pour y remédier, plusieurs problématiques alternatives peuvent être considérées. Il est par exemple possible d'augmenter la région qu'un point domine en introduisant une tolérance : il s'agit de la *relation de ε-dominance*. Nous montrerons que, même avec une tolérance très faible, il est possible d'obtenir une représentation très concise de l'ensemble des points non dominés tout en réduisant très sensiblement l'ordre de grandeur des temps de résolution. De fait, la prise de décision s'en trouve simplifiée car les alternatives similaires ont été omises.

Nous nous intéresserons également au calcul d'une borne de l'ensemble des points non dominés, le *point nadir*, dont la détermination est réputée difficile au delà de deux objectifs. Être capable de l'obtenir rapidement permettrait de pouvoir fournir au décideur une information préalable à la résolution de son problème multiobjectif. À cet effet, nous formaliserons les deux stratégies suivies par les différents algorithmes de la littérature. L'une de ces stratégies est particulièrement adaptée quand le nombre d'objectifs est faible, mais son efficacité diminue grandement au delà du cas triobjectif : nous l'exploiterons donc uniquement dans cette situation. Dans le cas général, nous emploierons la seconde stratégie, permettant de réduire significativement le nombre de points non dominés à énumérer pour déterminer le point nadir. Les comparaisons expérimentales avec les algorithmes récents de la littérature démontreront son efficacité.

Le premier chapitre introduit les notions principales utilisées dans ce document. Le théorème fondamental de l'optimisation multiobjectif, permettant d'exhiber un point non dominé dans une région de l'espace, ainsi que la notion de région de recherche, région à explorer pour déterminer les points non dominés, y seront notamment détaillés. Nous présenterons dans le deuxième chapitre trois approches simples mais représentatives des différents algorithmes générant les points non dominés : celles de Sylva et Crema (2008), Lokman et Köksalan (2013) et de Kirlik et Sayin (2014). En nous appuyant sur ces approches, nous dégagerons différents concepts qui nous permettront de détailler, dans le troisième chapitre, un nouvel algorithme déterminant les points non dominés. Son efficacité sera démontrée en le comparant avec les approches récentes sur des instances de deux problèmes classiques d'optimisation combinatoire que sont les problèmes du *sac à dos* et *d'affectation* multiobjectifs, concluant ainsi la première partie de ce document. Nous présenterons dans le chapitre suivant comment utiliser la notion de *ε-dominance* pour obtenir rapidement une représentation concise des points non dominés. Enfin, nous nous intéresserons, dans le dernier chapitre, à la recherche du *point nadir*. Nous montrerons néanmoins que, dans le cas triobjectif, certaines propriétés simplifient grandement le problème, et nous proposerons deux algorithmes pour identifier ce point : l'un dédié aux problèmes ne comportant que trois critères, l'autre dédié aux cas plus généraux. Nous conclurons ce document en synthétisant nos contributions et en proposant certaines problématiques, détaillées en annexe, pouvant étendre ces travaux.

Chapitre 1.

Notions préliminaires et notations

Résumé

Nous introduisons dans cette section les problèmes d'optimisation multiobjectifs, ainsi que la notion de point non dominé. Le théorème fondamental de l'optimisation multiobjectif est détaillé. L'approche ε-contrainte, spécifique au cas biobjectif, est introduite, suivie du concept général de *région de recherche*. Enfin, nous présentons puis comparons deux façons de caractériser cette region, la caractérisation implicite, simple mais peu efficace, et la caractérisation explicite sur laquelle repose l'intégralité des algorithmes de ce document.

1.1. Concepts préliminaires et notations

1.1.1. Formulation du problème

Considérons le problème d'optimisation multiobjectif consistant à minimiser p fonctions linéaires sur un ensemble discret et non vide de solutions réalisables X.

$$\begin{cases} \min & f(x) = (f_1(x), \dots, f_p(x)) \\ \text{s.c.} & x \in X \end{cases} \tag{1.1}$$

Les élément $x \in X$ sont les *solutions réalisables* et leurs images respectives $y = (f_1(x), \dots, f_p(x))$, sont les *points réalisables*. L'ensemble $\mathbb{R}^p$ est appelé *espace des objectifs* et nous considérons l'ensemble $Y = f(X)$ qui est l'ensemble des images de chaque solution réalisable dans l'espace des objectifs. Le problème (1.1) peut être reformulé dans l'espace des objectifs de la manière suivante :

$$\begin{cases} \min & y = (y_1, \dots, y_p) \\ \text{s.c.} & y \in Y \end{cases} \tag{1.2}$$

Par souci de concision, nous considérerons dans la suite la formulation (1.2).

Étant donné deux points $(y, y') \in (\mathbb{R}^p)^2$, *$y$ domine y' au sens de Pareto* si et seulement si y est au moins aussi bon que y' sur chaque critère, et strictement meilleur sur au moins l'un d'entre eux :

$$y \preceq y' \Leftrightarrow \begin{cases} \forall i \in [\![1, p]\!] & y_i \leq y'_i \\ \exists k \in [\![1, p]\!] & y_k < y'_k \end{cases}$$

Nous utiliserons également dans la suite l'extension de la dominance au sens large (y est au moins aussi bon que y' sur chaque critère) et au sens strict (y est strictement meilleur que y' sur chaque critère).

$$y \leqq y' \Leftrightarrow \forall i \in [\![1, p]\!],\ y_i \leq y'_i$$
$$y \prec y' \Leftrightarrow \forall i \in [\![1, p]\!],\ y_i < y'_i$$

Nous nous intéressons ici à la détermination de l'ensemble Y_{ND} contenant les points non dominés de Y, ainsi qu'à la détermination d'une solution associée pour chacun d'eux.

$$Y_{ND} = \{y \in Y : \nexists y' \in Y, y' \preceq y\}$$

Nous faisons l'hypothèse que l'ensemble Y_{ND} est *fini* et qu'il est *stable extérieurement* (Ehrgott, 2005, Sawaragi et al., 1985), c'est-à-dire qu'il satisfait la propriété suivante.
Définition 1.1. *L'ensemble Y_{ND} est* stable extérieurement *si et seulement si :*

$$\forall y \in Y \setminus Y_{ND},\ \exists y' \in Y_{ND} \textit{ tel que } y' \preceq y$$

Une solution réalisable dont l'image est non dominée est dite *efficace*. Dans la suite, X_E fera référence à l'ensemble des solutions efficaces.

$$X_E = \{x \in X, f(x) \in Y_{ND}\}$$

L'ensemble Y_{FND} fait référence à l'ensemble des points *faiblement non dominés*, c'est à dire :

$$Y_{FND} = \{y \in Y : \nexists y' \in Y, y' \prec y\}$$

Proposition 1.1. $Y_{ND} \subseteq Y_{FND}$

Démonstration. Le résultat découle du fait que $y \prec y' \implies y \preceq y'$. □

Étant donné un point y, nous notons $d(y)$ la région de l'espace dominée par y, c'est à dire :

$$d(y) = \{y' \in \mathbb{R}^p,\ y \preceq y'\}$$

À l'inverse, nous notons $z(y)$ la région de l'espace dominant fortement y :

$$z(y) = \{y' \in \mathbb{R}^p,\ y' \prec y\}$$

Soit y^I le point idéal de Y, défini par $y^I_i = \min_{y \in Y}\{y_i\}$, que l'on peut obtenir en résolvant p problèmes minimisant chacun des objectifs. y^I fournit une borne inférieure sur chaque critère. Le terme *optimum lexicographique* fera référence à tout point non dominé $y \in Y_{ND}$ contribuant à une composante du point idéal, c'est à dire tel que $y_i = y^I_i$ pour un $i \in [\![1, p]\!]$. Enfin, soit M une borne supérieure de chaque critère, pouvant être la plus grande valeur représentable par un ordinateur ou bien être obtenue par p problèmes de maximisation.

Nous noterons par $y_{-k} \in \mathbb{R}^{p-1}$ la projection orthogonale de y dans la direction k. En d'autres termes, la composante k de y a été omise :

$$y_{-k} = (y_1, \ldots, y_{k-1}, y_{k+1}, \ldots, y_p)$$

Nous emploierons également les notations $\prec$, $\preceq$ et $\leqq$ lorsque l'on comparera deux projections orthogonales.

1.1.2. Théorème fondamental de l'optimisation multiobjectif

Un résultat fondamental en optimisation multiobjectif présente une manière d'obtenir un point non dominé dans un sous ensemble de Y borné supérieurement : il suffit d'optimiser une fonction *fortement monotone* sur cet ensemble.

Définition 1.2. (Forte monotonie) *Une fonction* $\Phi : Y \to \mathbb{R}$ *est* fortement monotone *si et seulement si, pour tout couple de points* $(y, y') \in Y^2$*, nous avons :*

$$y \preceq y' \implies \Phi(y) < \Phi(y')$$

Le problème suivant exhibe un point non dominé, dominant une borne supérieure locale u, s'il existe un point réalisable dans $z(u)$.

Théorème 1.1. *(Chankong et Haimes, 1983, page 114) Soit $u \in \mathbb{R}^p$ et $\Phi : Y \to \mathbb{R}$ une fonction* fortement monotone. *Soit le programme $P(u)$ défini de la manière suivante :*

$$P(u) \begin{cases} \min & \Phi(y) \\ s.c. & y \in Y \\ & y \leqq u \end{cases}$$

Si $P(u)$ est réalisable, le point retourné est non dominé.

Démonstration. Supposons $P(u)$ réalisable, et soit y^* l'un de ses optima. Supposons par l'absurde que y^* est dominé, c'est à dire qu'il existe un point $y \in Y$ tel que $y \preceq y^*$. Puisque nous avons $y^* \leqq u$, nous avons également $y \leqq u$ et donc y est réalisable pour $P(u)$. De plus, puisque Φ est fortement monotone, nous avons $\Phi(y) < \Phi(y^*)$, ce qui contredit l'optimalité de y^*. □

Remarque 1.1. *Au lieu d'utiliser $y \leqq u$ dans le programme P, la plupart des algorithmes générant les points non dominés font intervenir une contrainte de la forme $y \prec u$ se traduisant par les p inégalités $y_i < u_i$, $i \in [\![1, p]\!]$. Cependant, ces contraintes ne peuvent être ajoutées en l'état dans un programme linéaire, car il s'agit d'inégalités strictes. Pour pallier ce problème, nous les remplaçons par des contraintes de la forme $y_i \leq u_i - \varepsilon_i$, où ε_i est une valeur plus petite que la plus petite différence entre deux points réalisables sur l'objectif i. Si les instances ne font intervenir que des valeurs entières - ce qui est le cas ici - nous pouvons choisir $\varepsilon_i = 0.5$, $i \in [\![1, p]\!]$.*

Le résultat suivant garantit que tout point non dominé peut être atteint en résolvant un programme $P(u)$.

Théorème 1.2. *Soit $y^* \in Y_{ND}$. Il existe $u \in \mathbb{R}^p$ tel que y^* soit optimal pour $P(u)$.*

Démonstration. Il suffit de choisir $u = (y_1^*, \ldots, y_p^*)$. □

Les fonctions suivantes sont fortement monotones et souvent utilisées en optimisation multiobjectif :

— *Somme pondérée* : $\lambda y = \sum_{i=1}^{p} \lambda_i y_i$ avec $\lambda_i > 0,\ \forall i \in [\![1, p]\!]$

— *Optimisation lexicographique* : $\text{lexmin}\{y_k, \sum_{\substack{i=1 \\ i \neq k}}^{p} \lambda_i y_i\}$ avec $\lambda_i > 0,\ \forall i \in [\![1, p]\!], i \neq k$.

Optimiser une somme pondérée est trivial car il suffit de résoudre le programme mono-critère sous-jacent. En revanche, obtenir un point non dominé minimisant l'objectif k est plus délicat à mettre en oeuvre, mais permet d'obtenir un point très particulier, car garantissant qu'aucun point de la région explorée n'est meilleur sur f_k. Il existe en pratique deux façons de procéder : la *méthode en deux phases* et la *méthode directe*.

Méthode en deux phases Le point non dominé minimisant l'objectif k est obtenu en résolvant deux programmes en nombres entiers consécutifs. Tout d'abord, le programme P^1 est résolu :

$$P^1(k,u)\begin{cases} \min & y_k \\ \text{s.c.} & y \in Y \\ & y \leqq u \end{cases}$$

Le point $\hat{y}$, optimal pour P^1 s'il existe, est faiblement non dominé : il peut être dominé par un point ayant la même performance sur l'objectif k et de meilleurs performances sur les autres objectifs. Il est donc nécessaire de résoudre un deuxième programme, pour obtenir un point non dominé.

$$P^2(\hat{y})\begin{cases} \min & \sum\limits_{\substack{i=1 \\ i\neq k}}^{p} y_i \\ \text{s.c.} & y \in Y \\ & y_k = \hat{y}_k \\ & y_{-k} \leqq \hat{y}_{-k} \end{cases}$$

Le point optimal de P^2 est non dominé, et minimisera également la composante k. Il est également important de remarquer que $\hat{y}$ est réalisable pour P^2, et il est donc possible de l'utiliser comme solution de départ, ce qui améliore sensiblement les temps de résolution.

Méthode directe Il est également possible de fusionner les deux programmes précédents pour obtenir directement un point lexicographiquement optimal sur l'objectif k. Il suffit d'optimiser une somme pondérée des objectifs, où le poids de y_k est suffisamment grand pour qu'il soit impossible de compenser une dégradation de performance sur l'objectif k malgré d'excellentes valeurs sur les autres objectifs. Si toutes les valeurs considérées sont entières, le résultat suivant fournit une borne inférieure du poids de y_k.

Proposition 1.2. *Si les valeurs de chaque objectif sont entières, le programme suivant retourne un point non dominé minimisant l'objectif k.*

$$P'(k,u)\begin{cases} \min & \Delta y_k + \sum\limits_{\substack{i=1 \\ i\neq k}}^{p} y_i \\ s.c. & y \in Y \\ & y \leqq u \end{cases}$$

avec $\Delta > \sum\limits_{\substack{i=1 \\ i\neq k}}^{p} (U_i - L_i)$ *où* U_i *et* L_i *sont respectivement une borne supérieure et inférieure des valeurs prises par les solutions réalisables dominant u sur l'objectif i.*

Démonstration. D'après la Proposition 1.1, P' retourne un point non dominé. Soit y_k^* la valeur minimale de f_k sur $z(u)$. La pire situation possible serait l'existence des deux points réalisables

suivants dans $z(u)$:

$$y^1 = (U_1, \ldots, U_{k-1}, y_k^*, U_{k+1}, \ldots, U_p)$$
$$y^2 = (L_1, \ldots, L_{k-1}, y_k^* + 1, L_{k+1}, \ldots, L_p)$$

Nous devons choisir Δ de manière à ce que y^1 soit choisi avant y^2, c'est à dire tel que :

$$\Delta y_k^* + \sum_{\substack{i=1 \\ i \neq k}}^{p} U_i < \Delta(y_k^* + 1) + \sum_{\substack{i=1 \\ i \neq k}}^{p} L_i$$

Ce qui entraine la contrainte : $\Delta > \sum_{\substack{i=1 \\ i \neq k}}^{p} (U_i - L_i)$ □

Potentiellement, la méthode directe est plus rapide que celle en deux phases, car elle ne nécessite la résolution que d'un seul programme. Cependant, si Δ est trop grand, P' peut retourner un point faiblement non dominé. En effet, la précision des ordinateurs étant limitée, les autres coefficients de la fonction objectif deviennent négligeables devant Δ. En conséquence, seul l'objectif f_k est optimisé et les autres objectifs sont ignorés. Le point retourné par P' a donc la garantie d'être optimal sur f_k, mais peut être dominé sur les $p - 1$ objectifs restants. Ainsi, utiliser la méthode directe pour énumérer des points peut conduire à un sur-ensemble des points non dominés, et nécessite une procédure additionnelle pour filtrer ceux qui ne le sont que faiblement. Cependant, il n'est nécessaire de comparer l'optimum de $P'(k, u)$ qu'avec les points ayant la même valeur sur l'objectif k, ce qui réduit sensiblement le nombre de comparaisons à effectuer.

Nous utiliserons dans la suite la notation $F(P)$ pour se référer à l'ensemble des points réalisables d'un programme mathématique P.

1.2. Méthode ε-contrainte bicritère

La méthode ε-contrainte, proposée notamment par Haimes (1971) puis par Chankong et Haimes (1983), a vocation de déterminer l'ensemble des points non dominés d'un problème biobjectif . Étant donné deux fonctions objectif f_1 et f_2, elle va générer l'ensemble des points triés selon f_1 en contraignant successivement f_2. L'algorithme 1.1 détaille l'approche.

La figure 1.1 présente les différentes situations possibles. Le premier point est obtenu en résolvant le programme suivant (Figure 1.1a) :

$$P_1 \begin{cases} \text{lexmin} & \{y_1, y_2\} \\ \text{s.c.} & y \in Y \end{cases}$$

D'après le théorème 1.1 il est nécessaire d'optimiser lexicographiquement f_1 puis f_2, pour éviter d'obtenir un point minimisant f_1, mais étant dominé sur f_2.

Algorithme 1.1 : ε-contrainte

Input : Ensemble des points réalisables Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$u \leftarrow M, N \leftarrow \emptyset$
Soit $P = \begin{cases} \text{lexmin} & \{y_1, y_2\} \\ \text{s.c.} & y \in Y \\ & y_2 < u \end{cases}$
while *P est réalisable* **do**
  y^* est l'optimum de P
  $N \leftarrow N \cup \{y^*\}$
  /* La contrainte sur y_2 est translatée pour devenir la plus restrictive possible. */
  $u \leftarrow y_2^*$
$Y_{ND} \leftarrow N$

Si P_1 est infaisable, cela signifie que le problème de départ n'a pas de solution, et donc *a fortiori* aucun point non dominé. Sinon, soit y^1 l'optimum retourné par P_1. Puisque y^1 est optimal sur f_1, il domine tous les points moins bons que lui sur f_2. En d'autres termes, tous les points non dominés par y^1 sont meilleurs que lui sur f_2. La contrainte correspondante est donc ajoutée (figure 1.1b). Les itérations se poursuivent et la contrainte sur f_2 est progressivement translatée pour obtenir un nouveau point non dominé à chaque fois (figure 1.1c).

Les itérations se poursuivent ainsi jusqu'à ce qu'un programme P soit infaisable, indiquant que tous les points non dominés ont été exhibés (figure 1.1d). La méthode énumère donc un point non dominé différent à chaque itération, excepté à la dernière qui sert à prouver que tous les points ont été obtenus. Observons que chaque modèle est de taille constante : ils comportent autant de variables que le domaine initial et seulement une contrainte de plus.

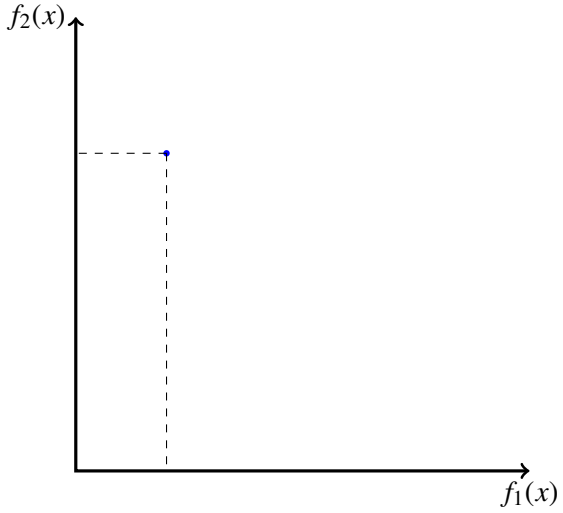
Avant d'étudier les différentes généralisations de cette méthode quand le nombre de critères est arbitraire, nous rappelons dans la suite la définition de la région de recherche, concept fondamental sur lequel elles reposent.

1.3. Région de recherche

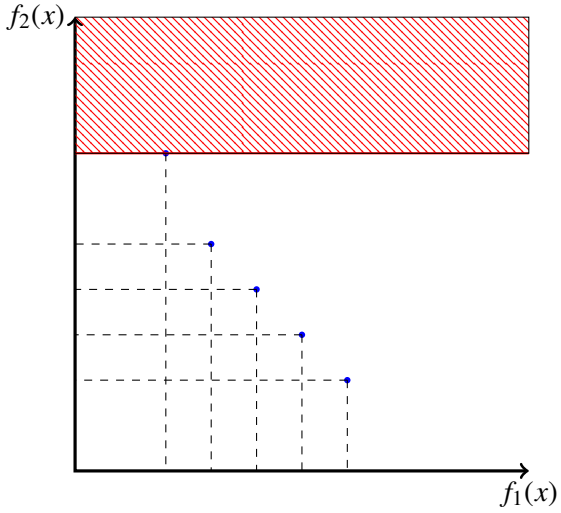
Avant de proposer une généralisation de la méthode ε-contrainte au cas où le nombre de critères est quelconque, nous introduisons certains concepts de la littérature.

Définition 1.3. *(Klamroth et al. (2015)) Étant donné un ensemble de points réalisables $N \subset Y$, la* région de recherche induite par *N est la région de l'espace susceptible de contenir les points non dominés par l'ensemble N. En d'autres termes, il s'agit de l'ensemble des points qui ne sont dominés par aucun élément de N :*

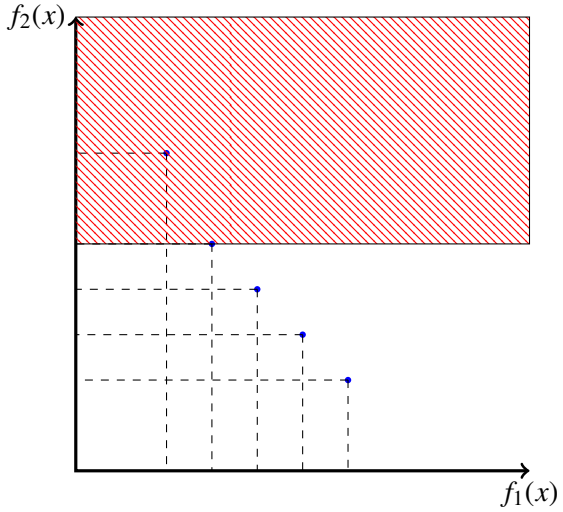
$$S(N) = \{y \in \mathbb{R}^p,\ \nexists y' \in N,\ y' \leqq y\}$$



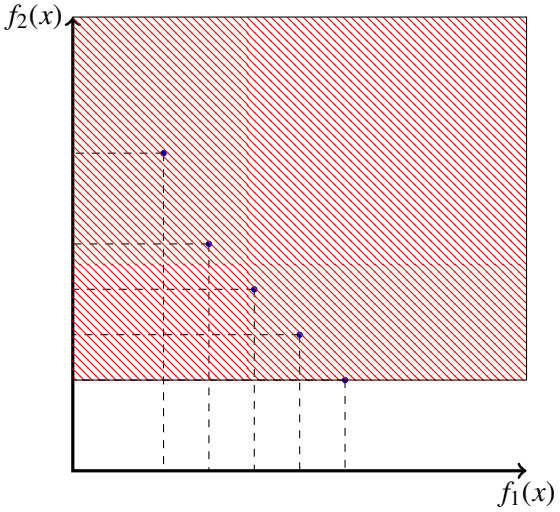
(a) Le point y^1 minimisant lexicographiquement f_1 est trouvé.



(b) La contrainte imposant d'améliorer y_2^1 est ajoutée, contraignant le prochain point à se trouver hors de la région hachurée.



(c) Les itérations se poursuivent et un nouveau point est trouvé. On translate donc la contrainte.



(d) Le programme est infaisable : tous les points non dominés ont été énumérés et l'algorithme s'arrête.

FIGURE 1.1. – Quelques itérations de l'approche ε-contrainte.

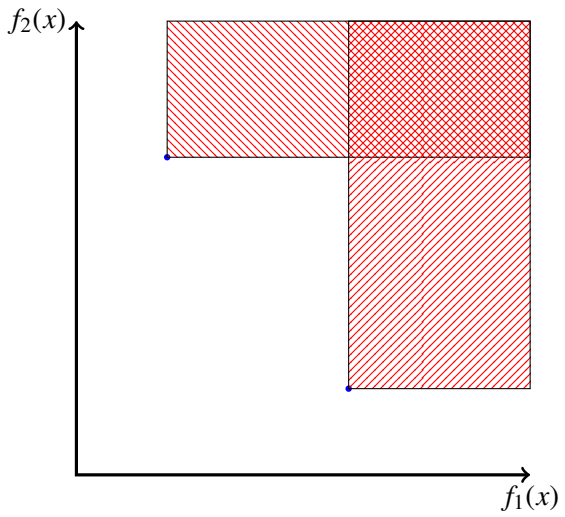


FIGURE 1.2. – Région de recherche ($p = 2$ et $|N| = 2$)

La figure 1.2 présente la région de recherche lorsque deux points ont été énumérés.

Comme remarqué dans Klamroth et al. (2015), on a $N \cap S(N) = \emptyset$ (car chaque point de N se domine lui même au sens large). Ainsi, les méthodes génériques exhibent itérativement un point non dominé de la région de recherche, puis mettent cette dernière à jour en retirant la région que le point trouvé domine au sens large (algorithme 1.2). Leur remarque garantit deux choses : d'une part, ces méthodes n'exhiberont pas plusieurs fois le même point ce qui prouve leur convergence, et d'autre part, nous pouvons affirmer que tous les points non dominés sont dans N quand la région de recherche ne contient plus de solution réalisable.

Algorithme 1.2 : Schéma général des approches

Input : Ensemble des points réalisables Y
Output : Ensemble des points non dominés de Y : Y_{ND}

1 $N \leftarrow \emptyset$
2 $S(N) \leftarrow \mathbb{R}^p$
3 **while** $S(N) \neq \emptyset$ **do**
  `/* On trouve un nouveau point non dominé dans la région de recherche */`
4 $y^* \leftarrow$ explorer $S(N)$
  `/* On calcule la nouvelle région de recherche en retirant les régions de` $\mathbb{R}^p$ `dominées par un point de` N `*/`
5 $S(N \cup \{y^*\}) \leftarrow S(N) \setminus \{y \in S(N), y^* \leqq y\}$
6 $N \leftarrow N \cup \{y^*\}$
7 $Y_{ND} \leftarrow N$

Les différentes caractérisations de la région de recherche utilisées par les algorithmes propo-

sés dans la littérature se décomposent pour la plupart en deux catégories : les caractérisations *implicites* et *explicites*.

1.3.1. Caractérisation implicite

Présentation

Les caractérisations implicites présentent l'avantage d'être très simples à modéliser, à l'instar de celle proposée par Klein et Hannan (1982) puis approfondie par Sylva et Crema (2004), qui repose sur un modèle linéaire contenant des contraintes imposant à la solution optimale de n'être dominée par aucun point de N.

Pour chaque point $y^k \in N,\ k \in [\![1, |N|]\!]$, nous introduisons p variables binaires $z_i^k,\ \forall i \in [\![1, p]\!]$.

$$z_i^k = \begin{cases} 1 & \text{si la contrainte } y_i < y_i^k \text{ est imposée} \\ 0 & \text{sinon} \end{cases}$$

Ce comportement se modélise par les p contraintes suivantes :

$$y_i < y_i^k + M(1 - z_i^k),\ \forall i \in [\![1, p]\!]$$

avec M une borne supérieure de toutes les valeurs atteintes. En effet, si $z_i^k = 0$, la contrainte se désactive en devenant $y < y_i^k + M$ qui n'est pas contraignant.

Pour garantir que la solution considérée ne sera pas dominée par y^k, il faut et il suffit d'imposer qu'elle améliore strictement au moins l'une de ses composantes, ce qui se modélise par l'ajout de la contrainte suivante :

$$\sum_{i=1}^{p} z_i^k \geq 1$$

En résumé, étant donné un ensemble N de points, la région de recherche associée est décrite par le modèle en nombres entiers suivant :

$$S(N) \begin{cases} y_i < y_i^k z_i^k + M(1 - z_i^k) & \forall i \in [\![1, p]\!],\ \forall k \in [\![1, |N|]\!] \\ \sum_{i=1}^{p} z_i^k \geq 1 & \forall k \in [\![1, |N|]\!] \\ z_i^k \in \{0, 1\} & \forall i \in [\![1, p]\!],\ \forall k \in [\![1, |N|]\!] \end{cases} \tag{1.3}$$

Le nombre de contraintes et de variables ajoutées est donc de l'ordre de $p|N|$.

Critique

Cette caractérisation implicite présente l'avantage que la mise à jour se fait simplement par l'ajout de $(p + 1)$ nouvelles contraintes à chaque itération et donc de $(p + 1).|N|$ contraintes au

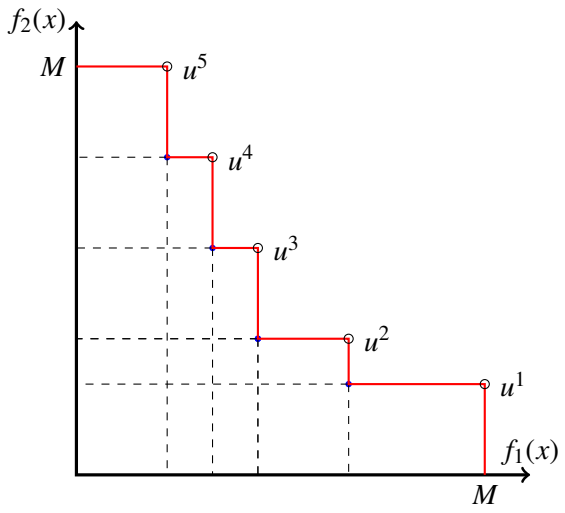


FIGURE 1.3. – Représentation de la région de recherche sous la forme de l'union des zones $z(u^i),\ i \in [\![1, 5]\!]$.

total. De plus, chaque itération (à l'exception de la dernière) retourne un nouveau point non dominé sans aucune répétition (ce qui est caractéristique de la méthode ε-contrainte bicritère). En revanche, il est nécessaire de représenter la région de recherche dans son intégralité (en l'occurrence, la taille du modèle peut croître significativement) sans pouvoir ne s'intéresser qu'à une sous région spécifique pour en exploiter certaines propriétés.

Ainsi, bien qu'étant très élégante, cette caractérisation est très sommaire et n'est donc pas efficace en pratique car très limitée.

1.3.2. Caractérisation explicite

Présentation

Suite à la critique précédente, de nombreuses caractérisations plus explicites ont été proposées, notamment par Przybylski et al. (2010) puis formalisé par Klamroth et al. (2015) qui ont suggéré de décomposer la région de recherche en sous-ensembles pouvant être investigués séparément. Ces sous-ensembles que nous appellerons *zones de recherche* sont définis par un point virtuel, une *borne supérieure locale* dont les composantes sont déterminées par les points non dominés déjà trouvés. La figure 1.3 présente cette décomposition en zones de recherche dans le cas biobjectif.

Cette décomposition explicite de la région de recherche est liée à la caractérisation implicite décrite à la section précédente : considérant l'ensemble des points non dominés que nous avons déjà trouvés, la région de recherche peut se modéliser par l'intersection des ensembles de points

améliorant au moins un critère de chaque point de N. :

$$S(N) = \bigcap_{y^* \in N} \bigcup_{i=1}^{i=p} \{y \in \mathbb{R}^p,\ y_i < y_i^*\}$$

Distribuer puis simplifier l'expression précédente nous permet de reformuler la région de recherche comme une union de *zones de recherche* bornées supérieurement. Un point appartient à une zone s'il améliore strictement toutes les composantes de la borne associée. Dans la suite, $U(N)$ fera référence à l'ensemble des bornes nécessaires pour représenter $S(N)$. En d'autres termes, $U(N)$ est tel que :

$$S(N) = \bigcup_{u \in U(N)} \bigcap_{i=1}^{i=p} \{y \in \mathbb{R}^p,\ y_i < u_i\}$$

Mise à jour de la région de recherche

Construire la région de recherche engendrée par un ensemble de points non dominés se fait itérativement (Proposition 2 de Przybylski et al. (2010)). Considérant un ensemble de bornes $U(N)$, un point y^* appartient à la région de recherche s'il appartient à une ou plusieurs zones, c'est à dire qu'il domine strictement une ou plusieurs bornes de $U(N)$. Chaque borne $u \in U(N)$ telle que $y^* \prec u$ sera mise à jour en créant p nouvelles bornes u^k, $k \in [\![1, p]\!]$, chacune étant identique à u à l'exception d'une composante qui sera partagée avec y^*, c'est à dire :

$$u^k = (u_1, \ldots, u_{k-1}, y_k^*, u_{k+1}, \ldots, u_p)$$

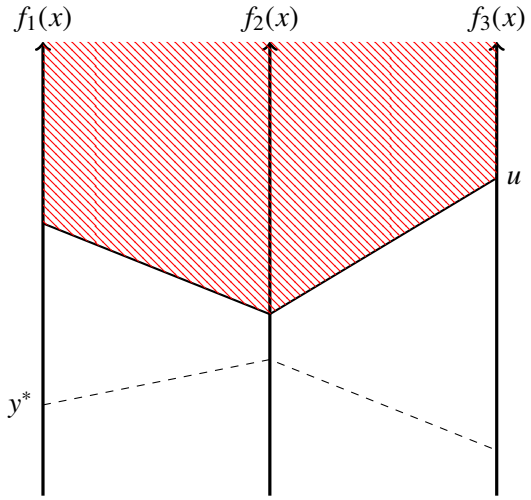
La borne u^k est appelée *k^e fille de u*. La figure 1.4 illustre cette subdivision dans le cas triobjectif

Bien que cette procédure de mise à jour de la région de recherche soit suffisante, elle peut être raffinée pour réduire le nombre de zones générées. En effet, il est possible que certaines des nouvelles subdivisions soient incluses dans d'autres zones, ce qui induirait des redondances (car elles couvriraient la même région de l'espace) et augmenterait inutilement le nombre de zones considérées. L'exemple qui suit illustre ce phénomène :

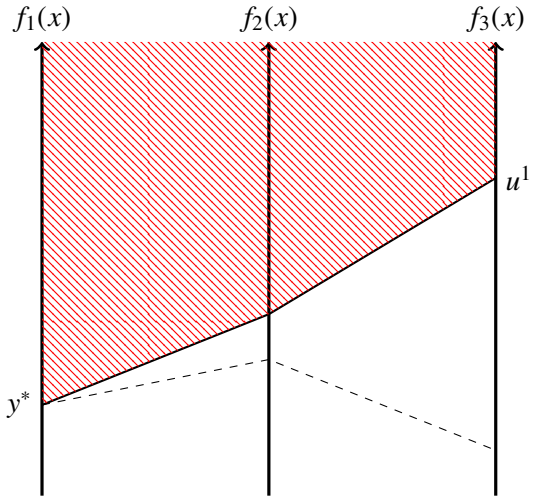
Exemple 1.1. (Mise à jour de la région de recherche) *Dans le cas tricritère, soit la région de recherche ne contenant qu'une seule zone bornée par $u = (\infty, \infty, \infty)$. Soit le point $y^1 = (2, 2, 10)$. La procédure de mise à jour induira trois nouvelles zones bornées respectivement par $u^1 = (2, \infty, \infty)$, $u^2 = (\infty, 2, \infty)$ et $u^3 = (\infty, \infty, 10)$. Soit le nouveau point $y^2 = (10, 1, 1)$. Puisque $y^2 \prec u^2$ et $y^2 \prec u^3$, nous devons subdiviser ces deux zones pour mettre à jour la région de recherche :*

$$\begin{aligned} u^2 = (\infty, 2, \infty) &\implies u^{2,1} = (10, 2, \infty),\ u^{2,2} = (\infty, 1, \infty),\ u^{2,3} = (\infty, 2, 1) \\ u^3 = (\infty, \infty, 10) &\implies u^{3,1} = (10, \infty, 10),\ u^{3,2} = (\infty, 1, 10),\ u^{3,3} = (\infty, \infty, 1) \end{aligned}$$

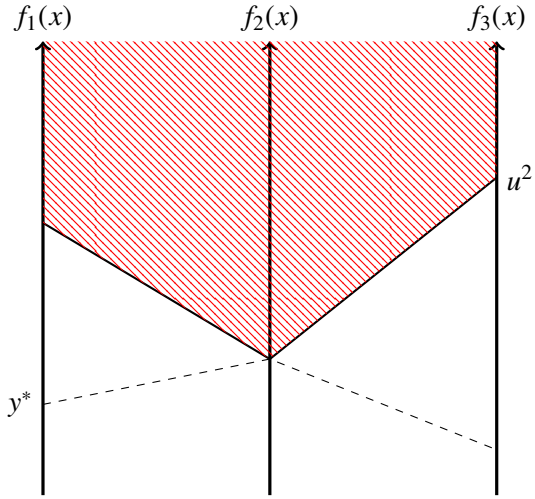
Nous pouvons dès lors constater l'apparition de zones redondantes. En effet, nous avons $z(u^{3,2}) \subset z(u^{2,2})$ (car $u^{3,2} \leqq u^{2,2}$) et $z(u^{2,3}) \subset z(u^{3,3})$ (car $u^{2,3} \leqq u^{3,3}$). Ces deux zones peuvent donc être retirées sans modifier la région de recherche.



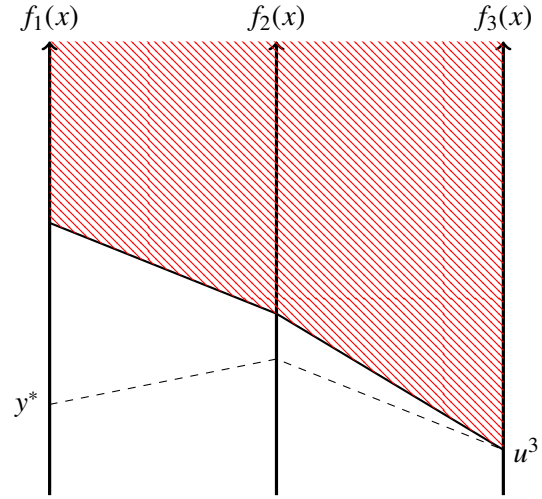
(a) Le point y^* appartient à $z(u)$ car $y^* \prec u$. La zone $z(u)$ est donc subdivisée.



(b) La zone $z(u^1)$ contient les points de $z(u)$ non dominés par y^* améliorant f_1.



(c) La zone $z(u^2)$ contient les points de $z(u)$ non dominés par y^* améliorant f_2.



(d) La zone $z(u^3)$ contient les points de $z(u)$ non dominés par y^* améliorant f_3.

FIGURE 1.4. – Exemple de mise à jour d'une zone dans le cas triobjectif : lors de la découverte d'un point dans une zone, celle ci est subdivisée en trois zones filles.

Les bornes *maximales* sont définies formellement de la manière suivante :
Définition 1.4. (Maximalité) *Une borne $u \in U(N)$ (ou la zone associée $z(u)$) est* maximale *si elle n'est dominée par aucune borne $u' \in U(N)$, c'est à dire :*

$$\nexists u' \in U(N),\ u \preceq u'$$

Nous supposerons dans la suite que toutes les zones présentes dans $U(N)$ sont maximales.

Plusieurs approches pour filtrer les redondances lors de la procédure de mise à jour de la région de recherche ont été étudiées dans la littérature. Celle que nous utiliserons repose sur le résultat suivant qui établit le lien entre les zone de recherche maximale et les points non dominés énumérés.
Proposition 1.3. (Proposition 4.1 de Klamroth et al. (2015)) *Soit $u \in U(N)$. u est une borne maximale si et seulement si, pour chaque composante bornée de $u_k \neq M$, il existe un point $y \in N$ tel que $y_k = u_k$ et $y_{-k} \prec u_{-k}$.*

Démonstration. Soit $u \in U(N)$ une borne maximale dont la composante k est bornée : $u_k \neq M$. Soit $\hat{u} = (u_1, \ldots, u_{k-1}, u_k + \varepsilon, u_{k+1}, \ldots, u_p)$ pour $\varepsilon > 0$. Observons que $z(u) \subset z(\hat{u})$. Puisque u est maximale, cela signifie qu'il existe un point $\hat{y} \in N$ appartenant à $z(\hat{u})$, mais qui n'appartient pas à $z(u)$. En d'autres termes, nous avons :

$$\hat{y}_{-k} \prec \hat{u}_{-k} = u_{-k} \tag{1.4}$$
$$u_k \leq \hat{y}_k < u_k + \varepsilon \tag{1.5}$$

La première inégalité est équivalente à $\hat{y}_{-k} \prec u_{-k}$. Du fait que la seconde inégalité est vraie pour tout ε strictement positif, celle-ci est équivalente à $\hat{y}_k = u_k$. □

Définition 1.5. (Point définissant) *Soit $u \in U(N)$ une borne supérieure locale. Les points y satisfaisant la Proposition 1.3 pour une composante $u_k \neq M$ donnée, c'est à dire tels que :*

$$\begin{cases} y_k = u_k \\ y_{-k} \prec u_{-k} \end{cases} \tag{1.6}$$

sont appelés points définissant la k^e composante de u

Observons que plusieurs points peuvent définir la même composante bornée d'une zone. Ce phénomène se produit quand plusieurs points peuvent atteindre la même valeur sur un objectif. Dans la suite, $N_k(u)$ fera référence à l'ensemble des points définissant la k^e composante de u.

L'algorithme 1.3 synthétise comment utiliser cette propriété pour mettre à jour la region de recherche en évitant de considérer les redondances. Il est à noter que d'autres approches ont été proposées pour ce faire (*cf* Dächert et al., 2017), notamment en introduisant une relation de voisinage.

Algorithme 1.3 : Mise à jour de la région de recherche en évitant certaines subdivisions (Klamroth et al. (2015)).

Input : $U(N)$, ensemble des bornes maximales décrivant l'actuelle région de recherche.
y^*, le nouveau point trouvé.

Output : $U(N \cup \{y^*\})$, ensemble des bornes maximales décrivant la nouvelle région de recherche.

$U(N \cup \{y^*\}) \leftarrow \emptyset$
/* Mise à jour de la région de recherche */
foreach $z(u) \in U(N)$ **do**
  /* Zones qui seront subdivisées */
  if $y^* \prec u$ **then**
    foreach $i \in [\![1, p]\!]$ **do**
      $u^i \leftarrow (u_1, \ldots, y^*_i, \ldots, u_p)$
      $N_i(u^i) \leftarrow \{y^*\}$
      $N_k(u^i) \leftarrow \{y \in N_k(u), y_i < y^*_i\},\ \forall k \in [\![1, p]\!],\ k \neq i,\ u_k \neq M$
      /* Verification qu'au moins un point définit chaque composante bornée de u^i */
      if $N_k(u^i) \neq \emptyset\ \forall k \in [\![1, p]\!], u_k \neq M$ **then**
        $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \cup \{u^i\}$
  /* Sinon, on regarde si y^* définit une composante de u */
  else if $y^*_k = u_k \wedge y^*_{-k} \prec u_k$ *pour un certain* $k \in [\![1, p]\!]$ **then**
    $N_k(u) \leftarrow N_k(u) \cup \{y^*\}$
    $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \cup \{u\}$
  else
    $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \cup \{u\}$

Critique

Caractériser explicitement la région de recherche complexifie l'opération de mise à jour : d'une part, un point trouvé peut appartenir à plusieurs zones, et d'autre part, des redondances peuvent apparaître. De plus, le nombre de zones générées croît significativement en fonction du nombre de critères et de points non dominés. Dans le cas biobjectif, si $|N|$ points ont été énumérés, $|U(N)| = |N| + 1$. Ce résultat a été étendu au cas triobjectif par Dächert et Klamroth (2015) qui montrent que $|N|$ éléments induisent au plus $2|N| + 1$ bornes. Dans le cas général, le résultat a été prouvé par Klamroth et al. (2015), qui, s'inspirant de Kaplan et al. (2008), montrent que si p objectifs sont considérés, $|U(N)| = O(|N|^{\lfloor \frac{p}{2} \rfloor})$.

En revanche, en plus de présenter des propriétés très intéressantes, comme la notion de *points définissants la composante d'une zone*, cette décomposition nous offre la possibilité de nous focaliser sur des sous-ensembles de la région de recherche. Ces caractéristiques nous seront très utiles pour générer l'ensemble des points non dominés, comme nous l'observerons dans la suite.

Première partie

De la génération des points non dominés

Chapitre 2.

État de l'art

Résumé

Parmi les différents algorithmes générant l'ensemble des points non dominés, nous en présentons trois en détail dans un cadre unifié. Ces trois approches ont été choisies car elles reposent sur différentes idées majeures et sont donc représentatives de l'état de l'art.

La première, repose sur une caractérisation implicite de la région de recherche. En utilisant un second programme mathématique, les auteurs arrivent à séparer *mise à jour* et *exploration* de la région de recherche. Cependant, la taille de ce nouveau programme croît très rapidement, rendant cette approche très limitée.

La seconde s'inspire également de la caractérisation implicite de la section précédente, mais tente de décomposer le programme initial en un ensemble de sous programmes de taille constante à résoudre. Ainsi, à chaque itération, un ensemble de programmes mathématiques sont résolus pour obtenir un nouveau point. Même si cette approche est meilleure que la précédente, il est de plus en plus difficile de trouver un nouveau point car le nombre de programmes à résoudre explose avec le nombre d'objectifs.

La dernière est la plus efficace : elle partitionne la région de recherche sous la forme d'une liste de rectangles de dimension $p - 1$. Ainsi, il n'est pas nécessaire de recalculer la région de recherche intégralement à chaque nouveau point trouvé comme dans les approches précédentes. Cependant, le nombre de rectangles augmente énormément avec le nombre d'objectifs, et il est difficile de résoudre des instances avec plus de trois objectifs, même si la taille de l'ensemble final est faible.

Nous concluons ce chapitre par quelques simulations montrant l'évolution de la taille de la région de recherche en fonction du nombre de points non dominés énumérés.

2.1. Présentation

Comme indiqué au Chapitre 1, les algorithmes générant l'ensemble des points non dominés se différencient par la caractérisation de la région de recherche qu'ils manipulent ainsi que par la procédure utilisée pour l'explorer. Cependant, plusieurs catégories d'approches peuvent être distinguées.

Les approches récursives, comme celle proposée par Dhaenens et al. (2010), reposent sur l'observation que certains points non dominés le sont également pour des sous-problèmes ne faisant intervenir qu'un sous ensemble des critères. Après avoir déterminé ces points, la région de recherche est séparée en zones qui seront investiguées séparément. Cependant, le nombre de sous-problèmes augmente exponentiellement en fonction du nombre de critères, et certains points peuvent être non dominés dans plusieurs d'entre eux, ce qui peut potentiellement induire un grand nombre de redondances.

D'autres approches s'appuient sur des caractérisations implicites de la région de recherche. Celle-ci présente l'avantage de pouvoir être très simplement mise à jour lors de la découverte de nouveau points, mais présente l'inconvénient de complexifier le coût des itérations au fur et à mesure de l'énumération des points. Ainsi, bien que pouvant générer théoriquement un grand nombre de points, ces approches sont peu applicables en pratique.

Deux familles d'approches reposant sur des caractérisation explicites peuvent être distinguées : celles utilisant des bornes de dimensions p, comme celles introduites par Klamroth et al. (2015) et celles utilisant des bornes projetées sur un espace de dimension $p - 1$, généralement orthogonalement à un des objectifs. Puisqu'une contrainte est omise, l'utilisation de bornes projetées présente l'avantage de réduire le nombre de programmes infaisables résolus qui sont généralement plus long à résoudre que les programmes réalisables (comme souligné par Boland et al. (2016)) : en effet, lors de la découverte d'une solution, le solveur peut élaguer une partie des branches de l'arbre de résolution et donc réduire le nombre de cas à considérer. En revanche, ces caractérisations sont en réalité des sur-ensembles de la région de recherche, et il est donc possible d'énumérer plusieurs fois le même point non dominé. Ainsi, il est nécessaire de mettre en œuvre une procédure additionnelle pour filtrer les éventuels duplicatas, ce qui n'est pas le cas lors de l'emploi de bornes de dimension p.

Concernant la procédure utilisée pour explorer la région de recherche, les approches se distinguent généralement entre celles qui optimisent une somme pondérée des objectifs et celles qui privilégient un objectif en particulier. Les premières sont très simples à utiliser tandis que les secondes nécessitent de résoudre plusieurs programmes mathématiques à chaque itération, ce qui en complexifie le coût (à l'exception des techniques *directes* évoquées au Chapitre 1 qui peuvent en revanche retourner des points faiblement non dominés). Cependant, le fait d'avoir un point optimal sur un objectif fournit une information supplémentaire pouvant être utilisée pour détecter la vacuité de certaines portions de l'espace, notamment lors de l'exploration d'une borne projetée.

Nous détaillons trois approches différentes dans ce chapitre. La première, Sylva et Crema (2008), repose sur une caractérisation implicite. La seconde, proposée par Lokman et Köksalan (2013),

utilise une caractérisation hybride de la région de recherche : les programmes en nombres entiers résolus sont de taille constante, mais le nombre de programmes à résoudre à chaque itération augmente lors de la découverte d'un nouveau point. Enfin, la dernière approche, proposée par Kirlik et Sayin (2014) utilise une caractérisation explicite s'appuyant sur des bornes projetées orthogonalement à un critère. Chaque algorithme est illustré par un exemple simple. Nous concluons ce chapitre par une analyse de l'évolution de la taille de la région de recherche en fonction du nombre de points non dominés énumérés et du nombre d'objectifs.

2.2. Applications de la caractérisation implicite de la région de recherche (Sylva et Crema, 2004, 2008)

2.2.1. L'algorithme de Sylva et Crema (2004)

Comme dit précédemment, l'approche proposée par Sylva et Crema (2004) repose sur la caractérisation implicite de la région de recherche (1.3) (page 20). Ainsi, considérant que l'on a généré un ensemble de points non dominés $N = \{y^1, \dots, y^{|N|}\}$, les $p \cdot |N|$ variables binaires z_i^k, $i \in [\![1, p]\!]$, $k \in [\![1, |N|]\!]$ suivantes sont introduites :

$$z_i^k = \begin{cases} 1 & \text{si la contrainte } y_i < y_i^k \text{ est imposée} \\ 0 & \text{sinon} \end{cases}$$

et le modèle suivant est résolu :

$$P(N) \begin{cases} \min : & \sum_{i=1}^{p} y_i & & \\ \text{s.c.} & y \in Y & & \\ & y_i < y_i^k + M(1 - z_i^k) & \forall i \in [\![1, p]\!], & \forall k \in [\![1, |N|]\!] \\ & \sum_{i=1}^{i=p} z_i^k \geq 1 & & \forall k \in [\![1, |N|]\!] \\ & z_i^k \in \{0, 1\} & \forall i \in [\![1, p]\!], & \forall k \in [\![1, |N|]\!] \end{cases}$$

Proposition 2.1. *Si $P(N)$ est réalisable, alors l'optimum y^* est non dominé. Si $P(N)$ est non réalisable, alors $N = Y_{ND}$.*

Démonstration. Les contraintes de $P(N)$, à l'exception de $y \in Y$, caractérisent $S(N)$. Si $P(N)$ est réalisable, le point obtenu est donc non dominé par un point de N. De plus, $P(N)$ peut se ramener au problème $P(u)$ du Théorème 1.1, où u contient certaines coordonnées de la forme $u_i = y_k^i$ (si l'une des variables $z_k^i = 1$) et d'autres de la forme $u_j = M$ (si $z_k^i = 0$, $\forall k \in [\![1, |N|]\!]$). Il s'ensuit que $y^* \in Y_{ND}$. En conséquence, y^* est un nouveau point non dominé.

Si $P(N)$ n'est pas réalisable, cela signifie que tous les points non dominés ont été énumérés. □

L'algorithme 2.1 formalise l'approche.

Algorithme 2.1 : Approche de Sylva et Crema (2004)

Input : Ensemble des points réalisables Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
while $P(N)$ *est réalisable* **do**
  $y^* \leftarrow$ résoudre $(P(N))$
  $N \leftarrow N \cup \{y^*\}$
$Y_{ND} \leftarrow N$

Il a néanmoins été observé que ce modèle devenait très rapidement extrêmement coûteux à résoudre. Cela est lié au grand nombre de contraintes disjonctives. C'est pourquoi cette approche a été améliorée par les mêmes auteurs (Sylva et Crema, 2008).

2.2.2. L'algorithme de Sylva et Crema (2008)

En fait, le programme $P(N)$ sélectionne une affectation des variables z_i^k, $i \in [\![1, p]\!]$, $k \in [\![1, |N|]\!]$ qui contraindra les performances de la solution sur chaque objectif y_i. Dès lors, il peut s'avérer judicieux de séparer la résolution du modèle $P(N)$ en deux sous modèles : un programme chargé de sélectionner une nouvelle affectation des variables z_i^k et un programme chargé d'explorer la zone induite par cette affectation pour y exhiber un éventuel point non dominé.

Proposition 2.2. *Soit une affectation des variables*

$$Z = \{z_i^k, \textit{ tels que } z_i^k = 1, i \in [\![1, p]\!], k \in [\![1, |N|]\!]\}$$

Alors, le modèle $P(N)$ peut se réduire en :

$$P(u) \begin{cases} \min & \sum_{i=1}^{p} y_i \\ s.c. & y \in Y \\ & y \prec u \end{cases}$$

$$\textit{avec } u_i = \begin{cases} M & \textit{si } z_i^k = 0,\ \forall k \in [\![1, |N|]\!] \\ \min\limits_{\substack{k \in [\![1,|N|]\!] \\ z_i^k \in Z}} \{y_i^k\} & \textit{sinon} \end{cases}$$

Démonstration. Considérons les $|N|$ contraintes de budget sur l'objectif i :

$$y_i < y_i^k + M(1 - z_i^k) \quad \forall i \in [\![1, p]\!],\ \forall k \in [\![1, |N|]\!]$$

Si, pour un i donné, toutes les variables z_i^k sont nulles, alors on a une contrainte $y_i < M$. Sinon, il suffit de prendre la plus restrictive, qui est par définition $y_i < \min\limits_{\substack{k \in [\![1,|N|]\!] \\ z_i^k \in Z}} \{y_i^k\}$. □

Les affectations des variables z_i^k sont énumérées à l'aide du programme mathématique suivant :

$$Q \begin{cases} \min & 1 \\ \text{s.c.} & \sum_{i=1}^{p} z_i^k \geq 1 \quad \forall k \in [\![1, |N|]\!] \end{cases}$$

Pour chacune des affectations Z retournées par Q, le modèle $P(u)$ associé est résolu. Cependant, il n'y a désormais plus de garantie que ce dernier retourne un nouveau point non dominé, car il se peut qu'aucun point réalisable ne satisfasse la borne supérieure locale. Dans ce cas, il est nécessaire d'ajouter au programme Q une contrainte pour éviter que l'affectation des z_i^k correspondante ne réapparaisse. Cela se traduit par l'ajout de la contrainte suivante.

$$\sum_{z_i^k \in Z} z_i^k \leq |Z| - 1 \tag{2.1}$$

En effet, l'affectation Z viole la contrainte car $\sum_{z_i^k \in Z} z_i^k = |Z|$.

Si, à l'opposé, un nouveau point y^{n+1} est trouvé, il suffit d'introduire les p nouvelles variables binaires z_i^{n+1}, $i \in [\![1, p]\!]$ correspondantes, ainsi que la contrainte $\sum_{i=1}^{p} z_i^{n+1} \geq 1$.

Ainsi, Q correspond à la région de recherche. L'ajout de contraintes lors de la découverte - ou non - d'un nouveau point y^* par P revient à mettre à jour la région de recherche en retirant des régions dominées par y^* ou des régions réputées vides. En réalité, chaque affectation des variables z_i^k correspond à une zone de recherche selon Klamroth et al. (2015), sans aucune garantie de maximalité.

Quelques améliorations ont été apportées à cette idée de base. Il est intéressant d'observer que toutes les affectations de z_i^k ne sont pas nécessairement pertinentes. En effet, si $y_i^k < y_i^l$, alors nous souhaitons imposer que $z_i^k = 1 \implies z_i^l = 1$ par définition des variables z_i^k.

Proposition 2.3. *Soit σ_i une permutation de $[\![1, |N|]\!]$ dans $[\![1, |N|]\!]$ triant les points de N selon l'objectif i, c'est à dire :*

$$y_i^k \leq y_i^l \implies \sigma_i(k) \leq \sigma_i(l)$$

Alors, les $|N|$ contraintes :

$$z_i^{\sigma_i(1)} \leq \ldots \leq z_i^{\sigma_i(|N|)}$$

imposent que si $y_i^k \leq y_i^l$ alors $z_i^k = 1 \implies z_i^l = 1$.

Démonstration. $y_i^k \leq y_i^l \implies \sigma_i(k) \leq \sigma_i(l)$. Donc la contrainte $z_i^k \leq z_i^l$ est ajoutée, et donc $z_i^k = 1 \implies z_i^l = 1$. □

L'autre amélioration proposée par les auteurs de la méthode repose sur l'observation que la contrainte (2.1) élimine l'affectation $Z = \{z_i^k$, tels que $z_i^k = 1\}$, mais également toute affectation où les variables égales à 1 forment un sur-ensemble de Z. Dès lors, les auteurs suggèrent de chercher à trouver les affectations comportant le plus faible nombre de variables z_k^i égales à

1 afin d'éliminer le plus d'affectations possible. Pour ce faire, ils proposent de choisir pour fonction objectif du programme Q :

$$\min \sum_{\substack{i\in[\![1,p]\!]\\ k\in[\![1,|N|]\!]}} z_i^k$$

L'approche est synthétisée dans l'algorithme 2.2.

Algorithme 2.2 : Approche proposée par Sylva et Crema (2008)

Input : Ensemble des points réalisables Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
$n \leftarrow 0$
while Q *est réalisable* **do**
  $Z \leftarrow \{z_i^k$, tels que $z_i^k = 1, i \in [\![1,p]\!], k \in [\![1,|N|]\!]\}$
  foreach $i \in [\![1,p]\!]$ **do**
    $u_i \leftarrow \begin{cases} M & \text{si } z_i^k = 0,\ \forall k \in [\![1,|N|]\!] \\ \min_{y^k\in N}\{y_i^k\} & \text{sinon} \end{cases}$
  $y^{n+1} \leftarrow$ résoudre $P(u)$
  if $P(u)$ *est réalisable* **then**
    $n \leftarrow n+1$
    $N \leftarrow N \cup \{y^{n+1}\}$
    Ajouter les p variables $z_i^{n+1} \in \{0,1\}$
    Ajouter la contrainte $\sum_{i=1}^{p} z_i^{n+1} \geq 1$ au programme Q
    Mettre à jour les p permutations σ_i et les contraintes de la proposition 2.3
  else
    Ajouter la contrainte $\sum_{z_i^k\in Z} z_i^k < |Z| - 1$
$Y_{ND} \leftarrow N$

Il est intéressant de remarquer que suite à ces améliorations, le programme Q n'énumère que des bornes *maximales* selon Klamroth et al. (2015).

Proposition 2.4. *Soient deux affectations réalisables Z et Z' ainsi que les bornes qui leur sont respectivement associées u et u'. Si $u' \preceq u$ alors, $Z \subset Z'$.*

Démonstration. Soit $i \in [\![1,p]\!]$. Supposons que $u'_i < u_i$ et $u_i \neq M$. Par définition, $\exists (k,k') \in [\![1,|N|]\!]^2$ tels que $u_i = y_i^k$ et $u'_i = y_i^{k'}$ avec $y_i^{k'} < y_i^k$. En conséquent, la contrainte $z_i^{k'} \leq z_i^k$ est présente dans le programme Q. En conséquence, z_i^k et $z_i^{k'}$ appartiennent à Z', mais seul z_i^k appartient à Z. Nous avons donc $\{z_i^k \in Z\} \subset \{z_i^{k'} \in Z'\}$.

Si $u'_i < u_i$ et $u_i = M$, alors $\{z_i^k \in Z\} = \emptyset$ et $\{z_i^{k'} \in Z'\} \neq \emptyset$. Ainsi, dans ce cas également, nous avons $\{z_i^k \in Z\} \subset \{z_i^{k'} \in Z'\}$.

Si $u'_i = u_i$, alors $\{z_i^k \in Z\} = \{z_i^{k'} \in Z'\}$. □

Corollaire 2.1. *Soit Z^* une affectation optimale pour Q et soit la borne associée u^*. Soient Z une autre affectation réalisable pour Q et u la borne associée. Alors, $u^* \not\preceq u$.*

Démonstration. Si $u^* \preceq u$, alors $Z \subset Z^*$ d'après la Proposition 2.4. Ainsi,

$$\sum_{\substack{k\in[\![1,p]\!] \\ z_i^k \in Z}} 1 < \sum_{\substack{k\in[\![1,p]\!] \\ z_i^k \in Z^*}} 1$$

Ce qui contredit l'optimalité de Z^*. □

2.2.3. Exemple

Considérons le problème d'optimisation multiobjectif suivant sur lequel nous déroulerons l'algorithme de Sylva et Crema (2008) :

$$\alpha \begin{cases} \min & y_1 = 8x_1 + 5x_2 + 6x_3 \\ \min & y_2 = 8x_1 + 5x_2 + 3x_3 \\ \min & y_3 = x_1 + 2x_2 + 4x_3 \\ \text{s.c.} & x_1 + x_2 + x_3 = 1 \\ & x_i \in \{0,1\} \qquad\qquad \forall i \in [\![1,p]\!] \end{cases}$$

Les trois points non dominés de ce problème sont trivialement atteints pour $x^1 = (1,0,0)$, $x^2 = (0,1,0)$ et $x^3 = (0,0,1)$ dont les images dans l'espace des critères sont :

$$y^1 = f(x^1) = (8,8,1)$$
$$y^2 = f(x^2) = (5,5,2)$$
$$y^3 = f(x^3) = (6,3,4)$$

Ces points sont bien non dominés, car chacun est optimal sur l'un des objectifs.

Le premier point trouvé minimise la somme des objectifs. Il s'agit de y^2. Nous introduisons donc les variables z_j^2, $j \in [\![1,3]\!]$ dans le programme Q.

$$Q \begin{cases} \min & z_1^2 + z_2^2 + z_3^2 \\ \text{s.c.} & z_1^2 + z_2^2 + z_3^2 \geq 1 \\ & z_j^2 \in \{0,1\} \qquad j \in [\![1,3]\!] \end{cases}$$

La solution $(1,0,0)$ est optimale. Nous résolvons donc le problème suivant [1] :

$$P \begin{cases} \min & y_1 + y_2 + y_3 \\ \text{s.c.} & y \in Y \\ & y_1 < 5 \end{cases}$$

1. Par souci de lisibilité, nous omettons volontairement les contraintes $y_i < M$

Le problème est infaisable, 5 étant l'optimum lexicographique sur l'objectif 1. Nous devons donc éliminer cette affectation en ajoutant la contrainte $z_j^2 \leq 0$:

$$Q \begin{cases} \min & z_1^2 + z_2^2 + z_3^2 \\ \text{s.c.} & z_1^2 + z_2^2 + z_3^2 \geq 1 \\ & z_1^2 \leq 0 \\ & z_j^2 \in \{0,1\} \qquad j \in [\![1,3]\!] \end{cases}$$

L'affectation $(0,1,0)$ est optimale pour Q. Nous résolvons donc le problème associé :

$$P \begin{cases} \min & y_1 + y_2 + y_3 \\ \text{s.c.} & y \in Y \\ & y_2 < 5 \end{cases}$$

Le point y^3 est optimal pour P. Nous introduisons donc les variables z_j^3, $j \in [\![1,3]\!]$ dans Q, la contrainte imposant d'améliorer au moins une composante de y^3 ainsi que les 3 contraintes appliquant la Proposition 2.3 :

$$Q \begin{cases} \min & \sum_{i=2}^{3} \sum_{j=1}^{3} z_j^i \\ \text{s.c.} & z_1^2 + z_2^2 + z_3^2 \geq 1 \\ & z_1^3 + z_2^3 + z_3^3 \geq 1 \\ & z_1^2 \leq z_1^3 \\ & z_2^3 \leq z_2^2 \\ & z_3^2 \leq z_3^3 \\ & z_1^2 \leq 0 \\ & z_j^i \in \{0,1\} \qquad i \in [\![2,3]\!],\ j \in [\![1,3]\!] \end{cases}$$

Une solution optimale est $z_2^2 = z_2^3 = 1$, et $z_j^i = 0$ partout ailleurs (la valeur de la fonction objectif est de 2). Le programme P est donc le suivant :

$$P \begin{cases} \min & y_1 + y_2 + y_3 \\ \text{s.c.} & y \in Y \\ & y_2 < 3 \end{cases}$$

Ce programme est infaisable car 3 est la meilleure valeur possible de f_2. Il faut donc retirer cette affectation réalisable de $F(Q)$ en ajoutant la contrainte $z_2^2 + z_2^3 \leq 1$.

$$Q\begin{cases} \min & \sum_{i=2}^{3}\sum_{j=1}^{3} z_j^i \\ \text{s.c.} & z_1^2 + z_2^2 + z_3^2 \geq 1 \\ & z_1^3 + z_2^3 + z_3^3 \geq 1 \\ & z_1^2 \leq z_1^3 \\ & z_2^3 \leq z_2^2 \\ & z_3^2 \leq z_3^3 \\ & z_1^2 \leq 0 \\ & z_2^2 + z_2^3 \leq 1 \\ & z_j^i \in \{0,1\} \qquad i \in [\![2,3]\!],\ j \in [\![1,3]\!] \end{cases}$$

Une solution optimale de Q est $z_3^2 = z_3^3 = 1$ et $z_j^i = 0$ partout ailleurs (la valeur de la fonction objectif est toujours de 2). Le nouveau problème résolu est donc :

$$P\begin{cases} \min & y_1 + y_2 + y_3 \\ \text{s.c.} & y \in Y \\ & y_3 < 2 \end{cases}$$

Ce problème fournit le point y^1. On ajoute donc les variables z_j^1, $j \in [\![1,3]\!]$ dans Q.

$$Q\begin{cases} \min & \sum_{i=2}^{3}\sum_{j=1}^{3} z_j^i \\ \text{s.c.} & z_1^1 + z_2^1 + z_3^1 \geq 1 \\ & z_1^2 + z_2^2 + z_3^2 \geq 1 \\ & z_1^3 + z_2^3 + z_3^3 \geq 1 \\ & z_1^2 \leq z_1^3 \leq z_1^1 \\ & z_2^3 \leq z_2^2 \leq z_2^1 \\ & z_3^1 \leq z_3^2 \leq z_3^3 \\ & z_1^2 \leq 0 \\ & z_2^2 + z_2^3 \leq 1 \\ & z_j^i \in \{0,1\} \qquad i \in [\![1,3]\!],\ j \in [\![1,3]\!] \end{cases}$$

Puisque tous les points non dominés ont été énumérés, aucune des affectations réalisables de Q ne conduira à un point. Il est en revanche nécessaire de toutes les tester afin de terminer l'algorithme. Cela se traduira par l'ajout d'un grand nombre de contraintes, complexifiant de plus en plus le programme Q.

2.2.4. Critiques

La révision proposée par Sylva et Crema (2008) a introduit une idée majeure, consistant à gérer de manière distincte *mise à jour* et *exploration* de la région de recherche. De plus, comme la difficulté de la résolution d'un programme en nombres entiers croît très fortement avec le

nombre de variables binaires, décomposer un gros problème en deux sous problèmes de taille plus faible permet de sensiblement réduire le temps passé à leur résolution.

Il est également intéressant d'observer que la mise à jour de la région de recherche reste toujours relativement peu coûteuse à l'instar de celle proposée dans l'approche initiale. Néanmoins, la taille du programme Q augmente rapidement, et, malgré ces améliorations significatives, il devient assez vite délicat de le résoudre. Dès lors, les procédures de *mise à jour* utilisées par les approches plus récentes sont plus coûteuses, mais permettent d'énumérer les bornes à explorer de manière plus explicite. Cela permet d'éviter de recourir à un programme dont la taille augmente au cours des itérations, comme nous allons le voir par la suite.

À noter que l'approche proposée ici n'est pas la seule manière de décomposer le programme $P(N)$. En introduisant des points virtuels, l'algorithme récent de Boland et al. (2017) sépare le programme initial $P(N)$ en plusieurs programmes. Chacun de ces programmes ne fait intervenir qu'un sous ensemble des points de N dans les contraintes disjonctives, ce qui permet de réduire le nombre de variables binaires et donc le temps de résolution. Bien que présentant les même caractéristiques que celle de Sylva et Crema (2004), l'approche de Boland et al. (2017) arrive à résoudre des instances avec un plus grand nombre d'objectifs.

2.3. Une caractérisation hybride de la région de recherche (Lokman et Köksalan, 2013)

2.3.1. Présentation

L'approche proposée par Lokman et Köksalan (2013) est motivée par l'idée de décomposer le problème $P(N)$ introduit par Sylva et Crema (2004) en sous problèmes dont la taille n'augmente pas au fil des itérations. L'idée consiste toujours à explorer des sous ensembles de la région de recherche qui sont bornés supérieurement. Cependant, contrairement à l'approche de Sylva et Crema (2008), les bornes sont générées explicitement sans recourir à un programme mathématique.

Considérons un ensemble de points non dominés $N \subset Y_{ND}$ déjà énumérés et cherchons à déterminer les bornes $u = (u_1, \ldots, u_p)$ qui sont pertinentes. Ces dernières sont construites itérativement : le choix de la borne u_i sur l'objectif i est contraint par les valeurs choisies sur les $i-1$ objectifs précédents.

Soit $\hat{u} = (u_1, \ldots, u_{i-1})$ le vecteur de dimension $i-1$ contraignant les $i-1$ premières composantes de la solution. Considérons l'ensemble des points $y^k \in N$ appartenant toujours à la région de l'espace délimitée par $\hat{u}$:

$$\hat{N} = N \cap z(\hat{u}) = \{y^k \in N,\ y^k_j < u_j,\ j \in [\![1, i-1]\!]\}$$

Ces points, ainsi que la région qu'ils dominent, doivent être éliminés de la future zone explorée. Les auteurs suggèrent de considérer la zone $\hat{u}^0 = (u_1, \ldots, u_{i-1}, M)$ ne contraignant pas le critère

i, ainsi que les $|\hat{N}|$ zones partielles à i composantes :

$$\hat{u}^k = (u_1, \ldots, u_{i-1}, y_i^k),\ \forall y^k \in \hat{N}$$

Chaque zone $\hat{u}^k$ retire de la région de recherche la région dominée par l'un des points $y^k \in \hat{N}$ en ajoutant la contrainte $y < y_i^k$. Évidemment, si $\hat{N} = \emptyset$, c'est à dire si aucun point dans N ne satisfait la borne partielle $\hat{u}$, il n'est pas nécessaire de contraindre les autres composantes et nous pouvons fixer $u = (u_1, \ldots, u_{i-1}, M, \ldots, M)$. On énumère ainsi l'ensemble des bornes définissant la région de recherche.

Il est également intéressant d'observer que la dernière composante, u_p est entièrement déterminée par les $p-1$ autres. En effet, pour retirer l'ensemble des points restants de $\hat{N}$, il est nécessaire d'imposer la contrainte la plus restrictive, c'est à dire :

$$u_p = \min_{y^k \in \hat{N}} \{y_p^k\}$$

L'algorithme 2.3 détaille la procédure *calculerBornes(*$N, (), 1, p$*)* pour générer les bornes à p dimensions définissant la région de recherche $S(N)$.

Comme montré par l'algorithme 2.4, l'idée consiste à explorer la région de recherche, et à recalculer les bornes si un nouveau point est trouvé. Lorsqu'à une itération, aucun point réalisable ne domine l'une des bornes définissant la région de recherche, l'ensemble des points non dominés a été énuméré. Cependant, en procédant ainsi, le nombre de bornes explose rapidement avec le nombre de points.

2.3.2. Améliorations

Suite à l'observation concluant la section précédente, les auteurs suggèrent deux améliorations pour réduire le nombre de modèles générés à chaque itération.

Il est en réalité possible de réduire la dimension des zones de recherche (et donc leur nombre) en choisissant un critère qui sera minimisé au lieu d'être contraint. Supposons sans perte de généralité qu'il s'agisse du critère p. L'idée consiste à déterminer, à chaque itération, le point de la région de recherche minimisant l'objectif p.

Désormais, chaque borne u est explorée à l'aide du programme $\Pi(u)$, minimisant la composante p tout en contraignant les $p-1$ restantes (l'optimisation lexicographique ne servant qu'à éviter les points faiblement non dominés). :

$$\Pi(u) \begin{cases} \text{lexmin} & \{y_p, \sum_{i=1}^{p} y_i\} \\ \text{s.c. :} & y \in Y \\ & y_{-p} \prec u_{-p} \end{cases}$$

Proposition 2.5. *Supposons que $N \subset Y_{ND}$ contienne les $|N|$ points ayant la plus petite valeur sur f_p. Alors, les bornes de $U(N)$ peuvent être définies sur $p-1$ dimensions en utilisant l'algorithme 2.3 :*

$$U(N) = \text{calculerBornes}(N, (), 1, p-1)$$

Algorithme 2.3 : Calcul récursif des bornes dans l'approche de Lokman et Köksalan (2013)

Procédure : *calculerBornes*$(N, \hat{u}, i, p)$
Input : Ensemble des points non dominés a considérer : N
Une borne partielle : $\hat{u}$
La dimension de $\hat{u}$ à calculer : i
La dimension finale des bornes : p
Output : Liste des bornes définissant la région de recherche $U(N)$ préfixées par $\hat{u}$

1 $U(N) \leftarrow \emptyset$
/* S'il ne reste à calculer que la dimension p, nous choisissons la contrainte la plus restrictive */
2 **if** $i = p$ **then**
3 $y_p^{min} \leftarrow \min_{y \in N}\{y_i\}$
4 $u \leftarrow (\hat{u}_1, \ldots, \hat{u}_{p-1}, y_p^{min})$
5 $U(N) \leftarrow \{u\}$
/* Sinon, nous générons les bornes partielles filles engendrées par $\hat{u}$ et chaque point de N. */
6 **else if** $N = \emptyset$ **then**
/* Il ne reste plus de points à filtrer dans N et on ne contraint pas les autres composantes */
7 $u \leftarrow (\hat{u}_1, \ldots, \hat{u}_{i-1}, M, \ldots, M)$
8 $U(N) \leftarrow \{u\}$
9 **else**
10 **foreach** $y \in N$ **do**
11 $\hat{u}' \leftarrow (\hat{u_1}, \ldots, \hat{u}_{i-1}, y_i)$
/* $\hat{N}$ contient uniquement les points qu'il est nécessaire de considérer pour compléter la zone */
12 $\hat{N} \leftarrow \{y^k \in N, y_i^k < y_i\}$
/* X contient toutes les zones complètes ayant $\hat{u}'$ pour préfixe */
13 $X \leftarrow$ calculerBornes$(\hat{N}, \hat{u}', i+1, p)$
14 $U(N) \leftarrow U(N) \cup X$
/* Cas spécial où l'objectif i n'est pas contraint */
15 $\hat{u}' \leftarrow (\hat{u}_1, \ldots, \hat{u}_{i-1}, M)$
16 $X \leftarrow$ calculerBornes$(N, \hat{u}', i+1, p)$
17 $U(N) \leftarrow U(N) \cup X$

Algorithme 2.4 : Génération des points non dominés en utilisant les bornes de Lokman et Köksalan (2013)

Input : Ensemble des points réalisables : Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$, $U(N) \leftarrow \{(M, \dots, M)\}$ **while** $U(N) \neq \emptyset$ **do**
  Choisir $u \in U(N)$
  if $P(u)$ *est infaisable* **then**
    $U(N) \leftarrow U(N) \setminus \{u\}$
  else
    y^* est l'optimum de $P(u)$
    $N \leftarrow N \cup \{y^*\}$
    $U(N) \leftarrow$ calculerBornes (N,(),1,p)
$Y_{ND} \leftarrow N$

Démonstration. Soit $y \in S(N)$. Par hypothèse, tout point de N est au moins aussi bon que y sur le critère p :

$$y_p^k \leq y_p, \ \forall y^k \in N$$

Soit $u = (u_1, \dots, u_p) \in U(N)$ une borne contraignant le critère p, c'est à dire $u_p \neq M$. Cela signifie qu'il existe un point de $y^k \in N$ tel que $u_p = y_p^k$. De fait, nous savons qu'aucun point de la région de recherche ne peut satisfaire cette contrainte et que le modèle correspondant sera infaisable. Il est donc nécessaire de ne considérer que les bornes u telles que $u_p = M$.

Soit $\hat{u} = (\hat{u}_1, \dots, \hat{u}_{p-2})$ une borne de partielle de dimension $p-2$ et soit $\hat{N} \subset N$ l'ensemble des points satisfaisant les $p-2$ contraintes associées :

$$\hat{N} = \{y^k \in N, \text{ tels que } y_i^k < \hat{u}_i, \ \forall i \in [\![1, p-2]\!]\}$$

Pour que la p^e composante ne soit pas bornée, il est nécessaire de choisir $\hat{u}_{p-1}$ tel que :

$$\{y^k \in \hat{N}, \ y_{p-1}^k < \hat{u}_{p-1}\} = \emptyset$$

et donc de choisir $\hat{u}_{p-1} = \min_{y^k \in \hat{N}} \{y_{p-1}^k\}$, ce que fait précisément la procédure *calculerBornes*$(N, (), 1, p-1)$ □

L'algorithme 2.5 présente l'approche plus en détail.

La seconde idée d'amélioration réside dans l'observation suivante, permettant de connaître le résultat de l'exploration d'une borne sans devoir résoudre le programme associé :

Proposition 2.6. *Soient u et u' deux bornes de dimension $p-1$ telles que $u' \leqq u$.*

1. *Si $\Pi(u)$ est infaisable, alors $\Pi(u')$ est infaisable également.*
2. *Sinon, soit y^* une solution optimale de $\Pi(u)$. Si $y^*_{-p} \prec u'$, alors y^* est aussi une solution optimale de $\Pi(u')$.*

Démonstration. Sachant que $F(\Pi(u')) \subset F(\Pi(u))$, les deux résultats sont triviaux. □

En conservant tous les programmes $\Pi(u)$ résolus au cours des itérations, il est donc possible d'éviter l'exploration de certaines bornes.

Algorithme 2.5 : Approche de Lokman et Köksalan (2013)

Input : Ensemble des points réalisables : Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \dots, M)\}$
while $U(N) \neq \emptyset$ **do**
  $y^{min} \leftarrow (M, \dots, M)$
  foreach $u \in U(N)$ **do**
    if $\Pi(u)$ *est réalisable* **then**
      $y^* \leftarrow \text{opt}(\Pi(u))$
      $y^{min} \leftarrow \min_{y_p}\{y^{min}, y^*\}$
  if $y_p^{min} = M$ **then**
    $U(N) \leftarrow \emptyset$
  else
    $N \leftarrow N \cup \{y^{min}\}$
    $U(N) \leftarrow \text{calculerBornes}(N, (), 0, p-1)$
$Y_{ND} \leftarrow N$

2.3.3. Exemple

Reconsidérons le problème triobjectif α :

$$\alpha \begin{cases} \min & y_1 = 8x_1 + 5x_2 + 6x_3 \\ \min & y_2 = 8x_1 + 5x_2 + 3x_3 \\ \min & y_3 = x_1 + 2x_2 + 4x_3 \\ \text{s.c.} & x_1 + x_2 + x_3 = 1 \\ & x_i \in \{0, 1\} \qquad \forall i \in [\![1, p]\!] \end{cases}$$

La borne initiale étant (M, M), le problème initial est :

$$\begin{cases} \min & y_3 \\ \text{s.c.} & y \in Y \end{cases}$$

qui retourne $y^1 = (8, 8, 1)$. Ce point induit deux nouvelles bornes, $u^1 = (8, M)$ et $u^2 = (M, 8)$. Les deux problèmes associés doivent donc être résolus :

$$\Pi(u^1) \begin{cases} \min & y_3 \\ \text{s.c.} & y \in Y \\ & y_1 < 8 \end{cases} \qquad \Pi(u^2) \begin{cases} \min & y_3 \\ \text{s.c.} & y \in Y \\ & y_2 < 8 \end{cases}$$

$\Pi(u^1)$ et $\Pi(u^2)$ retournent tous les deux $y^2 = (5, 5, 2)$. Ce nouveau point induit trois nouvelles bornes : $u^1 = (5, M)$, $u^2 = (8, 5)$ et $u^3 = (M, 5)$. Nous devons donc considérer les trois problèmes associés :

$$\Pi(u^1) \rightarrow \emptyset$$
$$\Pi(u^2) \rightarrow y^3 = (6, 3, 4)$$
$$\Pi(u^3) \rightarrow y^3 = (6, 3, 4)$$

Observons cependant que puisque $F(\Pi(u^2)) \subset F(\Pi(u^3))$, et que $\Pi(u^3)$ retourne y^3, il est inutile de résoudre $\Pi(u^2)$.

Les nouvelles bornes sont donc $u^1 = (5, M)$, $u^2 = (6, 5)$, $u^3 = (8, 3)$, $u^4 = (M, 3)$. Tous les problèmes associés étant infaisables, l'algorithme termine.

Nous pouvons observer deux inconvénients à cette méthode. D'abord, certains points peuvent être énumérés plusieurs fois. Deuxièmement, le nombre de programmes à résoudre pour trouver un nouveau point augmente avec la taille de N. Dans le cas triobjectif, $|N| + 1$ problèmes sont à résoudre lorsque $|N|$ points ont été énumérés. Mais ce nombre augmente grandement avec le nombre de critères. Illustrons cela sur un exemple.

Soient les quatre points non dominés suivants :

$$y^1 = (1, 5, 3, 3)$$
$$y^2 = (2, 2, 4, 4)$$
$$y^3 = (3, 3, 5, 1)$$
$$y^4 = (4, 4, 2, 2)$$

Les quinze bornes définissant la région de recherche, triées lexicographiquement, sont donc :

$$\begin{aligned}
u_1 = 1 &\rightarrow (1, M, M) \\
u_1 = 2 &\rightarrow (2, 5, M)\ (2, M, 3) \\
u_1 = 3 &\rightarrow (3, 2, M)\ (3, 5, 4)\ (3, M, 3) \\
u_1 = 4 &\rightarrow (4, 2, M)\ (4, 3, 4)\ (4, 5, 4)\ (4, M, 3) \\
u_1 = M &\rightarrow (M, 2, M)\ (M, 3, 4)\ (M, 4, 4)\ (M, 5, 2)\ (M, M, 2)
\end{aligned}$$

Bien que certaines bornes puissent être éliminées, on peut observer que le nombre de bornes à générer augmente considérablement.

2.3.4. Critiques

L'approche proposée par Lokman et Köksalan (2013) introduit trois nouvelles idées fondamentales.

Tout d'abord, il est possible - et moins coûteux - de générer les bornes délimitant la région de recherche explicitement. Éviter de recourir à un programme mathématique pour les déterminer offre un gain de temps considérable d'après les expérimentations. La taille des programmes résolus par cette approche reste constante, contrairement aux approches de Sylva et Crema.

La deuxième idée consiste à utiliser des bornes *projetées* sur l'espace de dimension $p-1$ en utilisant l'optimisation lexicographique. Cela présente l'avantage de sensiblement réduire le nombre de programmes à résoudre pour générer tous les points non dominés.

Enfin, conserver l'historique de tous les modèles résolus au cours des itérations permet de réduire encore plus le nombre de bornes à explorer. En effet, bien que cela induise une consommation de mémoire supplémentaire, cette idée permet de connaître à l'avance les solutions optimales de nombreux programmes et donc d'éviter des optimisations inutiles.

De nombreuses autres approches proposent différentes méthodes pour énumérer les bornes plus ou moins efficacement. L'idée de Özlen et Azizoğlu (2009) (reprise par Mavrotas (2009) puis améliorée par Zhang et Reimann (2014)) consiste à énumérer les bornes en testant toutes les valeurs possibles dans l'intervalle $[y_i^I; y_i^U]$ sur chaque critère $i \in [\![1, p]\!]$. Cette approche est néanmoins peu satisfaisante car un grand nombre de combinaisons sont redondantes. De plus, en fonction du pas utilisé, certains points peuvent être omis. L'idée proposée par Laumanns et al. (2005) considère toutes les combinaisons de points possibles pour former les bornes. Comme montré dans cette section, un grand nombre d'entre elles peuvent être évitées.

D'une manière générale, ces approches présentent rapidement des limites. Même si leur taille reste constante, le nombre de modèles à résoudre par itération explose, à la fois avec le nombre de critères et la taille de N, rendant difficile la résolution de larges instances. De plus, la procédure de mise à jour de la région de recherche, lors de la découverte d'un point y^*, est coûteuse car toutes les bornes doivent être recalculées. Les approches plus récentes proposent une méthode de mise à jour *incrémentale*, c'est à dire une procédure permettant de calculer $U(N \cup \{y^*\})$ directement à partir des bornes de $U(N)$. Cela permet d'éviter d'avoir à reconsidérer tous les points de N, et donc de simplifier énormément la procédure de mise à jour.

2.4. Une approche utilisant une caractérisation explicite de la région de recherche (Kirlik et Sayin, 2014)

2.4.1. Présentation

Parmi les approches proposées dans la littérature, celle proposée par Kirlik et Sayin (2014) semble être la plus efficace. Comme précédemment, l'idée principale réside dans la caractérisation *explicite* de la région de recherche sous la forme d'une liste de bornes $U(N)$. En revanche, celle ci peut être déterminée *incrémentalement* : lors de la découverte d'un nouveau point non dominé y^*, il est possible de calculer la nouvelle région de recherche $U(N \cup \{y^*\})$ uniquement à partir de $U(N)$.

Les auteurs proposent de considérer $U(N)$ comme une liste de *rectangles* (ou *boîtes*) de dimension $p-1$. Soient $u, l \in \mathbb{R}^{p-1}$ deux bornes de dimension $p-1$. Nous noterons par la suite $R(u,l)$ le *rectangle* borné supérieurement par u et inférieurement par l, autrement dit l'ensemble des points dont la projection est bornées par u et l :

$$R(u,l) = \{y \in R^p, \text{ tel que } l \leqq y_{-p} \prec u\}$$

Chaque rectangle $R(u,l)$ est exploré à l'aide du modèle $\Pi(u)$ qui permet de trouver le point minimisant l'objectif p sous la contrainte de satisfaire uniquement la borne supérieure locale u :

$$\Pi(u) \begin{cases} \text{lexmin} & \{y_p, \sum_{i=1}^{p-1} y_i\} \\ \text{s.c.} & y \in Y \\ & y_{-p} \prec u \end{cases}$$

Observons que l'éventuelle solution optimale de $\Pi(u)$ n'appartient pas nécessairement à $R(u,l)$. La proposition suivante permet d'exploiter cette propriété :

Proposition 2.7. *Soit le rectangle $R(u,l)$.*

1. *Si $\Pi(u)$ est infaisable, alors $R(u,l)$ est vide, ainsi que tout rectangle $R(u',l')$ tel que $u' \preceq u$*
2. *Sinon, soit y^* le point optimal de $\Pi(u)$. Alors, tout rectangle $R(u',l')$ tel que :*

$$y^*_{-p} \leqq l' \prec u' \preceq u$$

ne contient que des points dominés par y^*.

Démonstration. Soit une borne $u' \in \mathbb{R}^{p-1}$ telle que $u' \preceq u$. Nous avons $F(\Pi(u')) \subset F(\Pi(u))$.

1. Si $\Pi(u)$ est infaisable, alors $F(\Pi(u)) = \emptyset$, donc $F(\Pi(u')) = \emptyset$ et donc $\Pi(u')$ est infaisable.
2. Soit $y \in R'(u',l')$. Par construction, $y^*_{-p} \leqq y_{-p}$. De plus, si y^* est la solution optimale de $\Pi(u)$, et que $y^*_{-p} \prec u'$, cela signifie que $y^* \in F(\Pi(u'))$, et donc que y^* est optimal pour $\Pi(u')$. Donc $y^*_p \leq y_p$. En conséquence, y est dominé.

□

Il reste à définir une procédure pour subdiviser les rectangles lors de la découverte d'un nouveau point non dominé y^*. Les auteurs suggèrent d'utiliser l'algorithme 2.6, permettant de *partitionner* la projection de la région de recherche sur $\mathbb{R}^{p-1}$. Observons que si y^*_{-p} appartient entièrement à $R(u,l)$, cela induit 2^{p-1} subdivisions. Avant de mettre à jour la région de recherche, les rectangles qui satisfont la Proposition 2.7 sont éliminés.

Pour énumérer les points non dominés, il suffit donc de sélectionner un rectangle, puis de mettre à jour la région de recherche en fonction du résultat de son exploration. Bien que ce ne soit pas obligatoire, les auteurs suggèrent de choisir le rectangle maximisant l'hypervolume suivant :

$$R^*(u^*,l^*) = \underset{R(u,l)\in U(N)}{\text{argmax}} \{\prod_{i=1}^{p-1}(u_i - y^I_i)\}$$

Algorithme 2.6 : Subdivision des rectangles selon Kirlik et Sayin (2014)

Input : Ensemble des rectangles : $U(N)$
Le nouveau point non dominé : y^*
Output : Ensemble des rectangles mis à jour : $U(N \cup \{y^*\})$

$U(N \cup \{y^*\}) \leftarrow \emptyset$
foreach $R(u, l) \in U(N)$ **do**
  $T \leftarrow \{R(u, l)\}$
  foreach $i \in [\![1, p-1]\!]$ **do**
    if $l_i < y_i^* < u_i$ **then**
      $T' \leftarrow \emptyset$
      foreach $R(u', l') \in T$ **do**
        $\hat{u} \leftarrow (u'_1, \ldots, u'_{i-1}, y_i^*, u'_{i+1}, \ldots, u'_{p-1})$
        $\hat{l} \leftarrow (l'_1, \ldots, l'_{i-1}, y_i^*, l'_{i+1}, \ldots, l'_{p-1})$
        $T' \leftarrow T' \cup \{R(u', \hat{l}), R(\hat{u}, l')\}$
      $T \leftarrow T'$
    $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \cup T$

Ainsi, si deux rectangles $R(u, l)$ et $R'(u', l')$ avec $u' \preceq u$ sont présents dans la région de recherche, alors R sera toujours considéré avant R'. Cela est judicieux car le résultat de l'exploration de R pourrait appartenir à R'.

L'algorithme 2.7 présente l'approche en détail. Un exemple de son déroulement pas à pas est présenté dans la section suivante.

2.4.2. Exemple

La figure 2.1 présente le déroulement de l'algorithme sur le problème α :

$$\alpha \begin{cases} \min & y_1 = 8x_1 + 5x_2 + 6x_3 \\ \min & y_2 = 8x_1 + 5x_2 + 3x_3 \\ \min & y_3 = x_1 + 2x_2 + 4x_3 \\ \text{s.c.} & x_1 + x_2 + x_3 = 1 \\ & x_i \in \{0, 1\} \qquad \forall i \in [\![1, p]\!] \end{cases}$$

Tout d'abord, la region de recherche est initialisée avec un seul rectangle contenant tout l'espace des objectifs. Lors de son exploration, y^1, qui minimise f_3, est exhibé, induisant ainsi $2^{3-1} = 4$ nouveaux rectangles. Cependant, l'un d'eux ne contient que des points moins bons que y^1 sur f_1 et f_2. Puisque y^1 est optimal sur f_3, nous pouvons en conclure que le rectangle ne contient que des points dominés, et nous l'éliminons de la région de recherche (Figure 2.1a). Le rectangle ayant le plus grand volume est sélectionné pour la prochaine exploration (en gras).

Algorithme 2.7 : Approche de Kirlik et Sayin (2014)

Input : Ensemble des points réalisables Y
Point idéal : y^I
Point anti idéal : y^U

Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
$U(N) \leftarrow \{R(y^U, y^I)\}$
while $U(N) \neq \emptyset$ **do**
  solve $\Pi(u^*)$
  if $\Pi(u^*)$ *est infaisable* **then**
    /* Application de la Proposition 2.7.1 */
    Retirer les rectangles $R(u, l)$ avec $u \leqq u^*$
  else
    y^* est l'optimum de $\Pi(u^*)$
    if $y^* \in N$ **then**
      /* y^* a déjà été obtenu, on applique la Proposition 2.7.2 */
      Retirer les rectangles $R(u, l)$ avec $y_{-p} \leqq l \preceq u \leqq u^*$
    else
      /* y^* est un nouveau point non dominé */
      subdiviser ($U(N)$,y^*)
      /* Application de la Proposition 2.7.2 */
      Retirer les rectangles $R(u, l)$ avec $y_{-p} \leqq l \preceq u \leqq u^*$
      $N \leftarrow N \cup \{y^*\}$

Dans la figure 2.1b, le point y^2 a été énuméré et nous procédons à la subdivision. Comme précédemment, les rectangles ne contenant que des points dominés sont éliminés. En revanche, le rectangle en pointillés ne peut être retiré, car il peut contenir des points meilleurs que y^2 sur f_3.

La prochaine itération retourne y^3, et nous procédons de manière identique à précédemment (Figure 2.1c). Le rectangle maximisant le volume est donc sélectionné. Dans celui ci, y^2 est énuméré à nouveau. Dans la figure 2.1d, tous les points moins bons que y^2 sur les objectifs f_1 et f_2 ont été retirés du rectangle exploré. Il s'agit précisément du rectangle en pointillé dans la figure 2.1b dont nous avons désormais la confirmation qu'il ne contient que des points inférieurs à y^2 sur f_3.

Le programme associé au rectangle sélectionné est infaisable. Puisque Π ne contraint les objectifs que supérieurement, nous pouvons en déduire qu'il n'y a aucun point dans toute la région hachurée. Cette situation se reproduit aux iterations suivantes décrites dans les figures 2.1e et 2.1f.

2.4.3. Critiques

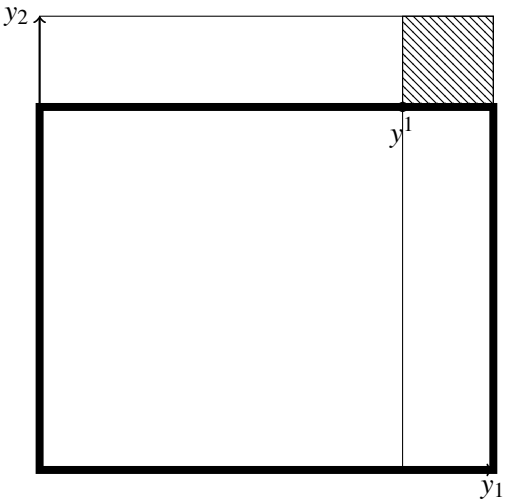
La mise à jour incrémentale de la région de recherche rend cette approche particulièrement efficace. De plus, l'exploration d'un rectangle peut apporter de l'information sur d'autres, ce qui permet d'en éliminer certains sans avoir à les considérer.

En revanche, cette approche présente deux inconvénients. Tout d'abord, le même point peut être obtenu plusieurs fois, et il est nécessaire de vérifier si la solution optimale de $\Pi(u^*)$ est déjà présente dans N. De plus, le nombre de rectangles explose très rapidement quand le nombre de critères croît. Pour y remédier, il est possible de remplacer le partitionnement de la région de recherche par un simple recouvrement. Puisque l'exploration de chaque rectangle ne tient pas compte de la borne inférieure, il suffit de ne considérer que les bornes supérieures.

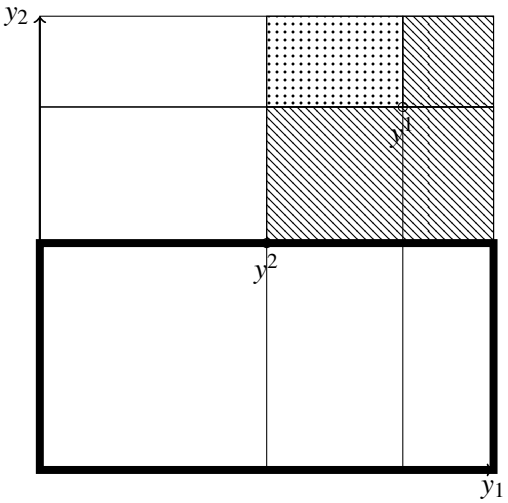
2.5. Synthèse des approches

Les courbes présentées à la Figure 2.2 montrent, en fonction du nombre de points de N et du nombre d'objectifs, l'évolution de la taille de la région de recherche pour les approches proposées par Kirlik et Sayin (2014) et par Klamroth et al. (2015). Les algorithmes correspondants ont été implémentés avec le langage de programmation *Haskell*. Chaque simulation a été faite sur un ensemble de points non dominés choisis aléatoirement. Les performances de chaque point sont tirées uniformément dans $[\![1, 1000]\!]$.

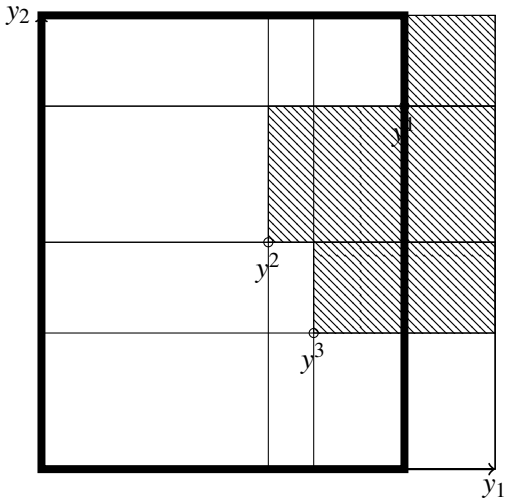
La Figure 2.2a présente le nombre théorique de bornes maintenues par l'algorithme de Kirlik et Sayin (2014). Comme évoqué précédemment, en pratique, ce nombre est un peu inférieur car lors de la génération d'un point, il est possible d'utiliser l'information obtenue sur le rectangle exploré pour réduire le nombre de subdivisions (Proposition 2.7). En revanche, ce partitionnement de la région de recherche augmente extrêmement vite avec le nombre d'objectifs (au point



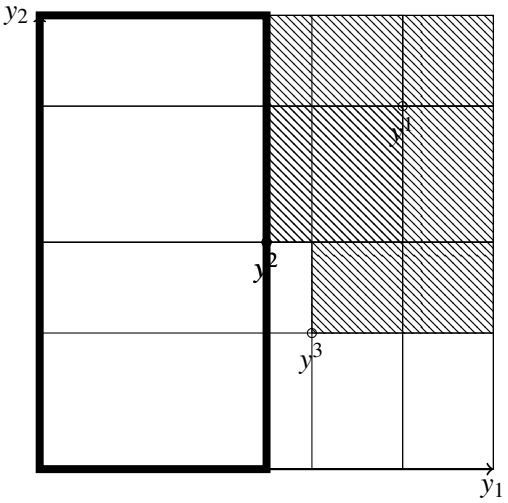
(a) y^1 est trouvé ce qui induit trois nouvelles boîtes.



(b) y^2 a été trouvé. On subdivise le plan verticalement et horizontalement ce qui induit 8 nouvelles boîtes. Parmi celles qui sont inclues dans la boîte explorée, nous retirons celles dominées par y^2.

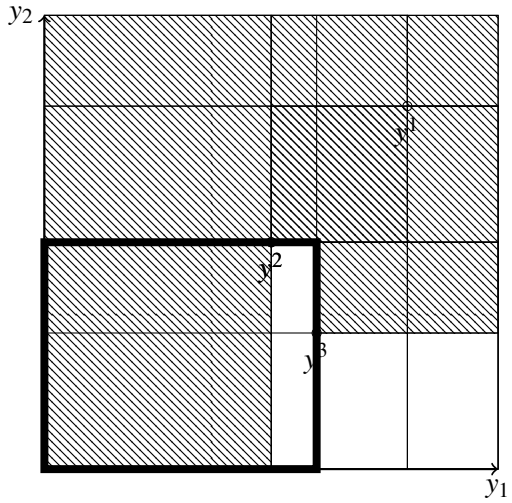


(c) Le point y^3 a été énuméré, la région de recherche est donc mise à jour.

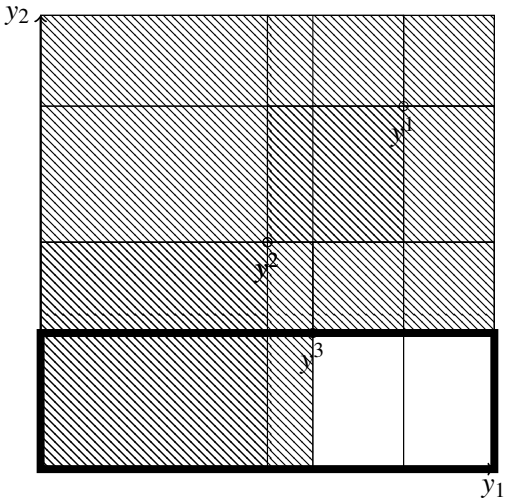


(d) y^2 a été obtenu à nouveau. La région dominée par y^2 a été retirée de la boîte explorée précedemment.

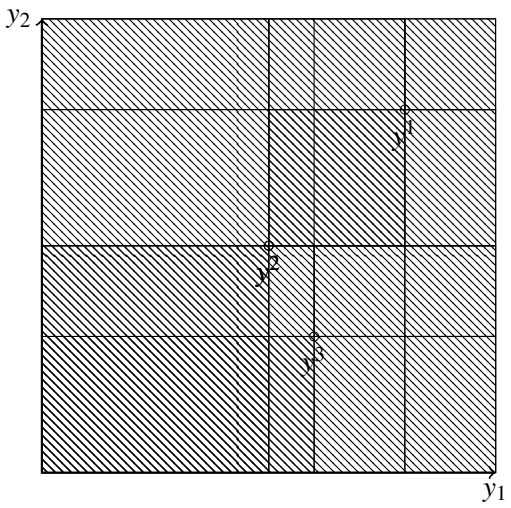
FIGURE 2.1. – Énumération des points non dominés avec l'algorithme proposé par G. Kirlik et S. Sayin.



(e) La boîte explorée étant vide (le problème n'admet pas de solutions), nous retirons toutes les boîtes dont la borne superieure domine celle explorée.



(f) Idem.

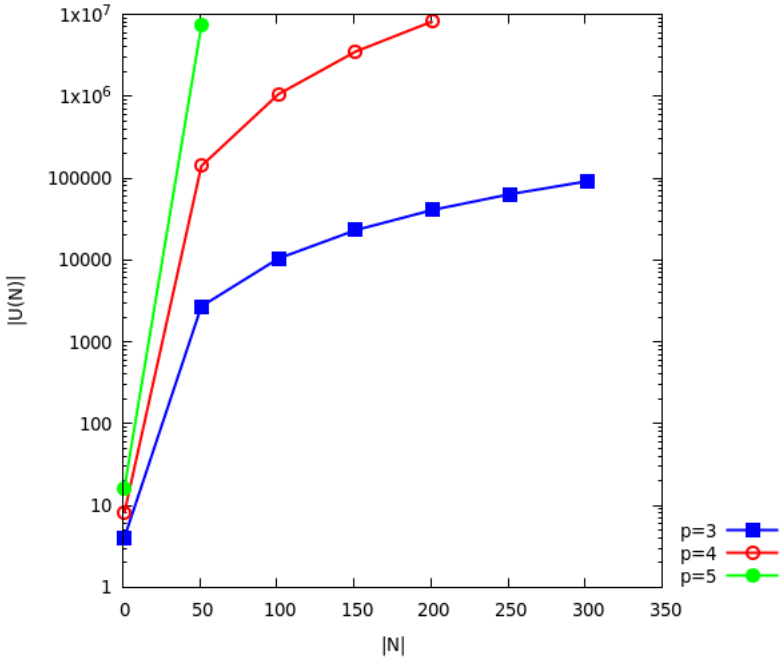


(g) Il ne reste plus de boîtes à explorer : tous les points non dominés ont été considérés.

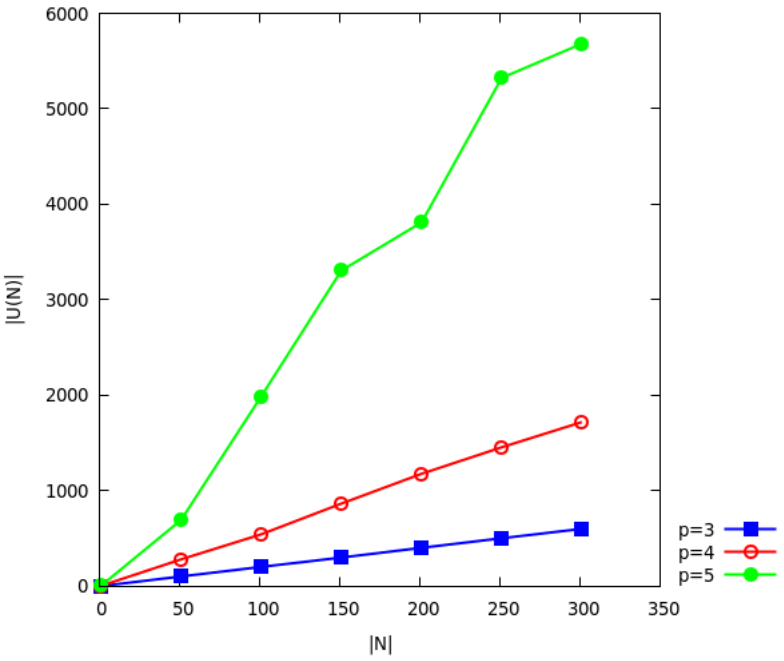
FIGURE 2.1. – Élimination des boîtes restantes.

qu'il fut impossible de terminer les simulations pour $p = 4$ et $p = 5$ avec 8 GB de mémoire). Nous verrons dans la suite que c'est également le cas expérimentalement.

Enfin, la Figure 2.2b présente le nombres de zones de recherches de Klamroth et al. (2015) nécessaires pour représenter la région de recherche dans chacun des cas. Nous pouvons observer que cette approche est plus stable que l'approche de Kirlik et Sayin (2014) quand le nombre de critère augmente (par exemple, moins de 6000 bornes sont générées pour l'instance à 5 objectifs et 300 points). Nous pouvons également observer que dans le cas triobjectif, nous avons systématiquement $2|N| + 1$ bornes à traiter quand $|N|$ points ont été énumérés, ce qui vérifie expérimentalement le résultat de Dächert et Klamroth (2015).



(a) Approche de Kirlik et Sayin (2014) (*échelle logarithmique*).



(b) La caractérisation de Klamroth et al. (2015) semble être la plus robuste.

FIGURE 2.2. – Evolution du nombre de bornes et de zones générées en fonction du nombre de points et de critères pour définir la région de recherche (simulation).

Chapitre 3.

Approche ε-contrainte généralisée

Résumé

Nous présentons dans ce chapitre un nouvel algorithme permettant la génération des points non dominés de programmes en nombres entiers multiobjectifs. Il repose sur la caractérisation de la région de recherche sous la forme d'une liste de zones mais n'en explore que les projections orthogonales selon un objectif. De ce fait, il est possible d'utiliser le résultat de l'exploration d'une zone pour détecter des zones vides.

De plus, nous montrons comment, en exploitant certains résultats, il est possible à la fois de garantir que les programmes mathématiques résolus sont réalisables et de leur fournir une solution de départ.

L'algorithme proposé apparaît significativement plus performant que les algorithmes récents de la littérature, comme le montrent les expérimentations que nous avons conduit sur des instances des problèmes du sac à dos et de l'affectation multiobjectifs.

3.1. Introduction

3.1.1. Une première approche

Puisque la caractérisation de la région de recherche proposée par Kirlik et Sayin (2014) montre rapidement ses limites quand le nombre d'objectifs augmente, nous proposons dans ce chapitre de bâtir une approche basée sur l'utilisation des zones de recherches proposées par Klamroth et al. (2015). L'algorithme 1.2 s'adapte très simplement de la manière suivante.

La région de recherche est initialisée avec une zone contenant l'intégralité de l'espace. Tout au long de son déroulement, l'algorithme maintient la liste des points N déjà énumérés ainsi que la région de recherche associée, sous la forme de la liste $U(N)$ des bornes supérieures locales.

À chaque itération, une zone $u \in U(N)$ est choisie, puis explorée à l'aide du programme $P(u)$ suivant :

$$P(u)\begin{cases} \min & \sum_{i=1}^{p} y_i \\ \text{s.c.} & y \in Y \\ & y \prec u \end{cases}$$

Le Théorème 1.1 garantit que l'optimum de $P(u)$, s'il existe, est non dominé. Deux cas de figure peuvent donc se présenter : si $P(u)$ est infaisable, la borne u est retirée de la région de recherche. Sinon, la region de recherche est mise à jour avec le point obtenu grâce à l'algorithme 1.3. Lorsque la liste de bornes est vide, tous les points non dominés ont été énumérés. L'algorithme 3.1 formalise la méthode.

3.1.2. Exemple

Reconsidérons l'exemple α :

$$\alpha\begin{cases} \min & y_1 = 8x_1 + 5x_2 + 6x_3 \\ \min & y_2 = 8x_1 + 5x_2 + 3x_3 \\ \min & y_3 = x_1 + 2x_2 + 4x_3 \\ \text{s.c.} & x_1 + x_2 + x_3 = 1 \\ & x_i \in \{0,1\} \qquad \forall i \in [\![1,p]\!] \end{cases}$$

Dont les points non dominés sont :

$$y^1 = f(x^1) = (8,8,1)$$
$$y^2 = f(x^2) = (5,5,2)$$
$$y^3 = f(x^3) = (6,3,4)$$

Algorithme 3.1 : Schéma général de l'approche utilisant des zones de recherche.

Input : Ensemble des points réalisables :Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \ldots, M)\}$
while $U(N) \neq \emptyset$ **do**
  /* On choisit la première zone de la région de recherche */
  $u \leftarrow$ head $(U(N))$
  /* On trouve un nouveau point non dominé dans la région de recherche */
  résoudre $P(u)$
  if $P(u)$ *est infaisable* **then**
    $U(N) \leftarrow U(N) \setminus \{u\}$
  else
    y^* est l'optimum de $P(u)$
    $N \leftarrow N \cup \{y^*\}$
    /* On met a jour incrémentalement la région de recherche avec l'algorithme 1.3 */
    mettre à jour $U(N)$
$Y_{ND} \leftarrow N$

La région de recherche est initialisée avec la zone $u = (M, M, M)$. Le point y^2, minimisant la somme des critères, est retourné par $P(u)$, ce qui induit trois nouvelles zones : $u^1 = (5, M, M)$, $u^2 = (M, 5, M)$ et $u^3 = (M, M, 2)$. La première zone est sélectionnée pour être explorée.

$P(u^1)$ est infaisable ce qui entraine l'élimination de la borne associée. $P(u^2)$ retourne y^3, ce qui induit :

- De la part de u^2 : $u^{2,1} = (6, 5, M)$, $u^{2,2} = (M, 3, M)$, $u^{2,3} = (M, 5, 4)$
- Puisque $y^3 \not\prec u^3$, la borne u^3 n'est pas modifiée.

Les programmes $P(u^{2,1})$, $P(u^{2,2})$ et $P(u^{2,3})$ sont infaisables, et les zones considérées seront donc éliminées successivement. Le programme $P(u^3)$ retournera y^1, induisant les trois nouvelles zones : $u^{3,1} = (8, M, 2)$ $u^{3,2} = (M, 8, 2)$, $u^{3,3} = (M, M, 1)$. Les trois programmes associés étant infaisables, les bornes sont éliminées. Puisque la région de recherche est désormais vide l'algorithme termine.

En résumé, 10 programmes mathématiques ont été résolus, ce qui est légèrement plus que l'approche de Kirlik et Sayin (2014). En revanche, cette dernière considère jusqu'à 10 bornes simultanément, contre 4 ici. Nous verrons dans la suite que cet écart se creuse énormément avec l'augmentation du nombre d'objectifs.

3.1.3. Remarques

Cette approche présente l'intérêt de garantir que chaque point non dominé ne sera énuméré qu'une seule fois. De plus, la taille de cette caractérisation de la région de recherche évolue plus lentement que celles utilisées par les autres approches quand le nombre de points non dominés et d'objectifs augmente.

Les modifications de cet algorithme que nous proposons dans la section suivante portent sur trois points : la procédure utilisée pour *explorer* une zone de recherche, la procédure utilisée pour *mettre à jour* la région de recherche et la procédure utilisée pour *sélectionner une zone* à chaque itération. Bien que ces trois étapes sont évidemment liées, nous les détaillerons séparément dans la section suivante.

3.2. Présentation

Une idée consiste à remplacer la résolution de $P(u)$ par celle du problème $\Pi(k, u)$ retournant un point non dominé minimisant la composante k dans u_{-k}.

$$\Pi(k,u)\begin{cases} \text{lexmin} & \{y_k, \sum\limits_{\substack{i=1\\ i\neq k}}^{p} y_i\} \\ \text{s.c.} & y \in Y \\ & y_{-k} \prec u_{-k} \end{cases}$$

Observons que $\Pi(k, u)$ repose sur l'optimisation lexicographique pour garantir que le point retourné, s'il existe, est non dominé. En effet, si seul l'objectif k est minimisé, alors il est possible d'obtenir un point *faiblement non dominé*, qui serait donc dominé sur le reste des objectifs (Théorème 1.1).

Dans un premier temps, nous présenterons comment modéliser cette optimisation lexicographique, puis nous présenterons certaines propriétés de ce modèle et enfin, nous montrerons comment s'en servir pour améliorer les performances de l'algorithme fondamental.

3.3. Optimisation lexicographique

Comme décrit à la Section 1.1.2, l'optimisation lexicographique peut être mise en oeuvre avec l'approche en *deux phases* ou bien de manière *directe*.

L'utilisation de bornes supérieures locales est particulièrement adaptée à la méthode directe. En effet, cela permet de diminuer la valeur du poids de l'objectif privilégié dans la fonction objectif.

Proposition 3.1. *Le modèle* $\Pi'(k,u)$ *suivant retourne un point non dominé dans* $z(u_{-k})$.

$$\Pi'(k,u)\begin{cases} \min & y_k + \sum\limits_{\substack{i=1\\ i\neq k}}^{p}(u_i - y_i^I - 1)y_k + y_i \\ s.c. & y \in Y \\ & y_{-k} \prec u_{-k} \end{cases}$$

où y^I *représente le point idéal.*

Démonstration. Puisque u_i est une borne supérieure des valeurs atteintes sur l'objectif f_i, il suffit d'appliquer la Proposition 1.2 avec $U_i = u_i$. □

Comme précisé à la Section 1.1.2, la méthode directe peut conduire à des points faiblement non dominés. Cependant, l'affinement de la pondération de y_k permet, en pratique, d'éviter ce phénomène.

3.4. Propriétés

Observons que les solutions réalisables de $\Pi(k,u)$ n'appartiennent pas forcément à la région de recherche, car la contrainte sur l'objectif k a été relâchée. En d'autres termes, cela signifie que la solution optimale obtenue a potentiellement déjà été énumérée. Ce problème est très fréquent dans les approches utilisant des bornes sur $p-1$ dimensions, qui doivent mettre en œuvre des procédures pour éliminer les duplicatas dans l'ensemble final. Par exemple, lors de la découverte d'un point, l'approche de Kirlik et Sayin (2014) doit le comparer avec tous les points déjà énumérés, avant d'éventuellement l'ajouter - ce qui peut signifier un grand nombre de comparaisons.

Dans le cas de l'approche présentée ici, il est possible d'exploiter plusieurs propriétés des zones de recherche pour réduire énormément, en pratique, le nombre de comparaisons nécessaires pour déterminer si le point généré est effectivement un nouveau point.

La première propriété permet de borner supérieurement la solution optimale y_k^* de $\Pi(k,u)$, si elle existe. Du fait que sa composante k n'est pas contrainte, on pourrait imaginer que $y_k^* > u_k$. Le résultat suivant prouve que cette situation est en réalité impossible.

Proposition 3.2. *Soit* $u \in U(N)$ *et soit* y^* *une solution optimale de* $\Pi(k,u)$ *(s'il est réalisable). Alors,* $y_k^* \leq u_k$.

Démonstration. Si $u_k = M$, alors le résultat est trivial. Sinon, u_k est borné. Puisque $U(N)$ ne contient que des bornes maximales, $N_k(u) \neq \emptyset$. Soit $y \in N_k(u)$. D'après (1.6), nous avons $y_{-k} \prec u_{-k}$ et $y_k = u_k$. Ainsi, y est réalisable pour $\Pi(k,u)$ et nous avons donc $y_k^* \leq y_k = u_k$. □

De la Proposition 3.2 découle le résultat suivant, montrant que si le point retourné par $\Pi(k,u)$ a déjà été énuméré, il est forcément présent dans $N_k(u)$.

Corollaire 3.1. *Soit $u \in U(N)$ et soit y^* une solution optimale de $\Pi(k,u)$ (s'il est réalisable). Alors, $y^* \in S(N)$ si et seulement si $y^* \notin N_k(u)$.*

Démonstration. D'après la Proposition 3.2, nous avons $y^*_k \leq u_k$. Si $y^*_k < u_k$, alors $y^* \in S(N)$ et $y^* \notin N_k(u)$. Dans le cas contraire, si $y^*_k = u_k$, y^* est un point définissant la composante k de u puisque y^* satisfait (1.6). Ainsi, si $y^* \notin N_k(u)$, cela signifie que y^* est un nouveau point non dominé.

À l'opposé, si $y^* \in S(N)$, il est trivial que le point n'a pas été énuméré et donc ne peut appartenir à $N_k(u)$. □

Il est important de noter qu'en réalité, seule une infime fraction des points de N définit chaque zone, ce qui réduit sensiblement le nombre de comparaisons nécessaire pour éliminer les duplicatas.

En pratique, nous vérifions si $y^*_k < u_k$, auquel cas nous avons la garantie que y^* est un nouveau point non dominé, et nous l'ajoutons donc immédiatement à N. C'est uniquement dans le cas où $y^*_k = u_k$ que nous vérifions si ce point définissant la composante k appartient à $N_k(u)$. En résumé, pour tester si le point généré est un nouveau point, au lieu d'effectuer $|N|$ comparaisons, il suffit d'en effectuer au plus $|N_k(u)| + 1$ est très souvent une seule.

Mais ce n'est pas tout. Il est en fait possible de garantir la réalisabilité de $\Pi(k,u)$ sous certaines conditions très peu restrictives. Il est important de noter qu'expérimentalement, un programme réalisable est bien plus rapide à résoudre qu'un problème infaisable. En effet, lors de la découverte de la première solution, certaines branches de l'arbre de résolution sous-jacent peuvent être éliminées car ne contenant que des solutions moins bonnes.

Proposition 3.3. *Soit $u \in U(N)$ tel que $u_k \neq M$. Alors, $\Pi(k,u)$ est forcément réalisable, et il existe un point $y \in N \cap F(\Pi(k,u))$.*

Démonstration. D'après la Proposition 1.3, puisque u est une borne maximale avec $u_k \neq M$, nous savons que $N_k(u)$ est non vide. Ainsi, n'importe quel point $y \in N_k(u)$ satisfait (1.6), et donc en particulier $y_{-k} \prec u_{-k}$, et est donc réalisable pour $\Pi(k,u)$. □

En plus de fournir la garantie que $\Pi(k,u)$ est réalisable dans certains cas, la Proposition 3.3 précise que n'importe quel point définissant la composante k de u peut être utilisé comme solution de départ. Comme évoqué précédemment, on observe expérimentalement que fournir une solution de départ permet de réduire sensiblement les temps de résolution des programmes. À notre connaissance, cette approche est la seule capable de fournir une solution réalisable en temps constant.

Il est également possible d'approximer les coordonnées d'une *borne inférieure locale* à l'aide du résultat suivant.

Proposition 3.4. (Borne inférieure locale) *S'il existe, l'optimum retourné par $\Pi(k,u)$ borne inférieurement la composante k de tous les points réalisables de la zone $z(u)$ ainsi que de chacune des subdivisions qu'elle engendre. Cela s'applique également à toute zone $z(u')$ telle que $u'_{-k} \leqq u_{-k}$.*

Démonstration. Puisque l'optimum de $\Pi(k, u)$ minimise l'objectif k sur l'ensemble des points réalisables dominant u_{-k}, il borne inférieurement tout sous ensemble. □

Cette borne inférieure locale peut être utilisée pour remplacer le point idéal lors de la détermination du poids utilisé dans la méthode de résolution directe de $\Pi(k, u)$ (Proposition 3.1). Cela permet de réduire les coefficients de la fonction objectif et donc de limiter les instabilités numériques qui pourraient en découler.

3.5. Règle de réduction

Nous présentons dans cette section différentes techniques exploitant les résultats successifs des modèles Π afin de prouver la vacuité de certaines zones de recherche sans avoir à les explorer.

Un premier résultat découle de la Proposition 3.4 : si $\Pi(k, u)$ exhibe un nouveau point non dominé, la k^e fille de u est réputée vide et il est inutile de la considérer.

Proposition 3.5. *Supposons $\Pi(k, u)$ réalisable, et soit y^* l'un de ses optima. Supposons que $y^* \prec u$. Alors, $z(u^k) \cap Y = \emptyset$.*

Démonstration. Par construction, $u^k_{-k} = u_{-k}$. Ainsi, $y^*_{-k} \prec u^k_{-k}$, et y^*_k est donc la valeur optimale de $\Pi(k, u^k)$. Or, $y^*_k = u^k_k$. En conséquence, $z(u^k)$ ne contient aucun point réalisable. □

Ce résultat peut être généralisé grâce à la proposition suivante.

Proposition 3.6. *Supposons $\Pi(k, u)$ réalisable, et soit y^* l'un de ses optima. Soit u' une borne telle que :*

$$\begin{cases} u'_{-k} \leq u_{-k} \\ u'_k \leq y^*_k \end{cases} \tag{3.1}$$

Alors, $z(u')$ ne contient aucun point réalisable.

Démonstration. Comme $F(\Pi(k, u')) \subset F(\Pi(k, u))$, y^* est une borne inférieure de toute solution réalisable de $\Pi(k, u')$. Puisque $u'_k \leq y^*_k$, $z(u')$ ne contient aucun point réalisable. □

Observons que la Proposition 3.5 est un cas particulier de la Proposition 3.6, en posant $u' = u^k$. Par construction, la borne u^k est générée *après* la borne u (autrement dit, si $u \in U(N)$, $u^k \notin U(N)$). De plus, le second membre de la propriété est satisfait à l'égalité, car $u^k_k = y^*_k$. La propriété suivante montre que ces deux caractéristiques sont vraies pour toute borne u' satisfaisant (3.1).

Proposition 3.7. *Soit $u \in U(N)$ et soit y^*_k la valeur optimale de $\Pi(k, u)$. Toute borne maximale u' satisfaisant (3.1) est telle que $u' \notin U(N)$ et $u'_k = y^*_k$.*

Démonstration. D'après (3.1) et la Proposition 3.2, nous avons $u'_k \leq y^*_k \leq u_k$. Puisque $u'_{-k} \prec u_{-k}$, nous avons $u' \leqq u$. Ainsi, puisque $U(N)$ ne contient que des bornes maximales, si $u \in U(N)$, $u' \notin U(N)$, ce qui prouve la première partie du résultat.

Satisfaire $u'_k \leq y^*_k$ implique que $u'_k \neq M$. Puisque u' est une borne maximale, d'après la Proposition 1.3, nous avons $N_k(u') \neq \emptyset$. Soit $y \in N_k(u')$ un point définissant la composante k de u'. D'après (1.6), nous avons $y \in F(\Pi(k, u'))$ et $y_k = u'_k$. Puisque $F(\Pi(k, u')) \subset F(\Pi(k, u))$, nous avons $y^*_k \leq y_k \leq u'_k$. Puisque y^* satisfait (3.1), nous avons $u'_k = y^*_k$, ce qui conclut la preuve. □

Si appliquer la Proposition 3.5 est trivial (il suffit de ne pas considérer la k^e fille de la borne u), appliquer les Propositions 3.6 et 3.7 nécessite de conserver l'historique de tous les modèles résolus. À cet effet, nous maintenons p arbres binaires de recherche $T_k,\ k \in [\![1, p]\!]$. L'arbre T_k contient la liste des modèles $\Pi(k, u)$ résolus triée selon la valeur de leur optimum. Lors de la création d'une nouvelle zone u', pour chacune de ses composantes bornées u'_k, nous cherchons dans T_k la liste des modèles $\Pi(k, u)$ tels que leur optima soit égal à u'_k. Cette recherche peut se faire en $O(log(|T_k|))$. Si l'une des bornes u satisfait (3.1), alors la borne u' est réputée vide et peut donc être retirée. Notons que l'ensemble des modèles de T_k ayant u'_k pour valeur optimale est significativement plus petit que T_k en pratique.

Il est également important de noter que la situation proposée par la Proposition 3.6 n'apparaît que quand plusieurs points partagent la même valeur sur un objectif. En effet, d'après la Proposition 3.7, u' doit apparaître *après* u. En vertu de cette observation, les règles de réduction sont inutiles dans le cas biobjectif, où cette situation est impossible.

3.6. Sélection d'une zone de recherche

Remarquons que le critère privilégié par le modèle $\Pi(k, u)$ n'est pas nécessairement constant tout au long de l'algorithme, mais qu'il peut être choisi dynamiquement en fonction de la zone de recherche sélectionnée. Bien qu'il ne s'agisse que d'une simple heuristique, nous aimerions qu'elle satisfasse deux caractéristiques. Tout d'abord, il faut que la composante sélectionnée soit bornée (excepté à la première itération où c'est évidemment impossible), afin de garantir que le programme Π soit réalisable et de pouvoir fournir une solution de départ (Proposition 3.3). Deuxièmement, nous aimerions augmenter les chances d'activer les règles de réduction décrites dans les Propositions 3.6 et 3.7. En d'autres termes, si u et u' sont dans la région de recherche et que $u'_{-k} \preceq u_{-k}$, nous aimerions que l'heuristique privilégie la résolution de $\Pi(k, u)$ à celle de $\Pi(k', u')$.

Sur la base de ces observations, nous avons choisi d'utiliser l'indicateur suivant, pouvant être interprété comme la mesure du volume du rectangle à $p - 1$ dimensions formée par u_{-k} et y^I_{-k}, qui est le point idéal.

$$h(k, u) = \prod_{\substack{i=1 \\ i \neq k}}^{i=p} (u_i - y^I_i) \tag{3.2}$$

avec y^I le point idéal.

Il suffit alors de choisir la zone u^* et la direction k^* maximisant ce volume :

$$(k^*, u^*) = \begin{cases} (1, (M, \dots, M)) & \text{si } N = \emptyset \\ \underset{\substack{u \in U(N) \\ k \in [\![1,p]\!] \\ u_k \neq M}}{\operatorname{argmax}} \{h(k,u)\} & \text{sinon} \end{cases} \tag{3.3}$$

Il est important de noter que si $N \neq \emptyset$, toute borne de $U(N)$ possède une composante bornée, ce qui permet de déterminer k^* et u^* à chaque itération. De plus, puisque la composante omise est bornée, cela permet d'appliquer la Proposition 3.3 et de fournir en temps constant une solution de départ à $\Pi(k^*, u^*)$.

Proposition 3.8. *Soient u et u′ deux bornes de U(N). Supposons qu'il existe un critère k tel que* $u'_{-k} \preceq u_{-k}$. *Alors,*

$$h(k, u') < h(k, u)$$

Démonstration. Si $u'_{-k} \preceq u_{-k}$, nous avons par définition $u'_i \leq u_i$, $\forall i \in [\![1, p]\!]$ sachant que l'une de ces inégalités est stricte. Ainsi, $u'_i - y^I_i \leq u_i - y^I_i$, $\forall i \in [\![1, p]\!]$ et $u'_i - y^I_i < u_i - y^I_i$ pour un certain i. Le résultat suit. □

Ainsi, la projection de la zone qui maximise le volume n'est incluse dans aucune autre et il semble raisonnable de penser que l'explorer en priorité permet d'augmenter l'impact de la règle de réduction.

L'algorithme 3.2 synthétise les différentes étapes de l'approche. Nous présentons dans la suite quelques éléments expérimentaux.

3.7. Comparaison avec les méthodes existantes.

Nous présentons dans cette section une analyse des performances expérimentales de différentes variantes présentées dans ce chapitre. Les tests ont été effectués sur un serveur muni d'un processeur Xeon E312xx cadencé à 2.4 GHz et de 7.8 GB de mémoire vive. L'algorithme est implémenté en *Haskell (2010)* en utilisant la bibliothèque *IBM Ilog Cplex*(tm) 12.7 et plus particulièrement la technologie *Concert*(C++) pour la résolution des modèles linéaires en nombres entiers.

3.7.1. Instances

Sac à dos multicritère

Le problème du sac à dos consiste à choisir un sous ensemble d'objets parmi un ensemble en contenant n. Chaque objet i comporte p valeurs v^k_i, $k \in [\![1, p]\!]$ et un poids ω_i, $i \in [\![1, n]\!]$. Le but est de maximiser la valeur du sac (qui est la somme des valeurs des objets choisis) sous contrainte que le poids total ne dépasse pas le poids maximal supporté par le sac, noté b. Le

Algorithme 3.2 : Énumération des points non dominés.

Input : Ensemble des points réalisables :Y
Output : Ensemble des points non dominés de Y : Y_{ND}

$N \leftarrow \emptyset$
/* Initialisation de la liste des bornes et de l'archive. */
$U(N) \leftarrow \{(M, \ldots, M)\}$
$T \leftarrow \emptyset$
while $U(N) \neq \emptyset$ **do**
  /* On choisit la projection d'une zone maximisant le volume (3.2) */
  choisir (k^*, u^*)
  $y^* \leftarrow$ résoudre $\Pi(k^*, u^*)$
  /* Le modèle est ajouté à l'archive */
  $T \leftarrow T \cup \{(k^*, u^*, y^*_k)\}$
  if $y^* \notin N_{k^*}(u^*)$ **then**
    /* y^* est un nouveau point non dominé. La région de recherche est mise à jour selon l'algorithme 1.3 */
    $U(N \cup \{y^*\}) \leftarrow$ updateSearchRegion $(U(N), y^*)$
    $N \leftarrow N \cup \{y^*\}$
  /* On applique les règles de réductions sur la région de recherche */
  foreach $u' \in U(N)$ **do**
    foreach $(k^i, u^i, y^{*i}) \in T$ **do**
      if $u'_{-k^i} \leqq u^i_{-k^i}$ *and* $y^{*i}_k \geq u'_k$ **then**
        $U(N) \leftarrow U(N) \setminus \{u'\}$
$Y_{ND} \leftarrow N$

modèle suivant contraint les variables $x_i \in \{0,1\},\ i \in [\![1,n]\!]$ pour qu'elles indiquent si l'objet i est choisi ou non :

$$(MOKP)\begin{cases} \text{"max"}: & \sum_{i=1}^{n} v_i^k x_i & k \in [\![1,p]\!] \\ \text{s.c.} & \sum_{i=1}^{n} \omega_i x_i \leq b & \\ & x_i \in \{0,1\} & \forall i \in [\![1,n]\!] \end{cases}$$

Dans sa version monocritère, ce problème est *faiblement NP difficile*. Afin d'accroître sa complexité, nous fixons b comme étant la demi somme de tous les poids considérés.

$$b = \frac{\sum_{i=1}^{n} \omega_i}{2}$$

Toutes les valeurs et tous les poids ont été générés aléatoirement dans l'intervalle $[\![1,100]\!]$.

Affectation multicritère

Considérant un ensemble I de n agents et un ensemble J de n tâches, nous disposons d'un graphe biparti où les arêtes reliant chaque agent $i \in I$ à chaque tâche $j \in J$ est valuée par un coût c_{ij}. Le but est d'associer chaque tâche à un et un seul agent de façon à minimiser le coût global de l'affectation, c'est à dire la somme des coûts individuels.

Dans sa version multicritère, les arêtes comportent p valeurs au lieu d'une seule. Le modèle suivant contraint les variables $x_{ij} \in \{0,1\},\ (i,j) \in I \times J$ pour qu'elles indiquent si l'agent i est affecté à la tâche j ou non :

$$(MOAP)\begin{cases} \text{"min"}: & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} & k \in [\![1,p]\!] \\ \text{s.c.} & \sum_{i=1}^{n} x_{ij} = 1 & \forall i \in [\![1,n]\!] \\ & \sum_{j=1}^{n} x_{ij} = 1 & \forall j \in [\![1,n]\!] \\ & x_{ij} \in \{0,1\} & \forall (i,j) \in [\![1,n]\!]^2 \end{cases}$$

Chaque coût a été généré aléatoirement dans l'intervalle $[\![1,20]\!]$.

3.7.2. Analyse des résultats

Les comparaisons expérimentales ont été effectuées sur la base du nombre de points non dominés retournés ($|Y_{ND}|$), du temps CPU en secondes (cpu), du nombre d'itérations (#Iter), du nombre de programmes infaisables résolus (#Infeasible), et des tailles maximales et moyennes de la région de recherche ($|U(N)|_{max}$ et $|U(N)|_{avg}$). Pour chaque type d'instance, les moyennes sont calculées

p	n	name	$\lvert Y_{ND}\rvert$ mean	cpu mean	#Iter mean	#Infeasible mean	$\lvert U(N)\rvert_{max}$ mean	$\lvert U(N)\rvert_{avg}$ mean
3	150	two-stage-dynamic	16 834, 4	2 957, 6	29 874, 8	0, 0	778, 0	386, 9
		direct-dynamic	16 834, 4	2 732, 0	29 443, 8	0, 0	847, 2	405, 7
		two-stage-static1	16 834, 4	3 117, 2	28 552, 9	251, 1	360, 4	259, 3
		direct-static1	16 834, 5	2 757, 9	28 322, 0	251, 1	382, 4	266, 0
4	30	two-stage-dynamic	274, 9	22, 1	1 268, 1	0, 0	480, 2	286, 0
		direct-dynamic	274, 9	20, 7	1 266, 3	0, 0	482, 2	286, 9
		two-stage-static1	274, 9	24, 0	1 252, 5	146, 9	428, 3	246, 3
		direct-static1	274, 9	22, 6	1 252, 1	146, 9	428, 6	246, 7
	50	two-stage-dynamic	2 627, 0	599, 3	12 172, 9	0, 0	3 773, 4	1 990, 9
		direct-dynamic	2 627, 0	477, 7	12 107, 2	0, 0	3 860, 9	2 029, 7
		two-stage-static1	2 627, 0	471, 7	11 969, 1	899, 4	2 827, 1	1 680, 1
		direct-static1	2 627, 0	415, 1	11 936, 2	899, 4	2 864, 6	1 695, 2
5	50	two-stage-dynamic	10 241, 2	7 822, 3	149 453, 8	0, 0	66 749, 1	39 528, 7
		direct-dynamic	10 241, 2	7 399, 1	149 128, 7	0, 0	67 338, 5	39 887, 3
		two-stage-static1	10 241, 2	8 705, 1	148 704, 7	11 250, 1	63 461, 8	36 770, 8
		direct-static1	10 241, 2	8 412, 9	148 519, 7	11 250, 1	63 725, 6	36 920, 3
6	20	two-stage-dynamic	349, 5	213, 2	13 538, 6	0, 0	9 344, 8	5 495, 9
		direct-dynamic	349, 5	222, 9	13 534, 2	0, 0	9 357, 9	5 508, 4
		two-stage-static1	349, 5	242, 7	13 520, 1	1 753, 3	9 155, 1	5 422, 6
		direct-static1	349, 5	251, 0	13 518, 3	1 753, 3	9 156, 5	5 426, 0
	30	two-stage-dynamic	2 166, 4	2 883, 3	81 491, 6	0, 0	50 827, 1	30 576, 8
		direct-dynamic	2 166, 4	2 944, 2	81 442, 0	0, 0	50 964, 5	30 654, 0
		two-stage-static1	2 166, 4	3 324, 2	81 353, 6	9 652, 6	47 893, 1	29 285, 1
		direct-static1	2 166, 4	3 529, 7	81 321, 5	9 652, 6	47 939, 7	29 328, 6

TABLE 3.1. – Comparaison entre les approches statiques et dynamiques (sac à dos multicritère).

sur un ensemble de 10 instances. La valeur N/A est renseignée si le temps de resolution d'une instance excède 24 heures ou si la consommation en mémoire excède 7.8 GB de mémoire vive.

Le tableau 3.1 compare différentes variantes de notre algorithme sur deux paramètres. Tout d'abord, la méthode utilisée pour modéliser l'optimisation lexicographique est étudiée : les approches en deux phases (préfixées par *two-stage*) et directes (préfixées par *direct*) sont comparées. Secondement, nous étudions l'impact du choix du critère optimisé : la variante *static1* optimise systématiquement l'objectif f_1 et la variante *dynamic* optimise l'objectif k maximisant (3.2).

Nous observons, dans le tableau 3.1, que les approches directes, bien qu'induisant potentiellement une instabilité numérique, ne retournent un point faiblement non dominé qu'une fois : lors d'une instance du sac à dos triobjectif à 150 éléments pour l'approche statique. Cela s'explique par l'utilisation de la Proposition 3.1 rendant le plus grand coefficient de la fonction objectif relativement faible en pratique. Comme nous nous y attendions, les approches directes semblent plus rapides que les approches en deux phases.

Bien que déterminant l'ensemble final en moins d'itérations, les approches statiques sont majoritairement plus lentes que les approches dynamiques à l'exception de quelques instances MOKP à 4 objectif et 50 éléments. Cela est lié au fait que des programmes infaisables, très couteux à

p	n	name	$\lvert Y_{ND}\rvert$ mean	cpu mean	#Iter mean	#Infeasible mean	$\lvert U(N)\rvert_{Max}$ mean	$\lvert U(N)\rvert_{Avg}$ mean
2	200	two-stage-dynamic	378,5	16,7	379,5	0,0	1,0	1,0
		dynamic-direct	378,5	12,4	379,5	0,0	1,0	1,0
		KS2014	378,5	17,3	378,5	0,0	2,0	2,0
		BCS2017	378,5	54,2	673,6	295,1		
3	150	two-stage-dynamic	16 834,4	2 957,6	29 874,8	0,0	778,0	386,9
		dynamic-direct	16 834,4	2 732,0	29 443,8	0,0	847,2	405,7
		KS2014	16 834,4	4 313,6	28 665,7	251,1	309 237,8	281 953,6
		BCS2017	16 834,4	16 246,1	37 669,4	20 835,0		
4	30	two-stage-dynamic	274,9	22,1	1 268,1	0,0	480,2	286,0
		dynamic-direct	274,9	20,7	1 266,3	0,0	482,2	286,9
		KS2014	274,9	99,2	1 269,3	146,9	803 691,3	669 430,9
		BCS2017	274,9	43,5	1 008,4	733,5		
	50	two-stage-dynamic	2 627,0	599,3	12 172,9	0,0	3 773,4	1 990,9
		dynamic-direct	2 627,0	477,7	12 107,2	0,0	3 860,9	2 029,7
		KS2014	2 627,0	11 768,4	12 248,8	899,3	21 113 829,8	19 521 170,0
		BCS2017	2 627,0	1 588,5	11 376,0	8 749,0		
5	50	two-stage-dynamic	10 241,2	7 822,3	149 453,8	0,0	66 749,1	39 528,7
		dynamic-direct	10 241,2	7 399,1	149 128,7	0,0	67 338,5	39 887,3
		KS2014	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	*N/A*	*N/A*	*N/A*	*N/A*		
6	20	two-stage-dynamic	349,5	213,2	13 538,6	0,0	9 344,8	5 495,9
		dynamic-direct	349,5	222,9	13 534,2	0,0	9 357,9	5 508,4
		KS2014	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	349,5	1 216,6	10 339,1	9 989,6		
	30	two-stage-dynamic	2 166,4	2 883,3	81 491,6	0,0	50 827,1	30 576,8
		dynamic-direct	2 166,4	2 944,2	81 442,0	0,0	50 964,5	30 654,0
		KS2014	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	*N/A*	*N/A*	*N/A*	*N/A*		

TABLE 3.2. – Comparaison entre les approches dynamiques et les méthodes récentes de la littérature (sac à dos multicritère).

résoudre, sont considérés par les approches statiques. À l'opposé, les approches dynamiques ne considèrent que des programmes réalisables et sont même en mesure de leur fournir une solution de départ ce qui réduit sensiblement le coût des itérations. Sur la base de ces observations, nous concluons que l'approche dynamique surclasse l'approche statique, qui ne sera donc plus traitée dans la suite de ce document.

Les tableaux 3.2 et 3.3 synthétisent les comparaisons expérimentales entre nos approches et celles de Kirlik et Sayin (2014) (KS2014) et de Boland et al. (2017) (BCS2017) sur des instances de MOKP et de MOAP respectivement. Les implémentations de ces méthodes ont été fournies par leurs auteurs respectifs. Pour l'approche KS2014, les colonnes $|U(N)|_{max}$ et $|U(N)|_{avg}$ représentent le nombre maximum et moyen de rectangles définissant la région de recherche au long des itérations. En revanche, puisque BCS2017 fait intervenir des objets plus complexes, nous omettons ces deux champs pour cette approche. Les expérimentations montrent que notre algorithme surclasse les deux autres. Ces derniers sont clairement plus lents, ce qui est probablement

p	$n \times n$	name	$\lvert Y_{ND}\rvert$ mean	cpu mean	#Iter mean	#Infeasible mean	$\lvert U(N)\rvert_{Max}$ mean	$\lvert U(N)\rvert_{Avg}$ mean
2	50×50	two-stage-dynamic	140,8	19,3	141,8	0,0	1,0	1,0
		direct-dynamic	140,8	13,8	141,8	0,0	1,0	1,0
		KS2014	140,8	18,1	140,8	0,0	2,0	2,0
		BCS2017	140,8	63,0	266,2	125,4		
	80×80	two-stage-dynamic	219,4	81,3	220,4	0,0	1,0	1,0
		direct-dynamic	219,4	67,6	220,4	0,0	1,0	1,0
		KS2014	219,4	64,2	219,4	0,0	2,0	2,0
		BCS2017	219,4	233,2	419,2	199,8		
3	30×30	two-stage-dynamic	6 181,3	1 209,3	9 744,6	0,0	248,8	174,1
		direct-dynamic	6 181,3	992,8	9 381,1	0,0	304,3	198,1
		KS2014	6 181,3	1 370,0	8 380,4	92,9	21 342,2	20 160,9
		BCS2017	6 181,3	2 508,1	11 942,4	5 761,1		
	40×40	two-stage-dynamic	14 679,7	4 337,0	22 463,6	0,0	357,7	259,8
		direct-dynamic	14 679,7	3 677,4	21 422,7	0,0	469,9	301,2
		KS2014	14 679,7	4 691,0	19 240,4	113,8	43 961,3	42 445,4
		BCS2017	14 679,7	11 307,7	27 924,1	13 244,4		
	50×50	two-stage-dynamic	24 916,8	11 074,2	37 943,9	0,0	468,5	337,4
		direct-dynamic	24 916,9	9 855,8	35 990,6	0,0	619,0	396,7
		KS2014	24 916,8	11 553,3	31 549,7	142,6	70 819,9	69 053,4
		BCS2017	24 916,8	32 835,7	47 045,0	22 128,2		
4	20×20	two-stage-dynamic	46 139,9	28 090,4	227 711,7	0,0	23 878,9	14 784,4
		direct-dynamic	46 139,9	24 514,8	226 038,3	0,0	24 695,9	15 178,2
		KS2014	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*

TABLE 3.3. – Comparaison entre les approches dynamiques et les méthodes récentes de la littérature (affectation multicritère).

dû au nombre de programmes infaisables qu'ils considèrent.

Bien que nécessitant légèrement moins d'itérations que notre approche pour calculer l'ensemble final, KS2014 est clairement plus lent à cause de l'énorme quantité de rectangles considérés. Par exemple, pour le MOKP à 4 objectifs et 50 éléments, leur région de recherche peut contenir jusqu'à 20 millions de rectangles quand seulement 4000 zones sont requises par notre approche. En conséquence, la procédure de mise à jour de la région de recherche de KS2014 peut rapidement devenir coûteuse, voire même saturer la mémoire. En pratique, KS2014 est incapable de résoudre des instances faisant intervenir plus de 5 objectifs, et ce même quand la taille de l'ensemble final est faible (par exemple, nos instances MOKP avec 6 objectifs et 30 éléments ne contiennent que 350 points non dominés en moyenne). En conclusion, KS2014 semble meilleur que BCS2017 pour les instances ayant au plus 3 objectifs, mais se fait surclasser à cause de sa consommation en mémoire.

À l'opposé, BCS2017 semble plus stable que KS2014 au sens où sa consommation en mémoire n'est pas démesurée, même quand le nombre d'objectifs augmente. Cependant, la détermination de l'ensemble final nécessite un temps de calcul relativement long (par exemple MOKP et MOAP biobjectif et triobjectif). Cela peut s'expliquer par le nombre de programmes infaisables résolus ainsi que de leur complexité : ils font en effet intervenir des contraintes disjonctives. En conclusion, BCS2017 et KS2014 échouent à résoudre les instances avec un grand nombre d'objectifs, mais pour différentes raisons : le temps de calcul pour le premier algorithme, et la consommation en mémoire pour le second.

Bien que tous ces algorithmes aient été créés pour résoudre des problèmes avec un nombre quelconque d'objectifs, nous nous sommes néanmoins intéressés à leur comportement dans le cas biobjectif. Notre approche partage un grand nombre de caractéristiques avec la méthode ε-contrainte habituelle : $|Y_{ND}| + 1$ itérations sont nécessaires pour déterminer $|Y_{ND}|$, et la région de recherche ne contient qu'une seule zone de recherche à chaque itération. La seule différence réside dans le choix de l'objectif optimisé : dans la méthode ε-contrainte, ce dernier reste constant tandis qu'il peut varier dans le cadre de notre approche. À partir de ces observations, nous pouvons assimiler notre approche à une généralisation de la méthode ε-contrainte.

3.8. Conclusion

Dans ce chapitre, nous avons proposé une nouvelle approche permettant de générer l'ensemble des points non dominés d'instances comportant un grand nombre d'objectifs. Cette approche peut être qualifiée de *hybride* : bien que reposant sur une caractérisation explicite de la région de recherche sous la forme d'une union de zones à p dimensions, l'exploration ne fait intervenir que $p - 1$ contraintes. Il est ainsi à la fois possible de représenter des régions de recherche de grande taille et d'explorer plusieurs zones avec un seul programme mathématique. Cette hybridation permet également de pouvoir choisir, en fonction de la zone considérée, les objectifs qui seront contraints. Sur la base de cette observation, nous avons montré qu'en tirant parti de certaines propriétés il est possible de fournir en temps constant une solution réalisable à tous les programmes résolus par l'algorithme (à l'exception du premier).

Il est intéressant d'observer que cette approche, bien que s'appuyant sur la programmation mathématique, est indépendante du solveur considéré. Il est seulement nécessaire de pouvoir optimiser une fonction linéaire sous des contraintes de budget. Si des algorithmes spécifiques à un certain type de problème satisfont cette propriété, il est très facile d'adapter notre approche pour les utiliser.

Enfin, nous nous sommes restreints dans ces travaux à la programmation mathématique en nombres entiers. Cependant, il pourrait être très intéressant d'essayer de l'adapter à la programmation continue, voire mixte. Étudier la généralisation de la notion de *zone de recherche* à ces situations pourrait certainement contribuer à ces domaines.

Deuxième partie

Résolution de problématiques associées

Chapitre 4.

Représentation de l'ensemble des points non dominés

Résumé

Au lieu d'énumérer tous les points non dominés, nous nous proposons, dans ce chapitre, d'en déterminer une ε-approximation. Une solution consiste à utiliser une extension de la relation de dominance appelée ε-dominance. Après avoir présenté cette relation, nous montrons qu'il suffit, lors de la découverte d'un nouveau point, de translater ce dernier lors de l'opération de mise à jour de la région de recherche. Ainsi, non seulement les points qu'il domine sont éliminés de la région de recherche, mais également les points non dominés relativement similaires. Par conséquent, il est possible de réduire grandement le nombre de points générés tout en garantissant, *a priori*, une qualité d'approximation. Les temps d'exécution s'en trouvent considérablement réduits comme le montrent les expérimentations que nous avons menées sur des instances du sac à dos multiobjectif.

4.1. Introduction

Comme évoqué précédemment dans ce document, générer l'ensemble des points non dominés peut se révéler très coûteux : chaque point est obtenu par la résolution d'un programme en nombres entiers, et le nombre de points peut être démesurément grand. De plus, l'intérêt de fournir à un décideur un grand ensemble de points, potentiellement très similaires, est parfois discutable.

De ces deux problèmes apparaît l'intérêt d'enrichir la relation de dominance afin d'obtenir une représentation plus concise de l'ensemble des points non dominés. Différentes propriétés intéressantes des représentations envisageables ont été dégagées, comme par exemple dans Sayın (2000) et dans Bazgan et al. (2017), mais certaines d'entre elles peuvent être conflictuelles ou requérir certaines conditions.

En l'absence d'information préalable sur les points désirés par le décideur, il est possible d'employer la relation de ε-dominance où, compte tenu d'une tolérance ε, un point ε-domine une région plus large que celle qu'il domine réellement. Cette approche permet d'obtenir une représentation de points bien répartis dans l'espace des objectifs, car les points non dominés ayant des performances similaires auront le même représentant.

Nous présentons dans ce document une adaptation simple de l'algorithme 3.2 à la relation de ε-dominance et nous comparerons les ensembles obtenus sur des instances du problème du sac à dos multiobjectif.

4.2. Résultats préliminaires

Nous introduisons tout d'abord la relation de ε-dominance entre deux points, notée $\preceq_\varepsilon$.

Définition 4.1. *Soient $\varepsilon > 0$ et deux points positifs $(y, y') \in (\mathbb{R}^+)^p \times (\mathbb{R}^+)^p$. La relation $\preceq_\varepsilon$ est définie par :*

$$y \preceq_\varepsilon y' \iff y_i \leq (1+\varepsilon)y'_i,\ \forall i \in [\![1, p]\!]$$

En d'autres termes, cela signifie que le cône de dominance de y est translaté, comme illustré à la figure 4.1.

Nous nous intéressons dans cette section à la génération d'une ε-approximation de Y, notée Y^ε_{ND}, vérifiant la propriété de couverture suivante :

$$\forall y \in Y,\ \exists y' \in Y^\varepsilon_{ND} : y' \preceq_\varepsilon y$$

Comme montré par Lacour (2014) à la section 4.3, adapter l'algorithme 3.2 est trivial : lors de la découverte du nouveau point non dominé y, il suffit de mettre à jour la région de recherche avec le point translaté $\hat{y} = \frac{y}{1+\varepsilon}$, permettant ainsi d'en retirer les points qui sont ε-dominés par y. Ainsi, quand la région de recherche $U(N)$ est vide, cela signifie que tous les points de Y sont ε-dominés par au moins un point de N. De plus, nous apportons la garantie que tous les points de l'ensemble final sont non dominés.

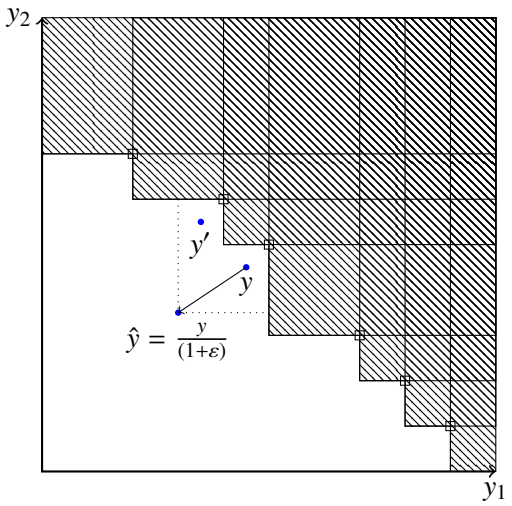


FIGURE 4.1. – Le point translaté $\hat{y}$ domine une région plus grande que celle du point y, dont le point non dominé y'.

Proposition 4.1. *Pour toute ε-approximation Y^{ε}_{ND}, on a $Y^{\varepsilon}_{ND} \subset Y_{ND}$*

Démonstration. Tous les points ajoutés à Y^{ε}_{ND} sont optimaux pour un programme Π. □

Il est intéressant de remarquer que la Proposition 1.3 qui établit le lien entre maximalité d'une zone et les points définissant ses composantes bornées est toujours vraie, à l'exception que ces derniers sont désormais les points translatés. Dès lors, il est impossible de garantir que tous les programmes sont réalisables et, *a fortiori* de fournir une solution de départ.

Nous allons voir, dans la section suivante, que cette adaptation simple de l'algorithme 3.2 permet d'obtenir une représentation concise et bien répartie des points non dominés.

4.3. Résultats expérimentaux

Le tableau 4.1 synthétise les résultats expérimentaux sur les instances du sac à dos que nous avions étudiées au Chapitre 3. Des approximations selon différentes valeurs de ε y sont considérées : $\varepsilon = 0.01$ (*approx-1*), $\varepsilon = 0.02$ (*approx-2*), $\varepsilon = 0.05$ (*approx-5*) et $\varepsilon = 0.1$ (*approx-10*). À titre informatif, la variante exacte est également spécifiée (*approx-0*). Pour chacune d'elles, outre les statistiques habituelles, nous déterminons la valeur de ε que nous calculons *a posteriori* ainsi que le pourcentage de points non dominés présents dans l'ensemble final ($\%|Y_{ND}|$).

La véritable valeur de ε, déterminée *a posteriori*, peut être déterminée car nous avons connaissance de l'ensemble total des points non dominés. Pour chaque point $y \in Y_{ND}$, nous déterminons son *meilleur* représentant dans l'ensemble Y^{ε}_{ND}. Pour déterminer la qualité d'un point $y' \in Y^{\varepsilon}_{ND}$

p	n	name	$\lvert Y_{ND}^{\varepsilon}\rvert$ mean	cpu mean	#Iter mean	#Infeasible mean	$\lvert U(N)\rvert_{max}$ mean	$\lvert U(N)\rvert_{avg}$ mean	ε mean	$\%\lvert Y_{ND}\rvert$ mean
3	150	approx-0	16 834, 4	2 957, 6	29 874, 8	0, 0	778, 0	386, 9	0, 0	100, 0
		approx-1	397, 2	37, 5	982, 1	14, 1	71, 5	47, 1	0, 9	2, 4
		approx-2	128, 8	11, 1	290, 1	11, 7	34, 6	21, 7	1, 9	0, 8
		approx-5	29, 5	2, 2	56, 7	7, 2	14, 1	8, 0	4, 6	0, 2
		approx-10	10, 1	0, 6	16, 0	1, 4	5, 7	3, 4	8, 4	0, 06
4	30	approx-0	274, 9	22, 1	1 268, 1	0, 0	480, 2	286, 0	0, 0	100, 0
		approx-1	121, 3	7, 6	630, 6	0, 7	227, 6	132, 7	0, 9	45, 6
		approx-2	64, 3	3, 1	285, 0	2, 4	113, 1	67, 2	1, 8	25, 2
		approx-5	22, 4	0, 7	67, 2	1, 8	34, 9	19, 6	4, 6	8, 6
		approx-10	8, 4	0, 2	17, 7	1, 0	11, 2	6, 0	8, 8	3, 4
	50	approx-0	2 627, 0	599, 3	12 172, 9	0, 0	3 773, 4	1 990, 9	0, 0	100, 0
		approx-1	483, 4	52, 5	2 628, 0	1, 8	977, 6	519, 8	0, 9	18, 8
		approx-2	188, 4	15, 9	865, 6	4, 1	340, 8	188, 7	1, 9	7, 6
		approx-5	44, 5	2, 4	137, 8	3, 5	65, 6	37, 6	4, 7	1, 8
		approx-10	14, 8	0, 5	32, 7	0, 9	21, 7	11, 5	9, 1	0, 6
5	50	approx-0	10 241, 2	7 822, 3	149 453, 8	0, 0	66 749, 1	39 528, 7	0, 0	100, 0
		approx-1	1 341, 3	755, 5	27 196, 4	63, 6	14 326, 6	8 490, 5	0, 9	13, 7
		approx-2	418, 0	144, 9	5 554, 1	51, 1	3 129, 0	1 902, 7	1, 9	4, 4
		approx-5	74, 0	11, 4	481, 5	12, 0	346, 1	193, 0	4, 8	0, 8
		approx-10	19, 8	1, 5	83, 9	3, 5	66, 0	34, 6	9, 7	0, 2
6	20	approx-0	349, 5	213, 2	13 538, 6	0, 0	9 344, 8	5 495, 9	0, 0	100, 0
		approx-1	247, 6	222, 2	15 121, 3	81, 4	10 203, 7	6 047, 9	0, 9	71, 0
		approx-2	166, 9	101, 9	7 542, 2	117, 8	5 065, 7	3 067, 1	1, 9	48, 5
		approx-5	57, 3	9, 2	921, 3	51, 9	828, 9	458, 9	4, 8	18, 6
		approx-10	21, 4	1, 3	159, 7	15, 5	198, 4	99, 2	9, 0	7, 9
	30	approx-0	2 166, 4	2 883, 3	81 491, 6	0, 0	50 827, 1	30 576, 8	0, 0	100, 0
		approx-1	784, 4	1 235, 7	48 297, 1	358, 9	30 668, 4	18 468, 1	0, 9	39, 6
		approx-2	329, 6	232, 0	11 847, 5	273, 0	8 139, 7	4 974, 7	1, 9	18, 1
		approx-5	71, 3	15, 6	1 030, 4	64, 6	895, 0	529, 0	4, 8	4, 3
		approx-10	18, 9	1, 6	123, 5	8, 8	120, 9	60, 5	9, 4	1, 2

TABLE 4.1. – Différentes approximations de Y_{ND} sur des problèmes MOKP

à représenter un point $y \in Y_{ND}$, nous calculons la valeur suivante :

$$\hat{\varepsilon}(y', y) = \max_{i \in [\![1,p]\!]} \{\frac{y'_i}{y_i} - 1\}$$

Le meilleur représentant de y est donc celui qui minimise cette métrique, c'est à dire :

$$\hat{\varepsilon}(y) = \min_{y' \in Y^{\varepsilon}_{ND}} \{\hat{\varepsilon}(y', y)\}$$

Pour évaluer la qualité de la représentation Y^{ε}_{ND}, nous calculons la pire valeur de $\hat{\varepsilon}$:

$$\hat{\varepsilon}(Y_{ND}, Y^{\varepsilon}_{ND}) = \max_{y \in Y_{ND}} \{\hat{\varepsilon}(y)\}$$

Cette valeur doit toujours inférieure à celle de la tolérance choisie, en d'autres termes on doit avoir $\hat{\varepsilon}(Y_{ND}, Y^{\varepsilon}_{ND}) \leq \varepsilon$, ce que confirment nos expérimentations.

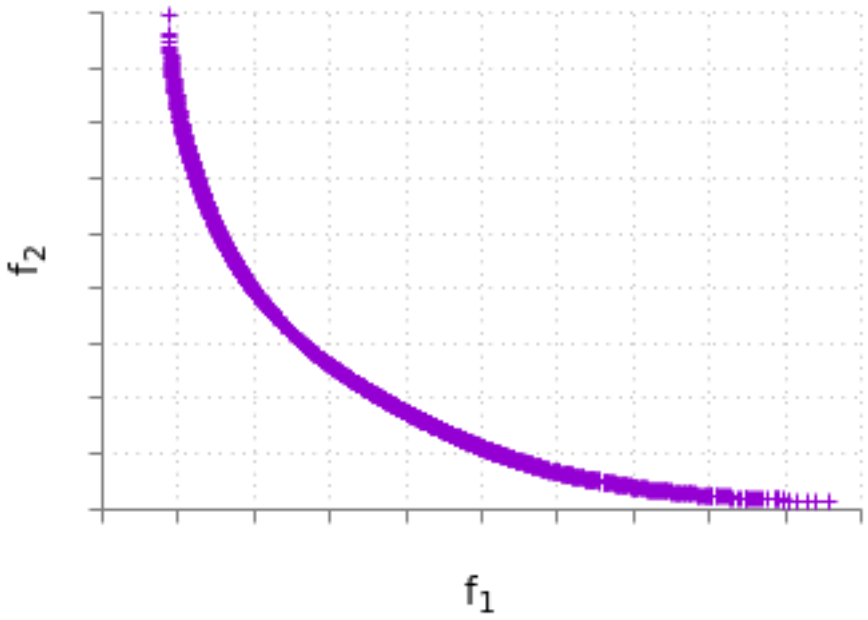
Nous observons un gain significatif de temps même pour des tolérances faibles (de l'ordre de 1%). Cela est principalement lié à la représentation de l'ensemble final, dont la cardinalité est très faible. Ainsi, très peu d'itérations sont nécessaires.

La Figure 4.1 illustre l'impact du choix de ε sur la représentation Y^{ε}_{ND} de l'ensemble des points non dominés dans le cas d'une instance du sac à dos biobjectif à 500 éléments. Nous observons que la taille de l'ensemble final diminue significativement et offre une représentation très concise de Y_{ND}, mais également bien répartie. Il est intéressant d'observer que la densité de points diminue fortement au centre de la représentation : en effet, dans cette région, une faible dégradation sur un objectif induit une faible amélioration sur l'autre et les points adjacents sont donc fortement similaires. À l'opposé, la densité augmente quand nous nous rapprochons des optima lexicographiques. Cela s'explique par le fait que, dans cette région, une légère dégradation sur un critère induit une très forte amélioration sur l'autre. Bien que cette dernière observation puisse être regrettable dans certaines situations où nous n'aurions aucun intérêt à conserver un point légèrement meilleur sur un critère et sensiblement plus mauvais sur l'autre, il est également possible qu'il soit pertinent d'examiner tous ces compromis.

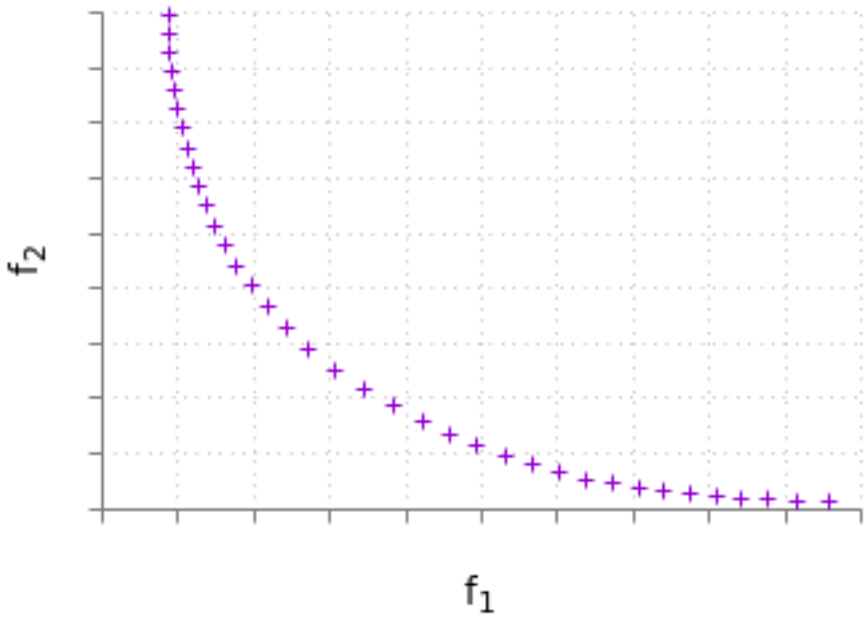
4.4. Conclusion et perspectives

Nous avons montré dans cette section qu'il était très simple d'adapter l'algorithme 3.2 pour générer une représentation de l'ensemble des points non dominés. En s'appuyant sur la relation de ε-dominance, il est possible d'obtenir, selon la tolérance souhaitée, un ensemble de points bien réparti.

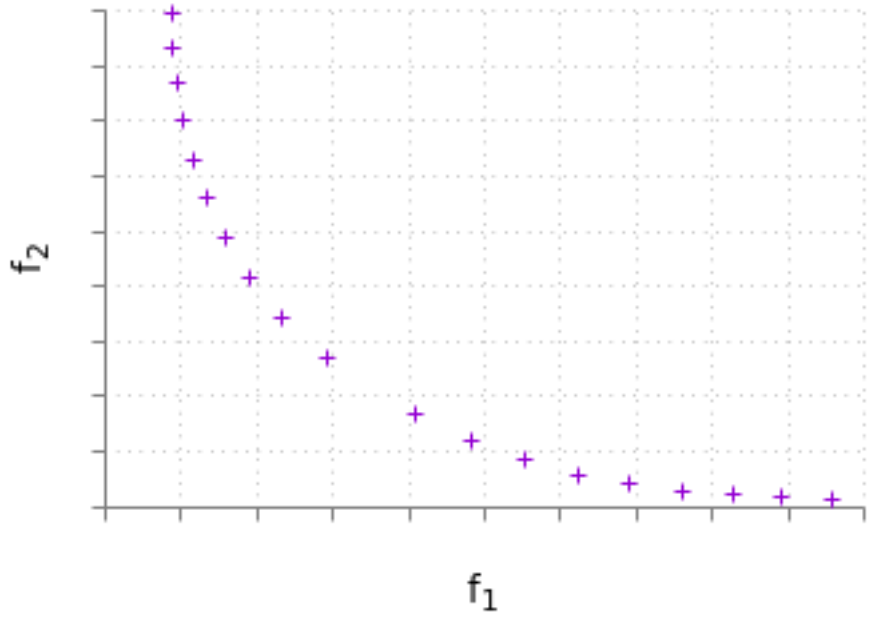
Plusieurs problématiques similaires mériteraient d'être approfondies. La première consisterait à modifier l'algorithme pour obtenir certaines propriétés sur la représentation. L'une d'elles consiste à déterminer un ensemble où les point ne se ε-dominent pas mutuellement. Cela est sensiblement plus complexe, car l'existence d'un tel ensemble n'est pas forcément garantie (Bazgan



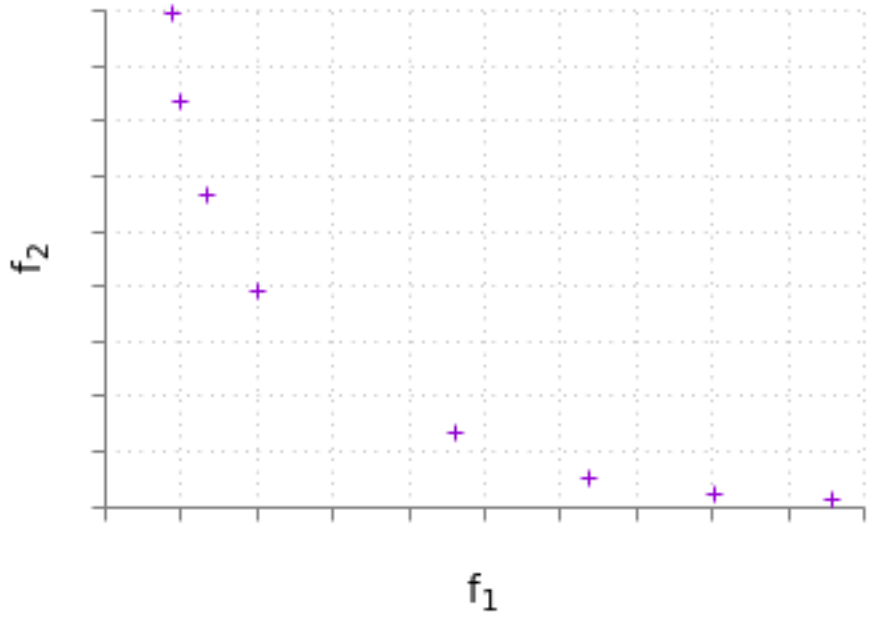
(a) Ensemble des points non dominés. $|Y_{ND}| = 1281$



(b) Approximation avec $\varepsilon = 1\%$. $|Y^{\varepsilon}_{ND}| = 37$



(c) Approximation avec $\varepsilon = 2\%$. $|Y_{ND}^{\varepsilon}| = 19$



(d) Approximation avec $\varepsilon = 5\%$. $|Y_{ND}^{\varepsilon}| = 8$

FIGURE 4.1. – Évolution de l'ensemble des points non dominés en fonction de ε (MOKP).

et al., 2017), la relation de ε-dominance n'étant pas transitive, et il est donc nécessaire d'imposer des hypothèses supplémentaires. Cette propriété permettrait, cependant, de réduire la cardinalité de la représentation. Une autre problématique, consisterait à intégrer les préférences d'un décideur à l'algorithme 3.2 afin de filtrer les points qui ne sont pas optimaux selon elle. Cette idée est approfondie en annexe.

Chapitre 5.

Détermination du point nadir

Résumé

Une problématique bien connue en optimisation multiobjectif consiste à déterminer la borne de l'ensemble des points non dominés appelée *point nadir*, dont les composantes sont les pires valeurs atteintes par ceux-ci sur chaque objectif. Au delà du cas biobjectif, ce problème est réputé difficile.

Après avoir présenté différentes propriétés de ce point, dont certaines sont spécifiques au cas triobjectif, nous présentons deux stratégies pour appréhender le problème et nous discutons de leurs avantages et inconvénients respectifs. Enfin, nous proposons deux algorithmes, pour déterminer le point nadir, l'un s'appuyant sur les propriétés particulières des instances triobjectif et l'autre plus efficace pour les dimensions supérieures. Enfin, nous montrons leur efficacité en les comparant avec les algorithmes récents de la littérature.

5.1. Introduction

Nous avons, dans ce document, fait très souvent référence à ces deux points particuliers que sont *le point idéal* (noté y^I) et *le point anti-idéal* (noté y^U), dont les composantes sont respectivement les valeurs minimales et maximales atteintes par les solutions réalisables sur chaque objectif. Ces valeurs sont très importantes pour initialiser les différents algorithmes générant l'ensemble des points non dominés que nous avons évoqués dans les chapitres précédents.

Une autre problématique consiste à déterminer *le point nadir*, noté y^N, dont les composantes sont les valeurs maximales atteintes par les solutions efficaces (voir la Figure 5.1). Dans la configuration biobjectif, déterminer le point nadir est trivial : il suffit de déterminer les optima lexicographiques. Cependant, ce n'est plus vrai à partir de trois objectifs et le problème devient très difficile.

En fait, par définition, un algorithme simple pour déterminer ce point consisterait à générer l'intégralité de Y_{ND} puis à déterminer la valeur maximale sur chaque composante. Cependant, cette approche n'est pas satisfaisante pour deux principaux motifs.

Tout d'abord, comme nous l'avons spécifié dans les chapitres précédents, déterminer Y_{ND} est très coûteux, notamment parce que sa cardinalité peut être très élevée. Or, une grande partie des points peut être ignorée, car ne contribuant pas à l'une des composantes du point nadir.

Secondement, un grand nombre d'approches se servent du point nadir pour fournir une information préalable au décideur sur la répartition des points non dominés et sur les valeurs qu'il peut espérer atteindre. De plus, certaines méthodes interactives - c'est à dire des méthodes exhibant itérativement des points en prenant en compte les préférences du décideur - utilisent le point nadir comme information préalable, comme par exemple dans Buchanan (1997) ou Miettinen et al. (2010). Enfin certaines approches utilisent simplement le point nadir comme point de départ avant de s'en éloigner le plus possible selon une certaine norme, par exemple la méthode proposée par Ehrgott et Tenfelde-Podehl (2003). D'une manière générale, le point nadir n'est qu'un indicateur de la répartition des points non dominés et il n'est guère envisageable de consacrer beaucoup de temps à sa détermination.

Le défi consiste donc à déterminer le point nadir en exhibant un nombre limité de points non dominés. Deux catégories d'approches, s'appuyant sur des stratégies différentes, peuvent être distinguées. La première stratégie repose sur le théorème proposé par Ehrgott et Tenfelde-Podehl (2003) montrant que les points déterminant une composante du nadir sont non dominés sur les $p-1$ composantes restantes. Ainsi, de telles approches résolvent les p sous-problèmes à $p-1$ objectifs puis déterminent le point nadir à partir des p ensembles de points non dominés obtenus. Ces approches sont très nombreuses dans la littérature car il s'agit d'adaptations d'algorithmes générant les points non dominés. C'est notamment le cas de l'algorithme proposé par Kirlik et Sayın (2015) qui repose sur l'algorithme décrit à la section 2.4 pour énumérer les points non dominés des p sous-problèmes de dimension $p-1$. Ces approches présentent un inconvénient : le nombre de sous-problèmes à résoudre et leur nombre de points non dominés augmentent avec le nombre d'objectifs. De fait, si on observe que cette stratégie est très efficace pour déterminer le point nadir de problèmes triobjectif, une sévère baisse de performances est observée à partir

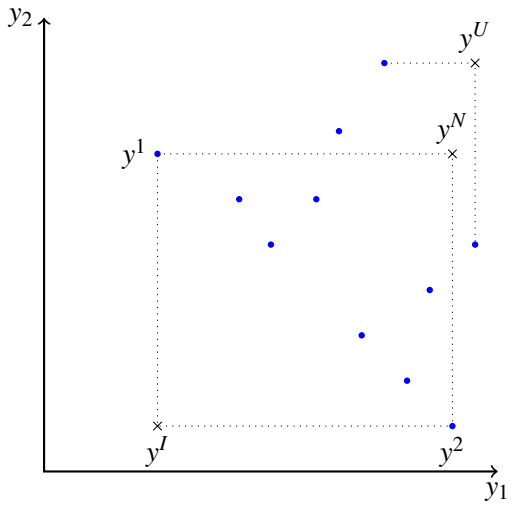


FIGURE 5.1. – Point idéal (y^I), point anti idéal (y^U) et point nadir (y^N). Dans le cas biobjectif, le point nadir est entièrement déterminé par les optima lexicographiques y^1 et y^2.

de 4 critères.

La seconde stratégie consiste à n'explorer que les régions de l'espace contenant des points réalisables améliorant l'estimation courante du point nadir. Ces régions sont identifiées en résolvant un programme mathématique puis sont explorées en utilisant le Théorème 1.1. Bien que cette étape préalable induise un coût supplémentaire, ces approches voient leur efficacité augmenter avec le nombre d'objectifs car elles ne résolvent plus intégralement les p sous problèmes à $p-1$ objectifs. Il s'agit du schéma suivi par les algorithmes résolvant la problématique plus générale consistant à optimiser une fonction linéaire sur l'ensemble des solutions efficaces (par exemple Boland et al., 2017, Jorge, 2009). L'adaptation de l'algorithme décrit à la section 2.3 qui a été proposée par Köksalan et Lokman (2015) (et reprise par Özpeynirci, 2017) appartient également à cette catégorie.

Après avoir présenté quelques propriétés intéressantes du point nadir, nous présenterons deux algorithmes permettant de le déterminer. Le premier algorithme, dédié aux problèmes triobjectifs, s'appuie sur des propriétés spécifiques à cette situation, tandis que le second voit son efficacité augmenter avec le nombre de critères. Ces méthodes semblent très prometteuses, comme le montrent les expérimentations que nous avons conduites.

5.2. Résultats préliminaires

Considérant un ensemble de points réalisables Y et l'ensemble des points non dominés associés Y_{ND}, nous cherchons à obtenir le *point nadir*, noté y^N, dont les composantes sont définies par :

$$y_k^N = \max_{y \in Y_{ND}} \{y_k\},\ k \in [\![1, p]\!]$$

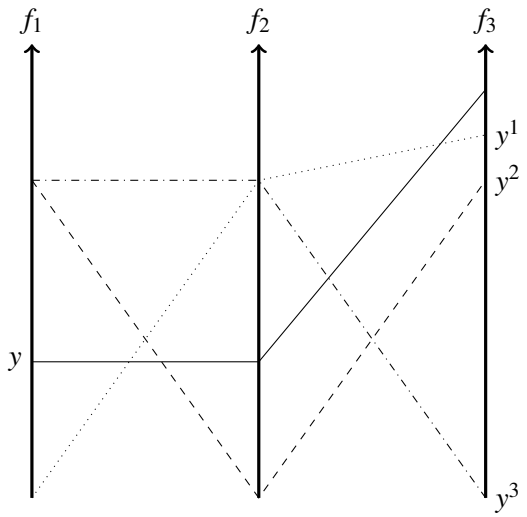


FIGURE 5.2. – Le point y n'est lexicographiquement optimal sur aucun objectif, mais définit pourtant y^N sur f_3.

Tous les points non dominés ne sont pas nécessaires à la détermination du point nadir. Seuls ceux ayant la pire valeur sur un objectif sont utiles.

Définition 5.1. (Point contribuant à la composante k) *Soit $y \in Y_{ND}$ un point non dominé atteignant la plus mauvaise performance sur l'objectif k, c'est-à-dire tel que :*

$$y_k = y_k^N$$

Alors, y est un point contribuant à la composante k *du point nadir.*

Évidemment, plusieurs points peuvent contribuer à la même composante du point nadir, et seul l'un d'entre eux est nécessaire.

Dans la configuration biobjectif, les deux optima lexicographiques sont les uniques points contribuant aux composantes du nadir. Cette propriété s'avère fausse pour un nombre arbitraire de critères comme le montre la proposition suivante, illustrée par la Figure 5.2.

Proposition 5.1. *Pour $p \geq 3$, les optima lexicographiques ne contribuent pas nécessairement à une composante de y^N.*

Démonstration. Soient les trois optima lexicographiques $y^1 = (0, 10, 10)$, $y^2 = (10, 0, 10)$ et $y^3 = (10, 10, 0)$. Soit le point $y = (5, 5, 15)$. y est également non dominé, et définit y_3^N. Cet exemple se généralise trivialement aux dimensions plus grandes : il suffit de supposer que les objectifs supplémentaires sont constants. □

Les résultats suivants présentent certaines conditions sur le nombre de composantes du nadir auxquelles peut contribuer un point non dominé.

Si un point contribue à $p-1$ composantes du nadir, il est forcément optimal sur l'objectif restant, comme montré par la Proposition 5.2.

Proposition 5.2. *Soit $y \in Y_{ND}$ tel que $y_{-k} = y^N_{-k}$. Alors, $y_k = y^I_k$.*

Démonstration. Supposons que $y_k > y^I_k$. Soit y^k un point optimal sur f_k. On a $y^k \preceq y$, contredisant l'hypothèse. □

Nous déduisons de cette constatation que la seule situation où un point peut contribuer à toutes les composantes du nadir apparaît quand le problème ne comporte qu'un seul point non dominé, comme montré par le Corollaire 5.1.

Corollaire 5.1. *Soit un point $y \in Y_{ND}$ tel que $y = y^N$. Alors, $y = y^I$ et $Y_{ND} = \{y\}$.*

Démonstration. Puisque $y = y^N$, nous avons en particulier $y_{-k} = y^N_{-k}$, $\forall k \in [\![1, p]\!]$. Par la Proposition 5.2, cela signifie que $y_k = y^I_k$ $\forall k \in [\![1, p]\!]$ et donc que $y = y^I$. Par définition, pour tout $y \in Y_{ND}$, on a $y^I \leqq y$. En conséquence, $Y_{ND} = \{y\}$. □

Le résultat suivant, proposé par Ehrgott et Tenfelde-Podehl (2003), établit une caractéristique des points non dominés pouvant contribuer a une composante k du point nadir : ceux-ci sont nécessairement non dominés sur les $p - 1$ objectifs restants $[\![1, p]\!] \setminus \{k\}$.

Proposition 5.3. *Soit $y \in Y_{ND}$ tel que $y_k = y^N_k$. Alors, $\nexists y' \in Y$ tel que $y'_{-k} \preceq y_{-k}$.*

Démonstration. Supposons, par l'absurde, qu'il existe $y' \in Y$ tel que $y'_{-k} \preceq y_{-k}$. Puisque Y_{ND} est *stable extérieurement*, $y' \in Y_{ND}$ ou y' est dominé par un point de Y_{ND}. Dans les deux situations, il existe $\hat{y} \in Y_{ND}$ tel que $\hat{y}_{-k} \preceq y_{-k}$. De plus, par définition du point nadir, $\hat{y}_k \leq y_k = y^N_k$. Ainsi, $\hat{y} \preceq y$, contredisant l'hypothèse que $y \in Y_{ND}$. □

Enfin, la Proposition 5.4 nous permet d'identifier des zones ne pouvant contenir de point contribuant à une composante de y^N. En effet, si une zone u est bornée sur la composante k ($u_k \neq M$), il existe un point $y \in N$ tel que $u_k = y_k$. Ainsi, il est possible d'éliminer certaines zones de la région de recherche sans les explorer.

Proposition 5.4. *Soit $N \subseteq Y_{ND}$ et soit $u \in U(N)$ une borne telle que $u_k \neq M$, $\forall k \in [\![1, p]\!]$. Alors, $z(u)$ ne contient aucun point non dominé contribuant à une composante du nadir.*

Démonstration. Soit $k \in [\![1, p]\!]$. Puisque $u_k \neq M$, nous avons $N_k(u) \neq \emptyset$ d'après la Proposition 1.3. D'après (1.6), chaque point $y \in N_k(u)$ est tel que $y_k = u_k$. Puisque y est non dominé et que tous les points de $z(u)$ sont meilleurs que y_k sur f_k, aucun d'entre eux n'apporte d'information supplémentaire sur le point nadir. □

Désormais, $U_{-k}(N)$ fera référence à l'ensemble des zones de $U(N)$ telles que $u_k = M$ et, nous supposons que N ne contient que des points non dominés ($N \subseteq Y_{ND}$). Un algorithme trivial consisterait à déterminer les composantes du point nadir séparément, en ne conservant, lors de la mise à jour de la région de recherche, que les zones non bornées sur le critère recherché (synthétisé par l'algorithme 5.1).

Il est intéressant d'observer que les Propositions 5.3 et 5.4 sont liées, comme le montre le résultat suivant.

Algorithme 5.1 : Algorithme général déterminant la composante k du point nadir.

Input : Ensemble des points réalisables : Y
Fonction fortement monotone : $\Phi : Y \to \mathbb{R}$
Composante du nadir à déterminer : k

Output : Composante du nadir : y_k^N

$N \leftarrow \emptyset$
$y_k^N = y_k^I$
$U_{-k}(N) \leftarrow \{(M, \dots, M)\}$
while $U_{-k}(N) \neq \emptyset$ **do**
  $u \leftarrow$ choisir une zone de $U_{-k}(N)$
  résoudre le programme $P(u)$ tel que

$$P(u) \begin{cases} \min & \Phi(y) \\ \text{s.c.} & y \in Y \\ & y_{-k} \prec u_{-k} \end{cases}$$

  if $P(u)$ *est infaisable* **then**
    $U_{-k}(N) \leftarrow U_{-k}(N) \setminus \{u\}$
  else
    y^* est l'optimum de $P(u)$
    $N \leftarrow N \cup \{y^*\}$
    Calculer $U(N)$
    $U_{-k}(N) \leftarrow \{u \in U(N),\ u_k = M\}$
    $y_k^N \leftarrow \max\{y_k^N, y_k^*\}$

Théorème 5.1. *1. L'algorithme 5.1 énumère* au moins *tous les points de Y_{ND} qui sont non dominés sur les $p-1$ critères restants.*

2. Si la fonction Φ satisfait la propriété suivante :

$$y_{-k} \preceq y'_{-k} \implies \Phi(y) < \Phi(y') \tag{5.1}$$

alors l'algorithme 5.1 énumère uniquement *les points de Y_{ND} non dominés sur les $p-1$ critères restants.*

Démonstration. 1. Soit $u \in U_{-k}(N)$ et soit $y \in z(u)$. Supposons par l'absurde qu'il existe un point $y' \in N$ tel que $y'_{-k} \preceq y_{-k}$. Soit le point $\bar{y}$ identique à y à l'exception de la composante k :

$$\bar{y} = (y_1, \dots, y_{k-1}, y'_k + 1, y_{k+1}, \dots, y_p)$$

Puisque $u_k = M$, $\bar{y} \in z(u)$. De plus, $y' \preceq \bar{y}$. En conséquence, $z(u)$ contient un point dominé par un point de N, contredisant l'hypothèse que $u \in U_{-k}(N)$. Ainsi, la région de recherche de l'algorithme 5.1 ne contient que des points non dominés par des points de N sur les $p-1$ critères restants, et l'algorithme énumèrera donc tous les points de Y_{ND} qui ne sont pas dominés sur ces critères.

2. Si Φ satisfait l'hypothèse (5.1), le Théorème 1.1 garantit que les optima de $P(u)$ sont non dominés sur les $p-1$ critères restants.

□

La Proposition 5.4 s'adapte également aux caractérisations de la région de recherche proposées par Lokman et Köksalan (2013) et Kirlik et Sayin (2014). Leurs adaptations respectives à la détermination du point nadir que sont Köksalan et Lokman (2015) et Kirlik et Sayın (2015) déterminent indépendamment chaque composante k de y^N : il suffit de générer les points non dominés en omettant d'explorer les modèles et rectangles contraignant l'objectif f_k (voir par exemple l'algorithme 5.2).

Avant de proposer une nouvelle méthode pour déterminer le point nadir, nous étudions, dans la suite, des propriétés spécifiques au cas triobjectif.

5.3. Cas triobjectif

La grande efficacité de l'adaptation triviale de l'algorithme 3.2 pour la résolution de problèmes triobjectif se justifie par différentes propriétés spécifiques à cette situation.

Le premier résultat établit que les composantes du point nadir peuvent être déterminées indépendamment si la région de recherche est initialisée avec les optima lexicographiques.

Proposition 5.5. *Soit $Y_I = \{y^1, y^2, y^3\}$ un ensemble de points non dominés tel que $y^k_k = y^I_k$, $\forall k \in [\![1,3]\!]$. Alors, toutes les bornes de $U(Y_I)$ ont exactement une composante non bornée.*

Algorithme 5.2 : Schéma de l'algorithme proposé par Kirlik et Sayın (2015)

Input : Ensemble des points réalisables : Y
Critère à optimiser : m
Composante du nadir à déterminer : k
Estimation du nadir : $\hat{y}^N$
Output : Composante du nadir : y^N_m
Précondition : $k \neq m$

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \ldots, M)\}$
$y^N_k \leftarrow \hat{y}^N_k$
Mettre à jour $U(N)$ avec $\hat{y}^N$
while $U(N) \neq \emptyset$ **do**
    Choisir $R(u, l) \in U(N)$
    Résoudre $\Pi(m, u)$
    if $\Pi(k, u)$ *est infaisable* **then**
        $U(N) \leftarrow U(N) \setminus \{R(u, l)\}$
    else
        y^* est l'optimum de $\Pi(m, u)$
        Mettre à jour $U(N)$
        $y^N_k \leftarrow \max\{y^N_k, y^*_k\}$
        $U(N) \leftarrow \{R(u, l) \in U(N), u_k = M\}$

Démonstration. y^1 induit deux nouvelles bornes : $u^2 = (M, y^1_2, M)$ et $u^3 = (M, M, y^1_3)$ (la borne fille u^1 étant éliminée car y^1 est optimal sur f_1).

Puisque y^2 est optimal sur f_2, nous avons $y^2_2 \leq y^1_2$. Si $y^2_2 = y^1_2$, alors $z(u^2) \cap Y = \emptyset$ et u^2 est éliminée. Sinon, $y^2 \prec u^2$ et u^2 doit être subdivisée. Deux nouvelles zones sont créées : $u^{2,1} = (y^2_1, y^1_2, M)$ et $u^{2,3} = (M, y^1_2, y^2_3)$ ($u^{2,2}$ est éliminée comme précédemment).

Si $y^2 \prec u^3$, deux zones sont générées, $u^{3,1} = (y^2_1, M, y^1_3)$ et $u^{3,3} = (M, M, y^2_1)$. Sinon, u^3 est conservée. Dans les deux situations, il ne reste donc qu'une seule borne ayant deux composantes non bornées, u^3 ou $u^{3,3}$, que l'on notera $\bar{u}^3$ pour traiter ces deux cas simultanément.

De manière similaire à précédemment, si $y^3_3 = \bar{u}^3_3$, alors $z(\bar{u}^3) \cap Y = \emptyset$ et $\bar{u}^3$ est éliminée. Sinon, $y^3 \prec \bar{u}^3$, ce qui induit deux nouvelles zones, $\bar{u}^{3,1} = (y^3_1, M, \bar{u}^3_3)$ et $\bar{u}^{3,2} = (M, y^3_2, \bar{u}^3_3)$, chacune ayant exactement une composante non bornée (la troisième fille est éliminée). □

Ainsi, si la région de recherche est initialisée avec les optima lexicographiques, nous pouvons déterminer le point nadir composante par composante car toutes les zones ont exactement une seule composante non bornée. Soit y^N_k la composante que nous cherchons. Dans la suite, nous supposerons que la région de recherche a été initialisée avec les optima lexicographiques (en d'autres termes, que $Y_I \subseteq N$). De plus,pour chaque zone $u \in U_{-k}(N)$, u_ℓ et u_m feront référence aux deux autres composantes de u que nous savons bornées (Proposition 5.5).

Afin de montrer que cette configuration est très adaptée aux règles de réduction, nous supposerons dans la suite que ℓ est l'objectif optimisé par le programme Π et que m est la seconde composante bornée de u. La formulation du programme Π est donc :

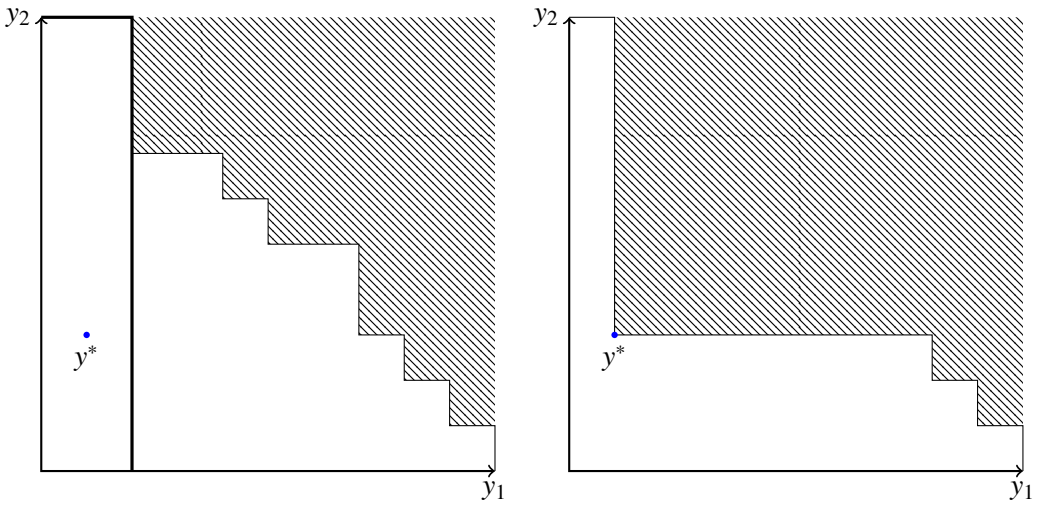
$$\Pi(\ell, u) \begin{cases} \text{lexmin} & \{y_\ell, y_m, y_k\} \\ \text{s.c.} & y \in Y \\ & y_m < u_m \end{cases}$$

Nous savons d'après la Proposition 3.3 (page 58) que $\Pi(\ell, u)$ est réalisable.

Si $\Pi(\ell, u)$ exhibe un nouveau point, nous pouvons appliquer la Proposition 3.6 (page 59) :
Proposition 5.6. *Soit $u \in U_{-k}(N)$ et y^* une solution optimale de $\Pi(\ell, u)$. Soit $u' \in U_{-k}(N)$ une borne telle que $u'_m \leq u_m$. Si $y^* \prec u'$, alors $z(u'^\ell) \cap Y = \emptyset$.*

Démonstration. Puisque $u'^\ell_{-\ell} = u'_{-\ell}$ et $u'^\ell = y^*_\ell$, la Proposition 3.6 s'applique, conduisant au résultat. □

En conséquence, si u_m est maximum, il est possible d'appliquer la Proposition 5.6 lors de la subdivision de toutes les bornes de $U_{-k}(N)$ contenant le point retourné par $\Pi(\ell, u)$ comme montré par le résultat suivant.
Corollaire 5.2. *Soit la borne $u' \in U_{-k}(N)$ telle que $u'_m = \max_{u \in U_{-k}(N)} \{u_m\}$. Soit y^* une solution optimale de $\Pi(\ell, u')$. Alors, $|U_{-k}(N \cup \{y^*\})| \leq |U_{-k}(N)|$*



(a) y^* minimise f_2 dans la zone.

(b) Tous les points dont la projection sur (f_1, f_2) est dominée par y^* ne peuvent contribuer à y_3^N.

FIGURE 5.3. – Utilisation de l'algorithme 5.3 pour déterminer la 3^e composante du nadir dans le cas triobjectif (toutes les zones sont non bornées sur le critère 3).

Démonstration. Par hypothèse, toutes les zones $u \in U_{-k}(N)$ sont telles que $u_m < u'_m$. En appliquant la Proposition 5.6, toutes les bornes u telles que $y^* \prec u$ sont remplacées par une seule de leurs filles, les autres ne sont pas modifiées. □

L'algorithme 5.3 synthétise l'approche. Il est intéressant d'observer qu'il énumère tous les points de Y_{ND} qui sont non dominés sur $p-1$ critères. En effet, même si ceux-ci n'améliorent pas l'estimation du point nadir, ils doivent nécessairement être considérés car tous sont candidats. En conséquence, cela revient à résoudre les p problèmes à $p-1$ critères.

La Figure 5.3 illustre le déroulement de l'algorithme lors de la détermination de y_3^N. Le point y^* est énuméré et la région de l'espace dominée par y^*_{-3} est retirée. En effet, tout les points de cette région sont moins bons que y^* sur f_1 et f_2 et doivent donc être meilleurs que y_3 sur f_3 pour être non dominés. En conséquence, ils ne peuvent contribuer à la troisième composante.

Cette approche est particulièrement efficace dans le cas triobjectif car il n'y que 3 sous problèmes à considérer. De plus, chaque point ne pouvant contribuer au plus qu'à 2 composantes - et le plus souvent qu'à une seule - chacun d'eux n'induit qu'un nombre limité de subdivisions. En revanche, cela n'est plus vrai pour $p \geq 4$, car chaque point peut désormais contribuer à 2 composantes du nadir, augmentant donc le nombre de bornes à considérer. En conséquence, dans cette situation, résoudre les p sous problèmes à $p-1$ critères est peu satisfaisant. Nous proposons dans la suite un nouvel algorithme n'explorant que des zones contenant des points réalisables susceptibles d'améliorer la meilleure estimation connue du point nadir, afin de réduire à la fois le nombre de points énumérés et le nombre de bornes considérées.

Algorithme 5.3 : Adaptation de l'Algorithme 3.2 déterminant le point nadir

Input : Ensemble des points réalisables Y
Output : Point nadir de Y : y^N

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \dots, M)\}$
/* Estimation du point nadir initialisée avec les valeurs atteintes par les optima lexicographiques */
$\tilde{y}^N \leftarrow \hat{y}^N$
while $U(N) \neq \emptyset$ **do**
  /* On choisit la projection d'une zone maximisant le volume (3.2) */
  choisir (k^*, u^*)
  $y^* \leftarrow$ résoudre $\Pi(k^*, u^*)$
  /* Le modèle est ajouté à l'archive */
  $T \leftarrow T \cup \{(k^*, u^*, y^*_k)\}$
  if $y^* \notin N_{k^*}(u^*)$ **then**
    /* y^* est un nouveau point non dominé. La région de recherche est mise à jour selon l'algorithme 1.3 */
    $U(N \cup \{y^*\}) \leftarrow$ updateSearchRegion $(U(N), y^*)$
    $N \leftarrow N \cup \{y^*\}$
    /* Mise à jour de l'estimation du point nadir */
    foreach $i \in [\![1, p]\!]$ **do**
      $\tilde{y}^N_i \leftarrow \max\{\tilde{y}^N_i, y^*_i\}$
  /* On ne conserve que les bornes ayant au moins une composante non bornée */
  $U(N) \leftarrow \{u \in U(N),\ \exists i \in [\![1, p]\!], u_i = M\}$
  /* On applique les règles de réductions sur la région de recherche */
  foreach $u' \in U(N)$ **do**
    foreach $(k^i, u^i, y^{*i}) \in T$ **do**
      if $u'_{-k^i} \leqq u^i_{-k^i}$ *and* $y^{*i}_k \geq u'_k$ **then**
        $U(N) \leftarrow U(N) \setminus \{u'\}$
$y^N \leftarrow \tilde{y}^N$

5.4. Une première adaptation de l'algorithme

L'algorithme est initialisé avec l'estimation du point nadir $\hat{y}^N$ obtenue à partir des optima lexicographiques sur chaque objectif. À chaque itération, une zone $u \in U(N)$ ayant une composante non bornée $u_k \neq M$ est sélectionnée. Le programme utilisé dans la phase d'exploration est similaire à $\Pi(k, u)$, mais impose en plus d'améliorer l'estimation du nadir sur la composante k.

$$\Omega(k, u, \tilde{y}^N) \begin{cases} \min & \sum_{i=1}^{p} y_i \\ \text{s.c.} & y \in Y \\ & y_{-k} \prec u_{-k} \\ & y_k > \tilde{y}^N_k \end{cases}$$

où $\tilde{y}^N$ est l'estimation du point nadir obtenue à partir des points non dominés déjà énumérés :

$$\tilde{y}^N_k = \max_{y \in N} \{y_k\}$$

Attention, le Théorème 1.1 ne s'applique pas à Ω qui peut retourner un point potentiellement dominé, comme le montre la proposition suivante :
Proposition 5.7. *S'il existe, y^*, le point optimal de Ω n'est pas forcément non dominé.*

Démonstration. S'il existe un point $y \in Y_{ND}$ tel que $y_{-k} \leqq y^*_{-k} \prec u_{-k}$ et $y_k \leq \tilde{y}^N_k$, alors $y \notin F(\Omega)$ mais $y \preceq y^*$. □

À cause de la Proposition 5.7, il est nécessaire de vérifier si l'optimum retourné par Ω est non dominé. Cela se fait en appliquant trivialement le Théorème 1.1 :
Proposition 5.8. *Soit y^* l'optimum retourné par Ω. Alors, le programme suivant retourne un point non dominé :*

$$P(y^*) \begin{cases} \min & \sum_{i=1}^{p} y_i \\ s.c. & y \in Y \\ & y \leqq y^* \end{cases}$$

Démonstration. Par le Théorème 1.1, l'optimum de $P(y^*)$ est non dominé, et, par construction, domine potentiellement y^*. □

Observons que l'optimum de Ω est réalisable pour P, et peut donc servir de solution de départ comme précédemment.

Considérons une borne $u \in U(N)$ telle que $u_k = M$ pour un certain $k \in [\![1, p]\!]$. Deux cas de figure peuvent se présenter.

Si le programme $\Omega(k, u, \tilde{y}^N)$ est réalisable, l'algorithme vérifie si son optimum est dominé ou non, en résolvant le programme P associé. Dans les deux cas, l'optimum de P appartient à la région de recherche et est donc un nouveau point non dominé. En effet, sa projection domine u_{-k} et donc le point domine u car $u_k = M$ par construction. Ainsi, la région de recherche est mise à

jour avec ce nouveau point de manière similaire à l'algorithme 1.3, à l'exception près que seules les zones ayant au moins une composante non bornée sont conservées. De plus, à chaque zone $u \in U(N)$, nous associons l'ensemble $\mathcal{K}(u) \subset [\![1, p]\!]$ des indices des composantes non bornées de u :

$$\mathcal{K}(u) = \{k \in [\![1, p]\!],\ u_k = M\}$$

Si le programme $\Omega(k, u, \tilde{y}^N)$ est infaisable, cela signifie qu'aucun point contribuant au nadir sur la composante k ne domine u_{-k}. Nous pouvons donc retirer l'indice k de $\mathcal{K}(u)$. Cependant, cela n'est pas suffisant pour éliminer la borne de $U(N)$: en effet, s'il reste des composantes non bornées dans $\mathcal{K}(u)$, il est nécessaire de les tester également. Si en revanche, $\mathcal{K}(u) = \emptyset$, la borne u peut être retirée de $U(N)$.

L'approche est décrite en détail dans l'algorithme 5.4.

Algorithme 5.4 : Une première approche pour la détermination du point nadir

Input : Ensemble des points réalisables Y
Output : Point nadir de Y : y^N

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \dots, M)\}$
`/* Estimation du point nadir initialisée avec le tableau des gains */`
$\tilde{y}^N \leftarrow \hat{y}^N$
while $U(N) \neq \emptyset$ **do**
  Choisir $u \in U(N)$
  Choisir $k \in \mathcal{K}(u)$
  $\mathcal{K}(u) \leftarrow \mathcal{K}(u) \setminus \{k\}$
  $y^* \leftarrow$ résoudre $\Omega(k, u, \tilde{y}^N)$
  if $\Omega(k, u, \tilde{y}^N)$ *est réalisable* **then**
    $y \leftarrow$ résoudre $P(y^*)$
    Mettre à jour la région de recherche
    Mettre à jour l'estimation du nadir
  else if $\mathcal{K}(u) = \emptyset$ **then**
    $U(N) \setminus \{u\}$
$y^N \leftarrow \tilde{y}^N$

Bien que cet algorithme semble plus efficace que l'algorithme 5.3 quand le nombre d'objectifs augmente, il présente un inconvénient : si une zone est non bornée sur plusieurs critères, mais qu'aucun des points qu'elle contient ne contribue au nadir, l'algorithme va devoir résoudre un programme Ω par composante non bornée. Pour pallier ce problème, nous proposons de généraliser cette idée dans la section suivante.

5.5. Un nouvel algorithme pour la détermination du point nadir

Le nouvel algorithme que nous proposons repose sur la modification suivante du programme Ω visant à exhiber un point améliorant une composante du nadir dans une zone $z(u)$. Pour cela, nous introduisons les $|\mathcal{K}(u)|$ variables binaires z_k définies par :

$$z_k = \begin{cases} 1 & \text{si y} > \tilde{y}_k^N \\ 0 & \text{sinon} \end{cases}$$

Le programme Ω devient donc :

$$\Omega(u, \tilde{y}^N) \begin{cases} \min & 1 & \\ \text{s.c.} & y \in Y & \\ & y \prec u & \\ & y_k > \tilde{y}_k^N - M(1 - z_k) & \forall k \in \mathcal{K}(u) \\ & \sum_{k \in \mathcal{K}(u)} z_k = 1 & \\ & z_k \in \{0, 1\} & \forall k \in \mathcal{K}(u) \end{cases}$$

Si $\Omega(u, \tilde{y}^N)$ n'a pas de solution, cela signifie qu'aucun point réalisable de $z(u)$ n'améliore l'estimation du nadir $\tilde{y}^N$: la borne u peut donc être retirée de la région de recherche. Sinon, le point retourné améliore l'estimation mais est potentiellement dominé : le programme P est donc résolu et la région de recherche est mise à jour avec le nouveau point non dominé obtenu, comme précédemment.

Il est important de remarquer que si une zone ne contient aucun point contribuant à une composante k du point nadir, c'est également le cas de ses filles. Il est donc intéressant de déterminer très vite les composantes non bornées d'une borne u sur lesquelles aucun point réalisable n'améliore l'estimation du nadir. Nous proposons donc d'utiliser la fonction objectif suivante dans le programme Ω :

$$\min \sum_{k \in \mathcal{K}(u)} k z_k$$

Ainsi, le programme Ω va chercher à améliorer l'estimation des composantes du nadir par ordre croissant des $k \in \mathcal{K}(u)$. Par conséquent, si la valeur optimale de $\Omega(u, \tilde{y}^N)$ est k^*, nous savons qu'aucun point réalisable de $z(u)$ ne peut contribuer sur une composante $k \in \mathcal{K}(u)$, $k < k^*$. Pour cette raison, il est intéressant d'explorer en priorité les zones ayant un grand nombre de composantes non bornées. L'algorithme 5.5 synthétise l'approche. Il est intéressant de remarquer qu'il se ramène à l'algorithme 5.4 dans le cas triobjectif pour lequel nous avons $|\mathcal{K}(u)| = 1$, à l'exception de la fonction objectif de Ω.

Algorithme 5.5 : Détermination du point nadir

Input : Ensemble des points réalisables Y
Output : Point nadir de Y : y^N

$N \leftarrow \emptyset$
$U(N) \leftarrow \{(M, \dots, M)\}$
/* Estimation du point nadir initialisée avec les valeurs atteintes par les optima lexicographiques */
$\tilde{y}^N \leftarrow \hat{y}^N$
while $U(N) \neq \emptyset$ **do**
  $u \leftarrow \underset{u^i \in U(N)}{\text{argmax}}\{|\mathcal{K}(u^i)|\}$
  $(k^*, y^*) \leftarrow$ résoudre $\Omega(u, \tilde{y}^N)$
  if $\Omega(k, u, \tilde{y}^N)$ *est réalisable* **then**
    $y \leftarrow$ résoudre $P(y^*)$
    $\mathcal{K}(u) \leftarrow \mathcal{K}(u) \setminus \{k < k^*, k \in [\![1, p]\!]\}$
    Mettre à jour la région de recherche. Noter que pour toute borne u' qui est subdivisée, $\mathcal{K}(u'^i) = \mathcal{K}(u'),\ \forall i \in [\![1, p]\!]$
    Mettre à jour l'estimation du nadir
  else
    $U(N) \setminus \{u\}$
$y^N \leftarrow \tilde{y}^N$

5.6. Résultats expérimentaux et critiques

Les Tableaux 5.1, 5.2 and 5.3 présentent les comparaisons expérimentales entre l'algorithme 5.5, noté *nadir-disj* et l'adaptation de l'approche *dynamique* en deux phases (Algorithme 5.3), notée *naive*. Il est important de remarquer que l'approche directe ne peut être adaptée car elle peut potentiellement retourner des points faiblement non dominés, contredisant ainsi l'hypothèse de la Proposition 5.4. Sont spécifiés le nombre de points non dominés générés ($|N|$), le temps de résolution en secondes (cpu), le nombre d'itérations (#Iter), le nombre de programmes infaisables considérés (#Infeasible), les tailles maximales et moyennes de la région de recherche ($|U(N)|_{max}$ et $|U(N)|_{avg}$) ainsi que le pourcentage de points non dominés énumérés ($\%|Y_{ND}|$) quand ce dernier est connu. La valeur N/A est renseignée si le temps d'exécution excède 10 heures. Comme dans le chapitre 3, les comparaisons sont faites avec les algorithmes récents de la littérature que sont celui proposé par Kirlik et Sayın (2015), noté *KS2015*, et celui proposé par Boland et al. (2017), noté *BCS2017*.

Comme ce n'est plus le programme $\Pi(k, u)$ qui est utilisé dans l'approche *nadir-disj*, les règles de réductions présentées précédemment ne peuvent plus s'appliquer. Nous observons donc, en conséquence, que des programmes infaisables sont résolus. Nous pouvons même constater qu'à partir de 4 objectifs, la majorité des programmes considérés par l'approche *nadir-disj* sont infaisables. En revanche, malgré cela, cette dernière semble être sensiblement plus efficace que les autres approches, et en particulier que l'algorithme *naive* , à l'exception du cas triobjectif. En effet, l'algorithme *naive* énumère tous les points non dominés sur $p - 1$ objectifs. Pour $p = 3$, cela fonctionne plutôt bien car chaque point induit un nombre limité de zones. En revanche, cela n'est plus vrai quand le nombre de critères augmente, par exemple pour le sac à dos à 7 objectifs et 20 éléments où le nombre de zones peut dépasser 32000. À l'opposé, l'approche *nadir-disj* n'énumère que des points améliorant l'estimation du nadir sur au moins une composante. Le nombre de points exhibés, et donc de zones manipulées, est drastiquement plus faible. Ainsi, il est possible de résoudre des instances du sac à dos 7 objectifs et 30 éléments en un peu moins de 1 minute quand l'approche *naive* nécessite plus de 10 heures.

D'une manière générale, l'approche *naive* domine expérimentalement toutes les autres lors de la résolution de problèmes triobjectifs tandis que l'approche *nadir-disj* semble être la meilleure dans les autres situations. Il est néanmoins intéressant d'observer que, si la capacité de l'algorithme *KS2015* à résoudre de larges instances semble limitée, celle de *BCS2017* est particulièrement efficace, au point de battre les approches que nous proposons sur les instances à 7 objectifs et 20 éléments. Cependant, ses limites sont atteintes lors de la résolution de problèmes *MOKP* à 4 objectifs et 20 éléments, que seule l'approche *nadir-disj* est en mesure de résoudre.

5.7. Conclusion

Après avoir proposé quelques résultats et un algorithme spécifiques au cas triobjectif, nous avons proposé une nouvelle approche pour résoudre la problématique du calcul du point nadir quand le nombre de critères excède 3. Au détriment des règles de réduction et de la garantie de ne

			$\|N\|$	cpu	#Iter	#Infeasible	$\|U(N)\|_{max}$	$\|U(N)\|_{avg}$	$\%\|Y_{ND}\|$
p	n	name	mean	mean	mean	mean	mean	mean	mean
3	50	naive	114,7	2,4	121,5	0,0	7,8	4,1	28,7
		nadir-disj	64,1	2,6	115,6	51,5	21,3	12,9	15,6
		KS2015	117,8	2,2	117,8	0,0	2,0	1,9	29,7
		BCS2017	49,1	1,7	195,1	20,7			13,0
	100	naive	463,2	18,4	470,4	0,0	10,0	4,0	9,5
		nadir-disj	380,9	36,6	545,2	164,3	47,6	28,9	7,1
		KS2015	484,5	18,0	484,5	0,0	2,0	1,9	38,7
		BCS2017	175,5	13,9	700,5	70,5			10,0
	150	naive	800,1	40,0	807,0	0,0	12,9	4,0	5,1
		nadir-disj	877,2	140,6	1 154,7	277,5	73,2	44,9	5,5
		KS2015	841,4	49,1	841,4	0,0	2,0	2,0	5,4
		BCS2017	328,5	38,6	1 311,6	153,4			2,1
4	50	naive	1 486,6	88,5	3 254,7	0,0	489,6	297,6	59,6
		nadir-disj	144,7	9,3	555,0	410,3	314,0	170,3	6,1
		KS2015	1 885,2	142,8	3 523,8	158,4	17 001,4	15 212,1	77,1
		BCS2017	200,6	31,9	989,0	9,9			8,4
5	50	naive	8 084,0	1 962,1	55 126,8	0,0	20 772,0	11 557,9	82,6
		nadir-disj	267,1	41,0	2 855,6	2 588,5	2 024,3	1 163,0	2,9
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	472,3	239,0	3 075,9	6,2			5,1
6	20	naive	348,2	140,6	10 024,8	0,0	6 599,5	3 958,5	99,8
		nadir-disj	51,9	5,5	923,0	871,1	686,1	384,2	19,1
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	54,2	5,9	403,2	1,5			20,0
	30	naive	2 127,2	1 616,6	56 279,5	0,0	32 959,1	20 411,4	98,8
		nadir-disj	119,9	17,9	2 780,4	2 660,5	2 081,8	1 180,9	8,5
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	139,1	34,6	1 051,8	3,1			9,9

TABLE 5.1. – Comparaisons expérimentales des différentes approches pour le calcul du nadir (instances MOKP)

p	n	name	$\|N\|$ mean	cpu mean	#Iter mean	#Infeasible mean	$\|U(N)\|_{max}$ mean	$\|U(N)\|_{avg}$ mean	$\%\|Y_{ND}\|$ mean
3	900	naive	301,4	27,7	305,7	0,0	5,6	3,0	4,9
		nadir-disj	734,6	185,1	914,2	179,6	60,5	38,4	12,0
		KS2015	369,9	36,6	369,9	0,0	2,0	2,0	6,1
		BCS2017	174,1	68,7	689,2	114,8			2,8
	1600	naive	418,7	56,5	422,6	0,0	5,1	2,9	3,0
		nadir-disj	1 533,0	627,2	1 800,6	267,6	94,8	61,3	10,8
		KS2015	535,1	75,8	535,1	0,0	2,0	2,0	3,8
		BCS2017	259,4	160,4	259,4	192,7			1,8
	2500	naive	515,2	105,0	518,9	0,0	4,7	2,9	2,1
		nadir-disj	2 566,2	1 651,3	2 909,2	343,0	108,8	68,8	10,3
		KS2015	700,5	150,8	700,5	0,0	2,0	2,0	2,8
		BCS2017	321,6	302,8	1 277,3	247,2			1,3
4	400	naive	9 954,4	1 805,1	19 479,4	0,0	888,3	623,1	21,5
		nadir-disj	957,9	309,7	2 828,0	1 870,1	1 223,2	621,6	2,2
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	1 024,5	881,5	5 774,6	89,7			2,4

TABLE 5.2. – Comparaisons expérimentales des différentes approches pour le calcul du nadir (instances MOAP)

résoudre que des problèmes réalisables, nous avons créé une méthode permettant de réduire grandement le nombre de points énumérés qui ne contribuent pas à une composante du point nadir et donc de diminuer, d'une part, le nombre de programmes à résoudre, et d'autre part, la taille de la région de recherche.

p	n	name	$\|N\|$ mean	cpu mean	#Iter mean	#Infeasible mean	$\|U(N)\|_{max}$ mean	$\|U(N)\|_{avg}$ mean
3	300	naive	2 418, 2	253, 9	2 425, 9	0, 0	15, 7	4, 0
		nadir-disj	4 364, 4	2 027, 1	5 144, 8	780, 4	198, 2	119, 3
		KS2015	2 580, 6	295, 9	2 580, 6	0, 0	2, 0	2, 0
		BCS2017	1 045, 1	284, 9	4 178, 3	563, 7		
	500	naive	5 509, 0	964, 2	5 516, 4	0, 0	16, 3	3, 4
		nadir-disj	11 000, 7	12 763, 8	12 818, 2	1 817, 5	261, 2	164, 8
		KS2015	5 982, 0	1 169, 8	5 982, 0	0, 0	2, 0	2, 0
		BCS2017	2 590, 1	1 228, 3	10 361, 6	1 604, 1		
4	100	naive	16 947, 5	2 388, 9	32 176, 0	0, 0	1 678, 5	1 022, 3
		nadir-disj	946, 3	201, 4	3 073, 3	2 127, 0	1 593, 5	771, 6
		KS2015	18 967, 4	4 183, 9	33 655, 2	478, 9	146 730, 8	100 385, 8
		BCS2017	1 558, 4	1 289, 0	9 275, 9	229, 7		
	200	naive	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		nadir-disj	5 412, 9	5 051, 0	14 747, 6	9 334, 7	6 630, 4	2 958, 7
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
7	10	naive	27, 2	5, 2	847, 2	0, 0	664, 5	363, 1
		nadir-disj	14, 9	1, 1	264, 8	249, 9	204, 7	110, 0
		KS2015	154, 5	978, 5	2 502, 8	841, 6	8 681 460, 3	4 622 450, 5
		BCS2017	14, 9	1, 1	264, 8	249, 9	204, 7	110, 0
	20	naive	553, 4	900, 0	43 093, 6	0, 0	32 508, 2	18 811, 9
		nadir-disj	69, 4	17, 9	2 820, 7	2 751, 3	2 165, 7	1 215, 6
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	65, 0	12, 2	538, 0	0, 7		
	30	naive	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		nadir-disj	147, 6	57, 4	9 127, 1	8 979, 5	7 658, 8	4 185, 1
		KS2015	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
		BCS2017	181, 2	79, 0	1 636, 3	2, 3		

TABLE 5.3. – Comparaisons expérimentales des différentes approches pour le calcul du nadir (instances MOKP où $|Y_{ND}|$ n'est pas connu).

Conclusion

Nous avons présenté, dans ce document, une nouvelle généralisation de la méthode ε-contrainte. En s'appuyant sur un certain nombre de résultats, il est possible de ne résoudre que des programmes réalisables et de fournir une solution de départ en temps constant. De plus, le nombre de zones considérées peut être réduit grâce à l'emploi de règles de réduction, en particulier lors de la résolution de problèmes triobjectifs. L'algorithme résultant semble très prometteur, comme le montrent les expérimentations que nous avons menées. Nous avons étendu cette approche pour intégrer une tolérance dans la relation de dominance et ainsi obtenir une représentation très concise extrêmement rapidement. Enfin, nous avons synthétisé les stratégies utilisées pour déterminer le point nadir, puis, après avoir présenté un certain nombre de résultats, dont certains sont spécifiques au cas triobjectif, nous avons proposé deux approches permettant de calculer cette borne sans énumérer tous les points non dominés. Ces deux approches semblent également très prometteuses.

Ces deux dernières problématiques peuvent être généralisées. La représentation des points non dominés peut, par exemple, s'appuyer sur un modèle de préférences construit au préalable avec le décideur. L'objectif serait donc d'obtenir l'ensemble des points non dominés selon la relation de préférence associée au modèle plutôt que selon la relation de dominance. Quant à la problématique de la détermination du point nadir, celle-ci peut se généraliser à l'optimisation d'une fonction linéaire sur l'ensemble des solutions efficaces. Ce problème est particulièrement complexe car plusieurs solutions efficaces peuvent être associées au même point. Les approches optimisant une telle fonction exhibent en général itérativement un point réalisable améliorant la meilleure valeur rencontrée, puis vérifient si ce point est non dominé. Dans le cas contraire, un point non dominé est énuméré, puis la solution efficace associée optimisant la fonction est déterminée. Les algorithmes itèrent ainsi jusqu'à ce qu'aucune solution réalisable n'améliore l'estimation. En conséquence, le nombre de programmes mathématiques à résoudre est sensiblement plus grand, car les approches examinent désormais des points dominés. Nous avons abordé ces deux dernières extensions et une ébauche d'algorithmes les résolvant est proposée en annexe. Enfin, il est important d'observer que nous nous sommes cantonnés aux problèmes d'optimisation discrète, et il serait intéressant d'étudier une possible généralisation aux problèmes d'optimisation mixtes, faisant intervenir à la fois des variables discrètes et continues.

Annexe A.

Modélisation des préférences

A.1. Problématique

Une approche différente de la relation d'ε-dominance consiste à chercher une représentation des points en incorporant les préférences d'un ou plusieurs décideur, le but étant de ne déterminer que les points qui sont à la fois non dominés et tels qu'aucun autre ne soit préféré à eux. Ce travail est le fruit d'une collaboration avec Sami Kaddani et fut l'objet de la publication Kaddani et al. (2016)

Le modèle que nous proposons ici repose sur l'idée que lors de la comparaison de deux points, si l'un est significativement meilleur que l'autre sur des critères *principaux* alors le décideur pourrait le préférer même s'il est moins bon - dans une certaine mesure - sur les autres critères. Il est d'ailleurs intéressant d'observer que si aucun critère n'est principal, ce modèle se ramène à une relation de ε-dominance. De même, si tous les critères sont principaux, alors il s'agit précisément d'une relation de dominance classique.

Nous présentons dans cette section une modification de l'algorithme 3.2 pour tenir compte d'un tel modèle de préférence, et ainsi réduire le nombre de points générés. En revanche, contrairement à la relation de ε-dominance, aucune garantie ne peut être apportée sur l'ensemble final qui peut être potentiellement vide. Certaines précautions doivent donc être prises pour éviter de telles situations. Nous nous intéresserons ici uniquement à la mise en œuvre algorithmique d'un tel modèle. Le lecteur intéressé par les propriétés de cette relation ainsi que par son élicitation peut se référer à la publication Kaddani et al. (2016) ou plus généralement à la thèse de Kaddani (2017).

A.2. Relation de préférence

La définition suivante généralise la notion de dominance :
Définition A.1. *Soit* $\Delta : \mathbb{R}^p \to \mathbb{R}^p$. *La relation* $\preceq_\Delta$ *induite par* Δ *est définie par :*

$$y \preceq_\Delta y' \iff \Delta(y) \preceq y'$$

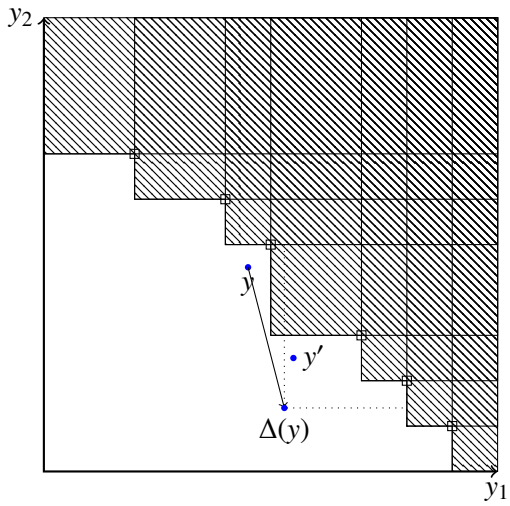


FIGURE A.1. – Généralisation de la dominance usuelle par la relation $\preceq_\Delta$

Comme montré par la Figure A.1, la relation $\preceq_\Delta$ peut être vue comme une translation du cône de dominance dans le plan. Observons que si Δ est l'application identité, $\preceq_\Delta$ correspond précisément à la relation de dominance habituelle. Dans cette section, nous restreindrons notre étude aux applications affines, c'est à dire telles que :

$$\Delta_i(y) = a_i y_i + b_i$$

Observons que si $0 < a_i < 1$ et $b_i = 0$, $\forall i \in [\![1, p]\!]$, la relation $\preceq_\Delta$ se ramène à la relation $\preceq_\varepsilon$ dans le cas où les valeurs des points sont toutes positives. La définition qui suit généralise la notion de *point non dominé*.

Définition A.2. *Soit Y^Δ un ensemble de points vérifiant la propriété suivante :*

$$\forall y \in Y^\Delta, \nexists y' \in Y, : y' \preceq_\Delta y$$

Y^Δ est l'ensemble des points non dominés selon Δ.

Nous nous intéressons ici à l'ensemble des points à la fois non dominés selon $\preceq$ et non dominés selon $\preceq_\Delta$. En d'autres termes, nous nous proposons de déterminer l'ensemble $Y_{ND} \cap Y^\Delta$. D'une manière générale, étant donné n applications Δ^i, nous nous intéressons à la détermination de l'ensemble Y^Δ_{ND} défini par :

$$Y^\Delta_{ND} = Y_{ND} \cap \bigcap_{i=1}^{n} Y^{\Delta^i}$$

Différentes propriétés de cette ensemble ont été étudiées dans Kaddani et al. (2016) et nous ne les détaillerons donc pas dans ce document. Il est toutefois important de noter que Y^Δ_{ND} peut ne contenir aucun élément : il est donc nécessaire d'apporter des garanties supplémentaires sur les applications Δ^i. Nous nous intéresserons dans la section suivante à l'intégration de ces relations dans l'algorithme 3.2.

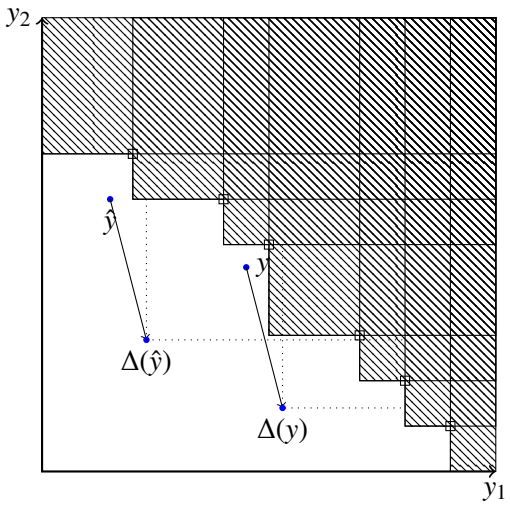


FIGURE A.2. – Les points $\Delta(\hat{y})$ et $\Delta(y)$ sont utilisés pour mettre à jour la région de recherche.

A.3. Génération des points

A.3.1. Une seule relation

Étudions la situation où une seule application Δ est considérée et où nous cherchons à déterminer $Y_{ND}^{\Delta} = Y_{ND} \cap Y^{\Delta}$. L'adaptation de l'algorithme 3.2 que nous proposons exhibe itérativement un point non dominé $y^* \in Y_{ND}$ avant de vérifier s'il est dominé selon $\preceq_{\Delta}$ en résolvant le programme suivant :

$$P_{\Delta}(y^*) \begin{cases} \min & \sum_{i=1}^{p} y_i \\ \text{s.c.} & y \in Y \\ & \Delta(y) \preceq y^* \end{cases}$$

Remarquons que $y^* \in F(P(y^*))$ et peut donc servir de solution de départ.

Si y^* est optimal pour $P_{\Delta}(y^*)$, nous avons $y^* \in Y^{\Delta}$ et donc $y^* \in Y_{ND}^{\Delta}$ et la région de recherche est mise à jour à partir des points y^* et $\Delta(y^*)$ pour retirer les points dominés par y^* selon $\preceq$ et $\preceq_{\Delta}$. Si, au contraire, $P_{\Delta}(y^*)$ a pour optimum $\hat{y} \neq y^*$, alors $\hat{y} \preceq_{\Delta} y^*$. Il est alors nécessaire de retirer deux ensembles de points de la région de recherche :

- L'ensemble $d_{\Delta}(\hat{y})$ des points dominés par $\hat{y}$ selon $\preceq_{\Delta}$.
- L'ensemble $d_{\Delta}(y^*)$ des points dominés par y^* selon $\preceq_{\Delta}$.

Puisque $\hat{y}$ est optimal pour $P_{\Delta}(y^*)$, nous avons $d(y^*) \subset d_{\Delta}(\hat{y})$. La région de recherche est donc mise à jour avec les points $\Delta(\hat{y})$ et $\Delta(y^*)$ (Figure A.2). De plus, si $\hat{y} \notin d_{\Delta}(\hat{y})$ et $\hat{y} \notin d_{\Delta}(y^*)$, le point $\hat{y}$ n'est pas retiré de la région de recherche et peut donc être exhibé à nouveau dans les itérations ultérieures car il peut s'avérer non dominé.

A.3.2. Cas général

La situation est similaire quand n relations $\preceq_{\Delta^i}$ sont considérées. Un point non dominé y^* est exhibé et les n programmes $P_{\Delta^i}(y^*)$ sont résolus. Si y^* est optimal pour chacun d'entre eux, alors $y^* \in Y_{ND}^{\Delta}$ et la région de recherche est mise à jour à partir de y^* et des points translatés $\Delta^i(y^*)$, $i \in [\![1, n]\!]$. Dans le cas contraire, un point $\hat{y}$ est exhibé et nous avons $\hat{y} \preceq_{\Delta^i} y^*$ pour un certain $i \in [\![1, n]\!]$. Il suffit donc de retirer les points dominés par y^* et $\hat{y}$ selon toutes les relations $\preceq_{\Delta^i}$. L'algorithme A.1 synthétise l'approche.

Algorithme A.1 : Extension de la relation de dominance

Input : Ensemble des points réalisables :Y,
Les n relations de dominance : Δ^i, $i \in [\![1, n]\!]$
Output : Ensemble des points non dominés de Y selon $\preceq_{\Delta^i}$: Y_{ND}^{Δ}

```
N ← ∅
U(N) ← {(M, ..., M)}
while U(N) ≠ ∅ do
    Résoudre Π(k, u)
    if Π(k, u) est réalisable then
        y* est l'optimum de Π(k, u)
        Soient ŷ^i l'optimum de P_{Δ^i}(y*), i ∈ [[1, p]]
        if ∃i ∈ [[1, n]] tel que ŷ^i ≠ y* then
            Mettre à jour U(N) avec les points Δ^j(ŷ^i), ∀j ∈ [[1, n]]
            Mettre à jour U(N) avec les points Δ^j(y*), ∀j ∈ [[1, n]]
        else
            Mettre à jour U(N) avec y*
            Mettre à jour U(N) avec les points Δ^j(y*), ∀j ∈ [[1, n]]
            N ← N ∪ {y*}
Y_ND^Δ ← N
```

Annexe B.

Extension à l'optimisation d'une fonction linéaire sur l'ensemble des solutions efficaces

B.1. Introduction

Nous nous sommes focalisés, dans ce document, sur l'optimisation dans l'espace des objectifs et avons étudié plusieurs approches permettant de déterminer l'ensemble des points non dominés ainsi que, pour chacun d'eux, une solution efficace associée. Cependant, nous ne faisons aucune distinction entre les antécédents d'un même point. En réalité, cette assomption est potentiellement fausse, et il peut arriver qu'une solution soit préférée à une autre. Ce *critère supplémentaire* aurait une priorité inférieure aux p autres, mais serait à optimiser dans un deuxième temps parmi toutes les solutions efficaces associées aux points non dominés. Si ce critère est linéaire, on parle d'*optimiser une fonction linéaire* Φ *sur l'ensemble des solutions efficaces*. Si la fonction Φ est une combinaison convexe des objectifs, le problème est trivial : il suffit simplement d'optimiser Φ sur l'ensemble des solutions réalisables. La solution obtenue sera efficace et optimisera Φ. En revanche, dans le cas contraire, le problème se complexifie. Une solution triviale consisterait à énumérer tous les points non dominés, déterminer pour chacun d'eux l'ensemble des solutions efficaces associées, puis à trouver celle optimisant Φ. Pour les raisons évoquées précédemment, cette approche n'est pas satisfaisante, non seulement à cause du coût de la détermination de Y_{ND}, mais également à cause du fait qu'un grand nombre de solutions efficaces peuvent être associées à chaque point. L'objectif est donc de résoudre le problème en considérant le plus petit sous ensemble de solutions efficaces. Il est intéressant d'observer que cette problématique généralise la détermination du point nadir.

Cette problématique a été régulièrement évoquée dans la littérature, et plusieurs personnes ont proposé des algorithmes la résolvant (Abbas et Chaabane, 2006, Boland et al., 2017, Chaabane et Pirlot, 2010, Jorge, 2009). Ces approches ont en commun qu'elles maintiennent des bornes sur les valeurs prises par Φ, cherchent itérativement une solution réalisable améliorant la meilleure valeur connue et déterminent si cette solution est efficace. Nous proposons ici une adaptation simple de l'algorithme 3.2 reposant sur ce schema afin de résoudre cette problématique.

B.2. Résultats préliminaires

Considérant les p objectifs $f_1, \ldots, f_p$ et la fonction $\Phi : X \mapsto \mathbb{R}$, le problème consistant à *optimiser* Φ *sur l'ensemble des solutions efficaces* peut se formaliser de la manière suivante :

$$\Gamma \begin{cases} \min & \Phi(x) \\ \text{s.c.} & f(x) \in Y_{ND} \end{cases}$$

La proposition suivante montre que ce problème est trivial à résoudre si Φ peut s'exprimer comme une combinaison convexe des objectifs.

Proposition B.1. *S'il existe* $\lambda_i \in \mathbb{R}_*^+$, $\forall i \in [\![1, p]\!]$ *tels que* $\Phi(x) = \sum_{i=1}^{p} \lambda_i f_i(x)$*, alors résoudre* Γ *est équivalent à résoudre :*

$$\begin{cases} \min & \Phi(x) \\ s.c. & f(x) \in Y \end{cases}$$

Démonstration. Les optima de ce programme minimisent Φ. De plus, Φ est une fonction strictement monotone sur les objectifs et donc, par le Théorème 1.1, les solutions de ce programme sont efficaces. □

Dans le cas général, néanmoins, optimiser sur les solutions efficaces est un problème difficile. L'approche que nous proposons repose sur la caractérisation explicite utilisée dans l'algorithme 3.2 et s'appuie sur trois problèmes en nombres entiers différents.

Le premier programme retourne un point réalisable améliorant une valeur de Φ précédemment trouvée, notée Φ^U. Il s'agit d'une généralisation du programme Ω utilisé lors de la détermination du point nadir.

$$\Omega(u, \Phi^U) \begin{cases} \min & \sum_{i=1}^{p} f_i(x) \\ \text{s.c.} & x \in X \\ & f(x) \prec u \\ & \Phi(x) < \Phi^U \end{cases}$$

Les solutions de Ω peuvent être dominées et il est donc nécessaire de le vérifier en résolvant un deuxième modèle, partant de l'optimum y^* retourné par Ω.

$$P(y^*) \begin{cases} \min & \sum_{i=1}^{p} y_i \\ \text{s.c.} & y \in Y \\ & y \leqq y^* \end{cases}$$

Il est important de remarquer que P est toujours réalisable, et que son optimum est nécessairement un nouveau point non dominé. En effet, son optimum domine y^* qui domine u.

Enfin, le programme suivant recherche une solution efficace minimisant Φ parmi celles associées à un point non dominé.

$$\Gamma(y)\begin{cases} \min & \Phi(x) \\ \text{s.c.} & x \in X \\ & f(x) = y \end{cases}$$

Nous présentons dans la section suivante comment utiliser ces trois programmes mathématiques pour déterminer un optimum de Φ.

B.3. Optimiser sur les solutions efficaces

L'algorithme que nous proposons maintient à chaque itération une borne supérieure Φ^U des valeurs prises par Φ sur Y_{ND}. À chaque itération, un point - potentiellement dominé- qui améliore Φ^U est exhibé de la région de recherche à l'aide du programme Ω. Le programme P est ensuite utilisé pour vérifier sa Pareto-optimalité en cherchant un point le dominant. Cette approche est synthétisée dans l'algorithme B.1.

Bien que l'algorithme 3.2 et l'algorithme B.1 sont relativement similaires, ce dernier est beaucoup plus coûteux : en effet, l'utilisation du programme Γ implique de considérer toutes les solutions efficaces associées aux points énumérés et cet ensemble peut être significativement grand. De plus, comme pour l'algorithme 5.5, les programmes résolus peuvent être infaisables : il est donc impossible de fournir une solution de départ. Enfin, les règles de réduction ne présenteraient aucun intérêt ici : si l'opérateur lexmin était utilisé dans le programme Ω, nous n'aurions toujours pas la garantie que l'optimum retourné minimise l'objectif principal sur $z(u) \cap Y$ à cause de la contrainte imposant d'améliorer Φ^U. Pour cette raison, nous le remplaçons par une somme pondérée, moins coûteuse à optimiser.

B.4. Conclusion

L'optimisation sur l'ensemble des solutions efficaces, bien que ne retournant qu'un seul point, constitue un problème d'optimisation très complexe.

Cependant, comme illustré ici, cette problématiques est très liée avec celle concernant la détermination de l'ensemble des points non dominés. Les algorithmes développés reposent sur les mêmes notions que sont le Théorème 1.1 et le concept de région de recherche.

Nous avons donc proposé dans ce chapitre une adaptation relativement simple de l'algorithme 3.2 permettant d'optimiser sur l'ensemble des solutions efficaces sans devoir nécessairement considérer l'ensemble des points non dominés ou des solutions efficaces associées.

Algorithme B.1 : Minimisation d'une fonction linéaire sur l'ensemble des solutions efficaces

Input : Ensemble des points réalisables : Y
Objectifs à minimiser : f_i, $i \in [\![1, p]\!]$
Fonction à minimiser sur les solutions efficaces : Φ

Output : Solution efficace minimisant Φ : x^*

$U \leftarrow \{(M, \ldots, M)\}$
$\Phi^U \leftarrow M$
while $U \neq \emptyset$ **do**
    choisir $u \in U$
    résoudre $\Omega(u, \Phi^U)$
    if $\Omega(u, \Phi^U)$ *est infaisable* **then**
        $U \leftarrow U \setminus \{u\}$
    else
        y est l'optimum de $\Omega(u, \Phi^U)$
        /* Recherche d'un point dominant y */
        y^* est l'optimum de $P(y)$
        /* Mise à jour de la région de recherche car y^* est un nouveau point non dominé appartenant à $z(u)$ */
        subdiviser les zones de U avec y^*
        /* Recherche de l'antécédent de y^* minimisant Φ */
        x est une solution optimale de $\Gamma(y^*)$
        /* Translation de la borne si elle est améliorée par x */
        if $\Phi(x) < \Phi^U$ **then**
            $\Phi^U \leftarrow \Phi(x)$
            $x^* \leftarrow x$

Bibliographie

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

M. A. Abo-Sinna et M. L. Hussein. An algorithm for generating efficient solutions of multiobjective dynamic programming problems. *European journal of operational research*, 80(1) : 156–165, 1995.

C. Bazgan, H. Hugot, et D. Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1) :260 – 279, 2009.

C. Bazgan, F. Jamain, et D. Vanderpooten. Discrete representation of the non-dominated set for multi-objective optimization problems using kernels. *European Journal of Operational Research*, 260(3) :814–827, 2017.

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

N. Boland, H. Charkhgard, et M. W. P. Savelsbergh. The *L*-shape search method for triobjective integer programming. *Math. Program. Comput.*, 8(2) :217–251, 2016.

N. Boland, H. Charkhgard, et 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(3) :904–919, 2017.

J. T. Buchanan. A naive approach for solving mcdm problems : The guess method. *Journal of the Operational Research Society*, 48(2) :202–206, 1997.

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

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

V. Chankong et Y. Haimes. *Multiobjective Decision Making : Theory and Methodology (North Holland Series in System Science and Engineering)*. 1983.

C. A. C. Coello, C. Dhaenens, et L. Jourdan. Multi-objective combinatorial optimization : Problematic and context. In *Advances in multi-objective nature inspired computing*, pages 1–21. Springer, 2010.

M. Cornu. *Local Search, data structures and Monte Carlo Search for Multi-Objective Combinatorial Optimization Problems.* PhD thesis, PSL Research University, Paris, France, 2017.

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

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

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

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

M. Ehrgott et X. Gandibleux. Multiobjective combinatorial optimization—theory, methodology, and applications. In *Multiple criteria optimization : State of the art annotated bibliographic surveys*, pages 369–444. Springer, 2003.

M. Ehrgott et D. Tenfelde-Podehl. Computation of ideal and nadir values and implications for their use in MCDM methods. *European Journal of Operational Research*, 151(1) :119 – 139, 2003.

J. R. Figueira, G. Tavares, et M. M. Wiecek. Labeling algorithms for multiple objective integer knapsack problems. *Computers & Operations Research*, 37(4) :700–711, 2010.

Y. Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3) :296–297, 1971.

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

S. Kaddani. *Partial preference models in discrete multi-objective optimization*. PhD thesis, PSL Research University, Paris, France, 2017.

S. Kaddani, S. Tamby, D. Vanderpooten, et J. M. Vanpeperstraete. Partial preference models using translated cones in discrete multi-objective optimization. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–8. IEEE, 2016.

H. Kaplan, N. Rubin, M. Sharir, et E. Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3) :982–1011, 2008.

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

G. Kirlik et S. Sayın. Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization*, 62(1) :79–99, 2015.

K. Klamroth, R. Lacour, et D. Vanderpooten. On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245(3) :767–778, 2015.

D. Klein et E. L. Hannan. An algorithm for the multiple objective integer linear programming problem. *European Journal of Operations Research*, 93, 378-385, 1982.

M. Köksalan et B. Lokman. Finding nadir points in multi-objective integer programs. *Journal of Global Optimization*, 62(1) :55–77, 2015.

R. Lacour. *Exact and approximate solving approaches in multi-objective combinatorial optimization, application to the minimum weight spanning tree problem.* PhD thesis, Université Paris Dauphine, 2014.

M. Laumanns, L. Thiele, et E. Zitzler. An adaptive scheme to generate the pareto front based on the epsilon-constraint method. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.

A. Liefooghe, J. Humeau, S. Mesmoudi, L. Jourdan, et E.-G. Talbi. On dominance-based multiobjective local search : design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18(2) :317–352, 2012.

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

E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2) :236–245, 1984.

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

K. Miettinen, P. Eskelinen, F. Ruiz, et M. Luque. Nautilus method : An interactive technique in multiobjective optimization based on the nadir point. *European Journal of Operational Research*, 206(2) :426 – 434, 2010.

M. Özlen et M. Azizoğlu. Multi-objective integer programming : a general approach for generating all non-dominated solutions. *European Journal of Operational Research*, 199(1) :25–35, 2009.

Ö. Özpeynirci. On nadir points of multiobjective integer programming problems. *Journal of Global Optimization*, 69(3) :699–712, 2017.

A. Przybylski et X. Gandibleux. Multi-objective branch and bound. *European Journal of operational research*, 260(3) :856–872, 2017.

A. Przybylski, X. Gandibleux, et 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.

T. K. Ralphs, M. J. Saltzman, et M. M. Wiecek. An improved algorithm for solving biobjective integer programs. *Annals of Operations Research*, 147(1) :43–70, 2006.

A. Rong, K. Klamroth, et J. R. Figueira. Multicriteria 0-1 knapsack problems with k-min objectives. *Computers & Operations Research*, 40(5) :1481–1496, 2013.

Y. Sawaragi, H. Nakayama, et T. Tanino. *Theory of multiobjective optimization*, volume 176. Elsevier, 1985.

S. Sayın. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87(3) :543–560, 2000.

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

M. Stiglmayr, J. R. Figueira, et K. Klamroth. On the multicriteria allocation problem. *Annals of Operations Research*, 222(1) :535–549, 2014.

J. Sylva et 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.

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

E.-G. Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009.

W. Zhang et 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.

Liste des Algorithmes

Liste des tableaux

Table des figures

Résumé

Le thème central de cette thèse est la détermination de l'ensemble des points non dominés de problèmes d'optimisation discrète multiobjectifs. Ce problème est réputé difficile en raison de la cardinalité potentiellement très grande de cet ensemble. Après avoir introduit un certain nombre de résultats fondamentaux, nous présentons une analyse critique de différents algorithmes représentatifs de l'état de l'art. Nous proposons ensuite une nouvelle méthode, présentant certaines propriétés remarquables, dont nous montros expérimentalement qu'elle est sensiblement meilleure que les algorithmes récents de la littérature.
Nous nous intéressons également à la détermination d'une description concise de l'ensemble des points non dominés à travers le concept d'approximation. L'introduction d'une tolérance permet en effet de ne pas énumérer les points très similaires. La cardinalité de l'ensemble généré, et par conséquent le temps de calcul, s'en trouvent très notablement réduits, tout en garantissant *a priori* la qualité de cet ensemble.
Enfin, nous proposons différents algorithmes pour déterminer le *point nadir*, qui est le point bornant les plus mauvaises performances atteintes par les points non dominés sur chaque critère. Ce problème très classique est réputé difficile à partir de trois critères. En s'appuyant sur différents résultats, nous proposons deux algorithmes très performants : l'un adapté au cas triobjectif tirant parti des spécificités de ce cas, l'autre dédié au cas général.

Mots Clés

optimisation multiobjectif, optimisation discrète, ensemble non dominé, ε-contrainte

Abstract

The central topic of this thesis is the determination of the set of non-dominated points of discrete multi-objective optimization problems. This problem is considered difficult because of the potentially very large cardinality of this set.
After introducing some fundamental results, we present a critical analysis of different algorithms representative of the state of the art. We then propose a new method, presenting some remarkable properties, which is shown experimentally to be significantly better than the recent algorithms of the literature.
We are also interested in determining a concise description of all non-dominated points through the concept of approximation. The introduction of a tolerance makes it possible not to enumerate very similar points. The cardinality of the generated set, and therefore the computation time, are very significantly reduced, while guaranteeing *a priori* the quality of this set.
Finally, we propose different algorithms to determine the *nadir point*, which is the point bounding the worse values reached by the non-dominated points. This very classic problem is known to be difficult when the number of objectives exceeds three. Based on different results, we propose two very efficient algorithms: one adapted to the triobjective case taking advantage of the specificities of this case, the other dedicated to the general case.

Keywords

multicriteria optimization, discrete optimization, nondominated set, ε-constraint