Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires

Julien Jorge

UNIVERSITÉ DE NANTES

UFR SCIENCES ET TECHNIQUES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE MATHÉMATIQUES

2010

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Julien JORGE

le 11 mai 2010

LINA

Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires

Jury

Président	:	Pr. Jin-Kao HAO	Université d'Angers
Rapporteurs	:	Pr. Daniel TUYTTENS, Professeur	Université de Mons, Belgique
		Pr. Vincent T'KINDT, Professeur	Université de Tours
		Pr. Mhand HIFI, Professeur	Université de Picardie
Examinateurs	:	M. Fabien LEHUEDE, Chargé de recherche	École des Mines de Nantes

Directeur de thèse : Pr. Xavier GANDIBLEUX
Laboratoire : LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE.
2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED 0366-...

NOUVELLES PROPOSITIONS POUR LA RÉSOLUTION EXACTE DU SAC À DOS MULTI-OBJECTIF UNIDIMENSIONNEL EN VARIABLES BINAIRES

New propositions for the exact solution of the unidimensional multi-criteria knapsack problem with binary variables

Julien JORGE

⋈

favet neptunus eunti

Université de Nantes

Julien JORGE

Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires

xii+178 p.

Ce document a été préparé avec LaTeX2e et la classe `these-LINA` version v. 1.30 de l'association de jeunes chercheurs en informatique LOGIN, Université de Nantes. La classe `these-LINA` est disponible à l'adresse :

`http://login.irin.sciences.univ-nantes.fr/`

Impression : these.tex – 1/6/2010 – 10:14

Révision pour la classe : these-LINA.cls,v 1.30 2005/08/16 17:01:41 mancheron Exp

« La seule révolution possible, c'est d'essayer de s'améliorer soi-même, en espérant que les autres fassent la même démarche. Le monde ira mieux alors. »

— Georges Brassens

Résumé

Ce travail porte sur la résolution exacte d'un problème d'optimisation combinatoire multi-objectif. Nous cherchons d'une part à confirmer l'efficacité de l'algorithme dit en deux phases, et d'autre part à poser une généralisation des procédures de séparation et évaluation, populaires dans le cadre mono-objectif mais presque absentes en multi-objectif. Notre étude s'appuie sur le problème multi-objectif de sac à dos unidimensionnel en variables binaires. Ce dernier est un classique de l'optimisation combinatoire, présent comme sous problème dans de nombreux problèmes d'optimisation.

La première partie de nos travaux porte sur un pré-traitement permettant de réduire la taille d'instances de ce problème. Nous mettons en évidence plusieurs propriétés permettant de déterminer *a priori* une partie de la structure de toutes les solutions efficaces. Nous nous attachons ensuite à décrire une procédure performante de type deux phases pour ce problème, tout d'abord dans le cas bi-objectif, où nous améliorons la procédure décrite par Visée *et al.* en 1998. Puis nous proposons un nouvel algorithme permettant de trouver plus efficacement les solutions recherchées durant la seconde phase. Nous étendons ensuite cette procédure pour des instances ayant trois objectifs ou plus. Les résultats obtenus sont comparés aux meilleurs algorithmes existants pour ce problème et confirment l'efficacité de l'approche en deux phases. La dernière partie de notre travail concerne la généralisation au cas multi-objectif d'une procédure de séparation et évaluation. Nous identifions plusieurs difficultés auxquelles nous répondons en proposant deux nouvelles procédures. Les expérimentations numériques indiquent que ces dernières permettent de résoudre des instances en des temps raisonnables, bien qu'elles n'atteignent pas les performances d'une procédure de type deux phases.

Mots-clés : optimisation combinatoire multi-objectif, problème de sac à dos unidimensionnel en variables binaires, résolution exacte, pré-traitement, méthode en deux phases, procédure de séparation et évaluation

Abstract

The purpose of this work is the exact solution of a problem from the field of multi-criteria combinatorial optimisation. Our goal his twofold. First, we aim at confirming the efficiency of the so-named two-phases algorithms. Then, we set a generalisation of the branch and bound procedures, popular in the mono-criteria case but almost non-existent in the multi-criteria case. Our work is based on the unidimensional multi-criteria knapsack problem with binary variables, a classic from combinatorial optimisation, found as a sub problem in many optimisation problems.

The first part concerns the reduction of the instances of the problem. We expose several properties allowing to *a priori* find some parts of the structure of all efficient solutions. Then, we describe an efficient two-phases procedure for this problem. Initially in the bi-criteria case, we improve the original procedure from Visée *et al.* (1998) before defining a new procedure to efficiently find the solutions in the second phase. This algorithm is extended to the tri- and multi-criteria case in the next part. Finally, the generalisation of the branch and bound procedure is the last part of our work. We focus on several difficulties, to which we answer with two new procedures. Numerical experiments show that these procedures can solve instances in acceptable time. Nevertheless, the two-phases algorithms outperform these procedures, just like the best known procedures for this problem.

Keywords: multi-objective combinatorial optimisation, unidimensional binary knapsack problem, exact resolution, preprocessing, two phase method, branch and bound

Remerciements

Je remercie en premier lieu monsieur Xavier Gandibleux, professeur à l'université de Nantes, pour avoir organisé et dirigé cette thèse. Mon profond respect va aussi à messieurs Jin-Kao Hao, professeur à l'université d'Angers, président du jury et examinateur et Fabien Lehuédé, maître de conférences à l'école des mines de Nantes, examinateur ; ainsi qu'à messieurs Mhand Hifi, professeur à l'université de Picardie, Daniel Tuyttens, professeur à l'université de Mons, en Belgique, et Vincent T'kindt, professeur à l'université de Tours, pour avoir accepté d'être rapporteurs de mon travail.

Une partie de mon travail n'aurait peut-être pas vu le jour sans l'apport de monsieur Anthony Przybylski, maître de conférences à l'université de Nantes, que je remercie à plusieurs égards. Tout d'abord pour m'avoir aidé à libérer ma réflexion des particularités des problèmes bi-objectifs, et d'autre part pour ses relectures rapides et ses retours pertinents lors de la rédaction de ce manuscrit. Bien que son nom n'apparaisse pas en couverture, Anthony a tenu un rôle semblable à celui d'un co-encadrant.

Je tiens aussi à remercier ci-après plusieurs personnes dont l'existence influence la mienne et qui ont donc, dans une certaine mesure, une part de responsabilité dans l'aboutissement de ce travail.

Florian Massuyeau est un ami et un membre de la famille que j'apprécie en particulier pour son esprit critique et sa manie de ne pas être d'accord avec celui qui parle ; pour nos intéressantes discussions sur la science, la musique et la vie, et pour les éclats de rire partagés quotidiennement.

Voilà plusieurs années que Sébastien Angibaud et moi-même travaillons sur des projets communs, années pendant lesquelles une sincère amitié s'est développée. Par son implication et nos discussions sur ces projets, Sébastien est une réelle source de motivation.

Julien Lassalle m'a appris à user du cognitif au profit de l'amusement et de l'originalité. C'est un ami aujourd'hui géographiquement distant mais toujours omniprésent avec lequel je partage le même sentiment sur l'existence, l'humanité et le sens de la vie en général.

Je ne remercierai jamais assez Marine Planson, sens de ma vie, femme de ma vie, pour son soutien et son naturel, pour son regard confiant. Marine me fait aimer mon quotidien et me donne envie d'être meilleur.

Je pense aussi bien sûr à mes parents et à mes sœurs, qui ont été présents depuis toujours, aux amis éloignés Benoît et Antoine, et aux collègues et amis qui m'ont accompagné ces dernières années.

Sommaire

Introduction

L'optimisation combinatoire est un domaine intensivement étudié en raison de son potentiel d'application à de nombreux problèmes rencontrés dans des situations réelles. Les recherches dans ce domaine sont longtemps restées confinées aux situations où un unique objectif est à optimiser. Cependant, de nombreux cas réels nécessitent l'optimisation simultanée de plusieurs objectifs conflictuels. Bien que des travaux sur le sujet aient été publiés au moins depuis les années 1970, l'intérêt de la communauté scientifique pour l'optimisation combinatoire multi-objectif n'apparaît réellement qu'à partir des années 1990. Ce regain d'intérêt trouve son origine dans le développement de nombreuses méta-heuristiques multi-objectifs, appliquées avec succès à des problèmes d'optimisation complexes. En fait, la résolution exacte de problèmes d'optimisation combinatoire multi-objectifs est intrinsèquement difficile, tant d'un point de vue théorique que pratique. En effet, ils sont tous $\mathcal{N}P$-complets ou $\mathcal{N}P$-difficiles, $\#P$-complets et intraitables, même lorsque la version mono-objectif du problème appartient à la classe de complexité $\mathcal{P}$. De plus, la plupart des méthodes de résolution exacte pour les problèmes d'optimisation combinatoires multi-objectifs sont souvent limitées au cas bi-objectif.

Nos travaux portent sur la résolution exacte de problèmes d'optimisation combinatoire multi-objectif. En particulier, nous nous intéressons aux schémas de résolution de ces problèmes avec d'un côté une méthode spécifique au cas multi-objectif et de l'autre une méthode d'exploration généralisée du cas mono-objectif vers le cas multi-objectif.

Les travaux de Ulungu [133] sont souvent cités en référence sur le sujet de l'optimisation combinatoire multi-objectif. Celui-ci a observé que les travaux sur la résolution de certains problèmes mono-objectifs avait produit des procédures très efficaces, profitant à chaque fois de la structure particulière du problème considéré. Ulungu [133] a cherché à exploiter la structure de ces problèmes dans le contexte multi-objectif, ce qui a mené au développement de la méthode dite en deux phases. Comme son nom l'indique, cette méthode procède au calcul des solutions efficaces en deux phases. La première calcule les solutions dites supportées tandis que la seconde détermine les solutions dites non supportées. Initialement appliquée au problème bi-objectif de sac à dos, l'auteur utilise une procédure de séparation et évaluation pour le calcul des solutions dans chaque phase. Elle a depuis été appliquée à de nombreux autres problèmes d'optimisation combinatoire multi-objectif. Cependant, les expérimentations numériques montrent que cette méthode peine à passer l'augmentation de la taille des instances résolues. De plus, elle est restée limitée au cas bi-objectif jusqu'aux récents travaux de Przybylski [111]. Ce dernier a proposé une généralisation de la méthode à trois objectifs et plus ainsi qu'une nouvelle procédure de calcul des solutions non supportées. Les expérimentations numériques de ses travaux appliqués au problème multi-objectif d'affectation montrent que la procédure en deux phases de Przybylski [111], utilisant une procédure dite de *ranking* pour le calcul des solutions non supportées, surpasse les méthodes précédentes pour ce problème. En effet, l'application d'une procédure de *ranking* permet d'explorer l'espace de recherche d'une manière performante, en découvrant les solutions dans un ordre favorisant les plus efficaces.

Du fait qu'elle s'appuie fortement sur la structure du problème à résoudre, il suffit d'une petite variation dans la formulation du problème pour qu'une procédure en deux phases perde son efficacité, voire

qu'elle ne puisse plus être appliquée. Il est alors nécessaire de s'orienter vers une méthode générique, par exemple de type ϵ-contrainte. Dans le cas de l'optimisation combinatoire mono-objectif, les procédures de séparation et évaluation (PSE) donnent de très bons résultats en tant que méthodes générales. Il est alors étonnant de constater que leur application au cas multi-objectif ne soit pas plus populaire. En effet, seules deux contributions [17, 126] portent sur de telles méthode pour la résolution de problèmes bi-objectifs, respectivement pour un problème d'ordonnancement et une application au problème d'arbre couvrant de poids minimal.

Nos travaux se situent au carrefour de ceux de Przybylski [111] et de ceux de Sourd et Spanjaard [126]. En nous appuyant sur le problème de sac à dos multi-objectif unidimensionnel en variables binaires, nous mettons en vis-à-vis l'application d'une procédure en deux phases et celle d'une procédure de séparation et évaluation. Nous cherchons d'une part à confirmer l'efficacité du schéma de type deux phases dans le cadre multi-objectif, et d'autre part à poser une généralisation à ce cadre des PSE, procédures classiques en optimisation mono-objectif.

Le problème de sac à dos est intensément étudié dans sa version mono-objectif depuis plus d'un siècle. L'intérêt qui lui est porté est en particulier dû au fait qu'il se retrouve en tant que sous problème dans de nombreux problèmes d'optimisation. Ainsi, c'est tout naturellement que de nombreux travaux ont été effectués pour sa version multi-objectif. Deux procédures se démarquent pour la résolution de ce problème. D'une part, Visée *et al.* [136] ont proposé une procédure en deux phases basée sur les travaux de Ulungu [133] et utilisant intensément une PSE mono-objectif de Martello et Toth [92]. Leur algorithme est en pratique restreint au cas bi-objectif et est dominé en terme de performances par les résultats récemment obtenus par Bazgan *et al.* [10]. Ces derniers ont proposé un algorithme très efficace, de type programmation dynamique, pour résoudre des instances du problème ayant jusqu'à trois objectifs. Cependant, bien qu'efficace en terme de temps de résolution, cette approche souffre des mêmes difficultés que son équivalent pour le problème mono-objectif. Elle demande énormément de mémoire et peine à résoudre les instances de plus grandes tailles. Il est à noter que le problème de sac à dos peut être interprété comme un problème de plus courts chemins, ce qui permet de lui appliquer des algorithmes spécifiques à ce dernier. Captivo *et al.* [22] ont en effet précédemment proposé une telle approche pour résoudre des instances du problème de sac à dos multi-objectif. Les performances de leur algorithme se situent entre celles de la procédure de Visée *et al.* [136] et de celle de Bazgan *et al.* [10].

Un point clé de l'efficacité des procédures appliquées au problème de sac à dos mono-objectif se trouve dans la détermination *a priori* d'éléments présents ou absents des solutions efficaces. Cela est effectué par une procédure de réduction du problème, procédure qui ne s'adapte pas immédiatement au cas multi-objectif. À notre connaissance, seuls Gomes da Silva *et al.* [64] ont proposé une telle procédure pour le problème de sac à dos bi-objectif, fondée sur des transformations du problème vers sa version mono-objectif.

En pratique, la procédure de programmation dynamique de Bazgan *et al.* [10] est aujourd'hui la plus performante pour résoudre des instances du problème de sac à dos multi-objectif. Cependant, étant donné ses difficultés à passer au delà de l'augmentation de leurs tailles, il est encore pertinent de chercher une meilleure procédure, possédant ses qualités en terme de temps de résolution, sans les inconvénients de la consommation de mémoire. Przybylski [111] a montré que le schéma des procédures en deux phases associé à une procédure de type *ranking* pour calculer les solutions non supportées était un bon candidat pour cela. De plus, Eppstein [45] a proposé une procédure de *ranking* pour le problème des k plus courts chemins, de fait applicable au problème de sac à dos mono-objectif.

La généralisation des PSE au cas multi-objectif n'est pas triviale. Ainsi, bien qu'elles soient parmi les algorithmes les plus performants pour le problème mono-objectif de sac à dos, aucune n'a été proposée pour le cas multi-objectif de ce problème. Les difficultés se retrouvent sur plusieurs points. D'une part, pour ce problème en particulier, un ordre pour l'exploration est aisément obtenu dans le cas mono-objectif, permettant d'accéder rapidement aux solutions optimales. Cet ordre est absent dans le cas multi-objectif. D'autre part, une autre difficulté pointée par Sourd et Spanjaard [126] se trouve dans la partie évaluation, consistant à juger si l'exploration en cours doit continuer ou être abandonnée. Alors que cette évaluation peut se limiter à déterminer si la recherche se dirige au delà d'une certaine valeur dans $\mathbb{R}$ pour le cas mono-objectif, elle se généralise au cas multi-objectif en déterminant si l'exploration se dirige, au moins en partie, dans une région de $\mathbb{R}^p$ décrite d'une manière non triviale, où p le nombre d'objectifs.

Une piste pour passer outre la difficulté venant de l'ordre dans lequel s'effectue l'exploration se trouve dans diverses généralisations de celui utilisé en mono-objectif. Bazgan *et al.* [10] ont observé que leur algorithme était plus rapide avec certains ordres qu'avec d'autres. Leurs résultats encouragent ainsi l'étude d'un tel procédé. Une autre façon de contourner cette difficulté est d'effectuer l'exploration d'une manière indépendante de l'ordre initial dans lequel elle se présente, en l'adaptant en fonction de ce qui a été observé jusqu'alors. À notre connaissance, aucune proposition n'a déjà été faite dans ce sens.

Sourd et Spanjaard [126] ont proposé de résoudre la difficulté venant de l'évaluation en utilisant une description de l'espace de recherche basée sur la notion d'ensembles bornant inférieurement et supérieurement, tels que définis par Ehrgott et Gandibleux [41]. Au cours d'une PSE, le premier de ces ensembles se détermine immédiatement depuis les résultats déjà obtenus durant l'exploration. Le second est par contre plus difficile à déterminer efficacement. Le calcul d'un ensemble bornant supérieurement permet d'évaluer, de manière optimiste, les résultats pouvant être obtenus en continuant l'exploration. Idéalement, il doit être rapide à calculer et au plus proche de la réalité. Sourd et Spanjaard [126] proposent pour cela de calculer les solutions supportées restreintes à l'espace à explorer. Cela se prête bien au problème d'arbre couvrant minimal mais peut être beaucoup plus coûteux pour d'autres problèmes. Ainsi, un calcul plus rapide, même s'il est éventuellement moins précis, sera parfois préférable.

Dans la suite de ce mémoire, nos travaux sont organisés autour de six chapitres.

Le premier chapitre présente le domaine de l'optimisation combinatoire multi-objectif et pose les principales définitions et propriétés. Les algorithmes classiques de résolution sont présentés et les principaux manques et difficultés, tant pratiques que théoriques, sont identifiés.

Le second chapitre présente le problème de sac à dos et ses variantes, en particulier la version unidimensionnelle, multi-objectif et à variables binaires, autour de laquelle la suite de ce document s'articule. Les méthodes classiques de résolution y sont décrites, autant pour la version mono-objectif du problème que pour le cas multi-objectif.

Nos travaux sur la réduction d'instances du problème sont présentés dans le troisième chapitre. La notion de variables régulières y est définie, puis plusieurs propriétés visant à déterminer ces variables sont posées. Leur efficacité est évaluée au regard de nombreuses instances extraites de la littérature sur le problème, que nous réutiliserons pour les chapitres suivants. Ces propriétés s'appuient sur des bornes connues sur la cardinalité des solutions efficaces, définies par Glover [62] puis généralisées au cas multi-objectif par Gandibleux et Fréville [53].

Le quatrième et le cinquième chapitre portent sur l'application d'une procédure en deux phases au problème de sac à dos. Le premier d'entre eux se concentre sur le cas bi-objectif, où la procédure présentée par Visée *et al.* [136] est améliorée par l'utilisation de bornes plus précises durant l'exploration.

Une procédure de *ranking* inspirée de Eppstein [45] est ensuite proposée pour le calcul des solutions non supportées, en remplacement de la PSE utilisée par Visée *et al.* [136]. Le second de ces chapitres porte quant à lui sur le cas du problème de sac à dos à trois objectifs et plus. Nous y proposons une adaptation de la procédure décrite par Przybylski [111].

Le sixième et dernier chapitre concerne l'application d'une PSE pour le problème de sac à dos multi-objectif. Nous proposons une première procédure, adaptée d'un algorithme classique utilisé dans le cas mono-objectif. Plusieurs ordres sont proposés pour l'exploration, ainsi que plusieurs fonctions d'évaluation. L'influence de chaque ordre et les performances de chaque fonction sont alors évaluées au regard de nombreuses instances. Nous proposons ensuite une seconde procédure dans l'optique de réduire l'influence de l'ordre initial ainsi que la taille de l'espace exploré. Cette procédure est évaluée sur les mêmes instances. Ces deux procédures sont comparées entre elles et avec l'algorithme en deux phases obtenu à l'issue du chapitre 5.

Notations

Notation	Signification
n	nombre de variables d'un problème d'optimisation
m	nombre de contraintes d'un problème d'optimisation
p	nombre de fonctions objectifs d'un problème d'optimisation multi-objectif
$x = (x_1, \ldots, x_n)$	solution, vecteur de valeurs pour les variables
$z(x) = (z_1(x), \ldots, z_p(x))$	vecteur des fonctions objectifs
$y = (y_1, \ldots, y_p)$	point, vecteur de valeurs pour les fonctions objectifs
X	ensemble réalisable d'un problème d'optimisation
$Y = z(X)$	ensemble réalisable dans l'espace des objectifs
y^I	point idéal
y^N	point nadir
y^U	point utopique
$y^1 > y^2$	$y^1_k > y^2_k$ pour $k = 1, \ldots, p$
$y^1 \geqq y^2$	$y^1_k \geq y^2_k$ pour $k = 1, \ldots, p$
$y^1 \geq y^2$	$y^1 \geqq y^2$ et $y^1 \neq y^2$
$\mathbb{R}^p_>$	$\{y \in \mathbb{R}^p : y > 0\}$
$\mathbb{R}^p_\geq$	$\{y \in \mathbb{R}^p : y \geq 0\}$
$\mathbb{R}^p_\geqq$	$\{y \in \mathbb{R}^p : y \geqq 0\}$
$\lambda \in \mathbb{R}^p_>$	poids

Notation	Signification
X_E	ensemble complet de solutions efficaces
X_{E_m}	ensemble complet minimal de solutions efficaces
X_{E_M}	ensemble complet maximal de solutions efficaces
X_{SE}	ensemble complet de solutions supportées
X_{SE_m}	ensemble complet minimum de solutions supportées
X_{SE_M}	ensemble complet maximum de solutions supportées
X_{SE1}	ensemble complet de solutions supportées extrêmes
X_{SE1_m}	ensemble complet minimum de solutions supportées extrêmes
X_{SE1_M}	ensemble complet maximum de solutions supportées extrêmes
X_{SE2}	ensemble complet de solutions supportées non-extrêmes
X_{SE2_m}	ensemble complet minimum de solutions supportées non-extrêmes
X_{SE2_M}	ensemble complet maximum de solutions supportées non-extrêmes
Y_N	ensemble des points non dominés
Y_{SN}	ensemble des points supportés
Y_{SN1}	ensemble des points supportés extrêmes
Y_{SN2}	ensemble des points supportés non-extrêmes
Y_{NN}	ensemble des points non supportés
S_N où $S \subset \mathbb{R}^p$	$S_N := \{s \in S : \nexists s' \in S : s' \geq s\}$
S^c où $S \subset \mathbb{R}^p$	complémentaire de S dans $\mathbb{R}^p$
$\mathrm{rint}(S)$	intérieur relatif de S
$\mathrm{adh}(S)$	adhérance de S
$\mathrm{conv}(S)$	enveloppe convexe de S

CHAPITRE 1

Problèmes d'optimisation combinatoire multi-objectif

La recherche opérationnelle est une discipline scientifique située au carrefour de l'informatique, des mathématiques appliquées et de l'économie. Elle s'adresse entre autres à la résolution de problèmes d'optimisation et fait largement usage d'outils informatiques.

Si certains problèmes peuvent s'exprimer facilement à l'aide d'un formalisme issu de la programmation mathématique, par exemple, ils peuvent se révéler difficiles à résoudre en pratique. Cette difficulté apparait encore plus grande lorsque le problème véhicule plusieurs critères (ou objectifs), éventuellement conflictuels.

Il n'existe pas aujourd'hui de méthode générale efficace pour résoudre ces problèmes. Néanmoins, pour de nombreuses classes de problèmes, il existe des algorithmes spécifiques donnant de bons résultats.

1.1 Introduction à la recherche opérationnelle

En amont de toute méthode de résolution, il y a un problème posé par celui que l'on nomme *le décideur*. Il s'agit de la personne, l'entreprise ou, plus généralement, l'entité qui fait face au problème. Le décideur n'est pas un formalisme mathématique. Il est incertain, fait preuve d'imprécisions et est incomplet dans la description du problème. Ainsi, il n'est pas nécessairement aisé pour lui d'exprimer son problème, ni les critères d'optimalité ; c'est-à-dire ce qui rend une solution meilleure qu'une autre. Pour ces raisons, de nombreux sujets de recherche s'adressent spécifiquement à la modélisation des préférences du décideur, afin de l'aider à exprimer ses souhaits et à choisir parmi les solutions.

La chemin allant de l'expression des souhaits du décideur jusqu'à la présentation des solutions est décomposé en quatre grandes étapes, composantes du domaine de l'aide à la décision. La première étape est la structuration du problème, suivie par sa modélisation. Viennent ensuite sa résolution puis l'interprétation des résultats obtenus. L'étape de la résolution du problème nous intéressera particulièrement par la suite.

Afin de présenter le domaine de façon accessible, la section suivante rapporte une histoire illustrant la place de la recherche opérationnelle dans notre quotidien. La situation souligne la difficulté du cheminement entre le problème initial et la solution désirée, depuis l'expression du problème jusqu'à la sélection

d'une solution, en passant par les difficultés numériques et l'intérêt d'avoir une méthode adaptée à un problème.

1.1.1 « C'est facile, il suffit de prendre l'optimal »

Notre décideur souhaite aller de Nantes à Montaigu, aussi vite que possible. Intuitivement, il prend son atlas et cherche le plus court chemin reliant les deux villes, à vue de nez. Il construit son itinéraire et prend la route.

Arrivé à Montaigu, il retrouve son voisin de palier, qu'il avait croisé sur le parking lors de son départ. Celui-ci est arrivé depuis dix minutes. Étonné, le décideur l'interpelle.

— Je pensais avoir trouvé le chemin le plus court, par où es-tu passé ?

— J'ai pris l'autoroute, répond-il.

— Mais c'est dix kilomètres de plus !

— Exact, mais je peux rouler plus vite.

Notre décideur est surpris en découvrant ainsi qu'il n'avait pas formulé son problème correctement. En effet, le trajet le plus court n'est pas nécessairement le plus rapide. « La route que je prendrai la prochaine fois sera la meilleure à tout point de vue », pense-t-il en ouvrant son atlas. Il commence à regarder les routes reliant Nantes et Montaigu. Certaines sont clairement mauvaises, mais il en reste beaucoup qui semblent intéressantes, trop pour pouvoir être examinées manuellement.

Insatisfait du résultat qu'il a obtenu jusqu'ici, il décide de s'aider de son ordinateur pour trouver le meilleur chemin. Il se rend sur *Open Street Map* * et récupère les données des routes des Pays de la Loire. Il note, pour chaque section de route, sa longueur en kilomètres et la vitesse maximale autorisée (ainsi, il pourra calculer le temps nécessaire pour la parcourir). Il note aussi le coût de la section en terme de carburant et péages.

Le décideur écrit rapidement un petit programme pour trouver le meilleur chemin sur la carte. Le programme considère les sections de chaque route les unes à la suite des autres. Pour chacune, il sépare la recherche en deux : d'un côté la section est ajoutée à la suite de celles déjà choisies et, de l'autre, elle est simplement ignorée. Dans les deux cas, chaque sections restantes est ensuite considéré de la même façon. Les meilleures solutions sont gardées pour être affichées à la fin du traitement.

Il lance son programme et patiente en réfléchissant. Voilà deux heures que le programme tourne et aucune solution n'est affichée. « Bizarre, je pensais que ça irait plus vite. Voyons donc, la carte décrit une centaine de sections de route. Chaque section n'a que deux états : soit elle est sélectionnée, soit elle ne l'est pas. Deux états pour la première section, puis deux pour la seconde et ainsi de suite, ça me fait $2 \times 2 \times \ldots \times 2 = 2^{100}$ solutions possibles. Disons que mon ordinateur traite un milliard de solutions par seconde, j'aurais le résultat dans… quatre cent milliards de siècles ! C'est bien plus que l'âge de l'univers † ! »

« Je ferais mieux d'améliorer mon algorithme », pense-t-il en lançant une petite recherche sur internet. Il y découvre l'algorithme de Martins [93], un programme qui cherche efficacement les meilleurs chemins selon plusieurs critères. « Exactement ce qu'il me faut », dit-il en souriant. « J'entre les données dans le programme, je valide et… cinq cent vingt-quatre solutions ! C'est beaucoup trop. »

En observant les résultats de plus près, le décideur remarque plusieurs choses. D'une part, les routes ne sont pas toutes égales. Certaines sont rapides et chères tandis que d'autres sont longues mais économiques. D'autre part, il y a beaucoup de routes équivalentes, c'est-à-dire qui ont la même durée et le même coût. Ces dernières ne diffèrent finalement que peu, principalement dans les villes traversées où,

∗. `http://www.openstreetmap.fr`, un site de cartographie libre

†. environ 13,7 milliard d'années

du fait de la densité des rues, plusieurs chemins équivalents permettent d'atteindre un lieu donné. « Je n'ai pas besoin de routes équivalentes », pense-t-il. Puis il demande à son ordinateur de ne garder qu'un trajet pour toute paire de valeur pour la durée et le coût. Une fois le filtrage effectué, il ne reste que cinq routes. « Comment vais-je choisir la meilleure ? »

La réponse à cette dernière question est qu'aucune des cinq solutions finales n'est « la » meilleure solution. En effet, les objectifs définis par le décideur sont conflictuels. Certaines routes sont moins chères que d'autres, mais sont plus longues à parcourir. Elles sont incomparables. Ainsi, chacune des solutions finales doit être considérée comme optimale. À partir d'ici, il faut injecter des informations subjectives, les préférences du décideur, pour l'orienter vers un choix. Connaissant maintenant les limites, il peut étudier ce qu'il est prêt à perdre sur un critère pour gagner sur l'autre ; c'est à lui d'arbitrer pour choisir la solution qu'il préfère, selon son expérience, son intuition, ou d'autres critères qui sont extérieurs au problème initial de plus court chemin.

1.1.2 Quelques problèmes classiques

L'histoire ci-dessus fait intervenir le problème classique du plus court chemin. Il s'agit d'un problème facile lorsque les routes sont évaluées au regard d'un seul critère et relativement facile avec plusieurs critères, bien qu'il existe des situations intraitables, comme illustré plus loin, dans l'exemple 1.8 page 19. Les exemples ci-dessous présentent quelques problèmes classiques.

Exemple 1.1 (Problème de plus court chemin). *Le dessin de la figure 1.1 représente des villes, sous forme de ronds, reliées par des routes. La longueur de chaque route est indiquée à côté de celle-ci. Le problème d'optimisation vise la recherche d'une plus court chemin dans ce réseau. Quel est le plus court chemin pour aller du point a au point m ?*

Un autre problème largement étudié est celui du voyageur de commerce. Il modélise une situation analogue à celle d'un voyageur de commerce qui souhaite visiter un ensemble de villes et revenir à son point de départ, sans passer deux fois par la même ville et en minimisant le coût nécessaire pour passer d'une ville à l'autre.

Exemple 1.2 (Problème du voyageur de commerce). *En partant du point a de la figure 1.1, quel est le plus court chemin passant une unique fois par chaque point et se terminant au point de départ a ?*

Le problème de *set packing* est un de ces problèmes qui s'écrivent facilement mais pourtant très difficiles à résoudre. Il modélise une situation dans laquelle le décideur souhaite obtenir un maximum de ressources parmis plusieurs ressources en conflit.

Exemple 1.3 (Problème de *set packing*). *Le décideur souhaite inviter ses amis à son anniversaire. Du fait de certains conflits parmi son entourage, il sait que certaines personnes ne viendront pas si d'autres sont présentes. Son but est d'inviter un maximum de personne.*
Chaque rond de la figure 1.2 représente une personne. Un trait reliant deux personnes indique qu'elles sont en conflit et ne viendront pas ensemble (on ignore pour l'instant les valeurs indiquées dans les ronds). Quelles sont les personnes à appeler pour avoir un maximum d'invités ?
Variante : on considère que tous les amis ne sont pas équivalents aux yeux du décideur. Chaque cercle de la figure 1.2 contient une valeur indiquant l'importance de la personne pour le décideur (une forte valeur correspond à une grande importance). Quelles sont les personnes à appeler pour avoir un groupe d'invités avec une importance maximum ?

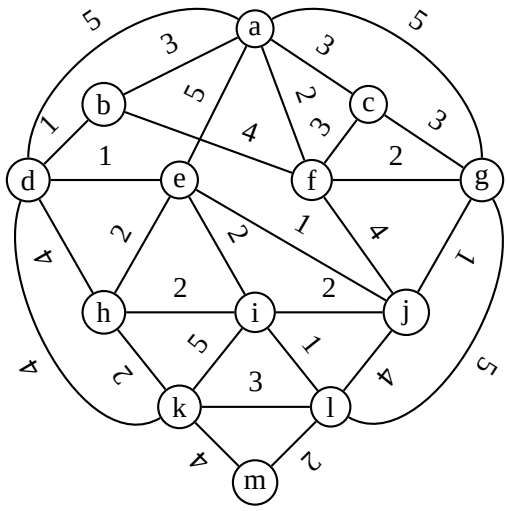

Figure 1.1 – Graphe illustrant les exemples de problèmes de plus court chemin et de voyageur de commerce.

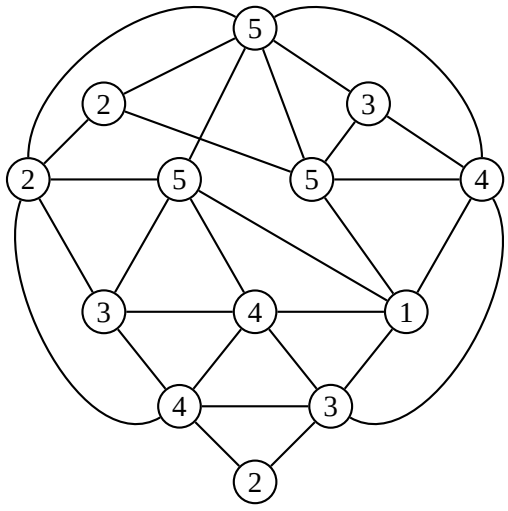

Figure 1.2 – Graphe illustrant les exemples de problèmes de *set packing*.

Numéro	Titre	Note	Poids (octets)
1	Cohiba Playa - Radiodays (ft. Dj Volto)	⋆⋆⋆⋆	4000000
2	Musetta - Ophelia's song	⋆⋆⋆⋆	5200000
3	Cartel - Real Talk by Brix	⋆⋆	2700000
4	Professor Kliq - Bust This Bust That	⋆⋆⋆⋆⋆	6800000
5	The Dada Weatherman - Painted Dream	⋆⋆⋆⋆⋆	8000000
6	Alexander Blu - Emptiness	⋆⋆⋆	6200000
7	UltraCat - Disco High	⋆⋆⋆	6300000
8	Josh Woodward - Effortless	⋆⋆	5300000
9	Tryad - Struttin'	⋆	5600000
10	Silence - Cellule	⋆	5900000

Table 1.1 – La bibliothèque musicale du décideur

La section 2.1.1.3 page 34 présente une application concrète du problème de *set packing* dans le cadre de l'optimisation de traffic ferroviaire.

Il existe de nombreux autres problèmes classiques, trop pour pouvoir tous les présenter dans cette thèse. Le lecteur intéressé pourra se référer à [139] sur le sujet. Le dernier problème présenté ici est celui du sac à dos. Il modélise une situation dans laquelle le décideur possède sa disposition un ensemble d'objets parmi lesquels il doit faire une sélection, en respectant une contrainte de capacité. L'objectif est de maximiser le profit apporté par les objets sélectionnés.

Exemple 1.4 (Problème de sac à dos). *Le décideur possède une bibliothèque musicale contenant 10 titres*[‡]*. Le logiciel qu'il utilise pour gérer sa bibliothèque lui permet de donner une note dans un intervalle de 1 à 5 étoiles à chaque titre, 5 étant la meilleure note. Les informations sur les titres sont récapitulées dans la table 1.1.*
Son lecteur multimédia portable a une capacité de 16000000 octets[§]*. Il souhaite le remplir de manière à avoir la meilleure liste de lecture possible à partir de sa bibliothèque. C'est à dire que la somme des notes des titres choisis doit être maximale. Quels titres doit-il sélectionner ?*

Ce dernier problème connait de nombreuses variantes. Certaines d'entre elles seront présentées dans la section 2.1.1 page 32, ainsi qu'une solution de cet exemple. Des solutions pour les autres exemples sont rapportées dans l'annexe A.

1.2 Problème et solutions en optimisation combinatoire multi-objectif

Un programme linéaire en nombres entiers est une description mathématique d'un problème d'optimisation. Dans celle-ci, les *variables* représentent les objets sur lesquels le choix s'effectue. L'ensemble des valeurs qu'elles peuvent prendre est leur *domaine*. C'est en général un sous-ensemble de $\mathbb{N}$, en particulier l'ensemble $\{0, 1\}$.

Une solution au problème est obtenue en attribuant une valeur de son domaine à chacune des variables. Lorsque les valeurs de plusieurs variables sont dépendantes les unes des autres, elles sont liées par des *contraintes*.

‡. une valeur si faible qu'elle en est irréaliste, mais idéale pour les besoins de l'exemple
§. il l'a acquis à la fin des années 90

Une solution est évaluée via la *fonction objectif*. Celle-ci quantifie la performance de la solution. Enfin, cette fonction est précédée par la *direction de la recherche*, indiquant le genre de valeur recherchée pour qu'une solution soit optimale.

Exemple 1.5 (Formulation d'un problème d'optimisation combinatoire). *Cet exemple s'appuie sur le problème de plus courts chemins, évoqué dans la section 1.1.2. Dans ce problème, le choix s'effectue sur les routes entre les villes. Celles-ci peuvent ainsi être représentés par des* variables $x_{i,j}$*, où* i *et* j *sont respectivement la ville de départ et la ville d'arrivée.*
Pour simplifier les notations, chaque ville sera ici représentée par un nombre entier, en suivant l'ordre alphabétique (a=1, b=2, . . . , m=13). De même, on nommera $l_{i,j}$ *la longueur de la route allant de la ville numéro* i *vers la ville numéro* j [¶]*. Ainsi, la route allant de la ville* a *vers la ville* c *correspond à la variable* $x_{1,3}$ *et sa longueur est* $l_{1,3} = 3$*.*
Dans toute solution, chaque route ne peut être que dans deux états : soit elle est sélectionnée, soit elle ne l'est pas. Le domaine *des variables se limite alors à* $\{0, 1\}$*, en considérant que* $x_{i,j} = 1$ *quand la route de* i *vers* j *est choisie, et* $x_{i,j} = 0$ *sinon.*
Un plus court chemin est une solution où la somme des longueurs des routes, notée L*, est minimale. Ceci définit la* fonction objectif *et la* direction de la recherche*, que l'on formulera ainsi :*

$$\textit{minimiser } L = \sum_{i=1}^{13} \sum_{j=1}^{13} (l_{i,j} \times x_{i,j})$$

Remarquons que toutes les routes interviennent dans l'évaluation de la fonction objectif. Cependant, étant donné qu'une route $x_{i,j}$ *non sélectionnée vérifie* $x_{i,j} = 0$*, sa longueur n'est pas comptée au final. De même, certaines variables correspondent à des routes qui n'existent pas (*$x_{1,13}$ *par exemple). Afin d'être sûr qu'elles ne seront pas choisies, leur longueur sera définie comme égale à l'infini.*
Le chemin recherché doit commencer par la ville a*. C'est une* contrainte *qui peut se traduire par « il y a exactement une seule route sélectionnée en partant de* a *», ou encore :*

$$\sum_{j=1}^{13} x_{1,j} = 1$$

De même, le chemin doit se terminer par la ville m *:*

$$\sum_{i=1}^{13} x_{i,13} = 1$$

Enfin, un chemin est fait de routes qui se suivent. Ainsi, pour toute ville j*, si une route y arrivant est choisie, alors il faut qu'il y en ait une de choisie qui en part. Sauf pour les villes de départ et d'arrivée :*

$$\sum_{i=1}^{13} x_{i,j} = \sum_{k=1}^{13} x_{j,k}, \quad \forall j \in \{2, \ldots, 12\}$$

¶. $l_{i,j}$ n'est pas une variable : sa valeur est donnée initialement.

Le problème complet s'écrit donc, au final

$$\begin{array}{lll} \min & L = \sum_{i=1}^{13}\sum_{j=1}^{13} l_{i,j} \times x_{i,j} & \\ s.c. & \sum_{j=1}^{13} x_{1,j} = 1 & \\ & \sum_{i=1}^{13} x_{i,13} = 1 & \\ & \sum_{i=1}^{13} x_{i,j} = \sum_{k=1}^{13} x_{j,k} & \forall j \in \{2,\ldots,12\} \\ & x_{i,j} \in \{0,1\} & \forall i,j \in \{1,\ldots,13\} \end{array}$$

Remarquons que ceci n'est que la traduction du problème en termes mathématiques. Cela ne donne aucune indication sur la valeur de la solution optimale, ni sur la méthode à appliquer pour l'obtenir.

1.2.1 Problème d'optimisation combinatoire mono/multi-objectif

Classiquement, un problème d'optimisation combinatoire mono-objectif est défini comme suit :

$$\begin{array}{lll} \text{opt} & z(x) & \\ s.c. & Ax\Delta b & \Delta \in \{\leq, =, \geq\} \\ & x \in \{0,1\}^n & \end{array}$$

où $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m\times 1}$ et où « opt » consiste, selon le problème, en une minimisation ou une maximisation de la valeur de la fonction objectif $f(x)$.

Cette formulation est celle d'un programme linéaire en variables binaires. À ce programme peut être associée une structure combinatoire, telle que la sélection d'un ensemble de routes (plus court chemin, problème du voyageur de commerce, *etc.*), ou encore la couverture d'un graphe (problèmes de l'arbre couvrant, coloration de graphe, *etc.*). C'est cette structure qui rend le problème particulier.

Le vecteur x représente les n variables du problème, le plus souvent binaires, indiquant si un objet i fait partie de la solution ($x_i = 1$) ou pas ($x_i = 0$). Les solutions dites réalisables sont celles vérifiant les contraintes $Ax\Delta b$.L'ensemble des solutions réalisables sera noté X et son image dans l'espace des objectifs sera $Y = \{z(x) : x \in X\}$.

Afin d'alléger les notations dans la suite du document, la direction de « opt » sera une maximisation et Δ sera $\leq$.

Définition 1.1 (Solution optimale)**.** *Une solution x^* est dite* optimale *si elle est réalisable et vérifie* $z(x^*) = \max_{x\in X}\{z(x)\}$.

Le plus souvent, le décideur souhaitera obtenir une ou plusieurs solutions optimales. Néanmoins, il peut s'avérer qu'une telle solution ne soit pas acceptable, pour des raisons diverses telles qu'une contrainte technique non formalisée dans le problème. Dans une telle situation, le décideur pourra apprécier une solution d'une qualité légèrement moindre. Il existe des procédures dites de *ranking* pour certains problèmes permettant d'obtenir des solutions dans un ordre décroissant de qualité (voir [23, 45, 84], entre autres). Le résultat de ces procédures est un ensemble ordonné $\{x^1,\ldots,x^k\}$ tel que, pour tout $i < j$, l'égalité $z(x^i) \geq z(x^j)$ est vérifiée.

Dans le cadre d'un problème d'optimisation combinatoire multi-objectif, plusieurs fonctions objectif, notées $z_1(x), \ldots, z_p(x)$, sont à considérer simultanément. Plusieurs types de problèmes peuvent alors apparaître. Dans un premier cas, tous les objectifs doivent être maximisés simultanément. Un tel problème se formule de la manière suivante :

$$\begin{array}{rl} \max & z(x) = (z_1(x), \ldots, z_p(x)) \\ s.c. & Ax \leq b \\ & x \in \{0,1\}^n \end{array}$$

ou, de manière équivalente :

$$\left.\begin{array}{rll} \max & z_j(x) & \forall j \in \{1, \ldots, p\} \\ s.c. & Ax \leq b & \\ & x \in \{0,1\}^n & \end{array}\right\} \quad MOP$$

Nous pouvons remarquer que le cas mono-objectif correspond au cas où $p = 1$ dans ces formulations.

Il est aussi envisageable d'associer des directions d'optimisation différentes aux composantes de $z(x)$. Par exemple, dans le cas d'un problème de gestion de portefeuille [42], les risques de l'investissement sont à minimiser tandis que les profits sont à maximiser.

Une autre situation possible consiste à maximiser la plus mauvaise performance obtenue sur un objectif :

$$\begin{array}{rl} \max & \min\limits_{j \in \{1,\ldots,p\}} z_j(x) \\ s.c. & Ax \leq b \\ & x \in \{0,1\}^n \end{array}$$

Il s'agit d'une formulation pessimiste où le décideur souhaite maximiser la pire des situations, parfois nommée problème de « *min-ordering* » et utilisée en particulier pour des problèmes de *location planning* [123].

D'autres formulations de problèmes multi-objectifs sont apparues, y compris des combinaisons des formulations ci-dessus. Néanmoins, dans la suite du manuscrit, nous appellerons « problème d'optimisation multi-objectif » tout problème correspondant à la formulation nommée MOP ci-dessus. Le lecteur désireux d'en apprendre plus sur les autres formulations pourra se référer à [35]. De plus, nous utiliserons le terme MOCO pour nommer les problèmes multi-objectifs d'optimisation combinatoire.

1.2.2 Solutions d'un problème d'optimisation multi-objectif

La notion d'optimalité d'une solution d'un MOP n'est pas aussi intuitive que dans le cadre d'un problème mono-objectif. En effet, soit deux solutions réalisables x^1 et x^2, telles que $z_1(x^1) < z_1(x^2), z_2(x^2) < z_2(x^1)$ et $z_j(x^1) = z_j(x^2), \forall j \in \{3, \ldots, p\}$. Ces solutions ont des performances différentes, elles ne sont donc pas équivalentes et, sans indication supplémentaire, aucune d'elles ne peut être considérée comme meilleure que l'autre. En effet, selon que le décideur ait une préférence sur le premier ou sur le second objectif, la solution optimale sera, respectivement, x^2 ou x^1.

Si aucune considération de préférence n'est donnée *a priori*, ces deux solutions doivent être conservées par une méthode de résolution exacte, ce qui conduit à reconsidérer le sens donné à la notion de solution optimale.

1.2.2.1 Efficacité et non dominance

Plusieurs interprétations peuvent être données à la notion de maximisation suivant le contexte de résolution (voir [35]). S'il existe un classement d'importance entre les objectifs, marquant une préférence absolue, l'appréciation de la qualité des solutions (la valeur des fonctions objectif) suit un ordre total appelé *ordre lexicographique*.

Définition 1.2 (Ordre lexicographique)**.** *Soient $y^1, y^2 \in \mathbb{R}^p$, alors $y^1 >_{lex} y^2$ s'il existe $l \in \{1, \ldots, p\}$ tel que pour tout $k < l, y^1_k = y^2_k$ et $y^1_l > y^2_l$.*

Si $y^1 >_{lex} y^2$ ou $y^1 = y^2$ alors on note $y^1 \geqq_{lex} y^2$.

Définition 1.3 (Optimalité lexicographique)**.** *Soit π une permutation de $\{1, \ldots, p\}$. Une solution admissible x est appelée* lexicographiquement optimale par rapport à π *si pour tout $x' \in X, z_\pi(x) \geq_{lex} z_\pi(x')$, où $z_\pi(x) = (z_{\pi_1}(x), \ldots, z_{\pi_p}(x))$.*

Une solution x^* est appelée lexicographiquement optimale *s'il existe une permutation π de $\{1, \ldots, p\}$ telle que x^* est lexicographiquement optimale par rapport à π.*

L'ensemble des solutions lexicographiquement optimales est noté X_{lex}.

En l'absence d'informations sur les préférences des décideurs, il n'est plus possible d'affirmer l'unicité liée à l'optimalité. C'est-à-dire que la valeur d'une solution ne peut être considérée comme la meilleure de toutes. Néanmoins, il reste possible de comparer des solutions deux à deux en utilisant l'ordre partiel donné par une relation de dominance de Pareto, définie ci-dessous.

Une solution Pareto optimale, ou encore appelée solution efficace, ne peut être améliorée sur un objectif sans dégrader la performance sur un autre objectif. Selon cette relation, la comparaison de la valeur de deux solutions indiquera soit que l'une est meilleure que l'autre, soit qu'elles sont incomparables.

Définition 1.4 (Dominance de Pareto)**.** *Soit $y^1, y^2 \in \mathbb{R}^p$, avec $p \geq 2$.*

– *Le point y^1 domine strictement le point y^2 ($y^1 > y^2$) si*

$$\forall i \in \{1, \ldots, p\} \quad y^1_i > y^2_i$$

– *Le point y^1 domine faiblement le point y^2 ($y^1 \geqq y^2$) si*

$$\forall i \in \{1, \ldots, p\} \quad y^1_i \geq y^2_i$$

– *Le point y^1 domine le point y^2 ($y^1 \geq y^2$) si*

$$\begin{array}{ll} & y^1 \geqq y^2 \\ \textit{et} & \exists i \in \{1, \ldots, p\} \quad \textit{t.q.} \quad y^1_i > y^2_i \end{array}$$

Nous utiliserons par la suite la notation $\mathbb{R}^p_{\geqq} = \{x \in \mathbb{R}^p : x \geqq 0\}$ et par analogie $\mathbb{R}^p_{\geq}$ et $\mathbb{R}^p_{>}$. La définition ci-dessous introduit la notion de solution efficace :

Définition 1.5 (Solution efficace (ou Pareto optimale))**.** *Soit $x \in X$. Si $\nexists x' \in X$ telle que $z(x') \geq z(x)$, alors x est appelée* solution efficace.

On dira qu'une solution est faiblement efficace *si son image n'est pas strictement dominée par l'image d'une autre solution.*

	$(x_1, x_2, x_3, x_4, x_5)$
x^1	$(0, 0, 1, 1, 1)$
x^2	$(0, 1, 0, 1, 1)$
x^3	$(0, 1, 1, 0, 1)$
x^4	$(0, 1, 1, 1, 0)$
x^5	$(1, 0, 0, 1, 1)$
x^6	$(1, 0, 1, 0, 1)$
x^7	$(1, 0, 1, 1, 0)$
x^8	$(1, 1, 0, 0, 1)$

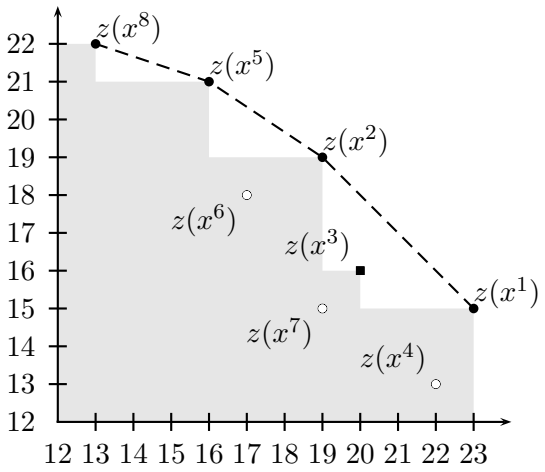

Figure 1.3 – À gauche, les solutions P_{ex}. À droite, leur images dans l'espace des objectifs.

Par la suite, l'ensemble des solutions efficaces sera noté X_E. Son image dans l'espace des objectifs sera nommé *frontière efficace* et sera noté $Y_N = \{z(x) : x \in X_E\}$.

Exemple 1.6 (Solutions efficaces d'un MOP bi-objectif). *Soit le problème bi-objectif d'optimisation ci-dessous :*

$$\left.\begin{array}{lll} \max & z_1(x) = & 2x_1 + 5x_2 + 9x_3 + 8x_4 + 6x_5 \\ \max & z_2(x) = & 8x_1 + 6x_2 + 2x_3 + 5x_4 + 8x_5 \\ s.c. & & 8x_1 + 7x_2 + 5x_3 + 4x_4 + 2x_5 \quad \leq 17 \\ & & x_1 + x_2 + x_3 + x_4 + x_5 \quad = 3 \\ & & x_i \in \{0, 1\} \quad \forall i \in \{1, \ldots, 5\} \end{array}\right\} \quad P_{ex}$$

Ce problème accepte 8 solutions réalisables, énumérées dans partie à gauche de la figure 1.3. La partie à droite représente leurs images dans l'espace des objectifs. Seules les solutions x^1, x^2, x^3, x^5 et x^8 sont efficaces, les autres solutions sont toutes dominées. x^1 et x^8 sont aussi lexicographiquement *optimales.*

Nous pouvons remarquer que, dans le cas bi-objectif, les points de Y_N sont naturellement ordonnés. En effet, soit $y, y' \in Y_N$, avec $y_1 > y'_1$, alors $y_2 < y'_2$. Plusieurs méthodes de résolution s'appuient sur cette propriété. Cependant, elle n'est plus valide à partir de trois objectifs où, si $y_1 > y'_1$, alors nous pouvons avoir $y_2 > y'_2$ et $y_3 < y'_3$ ou $y_2 < y'_2$ et $y_3 > y'_3$, indifféremment.

Dans la suite de ce manuscrit, nous considèrerons que les préférences du décideur ne sont pas connues au moment de la résolution. Ce qui correspond à un contexte de résolution *a posteriori*. Nous ne traiterons pas du contexte *a priori*, dans lequel ces préférences sont connues avant la résolution, ni du contexte interactif, dans lequel elles sont introduites pendant la résolution.

1.2.2.2 Classification des solutions efficaces

Une méthode communément répandue comme moyen de calcul de solutions efficaces repose sur l'utilisation d'une somme pondérée des fonctions objectifs.

Théorème 1.1 (Somme pondérée [61])**.** *Soit P un MOP avec p objectifs. Soit $\lambda \in \mathbb{R}^p_{\geq}$. On construit le problème mono-objectif P_λ :*

$$\left.\begin{array}{ll} \max & z^\lambda(x) = \lambda \cdot z(x) \\ s.c. & Ax \leq b \\ & x \in \{0,1\}^n \end{array}\right\} \quad P_\lambda$$

où l'opérateur $\cdot$ représente le produit scalaire.

Si x^* est une solution optimale de P_λ, alors on a les énoncés suivants.

1. *Si $\lambda \in \mathbb{R}^p_{\geq}$ alors x^* est faiblement efficace ;*
2. *Si $\lambda \in \mathbb{R}^p_{>}$ alors x^* est efficace ;*
3. *Si $\lambda \in \mathbb{R}^p_{\geq}$ et x^* est l'unique solution optimale de P_λ, alors x^* est efficace.*

L'utilisation d'une somme pondérée ne permet de trouver que les solutions efficaces dont les images se situent sur la frontière de l'enveloppe convexe de Y, notée $\mathrm{conv}(Y)$. Les autres solutions efficaces ne peuvent être obtenues quel que soit le poids λ. Néanmoins, la principale qualité de cette approche est que, si tous les objectifs sont du même type, tout problème P_λ est une instance du problème mono-objectif correspondant. Les algorithmes développés spécifiquement pour le cas mono-objectif peuvent donc être utilisés.

La popularité de cette méthode a mené à classifier les solutions efficaces en plusieurs catégories. Ainsi, les solutions optimales des problèmes P_λ sont appelées *solutions supportées* de MOP. On note leur ensemble X_{SE}. Les solutions appartenant à $X_E \setminus X_{SE}$ sont dites *non supportées* et leur ensemble sera noté X_{NE}. De plus, on distingue deux catégories de solutions supportées. Les *solutions supportées extrêmes*, notées X_{SE1}, sont les solutions dont les images se situent sur un des sommets de $\mathrm{conv}(Y)$. Les *solutions supportées non extrêmes*, dont les images sont sur les facettes de $\mathrm{conv}(Y)$. L'ensemble de ces dernière est noté $X_{SE2} = X_{SE} \setminus X_{SE1}$.

Exemple 1.7 (Somme pondérée). *La somme pondérée $(P_{ex})_\lambda$ obtenue en appliquant le poids $\lambda = (1,1)$ au problème de l'exemple 1.6 se formule comme suit :*

$$\left.\begin{array}{lll} \max \quad z(x) = & 10x_1 + 11x_2 + 11x_3 + 13x_4 + 14x_5 & \\ s.c. & 8x_1 + 7x_2 + 5x_3 + 4x_4 + 2x_5 & \leq 17 \\ & x_1 + x_2 + x_3 + x_4 + x_5 & = 3 \\ & x_i \in \{0,1\} & \forall i \in \{1,\ldots,5\} \end{array}\right\} \quad (P_{ex})_\lambda$$

Les solution $x^1 = (0,0,1,1,1)$ et $x^2 = (0,1,0,1,1)$ sont les uniques solutions optimales de ce problème, avec une valeur de 38. Ce sont des solutions supportées du problème initial. Les autres solutions supportées, x^5 et x^8, sont optimales pour $\lambda = (1,3)$, tandis qu'il n'existe pas de λ tel que x^4 soit optimale pour $(P_{ex})_\lambda$. Elle est donc non supportée.
Remarquons que les poids permettant d'obtenir ces solutions efficaces ne sont pas uniques. Par exemple, x^1 est optimale pour tout λ tel que $\lambda_1 \in [0,5;1[$ et $\lambda_2 = 1 - \lambda_1$.

Dans le contexte *a posteriori*, nous cherchons à déterminer « toutes » les solutions efficaces. Plusieurs façons de classifier X_E, selon cette notion de complétude de l'ensemble des solutions, sont possibles.

Notation	Signification	Image
X	ensemble des solutions réalisables	Y
X_E	ensemble complet de solutions efficaces	Y_N
X_{E_m}	ensemble complet minimal de solutions efficaces	Y_N
X_{E_M}	ensemble complet maximal de solutions efficaces	Y_N
X_{SE}	ensemble complet de solutions supportées	Y_{SN}
X_{SE_m}	ensemble complet minimal de solutions supportées	Y_{SN}
X_{SE_M}	ensemble complet maximal de solutions supportées	Y_{SN}
X_{SE1}	ensemble complet de solutions supportées extrêmes	Y_{SN1}
X_{SE1_m}	ensemble complet minimal de solutions supportées extrêmes	Y_{SN1}
X_{SE1_M}	ensemble complet maximal de solutions supportées extrêmes	Y_{SN1}
X_{SE2}	ensemble complet de solutions supportées non extrêmes	Y_{SN2}
X_{SE2_m}	ensemble complet minimal de solutions supportées non extrêmes	Y_{SN2}
X_{SE2_M}	ensemble complet maximal de solutions supportées non extrêmes	Y_{SN2}
X_{NE}	ensemble complet de solutions non supportées	Y_{NN}
X_{NE_m}	ensemble complet minimal de solutions non supportées	Y_{NN}
X_{NE_M}	ensemble complet maximal de solutions non supportées	Y_{NN}

Table 1.2 – Ensembles de solutions et leurs projections dans l'espace des objectifs

Définition 1.6 (Équivalence, ensemble complet [67])**.** *Deux solutions admissibles $x, x' \in X$ sont dites* équivalentes *si $z(x) = z(x')$.*

Un ensemble complet *X_E est un ensemble de solutions efficaces tel que toute solution $x \in X \backslash X_E$ est soit équivalente, soit dominée par une solution $x' \in X_E$. Un ensemble complet X_{E_m} sans solution équivalente est dit* minimal*, tandis que l'ensemble complet X_{E_M} contenant toutes les solutions équivalentes est dit* maximal.

Remarque.
1. Pour chaque point $y \in Y_N$ il existe au moins une solution $x \in X_E$ telle que $z(x) = y$;
2. tout ensemble complet contient un ensemble complet minimal ;
3. toute solution $x \in X \backslash X_{E_M}$ est dominée.

La table 1.2 récapitule les notations concernant les différents ensembles de solutions.

1.2.3 Intraitabilité de problèmes multi-objectifs

Pour de nombreux problèmes multi-objectifs, trouver un ensemble complet est très difficile. Bien que les solutions supportées puissent être obtenues à l'aide de sommes pondérées s'il existe un algorithme efficace pour la résolution du problème mono-objectif correspondant, il est aussi nécessaire, pour les MOCO, de déterminer des solutions non supportées. Or, les méthodes permettant leur détermination impliquent généralement la résolution de problèmes $\mathcal{NP}$-difficiles, même si le problème mono-objectif considéré peut être résolu en temps polynomial. Par conséquent, une difficulté pratique dans la résolution des MOCO provient de la détermination d'un ensemble X_{NE}. De plus, des expérimentations ont montré que les solutions non supportées, en plus d'être plus difficiles à obtenir, peuvent également être beaucoup plus nombreuses que les solutions supportées [136].

D'un point de vue théorique, la plupart des MOCO sont $\mathcal{NP}$-complets, $\#P$-complets et intraitables [35]. En particulier, Hugot a démontré l'intraitabilité du problème de sac à dos multi-objectifs [69]. Cependant, cette propriété apparaît même si le problème mono-objectif correspondant est dans la classe $\mathcal{P}$.

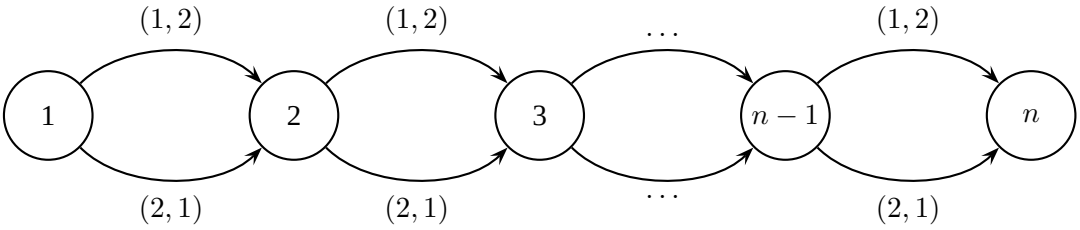

Figure 1.4 – Exemple de graphe pour lequel le problème de plus courts chemins accepte un nombre exponentiel de solutions efficaces.

C'est en particulier le cas pour les problèmes multi-objectifs de plus court chemin [67, 122], d'affectation [102, 122], d'arbre couvrant de poids minimum [66] et de flot de coût minimum en variables entières [117].

L'exemple ci-dessous présente une instance du problème de plus courts chemins multi-objectifs inspirée de [69], pour lequel toutes les solutions sont efficaces. Cet exemple est particulier du fait que ces solutions soient toutes supportées et que leurs images soient en nombre polynomial. Cependant, il existe des instances pour lesquelles toutes les solutions sont efficaces et présentent des performances différentes [67, 69].

Exemple 1.8 (Intraitabilité de problèmes multi-objectifs). *Le graphe de la figure 1.4 illustre l'intraitabilité intrinsèque de problèmes multi-objectifs.*
Dans ce graphe, tous les chemins allant du sommet 1 *au sommet* n *sont efficaces ; soit* 2^{n-1} *chemins. Il est inconcevable de présenter toutes ces solutions à un décideur. De plus, un algorithme cherchant les plus longs chemins va potentiellement les stocker en mémoire, entraînant une saturation rapide des ressources.*

1.2.4 Bornes

Dans un contexte mono-objectif, l'utilisation de bornes serrées sur la valeur de la solution optimale a permis le développement d'algorithmes efficaces. Il est donc intéressant de généraliser leur utilisation au contexte multi-objectif, c'est-à-dire pour encadrer l'ensemble des points non dominés. Cependant, cette généralisation n'est pas immédiate, comme le suggère l'exemple 1.9 ci-dessous.

Exemple 1.9 (Non généralisation des bornes du cas mono-objectif vers le cas multi-objectif [41]). *Ehrgott et Gandibleux [41] ont proposé de démontrer par ce contre-exemple que les propriétés obtenues sur des bornes dans le cas mono-objectif ne sont pas nécéssairement valides dans le cas multi-objectif. Soit le problème d'optimisation combinatoire bi-objectif ci-dessous.*

$$\left.\begin{array}{rl} \max & \sum_{i=1}^{n} c_i^1 x_i \\ \max & \sum_{i=1}^{n} c_i^2 x_i \\ s.c & \sum_{i=1}^{n} w_i x_i \leq \omega \\ & x_i \in \{0,1\}, \forall i \in \{1,\ldots,n\} \end{array}\right\} \; 2-01KP$$

où on suppose $0 < w_i < \omega, \forall i \in \{1, \ldots, n\}$ *;* $\sum_{i=1}^{n} w_i > \omega$*, et* $c_i > 0, \forall i \in \{1, \ldots, n\}$*.*
Ce problème est identifié comme le problème bi-objectif de sac à dos. Nous allons transformer ce problème en une version mono-objectif $2-01KP_\lambda$*, en utilisant une somme pondérée telle que définie dans le théorème 1.1. Ainsi, nous pourrons lui appliquer deux bornes définies pour le cas mono-objectif.*
Pour cet exemple, nous notons $c_i^\lambda = \lambda c_i^1 + (1-\lambda)c_i^2$*, avec* $0 \leq \lambda \leq 1$ *et nous supposons que*

$$\frac{c_1^\lambda}{w_1} \geq \frac{c_2^\lambda}{w_2} \geq \ldots \geq \frac{c_n^\lambda}{w_n}$$

Deux bornes, en particulier, sont définies pour ce problème, basées sur l'objet critique ; le premier objet qui ne rentre pas dans le sac en les prenant dans l'ordre établi ci-dessus. L'indice de cet objet sera noté $s = \min\{k : \sum_{i=1}^{k} w_i > \omega\}$*.*
La borne $U_1(\lambda)$ *est obtenue par résolution du problème* $2-01KP_\lambda$ *en remplaçant la contrainte* $x_i \in \{0,1\}$ *par* $x_i \in [0;1]$*. On obtient*

$$x_i^{LP} = \begin{cases} 1 & i \in \{1, \ldots, s-1\} \\ \frac{\omega - \sum_{j=1}^{s-1} w_j}{w_s} & i = s \\ 0 & \textit{sinon.} \end{cases}$$

et $U_1(\lambda) = \sum_{i=1}^{n} c_i^\lambda x_i^{LP}$*.*
La borne de Martello et Toth [92] $U_2(\lambda)$ *améliore la borne précédente en considérant les deux cas où l'objet* s *est inclus ou exclu de la solution, on considère donc deux solutions :*

$$x_i^0 = \begin{cases} 1 & i = 1, \ldots, s-1 \\ \frac{\omega - \sum_{j=1}^{s-1} w_j}{w_{s+1}} & j = s+1 \\ 0 & \textit{sinon} \end{cases}$$

$$x_i^1 = \begin{cases} 1 & i \in \{1, \ldots, s-2, s\} \\ \max\left\{0, 1 - \frac{w_s - (\omega - \sum_{j=1}^{s-1} w_j)}{w_{s-1}}\right\} & i = s-1 \\ 0 & \textit{sinon} \end{cases}$$

et $U_2(\lambda) = \max\left\{\sum_{i=1}^{n} c_i^\lambda x_i^0, \sum_{i=1}^{n} c_i^\lambda x_i^1\right\}$*.*
Il est connu que l'inégalité $U_2(\lambda) \leq U_1(\lambda)$ *est toujours valide dans le cas mono-objectif [92].*

On considère maintenant l'instance suivante de $2-01KP$ *avec* $n = 8, \omega = 102$ *et les poids et valeurs des objets donnés dans la table suivante.*

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
c^1	*15*	*100*	*90*	*60*	*40*	*15*	*10*	*1*
c^2	*15*	*100*	*90*	$\tilde{c}$	*40*	*15*	*10*	*1*
w	*2*	*20*	*20*	*30*	*40*	*30*	*60*	*10*

La table suivante contient les valeurs du rapport du profit sur le poids de chaque objet pour le problème $2-01KP_\lambda$ *avec* $\lambda = 1$*, et les solutions correspondantes* x^{LP}, x^0 *et* x^1*.*

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
$\frac{c^\lambda}{w}$	*7,5*	*5*	*4,5*	*2*	*1*	*0,5*	*1/6*	*0,1*
x^{LP}	*1*	*1*	*1*	*1*	*0,75*	*0*	*0*	*0*
x^0	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*
x^1	*1*	*1*	*1*	*2/3*	*1*	*0*	*0*	*0*

On a donc $U_2(\lambda) = 285$ (obtenu par x^1) et $U_1(\lambda) = 295$. En calculant les valeurs des points correspondants pour ces solutions, nous obtenons $z(x^{LP}) = (295, 235 + \tilde{c})$ et $z(x^1) = (285, 245 + \frac{2}{3}\tilde{c})$. Donc, $z(x^1)$ n'est meilleure que $z(x^{LP})$ que si $\tilde{c} \leq 30$. L'inégalité $U_2(\lambda) \geq U_1(\lambda)$ n'est donc pas valide dans le cas multi-objectif.

Cet exemple suggère que des bornes doivent être définies spécifiquement pour le cas multi-objectif. Les plus connues sont données par les deux points caractéristiques suivants.

Définition 1.7 (Point idéal, point nadir, point utopique)**.** *Étant donné un MOP avec p objectifs et au moins une solution efficace.*

1. *Le point $y^I = (y_1^I, \ldots, y_p^I)$, où $y_i^I = \max_{x \in X} z_i(x), \forall i \in \{1, \ldots, p\}$ est appelé* point idéal *;*
2. *Un point $y^U = y^I + \epsilon$, où $\epsilon > (0, \ldots, 0)$ est appelé* point utopique *;*
3. *Le point $y^N = (y_1^N, \ldots, y_p^N)$, où $y_i^N = \min_{x \in X_E} z_i(x), \forall i \in \{1, \ldots, p\}$ est appelé* point nadir.

Les points y^I et y^N sont au plus près possible de l'ensemble des points non dominés. Aucun de ces points n'est réalisable, sauf si tous les objectifs ont une solution optimale commune.

Le point idéal peut être obtenu en résolvant p problèmes mono-objectifs $P_i, \forall i \in \{1, \ldots, p\}$ définis par

$$\left.\max_{x \in X} z_i(x)\right\} \quad P_i$$

Si x^i est une solution optimale pour P_i, alors le point idéal est obtenu par $y_i^I = z_i(x^i), \forall i \in \{1, \ldots, p\}$.

La détermination du point nadir est en général plus difficile. Avec deux objectifs, il est obtenu en déterminant des solutions x^1 et x^2 respectivement lexicographiquement optimales par rapport à la permutation $(1, 2)$ et à $(2, 1)$: $y^N = (z_1(x^2), z_2(x^1))$. Cette égalité est due à l'ordre naturel des points non dominés dans le cas bi-objectif. Cependant, avec plus de deux objectifs, la détermination du point nadir n'est plus aussi aisée [34].

À cause de cette difficulté, des heuristiques sont utilisées (voir par exemple [34]), mais elles peuvent indifféremment surestimer ou sous-estimer le point nadir. Récemment, une caractérisation du point nadir a été proposée par Ehrgott et Tenfelde-Podehl [34] en considérant p sous problèmes.

Définition 1.8. *[34] Soit P un MOP avec p objectifs, p sous problèmes $P(i), \forall i \in \{1, \ldots, p\}$ sont définis par*

$$\left.\max_{x \in X}(z_1(x), \ldots, z_{i-1}(x), z_{i+1}(x), \ldots, z_p(x))\right\} \quad P(i)$$

Théorème 1.2 (Calcul du point nadir)**.** *[34] En notant Opt_{p-1} l'union des ensembles complets maximaux des problèmes $P(i), \forall i \in \{1, \ldots, p\}$, auquel on supprime toutes les solutions dominées pour le problème P, le point nadir y^N du problème P est donné par*

$$y_i^N = \min_{x \in Opt_{p-1}} z_i(x)$$

Si le point nadir peut donc être déterminé à l'aide du théorème 1.2, il reste coûteux à obtenir pour un MOP avec plus de deux objectifs.

Le point idéal et le point nadir sont en pratique peu utiles en tant que bornes, du fait qu'ils sont généralement « éloignés » des points non dominés. Ceci est dû au fait que ces derniers sont souvent nombreux et que les objectifs sont conflictuels. De plus, y^I et y^N représentent les meilleures et pires valeurs des points non dominés pour tous les objectifs. Une réponse à la faiblesse de ces bornes a été

discutée par Ehrgott et Gandibleux [38, 41] en proposant l'utilisation d'ensembles de bornes, lesquels permettent de mieux encadrer Y_N et sont aussi plus faciles à déterminer.

Les notions suivantes sont nécessaires pour introduire la définition des ensembles de bornes.

Définition 1.9 (Ensemble fermé)**.** *Un ensemble est dit* fermé *s'il contient sa frontière.*

Par exemple, l'intervalle $[0;1] \subset \mathbb{R}$ est fermé, tandis que $[0;1[$ est ne l'est pas. Ce dernier est dit *ouvert*.

Définition 1.10 (Ensemble $\mathbb{R}^p_{\leqq}$-fermé)**.** *Un ensemble S est dit $\mathbb{R}^p_{\leqq}$-fermé si $S - \mathbb{R}^p_{\leqq} = \{s - y : s \in S, y \in \mathbb{R}^p_{\leqq}\}$ est fermé.*

Définition 1.11 (Ensemble $\mathbb{R}^p_{\leqq}$-borné)**.** *Un ensemble S est dit $\mathbb{R}^p_{\leqq}$-borné si $\exists s \in \mathbb{R}^p$ tel que $S \subset s - \mathbb{R}^p_{\leqq}$.*

Définition 1.12 (Adhérence)**.** *L'*adhérence *d'un ensemble S, notée $\mathrm{adh}(S)$, est le plus petit ensemble fermé contenant S.*

Cette dernière propriété permet de forcer l'inclusion de la frontière d'un ensemble. Par exemple, l'adhérence de $]0;1[\subset \mathbb{R}$ est $[0;1]$.

La définition des ensembles de bornes proposée par Ehrgott et Gandibleux [41] est la suivante :

Définition 1.13 (Ensembles de bornes [41])**.** *Soit $\bar{Y} \subset Y_N$.*

1. *Un ensemble L bornant inférieurement $\bar{Y}$ est un ensemble $\mathbb{R}^p_{\leqq}$-fermé et $\mathbb{R}^p_{\leqq}$-borné tel que $\bar{Y} \subset \mathrm{adh}(\mathbb{R}^p \setminus (L - \mathbb{R}^p_{\leqq}))$ et $L = L_N$;*
2. *Un ensemble U bornant supérieurement $\bar{Y}$ est un ensemble $\mathbb{R}^p_{\leqq}$-fermé et $\mathbb{R}^p_{\leqq}$-borné tel que $\bar{Y} \subset U - \mathbb{R}^p_{\leqq}$ et $U = U_N$;*

Intuitivement, la condition $\bar{Y} \subset U - \mathbb{R}^p_{\leqq}$ pour un ensemble U bornant supérieurement $\bar{Y}$ signifie « U est situé au dessus de $\bar{Y}$ ». Remarquons que cette condition n'est pas suffisante, il est important qu'aucun point de U n'en domine un autre, sinon certains points de U pourraient être situés « en dessous » de certains points de $\bar{Y}$.

La condition $\bar{Y} \subset \mathrm{adh}(\mathbb{R}^p \setminus (L - \mathbb{R}^p_{\leqq}))$ pour un ensemble L bornant inférieurement $\bar{Y}$ signifie qu'aucun point non dominé n'est strictement dominé par un point de L. En particulier, tout ensemble de points réalisables dans lequel il n'y a aucun point qui en domine un autre est un ensemble bornant inférieurement $\bar{Y}$.

Des exemples importants d'ensembles bornant respectivement inférieurement et supérieurement Y_N sont $L = Y_{SN}$ et $U = \mathrm{conv}(Y_{SN})$.

Dans un article récent, Sourd et Spanjaard [126] ont présenté des résultats encourageants utilisant ces bornes pour la résolution du problème bi-objectif d'arbre couvrant de poids minimal.

1.3 Méthodes classiques de résolution

Peu de méthodes de résolution de problèmes mono-objectifs ont été étendues avec succès au cas multi-objectif. Cependant, des méthodes spécifiques au problèmes multi-objectifs ont été développées. Cette section présente les algorithmes classiques utilisés pour résoudre des problèmes multi-objectifs.

1.3.1 Méthodes de résolution exacte

1.3.1.1 Procédure de séparation et évaluation

Les procédures de séparation et évaluation (PSE) sont des méthodes classiques du domaine de l'optimisation mono-objectif. L'idée générale de ces procédures est de construire implicitement l'ensemble des solutions réalisables pour en extraire les solutions efficaces. Dans le cas d'un problème en variables binaires, l'étape de séparation consiste à sélectionner une variable x_k dont la valeur est indéterminée puis à diviser l'espace des solutions réalisables en deux sous-espaces où x_k prend la valeur 1 ou 0, respectivement. L'étape évaluation consiste quant à elle à mesurer la pertinence d'un sous-espace, avant de commencer son exploration, pour déterminer s'il contient potentiellement des solutions efficaces ou non, auquel cas il est inutile d'y continuer la séparation. Ce procédé est effectué de manière récursive sur chaque sous-espace, jusqu'à ce que chacun d'entre eux ne contienne qu'une unique solution.

Classiquement, dans le cas mono-objectif, cette dernière étape consiste à calculer une borne supérieure des performances des solutions du sous-espace, puis de comparer cette borne à la valeur de la meilleure solution connue.

Étonnamment, comme l'ont souligné Ehrgott et Gandibleux [37], peu de ces procédures ont été conçues pour des MOCO. À notre connaissance, seuls quelques articles proposent des procédures pour des problèmes multi-objectifs linéaires en nombres entiers (MOILP) [16, 17, 78, 90, 96], jusqu'à une contribution récente de Sourd et Spanjaard [126]. Seuls ces derniers ont proposé une telle généralisation, avec une application au problème d'arbre couvrant minimal.

1.3.1.2 Dichotomie

La dichotomie est une méthode de calcul des solutions supportées initialement proposée par Aneja et Nair [3] pour la résolution exacte du problème de transport bi-objectif. Elle consiste à trouver les poids $\lambda \in \mathbb{R}^2_>$ qui permettent d'obtenir tous les points supportés extrêmes. Bien sûr, cette méthode ne permet pas d'obtenir de solutions non supportées. Cette méthode n'utilise aucune spécificité du problème de transport et peut donc être appliquée à n'importe quel autre problème. Elle est ainsi souvent utilisée pour déterminer un ensemble X_{SE1_m} d'un MOCO bi-objectif.

L'initialisation de cette méthode consiste à déterminer des solutions x^1 et x^2 lexicographiquement optimales par rapport aux objectifs (1, 2) et (2, 1) respectivement. Une itération de cet algorithme consiste à résoudre un problème P_λ défini par $\lambda_1 = y^r_2 - y^s_2$ et $\lambda_2 = y^s_1 - y^r_1$, où y^r et y^s sont deux points consécutifs avec $y^r_1 < y^s_1$ (et donc $y^r_2 > y^s_2$). Le vecteur λ correspond à la normale à la droite joignant les points y^r et y^s (figure 1.5). Le point y^t obtenu par la résolution de P_λ est situé entre y^r et y^s selon deux situations :

1. si $\lambda \cdot y^t > \lambda \cdot y^r$, alors deux nouveaux problèmes P_λ définis par y^r et y^t, et y^t et y^s doivent être résolus (figure 1.6) ;
2. si $\lambda \cdot y^t = \lambda \cdot y^r$, la recherche s'arrête.

Pour la première itération, les points $y^1 = z(x^1)$ et $y^2 = z(x^2)$ sont utilisés. Remarquons que dans le premier cas, le point y^t n'est pas nécéssairement extrême et que dans le second cas, il peut très bien être un point non extrême situé entre y^r et y^s (ou être égal à y^r ou y^s).

Cette méthode est restée limitée au cas bi-objectif jusqu'à très récemment. Dans ses travaux de thèse, Przybylski [111] l'a étendue à trois objectifs et plus en proposant une autre façon de calculer les poids. Cette procédure sera détaillée dans le chapitre 5.

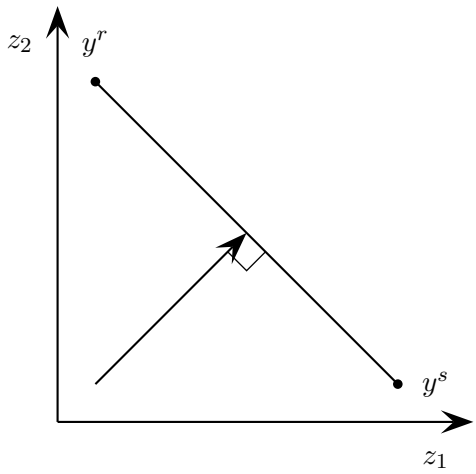

Figure 1.5 – Problème P_λ défini par y^r et y^s.

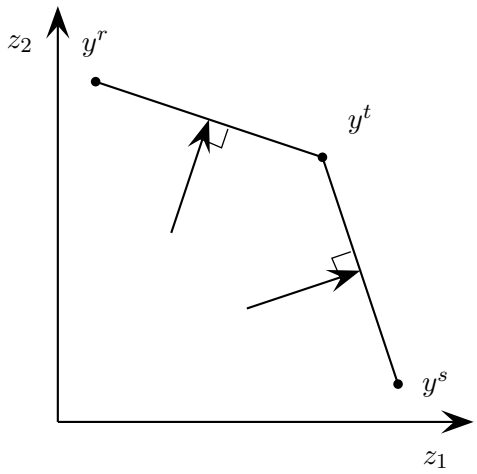

Figure 1.6 – Un nouveau point supporté est trouvé, la dichotomie continue.

1.3.1.3 Dichotomie par ajout de contraintes

Degoutin et Gandibleux [28] ont proposé une méthode générique de résolution de problèmes d'optimisation combinatoire bi-objectifs, faisant une utilisation intense d'un solveur de programmes linéaires en nombres entiers. Cette procédure calcule un ensemble complet minimal X_{E_m}.

Leur algorithme est initialisé par le calcul des solutions lexicographiquement optimales x^1 et x^2, avec $z_1(x^1) \leq z_1(x^2)$. Les auteurs construisent un problème mono-objectif paramétré $DG(y, y')$, défini comme suit

$$\left.\begin{array}{ll} \max & \lambda_1 z_1(x) + \lambda_2 z_2(x) \\ s.c. & Ax \leq b \\ & z_1(x) \geq y_1 + 1 \\ & z_2(x) \geq y'_2 + 1 \end{array}\right\} DG(y, y')$$

avec $\lambda \in \mathbb{R}^p$. Cette formulation reprend les contraintes du problème initial ($Ax \leq b$) et en ajoute deux pour assurer que des solutions correspondant aux points y et y' ne seront pas à nouveau trouvées. Ainsi, l'espace de recherche se réduit après chaque résolution d'un problème $DG(y, y')$. Les problèmes considérés par les auteurs ont tous des coefficients entiers (problèmes d'affectation, de sac à dos, de couverture d'ensemble et de *set packing*). Par conséquent, le décalage de 1 imposé par ces dernières contraintes est suffisant.

Ce problème est résolu par un appel à un solveur. Initialement, le problème $DG(y^1, y^2)$ est résolu, avec $y^1 = z(x^1)$ et $y^2 = z(x^2)$). Si aucune solution n'est trouvée, alors la procédure s'arrête. Autrement, la solution x^3 trouvée est conservée et deux nouveaux problèmes $DG(y^1, y^3)$ et $DG(y^3, y^2)$ sont construits, avec $y^3 = z(x^3)$. L'algorithme est ensuite appliqué de manière récursive sur ces deux problèmes. La figure 1.7 illustre le découpage dichotomique réalisé par cette procédure.

Cette approche a l'avantage d'être générale. Elle peut s'appliquer à n'importe quel MOCO bi-objectif à coefficients entiers. Son application à des problèmes à trois objectifs ou plus n'est pas immédiate. Tenfelde-Podehl a néanmoins proposé une méthode similaire pour des MOCO quelconques [130]. Ce-

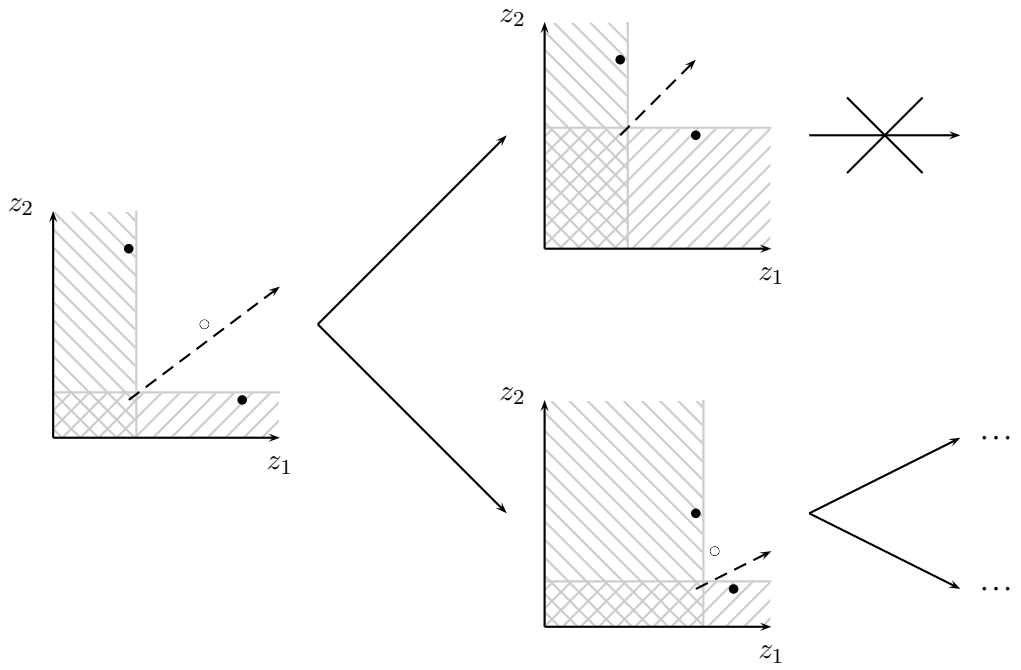

Figure 1.7 – Illustration de la procédure dichotomique de Degoutin et Gandibleux. Les rayures représentent les régions ôtée de l'espace de recherche par les contraintes ajoutées. Les ronds pleins (•) illustrent les points de référence pour $DG(y, y')$, tandis que le rond creux (○) représente l'image de la solution trouvée par le solveur. Quand celle-ci existe, l'espace est découpé de manière dichotomique. La direction de la maximisation est indiquée par les flèches en pointillés.

pendant, ces approches ne prennent aucunement en compte la structure du problème ; leurs performances ne dépendent donc que de la qualité du solveur utilisé.

1.3.1.4 Algorithme en deux phases

L'algorithme en deux phases est un cadre méthodologique pour les MOCO proposée par Ulungu et Teghem [133, 134]. L'idée générale est de construire l'ensemble des solutions efficaces X_E en séparant la recherche des solutions supportées X_{SE} et la recherche des solutions non supportées X_{NE}, et de réutiliser des algorithmes efficaces pour le cas mono-objectif quand il en existe. Ces algorithmes ayant été conçus pour profiter pleinement de la structure du problème, toute modification de cette structure empêche leur utilisation. C'est pourquoi on s'interdit avec cette méthode d'ajouter des contraintes sur les valeurs des fonctions objectifs afin de rechercher les solutions efficaces.

La première phase consiste en la construction de X_{SE} (au minimum X_{SE1_m}), en général à l'aide d'une variante de la méthode de Aneja et Nair [3] (section 1.3.1.2).

La seconde phase a pour but la détermination des autres solutions efficaces (X_{NE} et éventuellement X_{SE2}). Pour cela, les points supportés obtenus dans la première phase sont utilisés afin de déterminer une zone de recherche où les points non dominés peuvent éventuellement exister. Cette zone de recherche doit ensuite être explorée à l'aide d'une stratégie énumérative garantissant la détermination d'un ensemble complet. Dans le cas bi-objectif, la zone de recherche est donnée par l'ensemble des triangles dont

les hypoténuses sont définies par des points supportés adjacents (c'est-à-dire consécutifs par rapport à un objectif). En général, ces triangles sont explorés séparément par des techniques énumératives en utilisant des bornes inférieures et supérieures, des coûts réduits... Contrairement à la première phase, l'exploration de la zone de recherche doit utiliser les spécificités du problème pour être efficace.

Cette méthode a été appliquée sur un grand nombre de MOCO bi-objectifs : problème d'affectation, avec une approche de type séparation et évaluation en seconde phase [134] ou avec une approche de type *ranking* [113] ; problème de sac à dos, à nouveau avec une approche de type séparation et évaluation en seconde phase [136] ; problème de flot de coût minimum en variables entières [86, 121], d'arbre couvrant de poids minimum [115, 128], ainsi qu'à un problème d'ordonnancement bi-objectif [87].

Remarquons que ces deux phases exploitent fortement l'ordre naturel des points non dominés dans l'espace bi-objectifs (la première phase en tant que variante de la méthode de Aneja et Nair [3], la seconde dans la détermination de la zone de recherche). Par conséquent, l'application de cette méthode est restée jusqu'à récemment limitée au cas bi-objectif. Dans ses travaux de thèse, Przybylski [111] a levé ce verrou en étendant la méthode à trois objectifs et au delà. Le principe de sa méthode repose sur une extension de la dichotomie pour la première phase et sur une description de la zone de recherche pour la seconde phase, indépendante d'un ordre sur les points non dominés.

1.3.1.5 Autres méthodes exactes

Des méthodes générales de résolution de problèmes combinatoires ont été adaptées du cas mono-objectif vers le cas multi-objectif. Ainsi, plusieurs auteurs ont proposé des variantes multi-objectives de l'approche ϵ-contrainte [65]. Dans cette méthode, un seul objectif est minimisé, les autres étant transformés en contraintes. On considère donc un problème du type :

$$\begin{array}{ll} \max & z_j(x) \\ \text{s.c.} & z_i(x) \geq \epsilon_i \quad \forall i \in \{1, \ldots, p\} \setminus \{j\} \\ & x \in X \end{array}$$

où $\epsilon \in \mathbb{R}^p$.

Avec des choix appropriés pour le vecteur ϵ, toutes les solutions efficaces peuvent être obtenues [65]. En s'appuyant sur ces concepts, des méthodes de partitionnement ont été proposées [30, 88] pour résoudre des problèmes d'optimisation multi-objectif.

De nombreuses méthodes utilisent des techniques de scalarisation pour résoudre des problèmes multi-objectifs reformulés en problèmes mono-objectifs, à l'instar des sommes pondérées, telles que l'algorithme de Benson [12] ou la méthode des contraintes élastiques [36]. Le lecteur désireux d'en apprendre plus sur leur sujet pourra se référer à un récent article de Ehrgott [36].

Les algorithmes de programmation dynamique permettent de déterminer des solutions optimales pour plusieurs problèmes classiques d'optimisation mono-objectif, comme le problème de plus courts chemins, le problème de sac à dos et de nombreux problèmes d'ordonnancement [11, 83, 125]. Cependant, il n'existe pas de méthode générale à base de programmation dynamique pour résoudre des problèmes d'optimisation quelconques. Des méthodes spécifiques à certains problèmes ont néanmoins été généralisées avec succès au cas multi-objectif, comme pour le problème des plus courts chemins [93] ou de sac à dos [10, 22, 79].

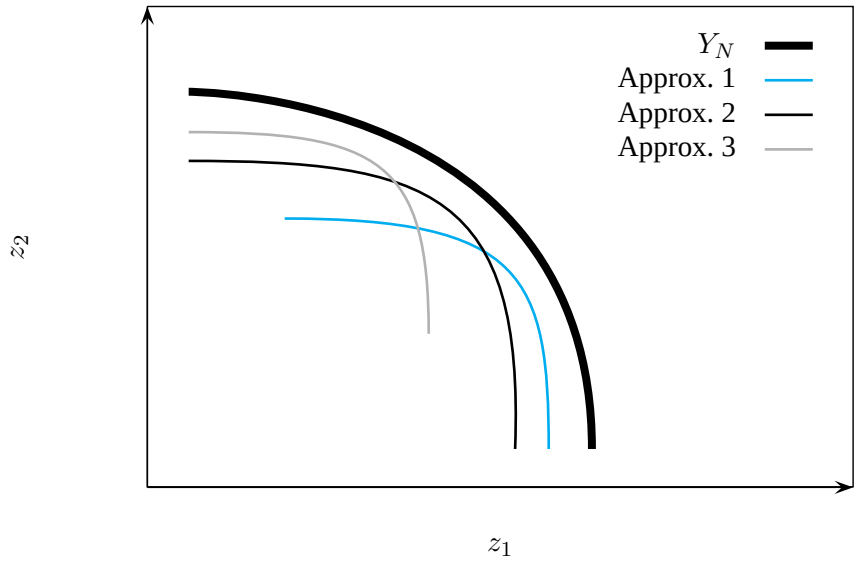

Figure 1.8 – Approximations d'une frontière efficace. Aucune de ces approximations n'est meilleure qu'une autre *a priori*. En effet, bien que la première soit la plus proche de Y_N sur le premier objectif, elles est plus éloignée que les autres sur le second objectif, où la troisième semble être la meilleure.

1.3.2 Méthodes de résolution approchée

Du fait de la difficulté de résolution des MOP, les méthodes exactes sont très limitées en ce qui concerne les problèmes qu'il est possible de résoudre. En effet, même si certaines approches peuvent être efficaces sur certains problèmes, le passage à une taille de problème plus grande représente un principal verrou. C'est pourquoi des méthodes approchées sont utilisées en vue d'obtenir une bonne approximation d'un ensemble complet en un temps raisonnable.

En principe, tout algorithme retournant des solutions réalisables pour une instance d'un MOP donné est un algorithme d'approximation. Cependant, il est évident que le décideur appréciera d'obtenir des solutions d'une qualité relativement proche de celle des solutions efficaces plutôt qu'une solution quelconque. Cette notion de qualité n'est pas évidente en optimisation multi-objectif. La figure 1.8 illustre le problème en présentant plusieurs approximations d'une même frontière efficace, à comparer avec la frontière elle-même. Aucune de ces approximations n'est meilleure qu'une autre *a priori*.

Ces méthodes de résolution se classent en deux grandes catégories : les algorithmes d'approximation et les méta-heuristiques.

1.3.2.1 Algorithmes d'approximation polynomiaux

Une façon intuitive de mesurer la distance entre une solution approchée et une solution optimale est la *garantie relative de performance* qui borne le ratio maximum entre l'approximation et la valeur de la solution optimale. Cette expression de la distance par un pourcentage de la valeur de la solution optimale a l'avantage d'être indépendante de l'ordre de grandeur des coefficients du problème.

Définition 1.14 (Garantie relative de performance (mono-objectif))**.** *Soit $z^*(I)$ la valeur optimale des solutions de l'instance I et $z^A(I)$ la valeur obtenue par l'algorithme d'approximation A.*

L'algorithme A est un algorithme d'approximation avec une garantie relative de performance *égale à $k, 0 < k < 1$ si l'inégalité $\frac{z^A(I)}{z^*(I)} \geq k$ se vérifie pour toute instance I.*

Un algorithme à garantie relative de performance égale à k sera appelé une *k-approximation*. Pour mettre en évidence la différence entre l'approximation et la valeur de la solution optimale, on définit $\epsilon = 1 - k$ et on parlera de *$(1-\epsilon)$-approximation*, vérifiant

$$\frac{z^A(I)}{z^*(I)} \geq 1 - \epsilon \quad \Longleftrightarrow \quad \frac{z^*(I) - z^A(I)}{z^*(I)} \leq \epsilon$$

pour toute instance I.

Cette définition s'étend naturellement au cas multi-objectif, comme défini ci-dessous :

Définition 1.15 (Garantie relative de performance (multi-objectif) [77])**.** *Soit une instance I d'un problème multi-objectif. Soit X^A les solutions réalisables non dominées obtenues par l'algorithme A.*

- *$x^A \in X^A$ est une* $(1-\epsilon)$-approximation *de $x^* \in X_E$ si $z_i(x^A) \geq (1-\epsilon) z_i(x^*)$ pour tout $i \in \{1, \ldots, p\}$;*
- *X^A est une* $(1-\epsilon)$-approximation de la frontière efficace *de I si $\forall x^* \in X_E, \exists x^A \in X^A$ telle que x^A soit une $(1-\epsilon)$-approximation de x^*.*

Il est assez naturel que le temps d'exécution d'un algorithme d'$(1-\epsilon)$-approximation augmente lorsque ϵ diminue. Plus la solution demandée doit être de bonne qualité, plus il faut du temps pour la trouver. Ainsi, il est intéressant de pouvoir limiter cette durée d'exécution dans une certaine mesure.

Définition 1.16 (algorithme d'approximation en temps polynomial)**.** *Un algorithme d'$(1-\epsilon)$-approximation est un* algorithme d'ϵ-approximation en temps polynomial *(PTAS, de l'anglais* polynomial time approximation scheme*) si sa complexité en temps d'exécution est polynomiale en n.*

Définition 1.17 (algorithme d'approximation en temps totalement polynomial)**.** *Un algorithme d'$(1-\epsilon)$-approximation est un* algorithme d'ϵ-approximation en temps totalement polynomial *(FPTAS, de l'anglais* Fully PTAS*) si sa complexité en temps d'exécution est polynomiale en n et polynomiale en $\frac{1}{\epsilon}$.*

La frontière efficace d'une instance d'un problème MOCO peut contenir un nombre arbitrairement grand de solutions. D'un autre côté, d'après [104], pour tout $\epsilon > 0$ il existe une $(1-\epsilon)$-approximation de la frontière efficace faite d'un nombre de solutions polynomial en n et en $\frac{1}{\epsilon}$. Par conséquent, un PTAS pour un MOCO a l'avantage de donner une approximation ayant une qualité garantie, mais aussi de présenter un ensemble de solutions relativement petit au décideur.

Dans [118], Safer et Orlin étudient les conditions nécessaires et suffisantes pour qu'il existe un FPTAS pour un problème MOCO. Ils présentent dans [119] un FPTAS pour divers problèmes de flot, de sac à dos et d'ordonnancement. En particulier, ils prouvent l'existence d'un FPTAS pour le problème de sac à dos multi-objectif.

Récemment, Bazgan *et al.* ont proposé un FPTAS pour ce problème [8, 9]. Glasser *et al.* ont aussi proposé un PTAS pour le problème de couverture de disque multi-objectif [24]. De même, Tsaggouris *et al.* ont présenté un FPTAS pour le problème de plus court chemin multi-objectif [132].

1.3.2.2 Méta-heuristiques multi-objectif

Des méta-heuristiques multi-objectifs sont souvent utilisées pour approximer un ensemble complet. Leur efficacité pour la résolution en pratique de problèmes très difficiles est d'ailleurs à l'origine du regain d'intérêt pour les MOCO dans les années 1990. Ehrgott et Gandibleux [39] classent ces méthodes

essentiellement en deux catégories : les algorithmes évolutionnaires et les algorithmes d'exploration de voisinage. Si les premières méthodes proposées appartenaient clairement à une seule de ces catégories, les méthodes récentes utilisent de plus en plus des caractéristiques de ces deux catégories. Des explications détaillées sur les nombreuses variantes de ces algorithmes sont données dans [40].

Les algorithmes évolutionnaires sont basés sur l'utilisation d'une population de solutions. En maintenant une telle population, cette méthode autorise la détermination de plusieurs solutions potentiellement efficaces parallèlement, en utilisant des mécanismes d'adaptation et de coopération. L'adaptation signifie qu'un individu évolue indépendemment des autres alors que la coopération implique un échange d'information entre ces individus. La population complète contribue au processus d'évolution vers un ensemble complet. On peut parler de méthode globalement convergente, le mécanisme de génération s'exécutant sur une population de solutions. Cette caractéristique rend ce type de méthode très séduisante pour la résolution de problèmes multi-objectifs avec l'avantage d'être peu dépendant du problème considéré.

Dans les algorithmes d'exploration de voisinage, une génération ne s'applique qu'à un seul individu, la solution courante x^k, et ses voisins $\{x \in \mathcal{N}(x^k)\}$. Cette méthode démarre avec une solution initiale et une direction de recherche, se traduisant par un poids $\lambda \in \mathbb{R}^p_>$. La procédure détermine une approximation des points non dominés correspondants à la valeur donnée par λ. Un mécanisme local de pondération des objectifs guide la recherche sur une partie de la frontière efficace Y_N. La convergence est donc locale. Ce principe est répété pour plusieurs directions de recherche pour obtenir une approximation complète de Y_N. Ce type d'algorithme a la caractéristique de converger rapidement grâce à l'effort économisé dans la dispersion. Cependant, cette économie rend plus difficile la couverture de Y_N.

Des exemples d'algorithmes évolutionnaires sont donnés par VEGA (*Vector Evaluated Genetic Algorithm* [120]) ou encore MOGA (*Multiple Objective Genetic Algorithm* [101]). Parmi les algorithmes d'exploration de voisinage, nous pouvons citer nos travaux sur le recuit simulé multi-objectif [55], basé sur la méthode originale de Ulungu [133]. Au même titre que cette dernière, d'autres méta-heuristiques ont été adaptées du cas mono-objectif vers le cas multi-objectif, telles que la recherche tabou MOTS [56, 53] ou bien GRASP [58]. Des méthodes hybridant algorithme évolutionnaire et recherche locale ont aussi été étudiés, comme par exemple MO-GLS (*Multiple Objective Genetic Local Search* [72]), MOGTS (*Multiple Objective Genetic Tabu Search* [6]) ou encore une récente application au problème bi-objectif de *set packing* [29]. Le lecteur souhaitant approfondir sur ces méthodes pourra s'intéresser à [40].

Bien qu'elles permettent souvent d'obtenir des solutions de bonne qualité dans des temps raisonnables, les méta-heuristiques n'offrent aucune garantie sur les performances des solutions produites. Initialement, Ulungu [133] a proposé une mesure de qualité d'une méta-heuristique consistant à compter le nombre des solutions trouvées dans les régions non dominées décrites par les solutions supportées adjacentes. Du fait de cette notion d'adjacence, cette méthode est évidemment limitée au cas bi-objectif. De plus, un facteur limitant son utilisation est qu'elle requiert de connaître les solutions efficaces du problème, ce qui n'est pas toujours possible en pratique. Ces limitations ont encouragé la proposition d'autres mesures, indépendantes de cette connaissance.

Une mesure de la qualité de plus en plus utilisée dans le cadre des méta-heuristiques est la mesure de l'hypervolume de l'espace dominé par la frontière efficace [20, 48, 142]. Cette approche a reçu beaucoup d'attention ces dernières années, en particulier parce qu'elle a la propriété d'être sensible à tout type d'amélioration. C'est-à-dire que lorsqu'une approximation F de la frontière efficace domine une approximation F', alors la mesure donne une valeur strictement meilleure pour F que pour F'.

Par conséquent, la mesure de l'hypervolume garantit que toute approximation F obtenant la meilleure mesure de qualité possible contient tous les points non dominés du problème.

Définition 1.18 (Fonction d'accomplissement)**.** *Soit F une approximation de la frontière efficace. La* fonction d'accomplissement $\alpha_F : \mathbb{R}^p \to \{0,1\}$ *de F est définie pour $y \in Y$ par*

$$\alpha_F(y) := \begin{cases} 1 & \textit{si } \exists y' \in F : y' \geqq y \\ 0 & \textit{sinon} \end{cases}$$

Définition 1.19 (Indicateur hypervolumique)**.** *Soit F une approximation de la frontière efficace et y^U un point utopique. L'*indicateur hypervolumique I_H^* *de F avec pour point de référence $(0,\ldots,0)$ est exprimé par*

$$I_H^*(F) := \int_{(0,\ldots,0)}^{y^U} \alpha_F(y)dy$$

Zitzler *et al.* [141] ont récemment proposé une version modifiée de I_H^* permettant d'introduire une information de précédence tout en conservant la notion de dominance de Pareto.

1.4 Questions ouvertes, conclusions

Bien qu'il existe des algorithmes exacts performants pour certains problèmes MOCO, il n'existe aujourd'hui aucune méthode efficace pour résoudre des problèmes MOCO quelconques, c'est à dire qui ne correspondent à aucune structure de problème en particulier. Une piste intéressante concerne la généralisation des procédures de séparation et évaluation (PSE) vers le cas multi-objectif. Des travaux en ce sens ont été effectués par Sourd et Spanjaard [126] pour le problème bi-objectif d'arbre couvrant minimal, avec des résultats expérimentaux très encourageants. Leur procédure utilise la notion de borne étendue au cas multi-objectif [41]. Cependant, bien que la définition de bornes multi-objectif soit bien posé, leur application directe reste coûteuse. Ainsi, le calcul et la comparaison de bornes est un domaine à approfondir et à améliorer.

De plus, Degoutin et Gandibleux ont montré dans [28] que, contrairement à plusieurs suppositions acceptées, Y_N n'est pas uniformément dense. Les caractéristiques des instances numériques jouent un rôle crucial sur la structure de Y_N et par conséquent sur la performance d'un algorithme. Dans le cas mono-objectif, l'étude d'instances générées aléatoirement a permis une bonne compréhension des propriétés des solutions du problème, comme par exemple pour le problème d'affectation [82]. Przybylski *et al.* ont été les premiers à proposer des résultats sur le sujet pour la version bi-objectif de ce problème, justifiant ainsi l'efficacité de leur méthode en deux phases [113, 112]. Cependant, peu de résultats sont disponibles pour d'autres problèmes.

Finalement, il serait illusoire de prétendre à résoudre exactement des problèmes quelconques avec une seule procédure s'il est déjà difficile de résoudre un problème particulier avec une telle méthode, voire avec une méthode spécifique. Dans la suite de nos travaux, nous utiliserons le problème de sac à dos multi-objectif comme support. Nous proposerons une étude sur l'influence des valeurs des objets dans les solutions efficaces ainsi que deux procédures en deux phases pour résoudre des instances. La première sera spécifique au cas bi-objectif tandis que la seconde portera sur le cas multi-objectif en général. Pour ce dernier cas, nous proposerons aussi deux PSE. Nous apporterons une attention particulière à l'influence des bornes sur les performances de ces dernières.

CHAPITRE 2

Problèmes de sac à dos multi-objectif

Le sac à dos est un problème $\mathcal{NP}$-complet fondamental de l'optimisation combinatoire. De part ses nombreuses variantes, il est au cœur de nombreux problèmes d'optimisation et, bien qu'il soit étudié depuis plusieurs décennies, il reste un sujet actif dans le domaine ; en attestent l'état de l'art effectué récemment par Kellerer *et al.* [77] et les non moins récentes publications portant tant sur sa résolution exacte [47, 98] que sur une résolution approchée [18, 50, 137].

L'intérêt du problème n'est pas que théorique. En effet, de nombreux cas pratiques se formulent sous la forme d'un problème de sac à dos ou similaire. Il est en particulier à l'origine de plusieurs algorithmes de chiffrement [97], bien que ces méthodes se soient avérées peu fiables [124, 135]. C'est aussi un modèle classiquement utilisé dans la gestion de portefeuilles [129], le transport [140] et bien d'autres domaines [19].

Dans ce chapitre, nous présentons le problème de sac à dos dans sa version la plus simple ainsi que les méthodes usuelles de résolution. Nous étendrons sur quelques unes de ses nombreuses variantes, telles que le sac à dos multi-dimensionnel et le problème de *set packing*. Nous détaillerons en particulier la variante du sac à dos multi-objectif unidimensionnel en variables binaires qui sera le problème sur lequel le reste de cette thèse s'appuiera.

2.1 Le problème de sac à dos

Le problème de sac à dos est un problème d'optimisation combinatoire modélisant une situation dans laquelle un décideur dispose d'un ensemble d'objets parmi lesquels il doit faire une sélection, en respectant une contrainte de capacité. L'objectif est de maximiser le profit apporté par les objets choisis.

Dans sa forme la plus simple, dénomée « sac à dos unidimensionnel en variables binaires », le problème se formule ainsi :

$$\left.\begin{array}{lll} \max & z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c. & \sum_{i=1}^{n} w_i x_i \leq \omega & \omega, w_i \in \mathbb{N}^* \\ & x_i \in \{0,1\} & \forall i \in \{1,\ldots,n\} \end{array}\right\} 01KP$$

où c_i est le profit apporté par la sélection de l'objet i et w_i est son poids. La contrainte exprime le fait que les objets sélectionnés doivent tenir dans la capacité ω du sac à dos. Selon qu'un objet i est sélectionné ou non, la variable associée x_i prend, respectivement, la valeur 1 ou 0.

Définition 2.1 (Tightness ratio)**.** *On appelle* tightness ratio *le ratio de la somme des poids des objets sur la capacité du sac :*

$$r = \frac{\sum_{i=1}^{n} w_i}{\omega}$$

Il est généralement admis que les instances pour lesquelles ce ratio est proche de 0,5 font intervenir une combinatoire plus grande.

Le problème de sac à dos est le plus simple des problèmes non triviaux d'optimisation linéaire en variables binaires : une seule contrainte et uniquement des coefficients entiers positifs. Cette simplicité lui confère une importance toute particulière dans le domaine de l'optimisation. En 1897, Mathews montrait déjà comment agréger plusieurs contraintes d'un problème d'optimisation en une seule contrainte de type sac à dos [94].

Malgré cette apparente simplicité, il est l'un des 21 problèmes $\mathcal{N}P$-complets cités par Richard Karp [74]. Il accepte cependant un algorithme de résolution d'une complexité pseudo polynomiale en $O(n\omega)$. Néanmoins, bien que les algorithmes dédiés aient profité de nombreuses améliorations durant les dernières décennies et bien que de nombreuses instances de référence soient aujourd'hui résolues dans des temps raisonnables, de nombreuses autres restent encore intraitables. En particulier, Pisinger [109] a récemment proposé une description de certaines de ces instances difficiles.

Le problème de sac à dos se retrouve en tant que sous problème de nombreux problèmes d'optimisation combinatoire ; ce qui n'est pas sans accentuer son importance [37]. Attestant de cet intérêt, Kellerer *et al.* ont récemment publié un livre [77] présentant un état de l'art de la résolution du problème et de ses variantes.

2.1.1 Variantes autour du problème

Il existe de nombreuses variantes du problème de sac à dos, selon le domaine des variables (valeurs binaires, entières ou réelles), le nombre de contraintes (unidimensionnel, bi-dimensionnel ou multi-dimensionnel), le nombre de profits associés à chaque objet (mono-objectif ou multi-objectif), le nombre de sacs, etc. Du fait de la quantité des paramètres intervenant dans la formulation, les variantes sont nombreuses. Cette section présente quelques unes d'entre elles. Le lecteur désireux de connaître plus de détails sur ces variantes ou sur d'autres pourra se référer à [77, 137].

2.1.1.1 Variables continues

Le problème de *sac à dos en variables continues* (LKP) est une variante dans laquelle il est possible de ne prendre qu'une fraction des objets. Sa résolution s'appuie sur les concepts d'efficacité d'un objet et d'élément bloquant, définis ci-dessous.

Définition 2.2 (Efficacité d'un objet)**.** *On appelle* efficacité d'un objet *le rapport de son coût sur son poids, noté* $e_i = \frac{c_i}{w_i}$.

Remarque. La notion d'efficacité d'un objet est à distinguer de la notion d'efficacité d'une solution dans le cadre multi-objectif.

Définition 2.3 (Élément bloquant)**.** *On appelle* élément bloquant *le premier objet ne pouvant tenir dans le sac lorsque les objets sont ajoutés par ordre décroissant d'efficacité. Son indice sera noté*

$$s = \min \left\{ k : \sum_{i=1}^{k} w_i > \omega \right\}, \textit{ pour les } e_i \textit{ triés en orde décroissant}$$

La valeur optimale $z^*(LKP)$ d'une instance est obtenue en prenant les objets dans l'ordre décroissant de leur efficacité, jusqu'à l'élément bloquant puis en ajoutant la fraction de cet élément permettant de saturer le sac [27] :

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

Ce problème correspond à la relaxation linéaire de $01KP$. Ainsi, si I est une instance de ce dernier et I^{LP} est l'instance obtenue à partir de I en remplaçant la contrainte $x_i \in \{0,1\}, \forall i \in \{1,\ldots,n\}$ par $x_i \in [0;1], \forall i \in \{1,\ldots,n\}$, alors l'inégalité $z^*(I) \leq z^*(I^{LP})$ se vérifie, où $z^*(I)$ est la valeur optimale des solutions d'une instance I. De plus, LKP appartient à la classe de complexité $\mathcal{P}$. Pour cela, il est souvent utilisé dans la résolution des variantes à variables entières, dans l'optique d'obtenir une borne supérieure de la solution optimale en un temps raisonnable.

2.1.1.2 Sac à dos multi-dimensionnel

Le problème de *sac à dos multi-dimensionnel* ($d-KP$), est une variante du problème ayant plusieurs contraintes de capacité. Il est à la frontière entre l'optimisation combinatoire et la programmation linéaire en nombres entiers. En effet, comme l'atteste sa formulation ci-dessous, il peut être considéré comme un programme linéaire en nombres entiers (ILP) classique sous la seule restriction que les coefficients soient positifs et les variables binaires. Son champ d'application est large, ce qui a grandement contribué à sa popularité.

$$\left.\begin{array}{lll} \max & z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c. & \sum_{i=1}^{n} w_i^j x_i \leq \omega_j & \omega_j, w_i \in \mathbb{N}^d_{\geq}, j \in \{1,\ldots,d\} \\ & x_i \in \{0,1\} & \forall i \in \{1,\ldots,n\} \end{array}\right\} d-KP$$

Comme le souligne un récent état de l'art de Fréville [51], cette variante a été intensément utilisée pour modéliser des situations de gestion de portefeuilles ou de sélection de projets. Il est, de plus, largement utilisé dans l'évaluation de méta-heuristiques [1], en particulier pour la recherche tabou et les algorithmes génétiques.

Cette variante est intrinsèquement difficile. Ainsi, les méthodes de résolution exacte actuelles se limitent à des instances de quelques centaines de variables pour une dizaine de contraintes. De plus, [60] et [81] ont prouvé qu'il n'existait pas de FPTAS pour ce problème (une preuve est plus facilement accessible dans [77]), bien qu'il accepte des PTAS. Ainsi, la meilleure solution pour résoudre une instance de $d-KP$ aujourd'hui consiste en l'utilisation d'heuristiques [49] ou de méta-heuristiques [21, 80, 99].

2.1.1.3 Problème de *set packing*

Le problème de *set packing* est une variante du problème de sac à dos multi-dimensionnel dans laquelle les poids sont binaires et les capacités sont toutes égales à 1.

$$\left.\begin{array}{lll} \max & z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c. & \sum_{i=1}^{n} w_i^j x_i \leq 1 & w_i^j \in \{0,1\}, j \in \{1,\ldots,d\} \\ & x_i \in \{0,1\} & \forall i \in \{1,\ldots,n\} \end{array}\right\} SPP$$

Très peu de méthodes exactes existent pour ce problème [25, 103], en particulier parce que sa relaxation linéaire ne donne pas de bonnes bornes. Ainsi, des heuristiques et méta-heuristiques sont en général utilisées pour résoudre des instances de ce problème avec un grand nombre de variables et de contraintes.

Gandibleux *et al.* [54, 73] ont récemment proposé une application de la méta-heuristique d'optimisation par colonie de fourmis (ACO, de l'anglais *Ant Colony Optimization*) pour résoudre des instances de ce problème, avec une application directe au problème d'infrastructure ferroviaire. Ce type de méta-heuristique entre dans la catégorie des algorithmes à population. Elles ont été inspirées par l'observation des traces de phéromones déposées par certaines espèces de fourmis [31]. Durant la recherche de nourriture, ces traces s'intensifient sur les chemins reliant la fourmilière à la réserve et tendent, à force de nombreux aller-retours, à décrire le chemin optimal pour aller de l'une à l'autre.

Le développement de cette méta-heuristique fait partie du projet RECIFE [116] visant à produire des outils d'aide à la décision s'appuyant sur les techniques issues de la recherche opérationnelle pour l'étude spécifique des nœuds ferroviaires ou des gares.

2.1.1.4 Sac à dos multi-objectif

Le problème de *sac à dos multi-objectif* ($01MOKP$), est une variante du problème où plusieurs objectifs sont à maximiser simultanément.

$$\left.\begin{array}{lll} \max & z_j(x) = \sum_{i=1}^{n} c_i^j x_i & j \in \{1,\ldots,p\}, c_i \in \mathbb{N}_{\geq}^{p} \\ s.c. & \sum_{i=1}^{n} w_i x_i \leq \omega & \omega, w_i \in \mathbb{N}^* \\ & x_i \in \{0,1\} & \forall i \in \{1,\ldots,n\} \end{array}\right\} 01MOKP$$

Cette variante possède toutes les difficultés inhérentes aux problèmes d'optimisation combinatoire multi-objectifs, telles que présentées dans le chapitre 1. Bien sûr, toutes les variantes peuvent être considérées dans le cas multi-objectif.

Ce problème sera approfondi à partir de la section 2.4.

2.1.1.5 Autres variantes

Une autre variante sur le domaine des variables consiste à autoriser plusieurs exemplaires pour les objets, voire même à autoriser un objet à être en quantité infinie. On parlera dans ce cas de *sac à dos en variables entières*, noté KP. Une instance de KP où chaque objet i est disponible en $b_i \in \mathbb{N}$ exemplaires

peut aisément être transformée en une instance de $01KP$ en considérant une variable binaire $x_{i,j}$ pour chaque exemplaire $j \in \{0, \ldots, b_i\}$ de chaque objet i. Cependant, une telle approche, en plus d'engendrer une augmentation de la taille du problème pouvant atteindre $O(n\omega)$ [77], introduit le problème des solutions dupliquées. En effet, si x est une solution de l'instance de $01KP$ telle qu'il existe i, j, k tels que $x_{i,j} \neq x_{i,k}$, alors cette solution est identique du point de vue de l'instance initiale de KP à la solution obtenue en remplaçant dans x la valeur de $x_{i,j}$ par $1 - x_{i,j}$ et celle de $x_{i,k}$ par $1 - x_{i,k}$. Par conséquent, il reste intéressant de développer des algorithmes spécifiquement pour ces variantes [47].

Un cas particulier de $01KP$ est celui dans lequel le coût et le poids des objets sont les mêmes ($c_i = w_i, \forall i \in \{1, \ldots, n\}$). Cette situation correspond au problème de la *somme des sous-ensembles*.

Il existe de nombreuses autres variantes qui ne seront pas détaillées ici. Telles que le *sac à dos à choix multiple*, dans lequel les objets sont répartis dans m classes. Il s'agit alors de choisir exactement un objet par classe de manière à maximiser le profit. Le *sac à dos multiple*, quant à lui, est une variante dans laquelle l'utilisateur dispose de plusieurs sacs à dos, parmi lesquels il doit répartir les objets ; tout en maximisant la somme des profits de tous les sacs cumulés. Le lecteur pourra se référer à Kellerer *et al.* [77] pour une étude récente de ces variantes.

2.2 Méthodes classiques de résolution

Le problème de sac à dos est étudié depuis plusieurs décennies, au cours desquelles de nombreuses méthodes de résolution ont vu le jour, tant exactes que approchées [138]. Les premières se regroupent en deux grandes familles : les procédures de séparation et évaluation et les algorithmes de programmation dynamique. Plusieurs auteurs ont néanmoins proposé des procédures hybridant ces deux approches [33, 110]. Nous présenterons ici ces deux approches, à la base de nombreuses méthodes exactes.

Bien que ces algorithmes bénéficient de nombreuses recherches actives et profitent ainsi de nombreuses améliorations, il existe toujours des instances pour lesquelles ils sont inefficaces [109]. Dans de telles situations, une procédure de résolution approchée s'impose comme seule alternative. La plus connue de ces procédures, et aussi une des moins performantes, est l'algorithme glouton. Celui-ci sera présenté dans cette section pour sa position historique.

L'exemple 2.10 ci-dessous reprend le problème de l'exemple 1.4 page 11 pour illustrer diverses approches élémentaires pour la résolution d'une instance du problème.

Exemple 2.10 (Résolution d'un problème de sac à dos mono-objectif). *Cet exemple reprend la situation présentée dans l'exemple 1.4 page 11. Celle-ci est rapportée ci-dessous.*

Le décideur possède une bibliothèque musicale contenant 10 titres. Le logiciel qu'il utilise pour gérer sa bibliothèque lui permet de donner une note dans un intervalle de 1 à 5 étoiles à chaque titre, 5 étant la meilleure note. Les informations sur les titres sont récapitulées dans la table 1.1.
Son lecteur multimédia portable a une capacité de 16000000 octets. Il souhaite le remplir de manière à avoir la meilleure liste de lecture possible à partir de sa bibliothèque. C'est à dire que la somme des notes des titres choisis doit être maximale.

Une approche naïve consiste à choisir les titres dans l'ordre décroissant de leurs notes, jusqu'à remplir le lecteur. La solution ainsi obtenue est constituée des titres numéros 4 et 5 et a une note totale de 10 étoiles pour un poids de 14800000 octets.

Une autre approche intuitive est de choisir les titres dans l'ordre décroissant du rapport de leur note sur leur poids. La solution obtenue avec cette méthode est alors constituée des titres numéros 1, 2 et 3, avec une note à nouveau égale à 10, pour un poids cette fois égal à 11900000 octets.
Ces deux procédés donnent des solutions équivalentes mais différentes. Cependant, aucune d'entre elles n'est optimale. Pour trouver une telle liste, il suffit de construire un problème de sac à dos où les objets sont les titres. Le coût c_i de l'objet numéro i est la note associée au titre et son poids w_i est sa taille en octets. La capacité ω du sac est égale à la capacité du lecteur. Le problème ainsi construit accepte une unique solution optimale composée des titres numéros 1, 2 et 4, avec une note de 13 étoiles pour un poids de 16000000 octets exactement.

2.2.1 Méthodes de résolution exacte

Il existe deux principaux modèles d'algorithmes efficaces pour résoudre exactement une instance du problème de sac à dos. Le premier consiste en une procédure de séparation et évaluation, le second est une approche de programmation dynamique. Les deux approches peuvent aussi être combinées pour obtenir un algorithme hybride [110]. La suite de cette section présente les concepts derrière ces deux approches.

2.2.1.1 Procédure de séparation et évaluation

Le concept algorithmique derrière les procédures de séparation et évaluation consiste en une énumération partielle de l'espace des solutions d'une manière intelligente. En général, seule une petite partie de l'espace est explorée explicitement et il est garanti que l'espace non considéré de manière explicite ne contient aucune solution optimale. Les solutions de cette partie sont dites énumérées implicitement.

Durant l'étape de *séparation*, un sous-ensemble $X' \subseteq X$ de l'espace des solutions est divisé en deux partitions X_1 et X_2, telles que $X_1 \cup X_2 = X'$. Le processus est répété jusqu'à obtenir des ensembles ne contenant qu'une seule solution ou jusqu'à ce qu'il soit montré que la poursuite de la recherche dans le sous-espace soit sans intérêt. Une solution optimale du problème est alors une meilleure solution parmi les solutions obtenues.

La partie *évaluation* consiste à calculer une borne inférieure $\underline{z}$ et une borne supérieure $\bar{z}(X')$ pour un ensemble X' donné, de sorte que

$$z(x) \leq \bar{z}(X'), \quad \forall x \in X'$$

Une borne inférieure peut être la valeur de la meilleure solution trouvée jusqu'à cet instant de la recherche, ou la valeur d'une solution obtenue avec une heuristique.

La borne supérieure est utilisée pour fermer certaines régions de l'espace à explorer. Supposons que $\bar{z}(X') \leq z(x)$, où x est la meilleure solution trouvée jusqu'à cet instant, alors l'ensemble X' ne contient aucune solution meilleure que x. Ainsi, il est inutile de continuer à explorer X'.

L'algorithme 2.1 illustre une procédure de séparation et évaluation pour $01KP$, sans le détail du calcul de la borne supérieure. De nombreuses procédures de séparation et évaluation ont été proposées ; les travaux de Martello et Toth [92] font référence sur ce sujet.

Une bonne borne supérieure doit être rapide à calculer tout en étant serrée, c'est à dire proche de la valeur de la meilleure solution de X'. Cependant, ces deux qualités s'opposent en pratique.

Procédure `pse`($\updownarrow x, \downarrow i, \updownarrow x^*$)
Paramètre $\updownarrow x$: la solution en cours de construction.
Paramètre $\downarrow i$: l'indice de l'objet sur lequel opérer.
Paramètre $\updownarrow x^*$: la meilleure solution trouvée.

- - La solution est-elle réalisable ? - -
1: **si** $\sum_{j=1}^{i-1} w_j x_j \leq \omega$ **alors**
- - Mise à jour de la meilleure solution connue. - -
2: **si** $z(x) > z(x^*)$ **alors**
3: $x^* \leftarrow x$
4: **fin si**
5: **si** $i \leq n$ **alors**
6: calculer une borne supérieure $\bar{z}$
- - Séparation de la recherche sur les sous-espaces disjoints vérifiant $x_i = 1$ *ou* $x_i = 0$. - -
7: **si** $\bar{z} > z(x^*)$ **alors**
8: $x_i \leftarrow 1$
9: `pse`$(x, i+1, x^*)$
10: $x_i \leftarrow 0$
11: `pse`$(x, i+1, x^*)$
12: **fin si**
13: **fin si**
14: **fin si**

ALG. 2.1 – Une procédure de séparation et évaluation pour $01KP$

Si i est l'objet le plus efficace, la borne la plus simple pour $01KP$ peut être obtenue en relâchant la contrainte d'intégrité $x_i \in \{0,1\}$ en $x_i \in \mathbb{R}$. Cela donne la borne triviale :

$$U_0 = \left\lfloor \frac{\omega c_i}{w_i} \right\rfloor \tag{2.1}$$

Si les objets ont été initialement triés en ordre décroissant d'efficacité, cette borne se calcule en $O(1)$. Autrement, il faut trouver l'objet le plus efficace en $O(n)$.

Une autre borne supérieure est obtenue en retirant la contrainte d'intégrité sur toutes les variables, de manière à obtenir le problème LKP (section 2.1.1).

$$U_1 = \lfloor z^*(LKP) \rfloor$$

Cette borne est plus serrée que U_0 et peut se calculer en $O(n)$ si les objets ont été initialement triés en ordre décroissant d'efficacité.

En considérant séparément le cas ou l'objet bloquant est ou n'est pas ajouté à la solution, Martello et Toth [92] ont défini une troisième borne supérieure :

$$U_2 = \sum_{i=1}^{s-1} c_i + \max\left\{ \left\lfloor \bar{\omega} \frac{c_{s+1}}{w_{s+1}} \right\rfloor, \left\lfloor c_s - (w_s - \bar{\omega}) \frac{c_{s-1}}{w_{s-1}} \right\rfloor \right\}$$

où s est l'indice de l'élément bloquant et $\bar{\omega} = \omega - \sum_{i=1}^{s-1} w_i$. Cette borne est plus fine que U_1 et il est montré que $U_2 \leq U_1 \leq U_0$.

D'autres bornes supérieures ont été définies dans [46, 91, 100] ou encore [32]. Ces bornes sont peu utilisées dans les procédures de séparation et évaluation du fait de la complexité nécessaire à leur calcul.

Nous reviendrons sur le cadre des PSE dans le chapitre 6.

2.2.1.2 Programmation dynamique

La programmation dynamique est une approche générale qui se retrouve comme technique de résolution en recherche opérationnelle. Cette technique peut être appliquée dès lors qu'une solution optimale consiste en une combinaison de solutions optimales de sous problèmes, ce qui est le cas pour le sac à dos.

Étant donné une instance du problème de sac à dos, on considère le problème restreint à ses j premiers objets et à une capacité d, défini comme suit :

$$\left.\begin{array}{lll} \max & z_{j,d} = \displaystyle\sum_{i=1}^{j} c_i x_i & \\ s.c. & \displaystyle\sum_{i=1}^{j} w_i x_i \leq d & \\ & x_i \in \{0,1\} \qquad i \in \{1,\ldots,j\} & \end{array}\right\} KP(j,d)$$

Si $z_{j-1,d}$ est connu pour toutes les capacités $d = 0,\ldots,\omega$, alors la valeur de $z_{j,d}$ est donnée par la formule suivante [11] :

$$z_{j,d} = \begin{cases} z_{j-1,d} & \text{si } d < w_j \\ \max\{z_{j-1,d}, z_{j-1,d-w_j} + c_j\} & \text{si } d \geq w_j \end{cases} \tag{2.2}$$

Le premier cas correspond à la situation dans laquelle le sac à dos est trop petit pour contenir l'objet j. Par conséquent, cet objet ne change pas la valeur optimale $z_{j-1,d}$. Si l'objet j peut être contenu dans le sac, ce qui correspond au second cas, il y a deux possibilités : soit l'objet n'appartient pas à la solution optimale, auquel cas la valeur $z_{j-1,d}$ est conservée, soit l'objet appartient à la solution optimale et contribue de c_j à sa valeur mais ne laisse qu'une capacité de $d - w_j$ pour les objets dans $\{1, \ldots, j-1\}$. Évidemment, cette capacité restante pour les objets précédents doit être complétée avec un profit maximum, ce qui est la valeur optimale $z_{j-1,d-w_j}$ du sous problème $KP(j-1, d-w_j)$.

En initialisant $z_{0,d} \leftarrow 0$ pour $d \in \{0, \ldots, \omega\}$, le calcul des valeurs $z_{j,d}$ par récursion de $j = 1$ à n dans l'algorithme 2.2 amène à la valeur $z_{n,\omega}$ des solutions optimales. Cet algorithme ne calcule pas le vecteur correspondant à une solution optimale. Néanmoins, il peut être déduit *a posteriori* si la table des valeurs $z_{i,d}$ est conservée. De nombreuses contributions ont été faites pour améliorer la complexité de la résolution de KP par programmation dynamique [68, 131] ou, plus récemment, [44, 106].

Procédure `programmation_dynamique`()
Sortie La performance des solutions optimales.

1: **pour** $d \in \{0, \ldots, \omega\}$ **faire**
2: $\quad z_{0,d} \leftarrow 0$
3: **fin pour**
4: **pour** j **de** 1 **à** n **faire**
5: $\quad$ **pour** d **de** 0 **à** ω **faire**
6: $\quad\quad$ **si** $d \geq w_j$ **alors**
7: $\quad\quad\quad z_{j,d} \leftarrow \max\{z_{j-1,d}, z_{j-1,d-w_j} + c_j\}$
8: $\quad\quad$ **sinon**
9: $\quad\quad\quad z_{j,d} \leftarrow z_{j-1,d}$
10: $\quad\quad$ **fin si**
11: $\quad$ **fin pour**
12: **fin pour**
13: **retourner** $z_{n,\omega}$

ALG. 2.2 – Programmation dynamique pour $01KP$

2.2.1.3 Analogie avec le problème du plus long chemin

Le problème de sac à dos peut être résolu de manière similaire à la recherche d'un plus long chemin dans un graphe. Classiquement, ce graphe correspondant à une instance du problème de sac à dos est obtenu en associant un sommet à chaque état $z_{j,d}$ de la programmation dynamique. Chaque sommet regroupe ainsi les solutions du sous problème $KP(j, d)$. Un ou deux arcs entrants sont créés pour chaque sommet $z_{j,d}, j > 1$. Le premier est construit depuis $z_{j-1,d}$ avec un poids de 0, quel que soit d. Le second est défini depuis $z_{j-1,d-w_j}$ si $d \geq w_j$, avec un poids égal à c_j. Dans le premier cas, la sélection de l'arc dans un chemin représente la non sélection de l'objet j à partir des solutions de $KP(j-1, d)$. Similairement, la sélection de l'arc pondéré du second cas représente la sélection de l'objet j à partir des solutions de $KP(j-1, d-w_j)$.

Un sommet puits z_{n+1} est aussi ajouté, relié à partir des sommets $z_{n,d}$ pour tout $d \in \{0, \ldots, \omega\}$.

En pratique, il n'est utile que de considérer les sommets atteignables à partir du sommet $z_{0,0}$ représentant une solution initiale vide. Les plus longs chemins entre les sommets $z_{0,0}$ et z_{n+1} correspondent aux solutions optimales du problème initial.

Ce modèle est populaire du fait de sa correspondance avec la programmation dynamique. En effet, on retrouve dans la construction du graphe l'équation 2.2 et la construction décrite par l'algorithme 2.2. Il existe cependant d'autres modèles permettant de transformer une instance du problème de sac à dos en problème de plus long chemins [59, 70, 79]. Ces différents modèles font varier le nombre de sommets, d'arcs, le degré minimal et maximal et, par conséquent, la performance des algorithmes.

Exemple 2.11 (Graphe classiquement associé à une instance du problème de sac à dos). *Soit l'instance de sac à dos suivante*

$$\begin{array}{lll} \max & z(x) = & 2x_1 + 9x_2 + 5x_3 + 8x_4 \\ s.c. & & 8x_1 + 5x_2 + 7x_3 + 4x_4 \quad \leq 12 \\ & & x_i \in \{0,1\} \qquad\qquad\quad \forall i \in \{1,\ldots,4\} \end{array}$$

Ce problème accepte une unique solution optimale $x^* = (0,1,0,1)$, avec $z(x^*) = 17$.
Le graphe associé à cette instance est illustré par la figure 2.1, où le plus long chemin au départ de $z_{0,0}$ est mis en évidence. Chaque arc non nul (« diagonal ») de ce chemin correspond à une variable de x^* égale à un. Les autres arcs (« horizontaux ») correspondent à des variables à zéro.

2.2.2 Méthodes de résolution approchée

Une procédure de résolution approchée de $01KP$ bien connue est un algorithme glouton consistant en la sélection des objets dans l'ordre décroissant de leur efficacité (définition 2.2). Il s'agit de la seconde approche proposée dans l'exemple 2.10.

Procédure `glouton`($\updownarrow x$)
Paramètre $\updownarrow x$: la solution construite par le glouton.

1: trier les objets par ordre décroissant d'efficacité
2: $\bar{\omega} \leftarrow \omega$
3: **pour** i **de** 1 **à** n **faire**
4: **si** $w_i \leq \bar{\omega}$ **alors**
5: $x_i \leftarrow 1$
6: $\bar{\omega} \leftarrow \bar{\omega} - w_i$
7: **sinon**
8: $x_i \leftarrow 0$
9: **fin si**
10: **fin pour**

ALG. 2.3 – Algorithme glouton $01KP$

Cette procédure est rapide mais peut donner dans le pire des cas des résultats particulièrement mauvais, comme le montre l'exemple 2.12. Elle est ainsi principalement utilisée en préparation pour une résolution plus coûteuse, par exemple pour obtenir une première borne inférieure.

Exemple 2.12 (Situation inadaptée de l'algorithme glouton). *Soit une instance de sac à dos telle que $n = 2, c_1 = 2, w_1 = 1$ et $c_2 = \omega, w_2 = \omega$, avec $\omega > 1$. L'algorithme glouton choisira en premier l'objet*

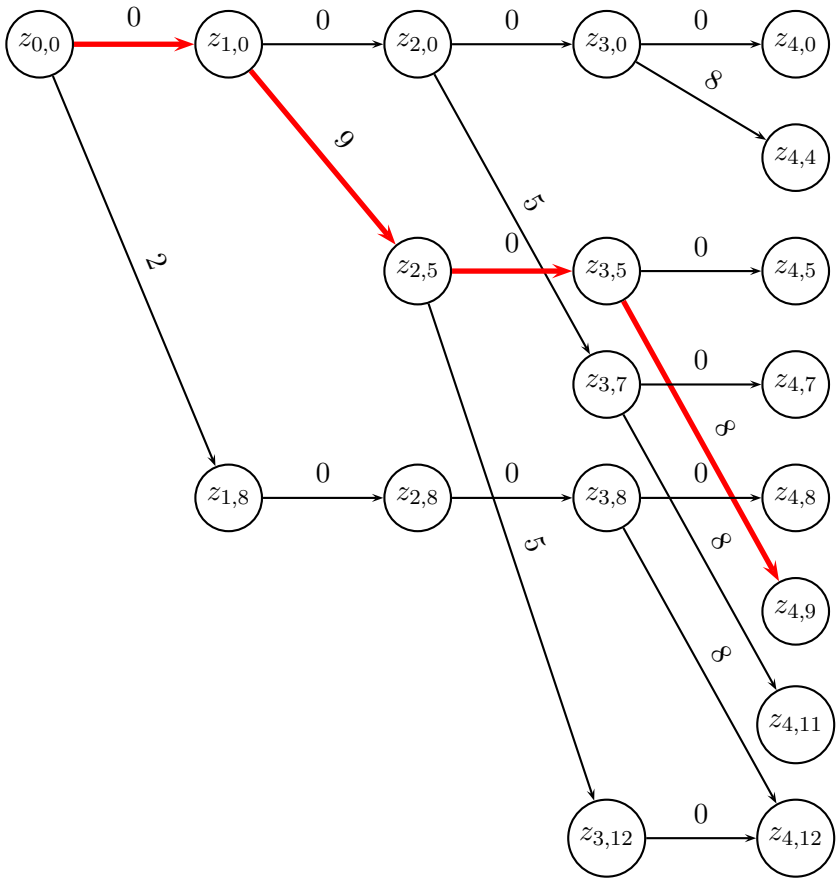

Figure 2.1 – Graphe associé à l'instance de l'exemple 2.11. Le plus long chemin au départ de $z_{0,0}$ est mis en évidence. Chaque arc non nul (« diagonal ») de ce chemin correspond à une variable de x^* égale à un. Les autres arcs (« horizontaux ») correspondent à des variables à zéro.

numéro 1, il est le plus efficace, empêchant ainsi la sélection du second. Or, la sélection de ce dernier aurait donné une solution d'une bien meilleure qualité.

En choisissant ω suffisament grand pour l'instance de cet exemple, le rapport $\frac{z(x^G)}{z(x^*)}$, avec x^* une solution optimale et x^G la solution retournée par l'algorithme glouton, peut être aussi proche de zéro que souhaité. Pour obtenir une approximation d'une meilleure qualité, il conviendra d'utiliser un algorithme d'approximation de type FPTAS comme [85, 89] ou, plus récemment, [75, 76].

2.3 Prétraitements pour le sac à dos mono-objectif

Il est clairement intéressant de réduire la taille d'une instance du problème avant de le résoudre, en particulier s'il s'agit d'une résolution exacte, mais aussi dans le cas d'une résolution approchée. La réduction du nombre de variables diminue considérablement le temps d'exécution des algorithmes pour quasiment toutes les instances pratiques, bien que cela ne change pas la complexité des algorithmes.

2.3.1 Réduction de la taille des instances

Les procédures de réduction consistent en une séparation de l'ensemble $\{1, \ldots, n\}$ des variables en trois sous-ensembles :

$$J^1 = \{i \in \{1, \ldots, n\} : x_i = 1, \forall x \in X^*\} \tag{2.3}$$
$$J^0 = \{i \in \{1, \ldots, n\} : x_i = 0, \forall x \in X^*\} \tag{2.4}$$
$$F = \{1, \ldots, n\} \backslash (J^0 \cup J^1) \tag{2.5}$$

où X^* est l'ensemble des solutions optimales.

L'instance initiale peut ensuite être remplacée par une instance réduites aux seules variables de F et où la capacité du sac est égale à $\omega - \sum_{i \in J^1} w_i$. Les solutions optimales du problème initial sont alors celles du problème réduit, auxquelles les objets de J^1 sont ajoutés.

L'idée principale pour le calcul de J^0 et de J^1 est la suivante. Si l'affectation $x_i \leftarrow b$ ($b = 0$ ou 1) implique une solution de moins bonne qualité que la meilleure solution connue, alors x_i doit prendre la valeur $1 - b$ dans toutes les solutions optimales.

Soit $\underline{z}$ la valeur d'une solution réalisable, obtenue par exemple à l'aide de l'algorithme glouton. Soit $\bar{z}_i^b$ une borne supérieure calculée en fixant $x_i = b$. Les ensembles J^0 et J^1 sont alors définis par

$$J^{(1-b)} = \{i \in \{1, \ldots, n\} : \bar{z}_i^b < \underline{z}\}$$

Toutes les bornes présentées en 2.2.1.1 peuvent être utilisées pour le calcul de $\bar{z}_i^b$. Ingargiola et Korsh [71] ont présenté le premier algorithme de réduction pour KP, mais de nombreuses améliorations ont vu le jour par la suite [91, 108].

2.3.2 Notion de *core* au problème

Pour de nombreuses instances de test, les expérimentations numériques indiquent que seul un sous-ensemble des objets intervient dans la construction effective des solutions optimales. En effet, les objets pour lesquels l'efficacité est très grande auront de grandes chances de faire partie des solutions optimales, tandis que ceux ayant une efficacité au plus proche de zéro seront probablement dans aucune de ces

solutions. Ceci se remarque d'autant plus que le nombre d'objets est grand. Ainsi, pour ces instances, l'exploration n'examine essentiellement que les objets dont l'efficacité est moyenne.

Cette observation a mené Balas et Zemel [4] à définir le *core* d'une instance de la manière suivante :

Définition 2.4 (*core* mono-objectif)**.** *Soit x^* une solution optimale d'une instance où les objets sont ordonnés par efficacité décroissante. Le* core *est donné par les objets dans l'intervalle $C = \{a, \ldots, b\}$, avec*

$$a = \min\{i : x_i^* = 0\}$$
$$b = \max\{i : x_i^* = 1\}$$

La taille du *core* est une petite fraction de n pour de nombreuses instances. Par conséquent, si les valeurs de a et b sont connues *a priori*, il est possible de ne résoudre que le problème réduit aux objets de C, avec une capacité de $\omega - \sum_{i=1}^{a-1} w_i$ et de compléter ensuite la solution de manière à ce que $x_i = 1, \forall i \in \{1, \ldots, a-1\}$ et $x_i = 0, \forall i \in \{b+1, \ldots, n\}$.

Cependant, le fait que le *core* soit défini à partir de la connaissance d'une solution optimale rend difficile son utilisation dans une méthode de résolution. Ainsi, plusieurs auteurs ont proposé diverses méthodes pour calculer une estimation du *core*, avant de pouvoir le résoudre avec un algorithme exact [4, 91, 107]. Plusieurs propositions ont été faites sur le choix de la taille du *core* à calculer. Golberg et Marchetti-Spaccamela [63] ont étudié théoriquement la taille minimale du *core*. Ils ont montré que sa taille augmentait logarithmiquement en fonction de la taille de l'instance.

2.4 Le problème de sac à dos multi-objectif

Le problème de sac à dos multi-objectif est une variante du problème de sac à dos dans laquelle plusieurs coûts sont associés à chaque objet. Cette variante est applicable à toutes celles présentées en 2.1.1. Néanmoins, nous ne nous intéresserons dans la suite qu'au problème de sac à dos multi-objectif unidimensionnel en variables binaires, noté $01MOKP$, dont la formulation est rappelée ci-dessous :

$$\left.\begin{array}{lll} \max & z_j(x) = \sum_{i=1}^{n} c_i^j x_i & j \in \{1, \ldots, p\} \\ s.c. & \sum_{i=1}^{n} w_i x_i \leq \omega & \omega, w_i \in \mathbb{N}^* \\ & x_i \in \{0,1\} & \forall i \in \{1, \ldots, n\} \end{array}\right\} 01MOKP$$

2.4.1 Méthodes de résolution exacte

Au regard de la littérature, il existe actuellement trois grandes techniques pour résoudre de manière exacte des problèmes multi-objectifs de sac à dos : les procédures en deux phases, la programmation dynamique et des algorithmes de chemins.

Les travaux portant sur ces deux dernières procédures, bien que proches dans l'esprit, bénéficient de publications distinctes. Classiquement, les travaux sur les algorithmes de programmation dynamique se concentrent sur le filtrage des états et sur la réduction de leur nombre, tandis que ceux sur les procédures de plus longs chemins portent sur la structure du graphe et sur l'algorithme appliqué.

2.4.1.1 Algorithme en deux phases, cas bi-objectif (Visée *et al.*, 1998)

Visée *et al.* [136] ont proposé un algorithme en deux phases (voir la section 1.3.1.4) dans lequel l'exploration est faite en utilisant une procédure de séparation et évaluation dérivée de Martello et Toth [92]. Le résultat de l'algorithme est un ensemble complet minimal. L'évaluation est effectuée sur diverses instances aux coefficients non corrélés générés aléatoirement.

La première phase s'attache au calcul d'un ensemble X_{SE} en suivant le principe de la dichotomie de Aneja et Nair [3]. À l'issue de cette phase, l'ensemble $Y_{SN} = \{y^1, \ldots, y^q\}$ est obtenu. Les autres solutions efficaces ont leur image dans l'espace des objectifs hors de $Y_{SN} - \mathbb{R}^p_{\leqq}$.

Dans le cas bi-objectif, tel qu'étudié par les auteurs, l'espace restant à explorer est composé d'un ensemble de triangles construits de la façon suivante : deux points y^r et y^s adjacents dans Y_{SN}, avec $y^r_1 < y^s_1$, définissent un triangle $\Delta(y^r, y^s)$ dont l'hypoténuse est donnée par le segment les reliant et dont l'angle droit est situé en (y^r_1, y^s_2), soit au point d'intersection des cônes de dominance $y^s - \mathbb{R}^p_{\geq}$ et $y^r - \mathbb{R}^p_{\geq}$. Par extension de la définition 1.7 page 21, ce point est nommé *point nadir local*.

Dans la seconde phase, un problème mono-objectif P_λ est associé à chaque triangle, avec $\lambda = (y^r_2 - y^s_2, y^s_1 - y^r_1)$, ce qui correspond à une direction de recherche orthogonale à l'hypoténuse. Ces problèmes sont ensuite résolus à l'aide d'une PSE de Martello et Toth [92] pour le sac à dos mono-objectif, dont la partie évaluation est adaptée de manière à explorer chaque triangle. Dans le cadre bi-objectif, la zone pertinente à explorer dans un triangle est délimitée par les trois bornes décrites ci-après, illustrées par la figure 2.2.

Si y^r et y^s sont les deux points décrivant le triangle, alors $\underline{z}_1 = y^r_1$ est une bonne borne inférieure au regard du premier objectif pour les solutions non supportées dont les images sont incluses dans le triangle. De même pour $\underline{z}_2 = y^s_2$ sur le second objectif. Enfin, une borne inférieure au problème P_λ est donnée par $\underline{z}^\lambda = \lambda \cdot y^N$, où y^N est le point nadir local défini par $y^N = (y^r_1, y^s_2)$.

Cette dernière borne est mise à jour au fur et à mesure que de nouvelles solutions potentiellement efficaces sont trouvées dans le triangle (figure 2.3). Soit $\{y^1, \ldots, y^m\}$ les points réalisables potentiellement non dominés connus dans le triangle, tels que $y^i_1 < y^{i+1}_1$ pour tout $i \in \{1, \ldots, m-1\}$. En renommant $y^0 = y^r, y^{m+1} = y^s$, la valeur de cette borne inférieure est $\underline{z}^\lambda = \min_{i \in \{0,\ldots,m\}} \left(\lambda_1 y^i_1 + \lambda_2 y^{i+1}_2\right)$.

L'évaluation des nœuds pendant la PSE se fait en calculant séparément une borne supérieure sur chacun des objectifs z_1, z_2 et z^λ. Si une de ces bornes est inférieure ou égale à la borne inférieure correspondante, alors le nœud est fermé.

Les auteurs appliquent un algorithme de réduction à chaque problème résolu, dans chaque phase. Dans la première, l'algorithme de réduction est immédiat ; il s'agit de problèmes mono-objectifs. Si, dans cette phase, y^r et y^s sont deux points supportés utilisés pour orienter la maximisation à une étape de la dichotomie, alors la borne inférieure utilisée dans l'algorithme de réduction ainsi que lors de l'initialisation de la PSE est $\underline{z}^\lambda = \lambda \cdot y^r = \lambda \cdot y^s$. En effet, à cette étape de la résolution, le segment (y^r, y^s) est une facette de l'enveloppe convexe des images des solutions supportées connues. Or, l'algorithme cherche une solution supportée x dont l'image est « au delà » de ce segment, dans une direction qui lui est perpendiculaire (voir l'algorithme de Aneja et Nair [3] présenté dans la section 1.3.1.2 page 23). Par conséquent, la valeur de $\lambda \cdot y^r$ doit borner inférieurement la valeur de $\lambda \cdot z(x)$.

Dans la seconde phase, l'algorithme de réduction consiste à calculer l'union des variables fixées par l'application de la réduction de problème mono-objectif en considérant séparément z_1, z_2 et z^λ avec les bornes correspondantes. Pour tout $b \in \{0,1\}, j \in \{1,2\}, J^b(j)$ est l'ensemble J^b obtenu pour le problème mono-objectif P_j et la borne inférieure $\underline{z_j}$ avec l'algorithme décrit dans la section 2.3.1 page

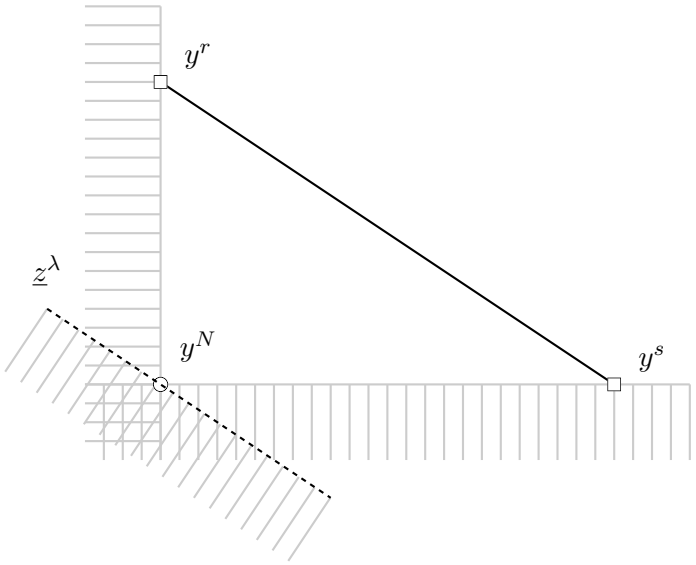

Figure 2.2 – Illustration des bornes lors de l'exploration d'un triangle dans la seconde phase. Les rayures représentent les régions ôtée de l'espace de recherche par ces bornes. Les carrés (□) représentent les performances des solutions décrivant le triangle. Le rond (○) illustre le point nadir local utilisé pour le calcul de la borne $\underline{z}^\lambda$, représentée ici par une ligne en pointillés.

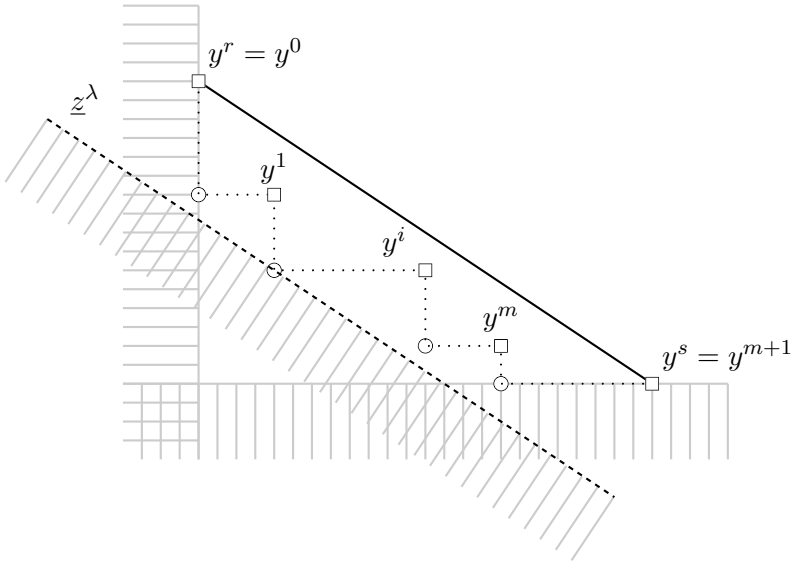

Figure 2.3 – Illustration de la mise à jour des bornes lors de l'exploration d'un triangle dans la seconde phase. Les rayures représentent les régions ôtée de l'espace de recherche par ces bornes. Les carrés (□) représentent les performances des solutions potentiellement efficaces trouvées jusqu'ici. Les ronds (○) illustrent l'ensemble des points nadirs locaux déduits de ces performances et sont utilisés pour mettre à jour la borne $\underline{z}^\lambda$, représentée ici par une ligne en pointillés.

42. Soit $J^b(\lambda), b \in \{0,1\}$ les ensembles obtenus de la même manière pour le problème P_λ et la borne $\underline{z}^\lambda$. Chacun de ces ensembles définit la valeur nécessaire de certaines variables pour obtenir des solutions dont l'image est au delà des bornes indiquées, c'est-à-dire des solutions dont l'image est dans le triangle. Les variables fixées à $b \in \{0,1\}$ avant l'exploration du triangle sont alors celles de $J^b = J^b(1) \cup J^b(2) \cup J^b(\lambda)$. L'exploration est alors réduite au seuls objets de l'ensemble $F = \{1,\dots,n\} \setminus (J^1 \cup J^2)$, avec une capacité de sac à dos égale à $\omega - \sum_{i \in J^1} w_i$.

2.4.1.2 Plus courts chemins multi-objectif (Captivo *et al.*, 2003)

L'analogie entre le problème de sac à dos et le problème de plus court chemin, telle que présentée dans la section 2.2.1.3 page 39, reste valide pour le cas multi-objectif. De plus, des algorithmes efficaces existent pour ce problème [67, 93]. Dans ce contexte d'algorithmes, Captivo *et al.* [22] ont proposé d'utiliser une procédure de calcul des plus courts chemins pour résoudre des problèmes bi-objectifs de sac à dos. Aucun pré-traitement n'est effectué pour réduire la taille du problème. Les auteurs ont évalué leur procédure sur de nombreuses instances générées aléatoirement selon différents degrés de corrélation.

Le graphe utilisé dans leur procédure est similaire à celui présenté précedemment pour le cas mono-objectif, à ceci près que les auteurs pondèrent les arcs avec l'opposé des profits associés aux variables ; de manière à pouvoir utiliser directement un algorithme de plus courts chemins.

L'algorithme implémenté est une version améliorée de l'algorithme 2.4. Il retourne néanmoins un ensemble complet minimum.

Procédure `plus_courts_chemins`($\downarrow G$)
Paramètre $\downarrow G$: le graphe construit depuis l'instance du problème.
Sortie Les performances des solutions efficaces.

- - $S_{j,d}$ est l'ensemble des performances obtenues au sommet $z_{j,d}$ (vide par défaut) - -
$S_{0,0} \leftarrow \{(0,0)\}$
pour j **de** 1 **à** n **faire**
  pour d **de** 0 **à** ω **faire**
    si $d \geq w_j$ **alors**
      $S_{j,d} \leftarrow S_{j-1,d} \cup \{y + c_j : y \in S_{j-1,d-w_j}\}$
      - - Suppression dans $S_{j,d}$ des points dominés par d'autres points de l'ensemble - -
      $S_{j,d} \leftarrow$ `supprimer_points_dominés`($S_{j,d}$)
    sinon
      $S_{j,d} \leftarrow S_{j-1,d}$
    fin si
  fin pour
fin pour
retourner supprimer_points_dominés($\bigcup_{d=0}^{\omega} S_{n,d}$)

ALG. 2.4 – Plus courts chemins pour $bi-01KP$

Il est à noter que cette méthode est valide quel que soit le nombre d'objectifs à optimiser. Cependant, les auteurs ont initialement implémenté l'algorithme uniquement pour le cas bi-objectif.

Figueira *et al.* [47] ont récemment présenté des résultats sur une approche similaire, basés sur des résultats de Klamroth et Wiecek [79] concernant la structure du graphe. Leurs travaux présentent plusieurs types de graphes construits à partir d'instances de divers problèmes de sac à dos. Figueira *et al.*

[47] ont utilisés ces structures pour le problème de sac à dos multi-objectif en variables entières, non borné, avant de leur appliquer une procédure de calcul des plus longs chemins multi-objectifs. Les auteurs résolvent des instances ayant jusqu'à sept objectifs pour vingt variables. Ils observent qu'un modèle de graphe nommé *Modèle IV* se démarque de par les très bonnes performances qu'il permet d'obtenir, apparemment du fait de sa structure similaire à une grille.

2.4.1.3 Programmation dynamique (Bazgan *et al.*, 2007)

Bazgan *et al.* [10] ont récemment présenté une procédure de résolution pour les instances du problème multi-objectif de sac à dos. Celui-ci est similaire à l'algorithme 2.4 et se démarque en particulier sur le filtrage des états. En effet, les auteurs utilisent plusieurs relations de dominance pour filtrer les points potentiellement non dominés durant la résolution.

La première de ces relations se base sur l'observation du fait que lorsque la capacité résiduelle $\omega - d$ associée à un état $S_{j,d}$ est supérieure ou égale à la somme des poids des objets restants, alors la seule façon d'obtenir une solution potentiellement efficace à partir de $S_{j,d}$ est d'y ajouter tous ces objets.

La deuxième relation est basée sur la dominance de Pareto (définition 1.4 page 15). Soit $s \in S_{j,d}$ et $s' \in S_{j,d'}$. Si $d \leq d'$, alors tous les objets pouvant être sélectionnés à partir de $S_{j,d'}$ peuvent aussi l'être à partir de $S_{j,d}$. Ainsi, si $s \geqq s'$, alors l'état s permettra d'obtenir des solutions au moins aussi bonnes qu'à partir de s', ce dernier peut donc être omis pour la suite.

Remarquons que l'utilisation de l'opérateur « $\geqq$ » dans cette dernière relation implique que le calcul de l'ensemble complet maximal n'est plus garanti.

La troisième et dernière relation est la plus coûteuse à évaluer. En effet, elle nécessite de calculer, pour chaque état auquel elle est appliquée, une borne supérieure sur chaque objectif ainsi qu'une solution réalisable. Soit $s \in S_{j,d}, s' \in S_{j,d'}$. Les auteurs calculent le point $\underline{z} = s + G^d(j)$ et le point $\bar{z} = s' + (U_2^{d'}(1,j), \ldots, U_2^{d'}(p,j))$, où $G^{\tilde{d}}(j)$ est un algorithme glouton multi-objectif inspiré de 2.2.2 page 40 et $U_2^{\tilde{d}}(i,j)$ consiste à appliquer la borne U_2 définie dans la section 2.2.1.1 page 36 uniquement sur l'objectif i. Dans les deux cas, la procédure appliquée est restreinte aux objets de l'ensemble $\{j, \ldots, n\}$ et à la capacité $\tilde{d}$.

Si $\underline{z} \geqq \bar{z}$, alors toutes les solutions obtenues à partir de s' sont dominées par $\underline{z}$, qui est une solution réalisable. Par conséquent, s' peut être supprimé.

Enfin, les auteurs ont poussé leur étude en direction de l'influence de l'ordre de sélection des variables et ont proposé plusieurs critères de tri. En effet, l'ordre décroissant d'efficacité utilisé pour le cas mono-objectif n'est plus applicable dans le cas multi-objectif du fait du caractère multi-dimensionnelle de celle-ci. Nous reviendrons sur le sujet de cet ordre dans le chapitre 6.

Les auteurs ont évalué leur procédure sur des instances bi- et tri-objectifs aux coefficients générés aléatoirement selon différents degrés de corélation. Leurs expérimentations numériques leur permettent de conclure que leur algorithme est le plus performant à ce jour.

2.4.2 Méthodes de résolution approchée

Le passage au cadre multi-objectif rend implicitement les problèmes plus ardus, en particulier du fait du nombre de solutions efficaces. Cela est aussi vrai pour le problème de sac à dos. Ainsi, alors que des algorithmes pour le cas mono-objectif peuvent résoudre des instances ayant plusieurs centaines de milliers de variables en un temps raisonnable [92], les résultats disponibles pour les algorithmes de la version multi-objectif se cantonnent à quelques centaines de variables. Il est alors naturel de s'orienter vers des méthodes approchées pour résoudre les plus grandes ou les plus difficiles des instances de

sac à dos multi-objectif. C'est dans cette optique que plusieurs méta-heuristiques ont été appliquées au problème. Ces méthodes sortent du cadre de nos travaux, nous nous limiterons donc à mentionner les principales contributions.

Ainsi, Gandibleux et Fréville ont proposé une recherche tabou [53] appliquée au problème bi-objectif. Czyzak et Jaskiewicz ont adapté une méthode de recuit simulé [26] et un algorithme génétique a été proposé sur le sujet par Gandibleux *et al.* [57]. Enfin, plusieurs auteurs ont proposé des méthodes hybrides, tels que Abdelaziz *et al.* [2] ou Barichard et Hao [6, 7], combinant algorithme génétique et recherche tabou.

Le principal inconvénient de ces procédures est la difficulté d'obtenir une bonne approximation de la frontière efficace, comme souligné dans la section 1.3.2 page 27. Pour répondre à cela, Bazgan *et al.* [9] ont récemment proposé un algorithme à garantie relative de performance de type *FPTAS* pour le problème de sac à dos multi-objectif. Ces travaux se basent fortement sur les résultats introduits dans [10], des même auteurs. L'idée est ici d'autoriser des états dominés dans une certaine marge d'erreur. Cette erreur est évaluée selon plusieurs fonctions introduites par les auteurs et se propage d'état en état jusqu'à obtenir une $(1+\epsilon)$-approximation telle que définie en 1.15 page 28.

2.4.3 Notions de *core* multi-objectif

Les algorithmes du cas mono-objectif tirant parti du *core* du problème sont parmi les plus performants [92]. Cependant, cette notion est basée sur l'ordre obtenu selon l'efficacité décroissante des objets ; un ordre qui ne se retrouve pas immédiatement dans le cas multi-objectif.

Gomes da Silva *et al.* ont néanmoins proposé une généralisation du core pour le problème de sac à dos bi-objectif dans [64].

Définition 2.5 (*Core* bi-objectif [64])**.** *Soit une famille de pondérations* Λ*. Le* core bi-objectif *d'une solution efficace* x *est le plus petit core obtenu en ordonnant les objets par ordre décroissant de leurs efficacités pondérées* $\frac{\lambda c_i}{w_i}$*, en considérant chaque pondération* $\lambda \in \Lambda$*.*

Remarquons que cette définition s'étend immédiatement au cas $p \geq 3$.

Les auteurs appuient leurs propos par de nombreuses expérimentations sur cinq familles d'instances. Ils observent que pour une taille d'instance donnée, la taille du *core* est relativement insensible à l'intérieur d'une même famille. Cependant, cette taille peut énormément varier d'une famille à une autre.

Enfin, les auteurs proposent deux schémas d'algorithmes de résolution utilisant le *core*, respectivement pour une résolution approchée et une résolution exacte.

En combinant ces résultats avec ceux de Puchinger *et al.* [114] sur le *core* pour le problème mono-objectif de sac à dos multi-dimensionnel, Mavrotas *et al.* [95] ont proposé une procédure de résolution pour le problème bi-objectif de sac à dos multi-dimensionnel.

2.5 Conclusion

Bien que le problème de sac à dos soit étudié depuis des décennies et bien qu'il existe des méthodes performantes pour résoudre de manière exacte des instances de sa version mono-objectif, peu de ressources sont disponibles pour la version multi-objectif. Actuellement, seules trois algorithmes de ce type sont disponibles pour ce problème : la procédure en deux phases proposée par Visée *et al.* [136], l'approche du calcul des plus courts chemins par Captivo *et al.* [22] et la programmation dynamique de Bazgan *et al.* [10].

Jusqu'aux récents travaux de Bazgan *et al.* [10], la résolution exacte d'instances du problème multi-objectif de sac à dos est restée limitée au cas bi-objectif. Cependant, leur procédure, basée sur de la programmation dynamique, souffre des mêmes limitations que la procédure de calcul des plus courts chemins de Captivo *et al.* [22], à savoir une excessive consommation de mémoire. Pour cette raison, ces procédures ne résolvent que des instances de quelques centaines de variables. D'un autre côté, l'algorithme en deux phases proposé par Visée *et al.* [136] a donné de bons résultats sur la version bi-objectif. Cependant, si la consommation de mémoire n'est pas un facteur limitant pour celui-ci, les temps de résolution sont au delà des procédures de Bazgan *et al.* [10] ou de Captivo *et al.* [22]. De plus, cet algorithme en deux phases est limité au cas $p > 2$.

À notre connaissance, seul Przybylski [111] a appliqué avec succès une méthode en deux phases sur un problème ayant plus de deux objectifs. De plus, Przybylski [111] a observé que l'utilisation d'une procédure de *ranking* dans la seconde phase entraînait une forte amélioration des performances de l'algorithme. Dans la mesure où une telle procédure existe pour le problème des plus courts chemins, pouvant de fait être appliquée à $01KP$, le schéma de la deux phases associé à un *ranking* semble être un bon candidat pour une résolution efficace des instances de $01MOKP$.

Enfin, il est étonnant de constater qu'aucune procédure de fixation de variables *a priori* n'ait été proposée pour le problème multi-objectif de sac à dos, ni aucune procédure de séparation et évaluation. En effet, ces algorithmes donnant de très bons résultats dans le cas mono-objectif, il nous semble intéressant de se pencher sur leur pendant multi-objectif.

Dans le cadre d'un système d'aide à la décision, la question de l'ensemble des solutions à calculer se pose. Les procédures présentées dans ce chapitre calculent toutes un ensemble complet minimal X_{E_m}. Or, dans le contexte de résolution *a posteriori* dans lequel nous nous plaçons, il est préférable de calculer l'ensemble complet maximal X_{E_M}, de manière à ne pas priver le décideur de solutions alternatives aux performances équivalentes.

Les chapitres suivants rapportent nos contributions sur les pistes ouvertes énoncées ci-dessus. En particulier, le chapitre 3 présente une technique de fixation de variables, tandis que les chapitres 4 et 5 portent respectivement sur une amélioration de la méthode en deux phases existante et sur sa généralisation à la situation où $p > 2$. Enfin, le chapitre 6 décrit deux procédures de séparation et évaluation appliquées au problème de sac à dos multi-objectif.

CHAPITRE 3

Pré-traitement et règles de réduction pour le sac à dos multi-objectif

Les procédures de réduction appliquées en pré-traitement permettent de réduire *a priori* la taille d'une instance du problème, en déterminant la valeur qu'auront certaines variables dans les solutions efficaces. Leur utilisation convient à la fois aux algorithmes de résolution exacte et approchée et peut rapidement améliorer les temps de résolution si le coût de son application est raisonnable par rapport à celui de la résolution.

Bien que de telles procédures soient disponibles pour le problème de sac à dos mono-objectif, (voir la section 2.3.1 page 42), ainsi que pour d'autres variantes, comme sa version multi-dimensionnelle [5, 52], très peu de contributions ont été faites pour $01MOKP$. À notre connaissance, seuls Gomes da Silva *et al.* [64] ont présenté des travaux visant à déterminer, lors d'un pré-traitement pour le sac à dos bi-objectif, la valeur qu'auront certaines variables dans les solutions efficaces. Les auteurs proposent dans cet article une extension de la notion de *core* du cas mono-objectif au cas bi-objectif (voir la section 2.4.3 page 48).

Ce chapitre se place dans cet axe de recherche en proposant de nouvelles propriétés visant à réduire *a priori* la taille des instances du problème de sac à dos multi-objectif. Ces propriétés résultent d'une relation de dominance sur les données du problème (les profits et le poids des objets) et de bornes connues sur la cardinalité des solutions efficaces. Elles permettent de fixer la valeur de certaines variables avant l'application d'un algorithme de résolution.

Dans la première section, des expérimentations numériques mettent en évidence le caractère régulier de différentes variables. Ces variables, dites régulières, n'acceptent qu'une unique valeur dans toutes les solutions efficaces. Dans la section suivante, une définition de ces variables est proposée et l'espace des données est introduit ; puis des résultats concernant la cardinalité des solutions sont rappelés. La troisième section présente plusieurs propriétés pour déterminer, pour une instance donnée, la valeur de plusieurs variables régulières. L'efficacité de ces propriétés est discutée au regard d'un ensemble d'instances du problème de sac à dos bi- et tri-objectif dans la quatrième section. Ces expérimentations numériques soulignent en particulier l'intérêt de telles propriétés pour les instances où les coefficients sont indépendants.

3.1 Observation du caractère régulier des variables

Afin d'illustrer les qualités des variables régulières, une attention toute particulière est donnée à une instance de 50 variables, aux coefficients générés aléatoirement et sans corrélation. Cette instance accepte 34 solutions efficaces, où 9 variables sont toujours égales à zéro, tandis que 18 autres sont toujours égales

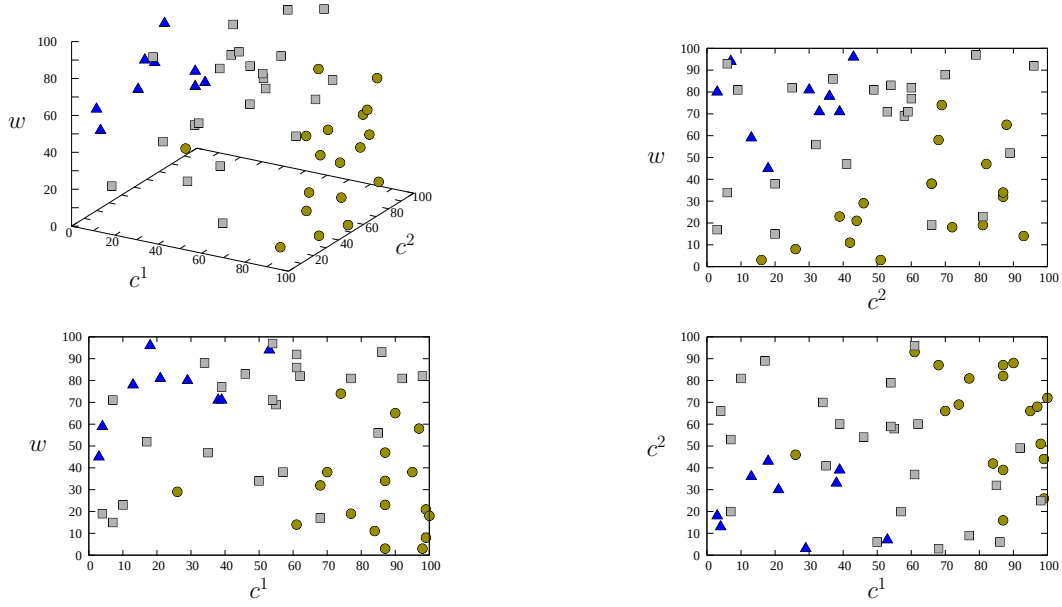

Figure 3.1 – L'image en haut à gauche illustre la régularité des valeurs des variables dans les solutions efficaces, pour une instance de sac à dos bi-objectif de taille 50. Les données des variables toujours à zéro sont indiquées par des triangles, celles des variables toujours à un sont illustrées par des cercles. Les données représentées par des carrés correspondent à des variables intervenant avec les deux valeurs dans X_{E_M}. Les autres images sont des projections de la première sur chaque paire d'axes.

à un. Les 23 variables restantes interviennent donc avec les deux valeurs. La figure 3.1 représente les profits et poids des objets, associés au caractère régulier des variables correspondantes.

Nous pouvons observer que certaines données semblent groupées selon le type des variables auxquelles elles sont attachées. En particulier, les variables toujours à zéro sont dans une partie de l'espace où les profits sont faibles et aux poids importants, alors que celles toujours égales à un sont associées aux grands profits et aux poids faibles. Il n'y a cependant aucune séparation nette de l'espace entre chaque groupe de variables. En effet, nous pouvons constater que des variables intervenant avec les deux valeurs sont présentes dans tout l'espace.

3.2 Variables régulières et données du problème

Cette section pose les définitions des éléments nécessaires pour les propriétés présentées dans la section suivante. Des bornes sur la cardinalité des solutions efficaces seront aussi rappelées.

La définition ci-dessous formalise le concept de variables régulières.

Définition 3.1 (Variables régulières)**.** *Les ensembles*

$$C_0 = \{i \in \{1, \dots, n\} : x_i = 0, \forall x \in X_{E_M}\} \tag{3.1}$$
$$C_1 = \{i \in \{1, \dots, n\} : x_i = 1, \forall x \in X_{E_M}\} \tag{3.2}$$

dénotent les indices des variables régulières*. C'est à dire les variables qui n'acceptent qu'une unique valeur dans toutes les solutions efficaces. Les autres variables sont dites* irrégulières*.*

L'espace déterminé par les données du problème de sac à dos est l'ensemble des vecteurs

$$V = \{v^i \in \mathbb{N}^{p+1} : v^i = (c_i^1, \ldots, c_i^p, -w_i), i \in \{1, \ldots, n\}\}$$

Nous utiliserons la relation de dominance ci-dessous sur les vecteurs de cet espace :

Définition 3.2 (Dominance des données)**.** *Soit $v^i, v^j \in V$. La donnée v^i domine la donnée v^j ($v^i \geq_V v^j$) si $c^i \geq c^j$ et $w_i \leq w_j$.*

Cette définition exprime le fait que l'objet j, s'il est choisi dans une solution où l'objet i est absent, peut clairement être substitué par ce dernier pour obtenir une meilleure performance.

Dans le cadre de la recherche d'un ensemble maximal de solutions, la relation de dominance de la définition 1.4 page 15 ne peut s'appliquer directement aux éléments de V. En effet, considérons par exemple les vecteurs $v^1 = (2, 2, -4)$ et $v^2 = (2, 2, -6)$ d'une instance quelconque de sac à dos bi-objectif. Selon la définition 1.4, nous avons $v^1 \geq v^2$. Cependant, la substitution de l'objet 2 par l'objet 1, si la capacité le permet, ne modifie en rien la performance d'une solution. Par conséquent, la seconde donnée n'est pas considérée comme dominée par la première.

Néanmoins, nous pouvons remarquer que bien que la séparation de la comparaison du profit et du poids soit impérative ici, elle ne l'est pas dans le cadre d'un ensemble complet non maximal. Aussi, les propriétés présentées dans la suite de ce chapitre seront applicables avec la relation de dominance classique dans ce cas.

Une définition des sous-ensembles de $\{1, \ldots, n\}$ correspondant à ceux de V dans lesquels la relation de dominance $\geq_V$ intervient est donnée ci-dessous.

Définition 3.3. *Soit $v^i \in V$.*

1. *L'ensemble des indices j des données v^j dominant la donnée v^i est nommé* ensemble préféré *et est noté $Pref(v^i) = \{j \in \{1, \ldots, n\} : v^j \geq_V v^i\}$.*
2. *L'ensemble des indices j des vecteurs v^j dominés par la donnée v^i est nommé* ensemble dominé *et est noté $Dom(v^i) = \{j \in \{1, \ldots, n\} : v^i \geq_V v^j\}$.*

L'équivalence suivante résulte directement de ces définitions

$$i \in Pref(v^j) \Leftrightarrow j \in Dom(v^i)$$

3.2.1 Bornes sur la cardinalité des solutions efficaces

Une borne inférieure et une borne supérieure de la cardinalité d'une solution optimale x pour le problème de sac à dos unidimensionnel mono-objectif ont été introduites par Glover [62]. Ces bornes sont définies comme suit

$$LB(\omega) = \max \left\{ s : \sum_{i=1}^{s} w_i \leq \omega \right\}, \text{ où } w_i \geq w_{i+1}, \forall i \in \{1, \ldots, n-1\} \tag{3.3}$$

$$UB(\omega) = \max \left\{ s : \sum_{i=1}^{s} w_i \leq \omega \right\}, \text{ où } w_i \leq w_{i+1}, \forall i \in \{1, \ldots, n-1\} \tag{3.4}$$

Pour calculer ces bornes, Fréville et Plateau [52] ont proposé une version modifiée de l'algorithme de tri rapide en $\Theta(n)$.

$LB(\omega)$ et $UB(\omega)$ sont clairement indépendantes des fonctions objectifs. Ainsi, Gandibleux et Fréville ont généralisé leur utilisation au cas multi-objectif à travers la proposition suivante [53] :

Proposition 3.1. *Si* $x \in X_{E_M}$ *alors* $LB(\omega) \leq \sum_{i=1}^{n} x_i \leq UB(\omega)$.

Dans la section suivante, ces bornes sont utilisées conjointement aux ensembles $Pref(.)$ et $Dom(.)$ pour dériver des propriétés des variables régulières depuis l'ensemble des solutions efficaces.

3.3 Propriétés des variables régulières

Dans cette section sont présentées des conditions sous lesquelles une variable est régulière dans l'ensemble des solutions efficaces.

Proposition 3.2. *Soit* $x \in X_{E_M}$. $\forall i \in \{1, \ldots, n\}$, *si* $x_i = 0$ *alors* $x_j = 0, \forall j \in Dom(v^i)$.

Démonstration. Immédiat à partir de la définition de dominance 3.2 sur V. □

Proposition 3.3. *Soit* $x \in X_{E_M}$. $\forall i \in \{1, \ldots, n\}$, *si* $x_i = 1$ *alors* $x_j = 1, \forall j \in Pref(v^i)$.

Démonstration. Immédiat à partir de la définition de dominance 3.2 sur V. □

En appliquant l'équivalence de la définition 3.2, ces propriétés peuvent être reformulées comme ci-dessous, respectivement :

- Soit $x \in X_{E_M}$. $\forall i, j \in \{1, \ldots, n\}$, si $x_i = 0$ et $i \in Pref(v^j)$ alors $x_j = 0$;
- Soit $x \in X_{E_M}$. $\forall i, j \in \{1, \ldots, n\}$, si $x_i = 1$ et $i \in Dom(v^j)$ alors $x_j = 1$.

Proposition 3.4. *Soit* $i \in \{1, \ldots, n\}$. *Si* $|Pref(v^i)| \geq UB(\omega)$ *alors* $x_i = 0, \forall x \in X_{E_M}$.

Démonstration. Par contradiction, supposons qu'il existe $x \in X_{E_M}$ telle que $x_i \neq 0$, c'est à dire telle que $x = (x_1, \ldots, x_i = 1, \ldots, x_n)$. Puisque $|Pref(v^i)| \geq UB(\omega)$ alors il existe un $j \in Pref(v^i)$ tel que $x_j = 0$. De plus, nous savons que $v^j \geq_V v^i$, ou de manière équivalente, $w_j \leq w_i$ et $c_j \geq c_i$. Par conséquent nous pouvons construire une solution réalisable $x' = (x_1, \ldots, x_i = 0, \ldots, x_j = 1, \ldots, x_n)$, avec $z(x') \geq z(x)$. Cela implique que $x \notin X_{E_M}$, ce qui est une contradiction. □

Proposition 3.5. *Soit* $i \in \{1, \ldots, n\}$. *Si* $\sum_{j \in Pref(v^i)} w_j + w_i > \omega$ *alors* $x_i = 0, \forall x \in X_{E_M}$.

Démonstration. Par contradiction, supposons qu'il existe $x \in X_{E_M}$ telle que $x_i \neq 0$, c'est à dire telle que $x = (x_1, \ldots, x_i = 1, \ldots, x_n)$. Puisque $\sum_{j \in Pref(v^i)} w_j + w_i > \omega$ alors il existe un $j \in Pref(v^i)$ tel que $x_j = 0$. De plus, nous avons $v^j \geq_V v^i$, ou de manière équivalente, $w_j \leq w_i$ et $c_j \geq c_i$. Par conséquent, nous pouvons construire une solution réalisable $x' = (x_1, \ldots, x_i = 0, \ldots, x_j = 1, \ldots, x_n)$, avec $z(x') \geq z(x)$. Cela implique que $x \notin X_{E_M}$, ce qui est une contradiction. □

Proposition 3.6. *Soit* $i \in \{1, \ldots, n\}$. *Si* $n - |Dom(v^i)| \leq LB(\omega)$ *alors* $x_i = 1, \forall x \in X_{E_M}$.

Démonstration. Par contradiction, supposons qu'il existe $x = (x_1, \ldots, x_i = 0, \ldots, x_n) \in X_{E_M}$ et que $n - |Dom(v^i)| \leq LB(\omega)$. Puisque $x_i = 0$, d'après la proposition 3.2, $x_j = 0, \forall j \in Dom(v^i)$. Par conséquent, il reste $n - |Dom(v^i)| - 1$ variables pour lesquelles l'affectation de la valeur 0 ou 1 reste à déterminer. Cependant, $n - |Dom(v^i)| - 1 < LB(\omega)$; par conséquent, selon la proposition 3.1, $x \notin X_{E_M}$, ce qui est une contradiction. □

Proposition 3.7. *Soit* $i \in \{1, \ldots, n\}$. *Si* $\sum_{j \notin Dom(v^i)} w_j \leq \omega$ *alors* $x_i = 1, \forall x \in X_{E_M}$.

i	1	2	3	4	5	6
x^1	1	0	1	0	0	1
x^2	1	0	0	0	1	1
x^3	0	0	1	0	1	1
x^4	0	0	1	1	0	1
x^5	0	0	0	1	1	1

Table 3.1 – Les solutions efficaces de l'exemple 3.13.

Démonstration. Par contradiction, supposons qu'il existe $x = (x_1, \ldots, x_i = 0, \ldots, x_n) \in X_{E_M}$ et que $\sum_{j \notin Dom(v^i)} w_j \leq \omega$. Puisque $x_i = 0$, d'après la proposition 3.2, $x_j = 0, \forall j \in Dom(v^i)$. Soit x' une solution avec $x'_j = 0, \forall j \in Dom(v^i)$, et $x'_j = 1, \forall j \notin Dom(v^i)$ (en particulier avec $x'_i = 1$). Puisque $\sum_{j \notin Dom(v^i)} w_j \leq \omega$, alors x' est réalisable. De plus, $z(x') \geq z(x)$, impliquant que $x \notin X_{E_M}$, ce qui induit une contradiction. □

Exemple 3.13 (Affectation de valeurs par dominance). *Soit l'instance de sac à dos bi-objectif ci-dessous.*

$$\begin{array}{ll} max & \left\{ \begin{array}{l} 2x_1 + 2x_2 + 5x_3 + 9x_4 + 8x_5 + 6x_6 \\ 8x_1 + 2x_2 + 6x_3 + 2x_4 + 5x_5 + 8x_6 \end{array} \right. \\ s.c. & 8x_1 + 8x_2 + 7x_3 + 5x_4 + 4x_5 + 2x_6 \quad \leq 17 \end{array} \tag{3.5}$$

Cette instance accepte 34 solutions réalisables, dont 5 solutions efficaces, listées dans la table 3.1. Le calcul des bornes indique que $LB(\omega) = 2$ et $UB(\omega) = 3$. De plus, chacune de ces 5 solutions vérifie $x_2 = 0$ et $x_6 = 1$.
Nous allons déduire les valeurs de ces variables régulières par application des propriétés présentées ci-dessus.
Dans cet exemple, $Pref(v^2) = \{1, 3, 4, 5, 6\}$ et $Dom(v^6) = \{1, 2, 3\}$. Puisque $|Pref(v^2)| > UB(\omega)$, la propriété 3.4 détermine la valeur $x_2 = 0$. Évidemment, la propriété 3.5 confirme cette valeur. De plus, $|Dom(v^6)| \not\leq LB(\omega)$. La propriété 3.6 ne permet pas ici de fixer $x_6 = 1$. Cependant, puisque $\sum_{j \notin Dom(v^6)} w_j = 11 \leq \omega$, alors la propriété 3.7 impose $x_6 = 1$.

3.4 Expérimentations numériques

Nous avons appliqué les propriétés de la section précédente sur un ensemble d'instances issues de la littérature. Cette section présente ces instances et les résultats obtenus. Nous noterons $C'_0 \subseteq C_0$ (respectivement $C'_1 \subseteq C_1$) l'union des variables fixées par les propriétés 3.4 et 3.5 (respectivement 3.6 et 3.7).

D'autre part, les instances ont été résolues à l'aide d'un algorithme exact qui détermine l'ensemble complet maximal des solutions efficaces. À partir de ces solutions, les ensembles C_0 et C_1 sont calculés. Leurs tailles servent à la fois à borner les résultats pouvant être obtenus à l'aide d'un algorithme de réduction, mais aussi à évaluer la qualité des propriétés.

La première partie de cette section présente les instances utilisées. Puis, dans la seconde partie, les résultats obtenus sur toutes les instances sont présentés.

Groupe	Nombre	Taille
A-1	10	50 à 500
A-2	10	50 à 500
A-3	10	50 à 500
A-4	10	50 à 500
B-1	2×10	50
B-2	15	50 à 1000
B-3	15	50 à 1000
C-1	7×10	100 à 700
C-2	8×10	600 à 4000
C-3	5×10	100 à 500
C-4	4×10	100 à 250

Table 3.2 – Tailles et nombres des instances de sac à dos bi-objectif utilisées pour les expérimentations. Les instances du sous-groupe B-1 diffèrent par les coefficients utilisés pour générer les coûts (voir la table 3.3). Le groupe C contient 10 instances pour chaque taille de problème.

Groupe	c^1	c^2	w
A-1 à A-4	$[1;100]$	$[1;100]$	$[1;100]$
B-1	$10 \times [1;300]$ et $10 \times [1;1000]$	$10 \times [1;300]$ et $10 \times [1;1000]$	$10 \times [1;300]$ et $10 \times [1;1000]$
B-2	$c_i^1 \in [c_i^2 - 100; c_i^2 + 100]$	$[111;1000]$	$[1;1000]$
B-3	$c_i^1 = w_i + 100$	$[1;1000]$	$[1;1000]$
C-1	$[1;1000]$	$[1;1000]$	$[1;1000]$
C-2	$[111;1000]$	$c_i^1 \in [c_i^2 - 100; c_i^2 + 100]$	$[1;1000]$
C-3	$[1;1000]$	$c_i^2 \in [\max\{900 - c_i^1, 1\}, \min\{1100 - c_i^1, 1000\}]$	$[1;1000]$
C-4	$[1;1000]$	$c_i^2 \in [\max\{900 - c_i^1, 1\}, \min\{1100 - c_i^1, 1000\}]$	$w_i \in [-200;200] + c_i^1 + c_i^2$

Table 3.3 – Coefficients utilisés pour générer les instances.

3.4.1 Instances du problème de sac à dos

La procédure de réduction a été testée sur des instances provenant de plusieurs sources de la littérature du problème de sac à dos [10, 22, 28, 136]. Les détails de ces instances sont explicités ci-dessous.

3.4.1.1 Instances du problème de sac à dos bi-objectif

Les instances du problème de sac à dos bi-objectif sont séparées en trois groupes selon plusieurs caractéristiques précisées ci-dessous. La table 3.2 indique le nombre d'instances dans chaque groupe ainsi que leurs tailles ; les intervalles des coefficients sont détaillés dans la table 3.3. Dans tous les cas, la capacité du sac à dos est $\omega = \lfloor 0.5 \times \sum_{i=1}^{n} w_i \rfloor$.

Les instances du groupe **A** viennent de [28]. Elles sont séparées en quatre sous-groupes, les trois derniers étant une variation du premier. Pour une taille d'instance donnée, le vecteur de poids est identique dans chaque sous-groupe. Le premier, A-1, est constitué de dix instances aux coefficients non corrélés et générés aléatoirement, comme décrit dans [136]. Les instances du sous-groupe A-2 ont été crées à partir de A-1 en remplaçant c^2 par c^1, en ordre inversé (c'est-à-dire que $c_i^2 = c_{n-i+1}^1, \forall i$). Les vecteurs des coûts de celles de A-3 ont des valeurs générées par plateaux de longueurs tirées aléatoirement dans l'intervalle$[1; \lfloor \frac{n}{10} \rfloor]$. Enfin, les instances du groupe A-4 ont été crées à partir de A-3 en remplaçant c^2 par c^1, en ordre inversé (comme A-2 par rapport à A-1).

Groupe	Nombre	Taille
D-1	55×10 instances	5 à 700
D-2	23×10 instances	10 à 350

Table 3.4 – Instances de sac à dos tri-objectif utilisées pour les expérimentations. Pour chaque taille de problème, 10 instances ont été générées.

Groupe	c^1	c^2	c^3	w
D-1	$[1; 1000]$	$[1; 1000]$	$[1; 1000]$	$[1; 1000]$
D-2	$[1; 1000]$	$c_i^2 \in [1; 1001 - c_i^1]$	$c_i^3 \in [\max\{900 - c_i^1 - c_i^2, 1\}, \min\{1100 - c_i^1 - c_i^2, 1001 - c_k^1\}]$	$[1; 1000]$

Table 3.5 – Coefficients utilisés pour générer les instances.

Remarque. L'instance de 50 variables utilisée pour illustrer les variables régulières dans la section 3.1 est tirée du sous-groupe A-1.

Toutes les instances du groupe **B** proviennent de [22]. Celles du sous-groupe B-1 ont leurs coefficients non corrélés. Les instances du sous-groupe B-2 sont des instances non conflictuelles où c^1 est corrélé positivement avec c^2. Enfin, celles du sous-groupe B-3 sont non conflictuelles où c^1 est positivement corrélé à w.

Les instances du dernier groupe, **C**, ont été produites par [10]. Le sous-groupe C-1 contient des instances aux coefficients non corrélés. Celles de C-2 sont des instances non conflictuelles où c^1 est corrélé positivement avec c^2. C-3 est constitué d'instances conflictuelles où c^1 et c^2 sont corrélés négativement, de même que C-4 où, en plus, w est corrélé positivement avec c^1 et c^2. Enfin, dans ce groupe, pour chaque sous-groupe, dix instances ont été construites pour chaque taille.

3.4.1.2 Instances du problème de sac à dos tri-objectif

Les instances du problème de sac à dos tri-objectif proviennent toutes de [10]. Elle seront représentées par le groupe **D** par la suite, contenant deux sous-groupes, nommés D-1 et D-2. Le premier est constitué d'instances non corrélées tandis que celles du second sont conflictuelles, où c^1 et c^2 sont corrélés négativement, de même que c^3 avec c^1 et c^2. La table 3.4 indique le nombre d'instances dans chaque sous-groupe ainsi que leurs tailles. Pour chaque sous-groupe, dix instances ont été construites pour chaque taille. Les coefficients ont été générés aléatoirement dans des intervalles détaillés dans la table 3.5. Dans tous les cas, la capacité du sac à dos est $\omega = \lfloor 0.5 \times \sum_{i=1}^{n} w_i \rfloor$.

3.4.2 Résultats expérimentaux

3.4.2.1 Instances bi-objectif

Les premières expérimentations concernent les dix instances bi-objectif du groupe A-1. Le nombre de variables régulières pour chaque instance est reporté par la figure 3.2. Plus de la moitié des variables sont régulières ; il est alors évident qu'identifier ces variables *a priori* facilitera la résolution de l'instance.

Les tables 3.6 et 3.8 présentent le nombre de variables régulières trouvées pour chaque instance, pour chaque propriété, ainsi que le nombre $|C'_0 \cup C'_1|$ de variables régulières différentes. Ce nombre est à comparer avec la cardinalité $|C_0 \cup C_1|$ calculé après une résolution exacte des instances et reporté dans la dernière colonne.

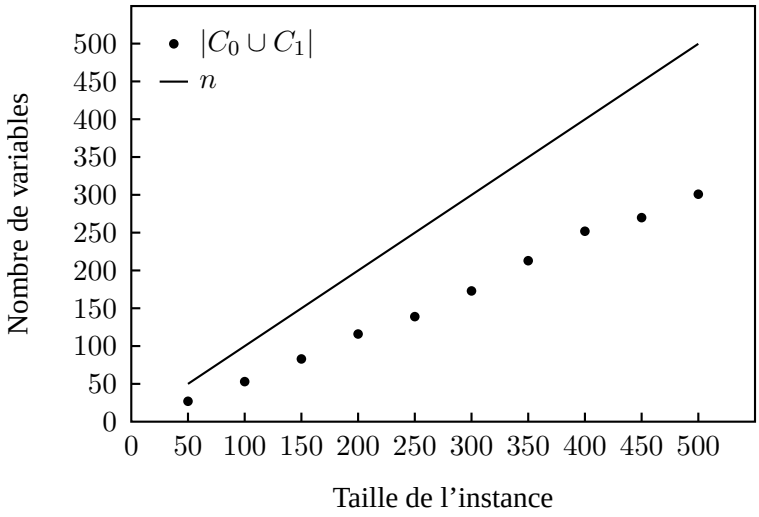

Figure 3.2 – Nombre de variables régulières pour chaque instance du groupe A-1.

Ces résultats montrent que $|C'_0 \cup C'_1|$ est loin de $|C_0 \cup C_1|$ mais reste néanmoins une quantité non négligeable. De plus, la propriété 3.5 est clairement plus performante que la 3.4 pour fixer des variables à 0. De même que la 3.7 est meilleure que 3.6 pour les fixer à 1.

Les propriétés n'apportent rien sur les instances des groupes B-3, C-3 et C-4. Ce résultat est attendu dans la mesure où, par construction, les données de ces instances ne présentent aucune dominance. Cependant, elles ne sont pas pour autant exemptes de variables régulières, comme le montrent les valeurs reportées dans la table 3.9.

3.4.2.2 Instances tri-objectif

La table 3.10 présente le nombre de variables régulières trouvées pour chaque instance, pour chaque propriété, ainsi que le nombre $|C'_0 \cup C'_1|$ de variables régulières différentes, à comparer avec la cardinalité de $|C_0 \cup C_1|$ indiquée dans la dernière colonne.

D'une manière encore plus marquante que pour le cas bi-objectif, $|C'_0 \cup C'_1|$ est loin de $|C_0 \cup C_1|$. Les propriétés 3.4 à 3.6 sont ici quasiment sans apport, seule la propriété 3.7 permet de déterminer dans tous les cas la valeur de quelques variables.

À nouveau, du fait de leur construction, les instances de D-2 ne véhiculent aucune dominance et, par conséquent, aucune des propriétés énoncées précédemment ne fixe de variables. Le nombre de variables régulières pour ces instances est reporté dans la table 3.11 et nous pouvons constater, à nouveau, que leur nombre est important.

3.5 Conclusion

Ce chapitre présente une approche originale basée sur une analyse des données pour fixer des variables du problème $01MOKP$ avant de calculer l'ensemble des solutions efficaces. Les concepts de variables régulières, d'espace des données et de dominance entre les éléments de cet espace sont introduits. En se basant sur une étude des relations parmi les données des instances du problèmes, des propriétés sont établies pour déduire *a priori* l'unique valeur de variables régulières dans X_{E_M}. La conséquence est

Sous-groupe	Taille	prop. 3.4	prop. 3.5	prop. 3.6	prop. 3.7	$\lvert C'_0 \cup C'_1 \rvert$	$\lvert C_0 \cup C_1 \rvert$
A-1	50	0	1	0	10	11	27
	100	0	0	1	5	5	53
	150	0	7	1	11	18	83
	200	2	3	0	14	17	116
	250	1	6	0	12	18	139
	300	0	3	2	23	26	173
	350	2	5	1	26	31	213
	400	1	7	5	26	33	252
	450	0	5	3	22	27	270
	500	0	13	3	34	47	301
A-2	50	0	2	1	7	9	24
	100	0	0	0	4	4	51
	150	0	1	1	8	9	78
	200	0	5	2	10	15	108
	250	1	5	0	13	18	145
	300	1	6	2	26	32	180
	350	4	8	3	23	31	203
	400	3	5	3	34	39	249
	450	2	6	3	25	31	249
	500	3	11	1	38	49	305
A-3	50	1	2	0	1	3	23
	100	3	7	0	4	11	61
	150	0	4	0	6	10	81
	200	0	3	1	16	19	105
	250	1	3	2	12	15	136
	300	5	10	0	27	37	186
	350	0	2	0	27	29	193
	400	4	15	4	32	47	239
	450	1	2	3	25	27	291
	500	14	18	9	66	84	364
A-4	50	0	1	0	1	2	19
	100	0	3	0	7	10	59
	150	2	4	0	6	10	75
	200	0	0	2	18	18	102
	250	1	4	3	17	21	146
	300	1	5	1	37	39	207
	350	0	14	1	31	45	239
	400	2	3	0	13	16	185
	450	6	11	7	27	38	328
	500	3	11	0	63	74	346

Table 3.6 – Nombre de variables fixées par les propriétés pour les instances du groupe A.

Sous-groupe	Taille	prop. 3.4	prop. 3.5	prop. 3.6	prop. 3.7	$\lvert C'_0 \cup C'_1 \rvert$	$\lvert C_0 \cup C_1 \rvert$
B-1	50	0	0	0	1	1	19
		0	3	0	2	5	25
		0	0	2	6	6	29
		0	3	0	4	7	28
		1	1	0	1	2	23
		1	1	0	3	4	26
		0	0	0	3	3	25
		1	2	1	3	5	27
		0	0	0	8	8	31
		1	2	0	4	6	27
		0	2	0	2	4	24
		1	2	0	7	9	30
		0	1	0	8	9	30
		1	3	0	2	5	25
		0	1	2	6	7	22
		0	0	0	3	3	25
		1	1	1	4	5	21
		1	4	1	8	12	33
		0	0	0	2	2	28
		1	1	1	3	4	22
B-2	50	3	9	2	17	26	46
	100	5	10	3	30	40	100
	150	7	14	3	37	51	139
	200	13	16	7	48	64	197
	250	15	19	8	58	77	237
	300	16	26	10	68	94	281
	350	20	28	13	88	116	328
	400	19	32	16	101	133	373
	450	20	35	18	111	146	420
	500	22	38	20	131	169	468
	600	25	51	24	148	199	560
	700	28	55	28	169	224	658
	800	32	63	30	182	245	751
	900	35	69	32	210	279	841
	1000	41	76	34	234	310	934

Table 3.7 – Nombre de variables fixées par les propriétés pour les instances du groupe B.

Sous-groupe	Taille	prop. 3.4	prop. 3.5	prop. 3.6	prop. 3.7	$\lvert C'_0 \cup C'_1 \rvert$	$\lvert C_0 \cup C_1 \rvert$
C-1	100	0,8	0,5	0,8	7,1	7,6	56,9
	200	1,0	0,4	1,2	12,6	13,0	116,4
	300	1,6	0,7	1,7	20,8	21,5	176,4
	400	2,7	1,5	2,5	26,2	27,7	238,3
	500	2,1	0,9	3,2	34,7	35,6	300,4
	600	2,5	0,9	3,2	41,5	42,4	366,4
	700	2,8	1,5	4,1	46,9	46,9	420,2
C-2	600	21,0	4,4	23,0	148,1	152,5	558,4
	700	24,9	4,9	27,3	161,7	166,6	656,7
	800	28,6	4,9	30,9	191,3	191,3	747,5
	900	33,9	6,4	34,0	216,9	223,3	848,4
	1000	37,2	7,6	38,5	241,1	248,7	941,2
	2000	75,1	15,8	75,0	477,7	493,5	1888,6
	3000	116,1	21,2	113,5	722,2	743,4	2838,8
	4000	153,6	29,0	150,4	963,0	992,0	3784,4

Table 3.8 – Nombre moyen de variables fixées par les propriétés pour les instances du groupe C.

Sous-groupe	Taille	$\lvert C_0 \cup C_1 \rvert$	Sous-groupe	Taille	$\lvert C_0 \cup C_1 \rvert$
B-3	50	29	C-3	100	28
	100	69		200	69,1
	150	100		300	111,4
	200	140		400	156,3
	250	180		500	192,7
	300	212	C-4	100	4,1
	350	—		150	7,9
	400	—		200	12,7
	450	—		250	15,9

Table 3.9 – Nombre de variables régulières par les instances bi-objectif où au moins un vecteur de coût est corrélé au vecteur des poids. Les solutions efficaces des instances du sous-groupe B-3 sont inconnues à partir de 350 variables.

Sous-groupe	Taille	prop. 3.4	prop. 3.5	prop. 3.6	prop. 3.7	$\lvert C'_0 \cup C'_1 \rvert$	$\lvert C_0 \cup C_1 \rvert$
	5	0,0	0,1	0,0	0,4	0,5	2,9
	10	0,1	0,0	0,0	0,6	0,6	3,0
	20	0,0	0,1	0,0	0,5	0,6	5,9
	30	0,1	0,0	0,1	0,6	0,6	9,7
	40	0,2	0,0	0,0	0,4	0,4	11,2
	50	0,1	0,0	0,0	0,6	0,6	25,4
	60	0,2	0,0	0,0	0,7	0,7	33,9
	70	0,0	0,0	0,0	1,2	2,2	37,2
	80	0,1	0,0	0,0	0,7	0,7	41,7
	90	0,1	0,0	0,0	1,0	1,0	48,2
	100	0,2	0,0	0,3	1,2	1,2	53,3
	110	0,0	0,1	0,1	1,8	1,9	62,2
	120	0,2	0,0	0,2	1,0	1,0	66,4
	130	0,2	0,0	0,0	1,8	1,8	70,1
	140	0,0	0,0	0,1	1,7	1,7	73,6
	150	0,2	0,0	0,0	1,5	1,5	81,0
	160	0,2	0,1	0,2	1,9	2,0	88,6
	170	0,0	0,2	0,0	2,0	2,2	94,4
	180	0,0	0,0	0,1	2,3	2,3	102,5
	190	0,0	0,0	0,1	2,8	2,8	111,5
	200	0,3	0,1	0,3	2,3	2,4	112,5
	210	0,0	0,1	0,1	2,9	3,0	116,5
	220	0,0	0,0	0,1	2,3	2,3	125,9
	230	0,0	0,0	0,0	2,0	2,0	128,7
	240	0,2	0,0	0,2	3,6	3,6	138,1
	250	0,1	0,2	0,2	3,8	4,0	143,2
	260	0,2	0,0	0,0	4,1	4,1	154,7
D-1	270	0,1	0,0	0,2	3,2	3,2	151,4
	280	0,0	0,0	0,2	4,2	4,2	163,8
	290	0,0	0,0	0,4	5,3	5,3	166,4
	300	0,1	0,0	0,3	3,7	3,7	172,7
	310	0,3	0,1	0,3	3,4	3,5	185,0
	320	0,1	0,0	0,2	4,4	4,4	187,3
	330	0,1	0,0	0,1	4,1	4,1	195,5
	340	0,2	0,2	0,2	4,8	5,0	194,4
	350	0,0	0,1	0,3	4,8	4,9	203,4
	360	0,2	0,3	0,3	5,2	5,5	208,5
	370	0,2	0,1	0,4	4,8	4,9	220,7
	380	0,2	0,1	0,2	4,2	4,3	225,1
	390	0,1	0,1	0,1	4,4	4,5	230,4
	400	0,1	0,0	0,6	4,7	4,7	236,5
	410	0,2	0,1	0,4	4,3	4,4	239,3
	420	0,0	0,2	0,3	6,7	6,9	250,2
	430	0,2	0,0	0,3	5,7	5,7	252,4
	440	0,1	0,1	0,3	5,5	5,6	262,1
	450	0,0	0,1	0,2	5,5	5,6	268,1
	460	0,4	0,1	0,3	5,2	5,3	269,8
	470	0,4	0,1	0,3	6,5	6,6	276,1
	480	0,1	0,1	0,5	8,2	8,3	285,1
	490	0,2	0,3	0,2	4,9	5,2	288,8
	500	0,2	0,0	0,5	8,0	8,0	299,2
	550	0,4	0,3	0,3	8,3	8,6	333,1
	600	0,3	0,2	0,9	7,9	8,1	363,4
	650	0,0	0,1	0,8	10,2	10,3	393,1
	700	0,1	0,1	0,6	10,6	10,7	423,2

Table 3.10 – Nombre moyen de variables fixées par les propriétés pour les instances du groupe D-1.

Taille	10	20	30	40	50	60	70	80	90	100	110	120
$\lvert C_0 \cup C_1 \rvert$	3,7	6,8	11,9	16,6	20,3	21,6	28,7	34,2	35,8	42,6	44,7	48,8
Taille	130	140	150	160	170	180	190	200	250	300	350	
$\lvert C_0 \cup C_1 \rvert$	55,9	63,8	65,6	68,6	73,5	79,5	81,1	89,9	116	138,2	163,3	

Table 3.11 – Nombre moyen de variables régulières pour les instances du groupe D-2.

que l'espace de recherche est réduit, ce qui laisse présager une diminution du temps de calcul nécessaire pour résoudre le problème. Les résultats théoriques sont évalués à la lumière de problèmes présentant deux objectifs ou plus, et appuyés par des expérimentations numériques.

Ces expérimentations montrent que la procédure de réduction est utile pour les instances où les coefficients sont indépendants, en particulier pour le cas bi-objectif. Cependant, de nombreuses variables n'obéissent pas aux propriétés dérivées, même si elles sont visiblement régulières après calcul de l'ensemble complet des solutions efficaces. Cette observation encourage la réalisation d'études plus approfondies sur des approches fixant ces variables *a priori*. De plus, la procédure de réduction n'est pas adaptée aux instances où les coefficients d'un objectif et de la contrainte sont corrélés. La question d'un pré-traitement pour ces instances reste ouverte et pertinente dans la mesure où celles-ci ont clairement des variables régulières.

CHAPITRE 4

Algorithme en deux phases pour le sac à dos bi-objectif

Ce chapitre présente deux approches pour le calcul des solutions X_{NE_M} dans une méthode en deux phases pour le problème de sac à dos bi-objectif, telle que décrite dans la section 2.4.1.1 page 44. La première utilise la procédure de séparation et évaluation (PSE) proposée par Visée *et al.* [136], pour laquelle nous renforçons le test évaluation des nœuds ; tandis que la seconde utilise une technique de *ranking*, inspirée de Przybylski *et al.* [113], et permettant d'obtenir les solutions de manière ordonnée selon leurs performances.

La première partie rappelle l'algorithme en deux phases, afin de positionner dans cette procédure les propositions décrites dans la suite. La seconde partie présente les nouveaux tests pour l'évaluation des nœuds, immédiatement suivis par des expérimentations numériques illustrant les gains apportés par ces modifications.

Les sections suivantes concernent l'application d'une procédure de *ranking* durant la seconde phase. Celle-ci utilise l'analogie entre le problème de sac à dos et celui des plus longs chemins, que nous détaillons en premier lieu. Puis la stratégie générale est décrite, suivie par les détails de son application à $2-01KP$.

Le chapitre se conclut par des expérimentations numériques. Une première analyse compare les méthodes en deux phases entre elles, puis d'une analyse du coût de la méthode de *ranking* en termes de temps processeur et de consommation de mémoire est effectuée. Enfin, les performances de cette dernière procédure sont comparées avec celles de la programmation dynamique de Bazgan *et al.* [10], celui-ci étant connu à ce jour comme le meilleur algorithme exact pour $01MOKP$. Le chapitre se termine sur une analyse des points forts et faibles de l'approche *ranking*, suivie par une discussion sur sa généralisation à plus de deux objectifs.

4.1 Description générale de l'algorithme en deux phases pour $2-01KP$

Nous rappelons dans cette section les éléments caractéristiques des procédures en deux phases, telles qu'elles ont été présentées dans la section 1.3.1.4 page 25 pour le cas général et dans la section 2.4.1.1 page 44 pour $2-01KP$.

Ces procédures séparent la recherche des solutions supportées (X_{SE}) des solutions non supportées (X_{NE}). Les premières sont en général obtenues avec une variante de la méthode d'Aneja et Nair [3]. En particulier, Visée *et al.* utilisent la PSE de Martello et Toth [92] pour résoudre les problèmes mono-

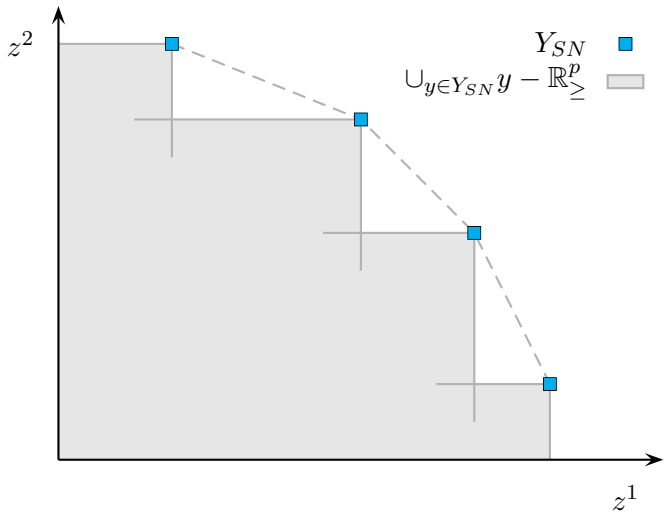

Figure 4.1 – L'espace de recherche pendant la seconde phase se décrit comme un ensemble de triangles. La ligne en pointillés représente l'enveloppe convexe de Y_N ainsi que les hypoténuses. Les angles droits des triangles se situent à l'intersection des cônes de dominance de points supportés adjacents.

objectifs construits durant cette phase. Remarquons cependant que n'importe quel algorithme exact peut être appliqué ici.

À l'initialisation de la seconde phase, les images $Y_{SN} = \{z(x) : x \in X_{SE}\}$ des solutions de X_{SE} obtenues à l'issue de la première phase sont utilisées pour réduire l'espace de recherche. Par définition, aucune solution de X_{NE} n'a son image dans $y - \mathbb{R}^p_{\geq}, \forall y \in Y_{SN}$. Nous pouvons remarquer sur la figure 4.1 que l'espace de recherche décrit ainsi se visualise comme un ensemble de triangles dans l'espace des objectifs. La seconde phase s'appuie sur cette représentation pour calculer les solutions restantes, en explorant chaque triangle individuellement. La suite de ce chapitre porte sur ces explorations.

Étant donnés $y^r, y^s \in Y_{SN}$ deux points supportés adjacents tels que $y^r_1 < y^s_1$, nous noterons $\Delta(y^r, y^s)$ le triangle décrit par ces points.

Classiquement, l'exploration de $\Delta(y^r, y^s)$ se fait en construisant une somme pondérée P_λ, avec $\lambda = (y^r_2 - y^s_2, y^s_1 - y^r_1)$. Ce problème est ensuite résolu avec un algorithme mono-objectif dans lequel les solutions sont évaluées et comparées selon leur caractère multi-objectif. Le calcul des bornes inférieures et supérieures est lui aussi adapté pour refléter le caractère multi-objectif des solutions.

4.2 Améliorations générales pour l'exploration des triangles

Cette section présente deux améliorations sur les bornes utilisées pendant l'exploration des triangles dans une méthode en deux phases, lors du calcul des solutions de X_{NE_M}. Ces bornes s'appliquent en particulier à la PSE utilisée par Visée *et al.* [136], présentée dans la section 2.4.1.1 page 44.

4.2.1 Une amélioration de la borne inférieure

La borne présentée ci-après a initialement été décrite par Przybylski [113] pour le problème d'affectation bi-objectif. Nous présentons ici son application à un problème de maximisation. Son utilisation est

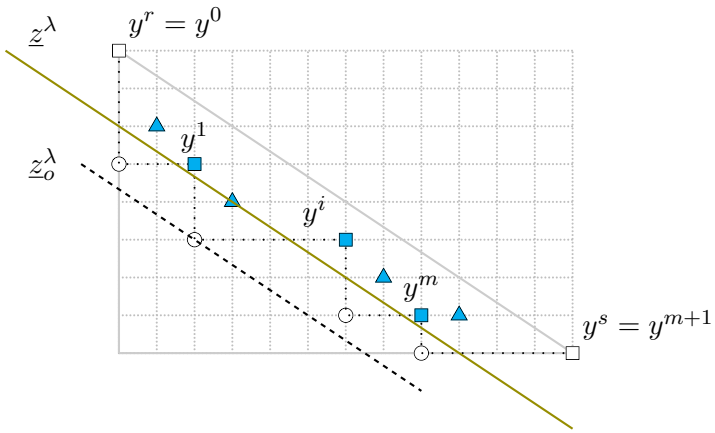

Figure 4.2 – La borne inférieure $\underline{z}_o^\lambda$, telle que définie dans [136] est représentée par la ligne en pointillés. Elle peut être améliorée en l'évaluant non plus sur les points nadirs locaux ($\circ$) mais sur les valeurs des solutions potentiellement efficaces restant à découvrir dans le triangle (points colorés).

basée sur l'hypothèse que, dans le problème considéré, les coefficients des contraintes et des fonctions objectifs sont des entiers positifs.

Soit un triangle $\Delta(y^r, y^s)$ défini par deux solutions supportées adjacentes x^r et x^s, avec $y^r = z(x^r), y^s = z(x^s)$ et $y_1^r < y_1^s$; soit l'ensemble des points potentiellement non dominés $\{y^1, \ldots, y^m\}$ correspondant aux meilleures solutions déjà trouvées dans ce triangle. En posant $y^0 = y^r$ et $y^{m+1} = y^s$, la meilleure borne présentée dans [136] est calculée comme indiqué dans la section 2.4.1.1 page 44 :

$$\underline{z}_o^\lambda = \min_{i \in \{0,\ldots,m\}} \left\{ \lambda_1 y_1^i + \lambda_2 y_2^{i+1} \right\}$$

La figure 4.2 illustre cette borne.

Puisque le problème de sac à dos n'utilise que des coefficients entiers, la borne décrite dans [113] peut être adaptée du cas d'une minimisation vers le cas d'une maximisation pour calculer une borne plus serrée sur l'objectif de la somme pondérée :

$$\underline{z}^\lambda = \min \left(\min_{i \in \{0,\ldots,m+1\}} \left\{ \lambda \cdot y^i \right\}, \min_{i \in \{0,\ldots,m\}} \left\{ \lambda_1 (y_1^i + 1) + \lambda_2 (y_2^{i+1} + 1) \right\} \right)$$

Cette valeur, représentée par la ligne continue de la figure 4.2, est la plus faible des valeurs que peuvent avoir les hypothétiques solutions efficaces restant à découvrir dans le triangle, dont les images sont représentées par les points colorés de la figure. Les solutions équivalentes sont conservées avec cette borne. Cette valeur minimale peut correspondre soit à un y^i connu, soit au point obtenu en ajoutant 1 sur chaque dimension d'un point nadir local.

Dans une situation où un ensemble complet maximal n'est pas indispensable, le calcul de la borne peut être restreint aux seuls points potentiellement non dominés pour lesquels aucune solution n'est connue, représentés par des triangles sur la figure 4.2.

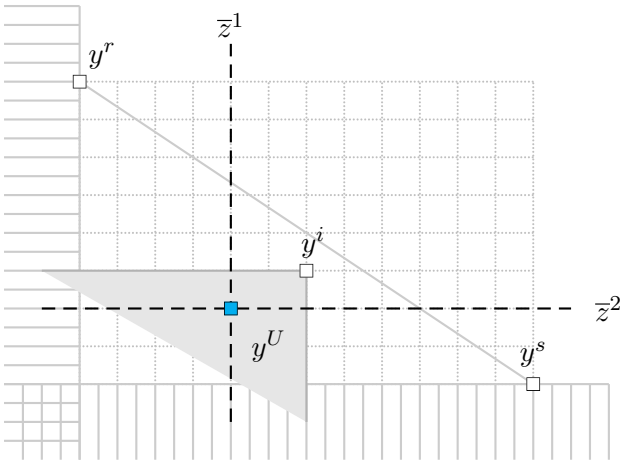

Figure 4.3 – Une situation où une observation indépendante des bornes $\overline{z}^1$ et $\overline{z}^2$ ne permet pas de fermer le nœud alors qu'un test de dominance sur le point utopique local $y^U = (\overline{z}^1, \overline{z}^2)$ confirme que le nœud n'amènera à aucune solution efficace. Nous supposons, sans le représenter, que $\overline{z}^\lambda \geq \underline{z}^\lambda$.

4.2.2 Une amélioration de l'évaluation des nœuds par une borne supérieure bi-objectif

L'évaluation des nœuds durant la procédure de séparation et évaluation, telle que décrite par [136], consiste à calculer une borne supérieure $\overline{z}^j$ sur chaque objectif j et une dernière, $\overline{z}^\lambda$, sur l'objectif de la somme pondérée. Si la valeur d'une de ces bornes est strictement inférieure à une borne inférieure connue pour l'objectif correspondant, le nœud est fermé.

Ainsi, pour une situation telle que représentée par la figure 4.3, aucun de ces tests ne permet de fermer le nœud, l'exploration du sous-arbre de la PSE continue. Cependant, par définition, nous savons que si x est une solution du sous-arbre, alors $z^1(x) \leq \overline{z}^1$ et $z^2(x) \leq \overline{z}^2$. En d'autres termes, nous notons $y^U = (\overline{z}^1, \overline{z}^2)$ un *point utopique local*, par analogie avec la définition 1.7 du point utopique de la page 21 ; « local » dans le sens ce point ne concerne que les solution du sous-arbre. Nous avons alors $z(x) \leqq y^U$.

L'évaluation d'un nœud peut être renforcée en observant simultanément les bornes supérieures $\overline{z}^1$ et $\overline{z}^2$ à travers y^U. En effet, si y^U est dominé par un point réalisable dans le triangle, comme illustré par la figure 4.3, alors aucune solution du sous-arbre n'est efficace. Par conséquent, il est inutile de poursuivre la séparation sur ce nœud.

4.2.3 Expérimentations numériques

Les algorithmes ont été testés sur les instances bi-objectifs des groupes A et B présentés dans la section 3.4.1.1. Ils ont été implémentés en C++ et exécutés sur un Pentium 4 cadencé à 3,73 GHz et accompagné de 3 Go de mémoire vive. Le temps de résolution a été limité à une heure par instance.

La figure 4.4 présente le temps nécessaire pour résoudre ces instances avec la procédure présentée par Visée *et al.* [136] (courbe « PSE ») et une version utilisant l'évaluation améliorée des bornes (courbe « aPSE »). Les instances sont ordonnées par groupe puis par taille croissante.

Ces expérimentations font ressortir une claire amélioration des performances en faveur de l'utilisation du test sur le point utopique local. Nous pouvons remarquer cependant que les deux versions de l'algorithme se comportent globalement de la même façon. Les temps de résolution croissent rapidement

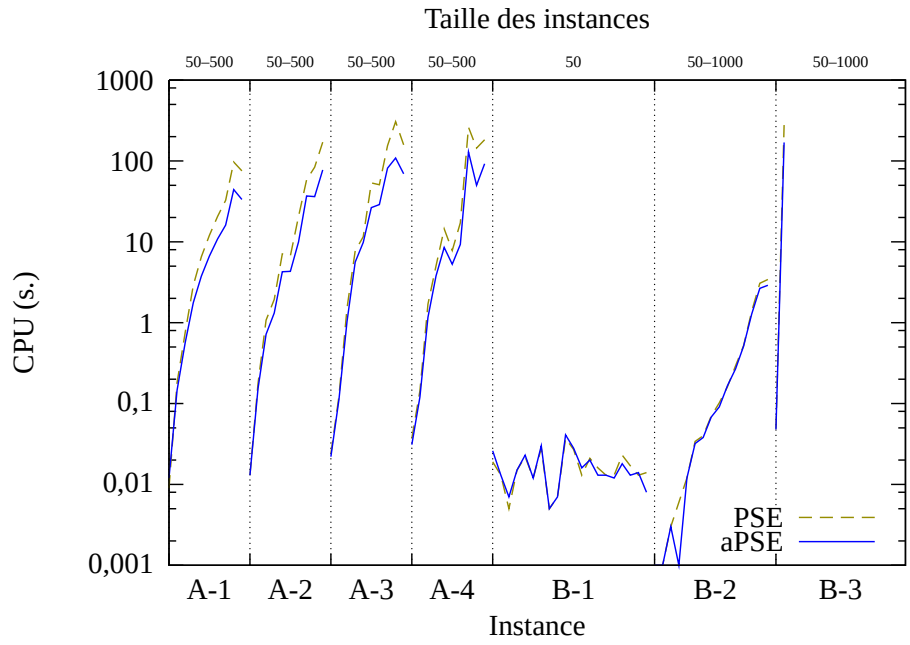

Figure 4.4 – Temps nécessaire pour calculer X_{E_M} avec une procédure en deux phases utilisant une PSE. La courbe « PSE » a été obtenue par notre implémentation de l'algorithme présenté dans [136]. L'évaluation décrite dans la section 4.2 a été implémentée pour obtenir la courbe « aPSE ».

avec la taille de l'instance et, de plus, la façon dont celle-ci a été générée influe directement sur ces temps. En effet, les instances du groupe B-3, dont un objectif est fortement corrélé avec le vecteur des poids, ne sont pas résolues au delà de 150 variables, tandis que celle du groupe A, allant jusqu'à 500 variables, ou celles des groupes B-1 et B-2, ce dernier allant jusqu'à 1000 variables, sont résolues sans difficulté.

Remarquons que l'algorithme apparaît très performant pour ces deux derniers sous-groupes. Cela n'est pas étonnant dans la mesure où les instances de B-1 sont de petite taille (50 variables) et que B-2, dont les instances sont générées avec une corrélation positive entre les objectifs, s'apparente à une famille d'instances mono-objectifs.

En pratique, le temps de résolution est principalement consommé durant la seconde phase. Par conséquent, nous nous concentrerons dans la suite de ce chapitre sur une technique d'exploration des triangles plus efficace.

4.3 Utilisation d'un *ranking* pour l'exploration des triangles

Les algorithmes de *ranking* permettent de calculer les solutions d'un problème en garantissant un ordre sur leur performance. Typiquement, en étant appliquée sur le problème P_λ construit pour l'exploration d'un triangle, une telle méthode va chercher les solutions selon leur valeur z^λ décroissante, jusqu'à ce qu'une borne inférieure soit atteinte. Un des principaux avantages d'une telle procédure est que les solutions dont les images sont les plus proches de l'hypoténuse, pouvant potentiellement apporter de grands incréments à la borne inférieure $\underline{z}^\lambda$, sont trouvées au plus tôt. De plus, toute solution efficace trouvée dans le triangle ne sera jamais remise en question ; contrairement au cas d'une exploration désor-

donnée telle que la PSE, où la performance d'une solution nouvelle peut dominer celle d'une ancienne solution.

Cette approche s'est déjà montrée très performante comme schéma d'énumération dans la seconde phase. En particulier, Przybylski *et al.* [113] l'ont appliqué avec succès sur le problème d'affectation bi-objectif. Nous présentons ici l'application d'une telle stratégie pour résoudre des instances de $01MOKP$ en s'appuyant sur la résolution de $01KP$ comme un problème de plus longs chemins. La section suivante présente les éléments nécessaires à cette résolution.

4.3.1 Résoudre KP comme un problème de plus longs chemins

Étant donnés une instance de $01MOKP$ et une pondération $\lambda \in \mathbb{R}^p$, nous construisons un graphe dans lequel tout chemin partant d'un sommet source initial correspond à une solution réalisable de l'instance, éventuellement incomplète si le chemin peut être étendu. La longueur de ce chemin sera égale à la valeur pondérée de la solution x correspondante : $z^\lambda(x) = \sum_{i=1}^{p} \lambda_i z_i(x)$. La construction du graphe suit le modèle décrit dans la section 2.2.1.3 page 39.

4.3.1.1 Construction du graphe

Soit $G_{KP}(I, \lambda) = (S, A)$ le graphe associé à une instance I donnée de $01MOKP$ et à un poids λ, où S est l'ensemble de ses sommets et $A \subset S \times S$ est l'ensemble de ses arcs.

Dans ce graphe, le passage par un arc correspond à l'affectation d'une valeur, 0 ou 1, à une variable précise. Nous noterons $s_{i,\varpi}$ le sommet où i est l'indice de l'objet considéré en prenant un de ses arcs sortants et ϖ est le poids des solutions, éventuellement incomplètes, correspondant aux chemins allant du sommet initial à $s_{i,\varpi}$. Le sommet initial est noté $s_{1,0}$ et représente une solution vide (c'est-à-dire une solution où aucune valeur n'a été affectée aux variables).

Soit $(s,t) \in A$. Nous noterons $l(s,t)$ la longueur de cet arc. De plus, si $a = (s,t)$, nous noterons $\text{queue}(a) = s$ son sommet de départ et $\text{tête}(a) = t$ son sommet d'arrivée. Enfin, dans un souci de lisibilité, nous noterons $l(a) = l(\text{queue}(a), \text{tête}(a))$.

Les notions de parent et de puits sont introduites par les définitions ci-dessous.

Définition 4.1 (Parent)**.** *Un sommet s est dit* parent *d'un sommet t si $(s,t) \in A$.*

Définition 4.2 (Puits)**.** *Soit un sommet s tel que $\nexists t \in S$ vérifiant $(s,t) \in A$, alors s est un* puits.

Le graphe $G_{KP}(I, \lambda) = (S, A)$ est défini comme suit. Nous rappelons que $c_i^\lambda = \sum_{j=1}^{p} \lambda_j c_i^j$.

$$\begin{aligned} S &= \{s_{i,\varpi} : i \in \{1,\dots,n+1\}, \varpi \in \{0,\dots,\omega\}\} \\ A &= \{(s_{i,\varpi}, s_{i+1,\varpi}) : i \in \{1,\dots,n\}, \varpi \in \{0,\dots,\omega\}\} \\ &\quad \cup \{(s_{i,\varpi}, s_{i+1,\varpi+w_i}) : i \in \{1,\dots,n\}, \varpi \in \{0,\dots,\omega - w_i\}\} \\ l(s_{i,\varpi}, s_{i+1,\varpi}) &= 0, \quad \forall (s_{i,\varpi}, s_{i+1,\varpi}) \in A \\ l(s_{i,\varpi}, s_{i+1,\varpi+w_i}) &= c_i^\lambda, \quad \forall (s_{i,\varpi}, s_{i+1,\varpi+w_i}) \in A \end{aligned} \tag{4.1}$$

Dans la suite du document, si cela n'introduit pas d'ambiguïté, nous omettrons de nommer l'instance et le poids dans la notation du graphe et $G_{KP}(I, \lambda)$ sera nommé G_{KP}.

Chaque sommet $s_{i,\varpi}$ peut être considéré comme un état de l'approche de la programmation dynamique (section 2.4.1.3 page 47) où la valeur des variables $x_j, j < i$, ont été fixées et où $\sum_{j=1}^{i-1} x_j w_j = \varpi$. De plus, puisque les solutions sont construites à partir d'une solution initiale vide et puisque l'ordre dans lequel les objets sont considérés n'a aucune influence sur l'efficacité d'une solution, l'ensemble S des sommets sera restreint aux seuls sommets atteignables depuis $s_{1,0}$.

Définition 4.3 (Chemin)**.** *Un* chemin $\phi = (a_1, \ldots, a_n)$ *est une suite d'arcs adjacents :* $\textit{tête}(a_i) = \textit{queue}(a_{i+1}), \forall i \in \{1, \ldots, n-1\}$*. On notera* $|\phi|$ *le nombre d'arcs dans le chemin et* $z^\lambda(\phi) = \sum_{i=1}^{n} l(a_i)$ *sa longueur.*

On notera $\varphi(s)$ l'ensemble des chemins allant de $s_{1,0}$ au sommet s.

Par construction, le graphe G_{KP} ne contient aucun cycle et chaque sommet a, au plus, deux arcs entrants et, au plus, deux arcs sortants. De plus, si un sommet a exactement deux arcs entrants, seul l'un d'eux a une longueur nulle. Il en va de même pour les deux arcs sortants. Enfin, pour un indice $i > 1$ donné, les sommets $s_{i,\varpi}$ sont séparés de $s_{1,0}$ par $i-2$ sommets, quel que soit ϖ. Les sommets $s_{i,\varpi}$ forment alors une « couche » dans la structure du graphe. Celui-ci est donc constitué de $n+1$ couches d'au plus $\omega + 1$ sommets, dont la taille augmente avec l'éloignement par rapport à $s_{1,0}$, unique sommet de la première couche. Les sommets de la dernière sont des puits.

Définition 4.4 (Suffixe commun)**.** *Deux chemins* $\phi = (a_1, \ldots, a_n), \phi' = (a'_1, \ldots, a'_n)$ *sont dits* i-suffixe-commun, $i \in \{1, \ldots, n\}$*, si les trois propriétés ci-dessous sont vérifiées :*

1. $a_i \neq a'_i$;
2. $a_j = a'_j, \forall j > i$;
3. $\textit{tête}(a_n) = \textit{tête}(a'_n)$.

Exemple 4.14 (Graphe associé à une instance). *Soit l'instance de* $2-01KP$ *ci-dessous*

$$\begin{array}{lll} \max & z^1 = & 1x_1 + 2x_2 + 2x_3 + 6x_4 + 4x_5 \\ \max & z^2 = & 8x_1 + 6x_2 + 6x_3 + 4x_4 + 4x_5 \\ \textit{s.c} & & 4x_1 + 4x_2 + 4x_3 + 6x_4 + 6x_5 \quad \leq 12 \\ & & x_i \in \{0,1\} \qquad\qquad\qquad\quad \forall i \in \{1, \ldots, 5\} \end{array}$$

Cette instance accepte deux solutions supportées, $x^r = (1,1,1,0,0)$ *et* $x^s = (0,0,0,1,1)$*. Nous posons* $y^r = z(x^r) = (5, 20)$ *et* $y^s = z(x^s) = (10, 8)$*. La pondération associée à* $\Delta(y^r, y^s)$ *est* $\lambda = (12, 5)$ *et le problème mono-objectif* P_λ *correspondant, utilisant donc les coûts* c_i^λ*, est défini comme suit*

$$\begin{array}{lll} \max & z^\lambda = & 52x_1 + 54x_2 + 54x_3 + 92x_4 + 68x_5 \\ \textit{s.c} & & 4x_1 + 4x_2 + 4x_3 + 6x_4 + 6x_5 \quad \leq 12 \\ & & x_i \in \{0,1\} \qquad\qquad\qquad\quad \forall i \in \{1, \ldots, 5\} \end{array}$$

Le graphe associé à cette instance est illustré par la figure 4.5. L'ensemble $\varphi(s_{6,12})$ *est mis en évidence, à titre illustratif. Il contient deux chemins :*

1. $\phi^1_{6,12} = ((s_{1,0}, s_{2,0}), (s_{2,0}, s_{3,0}), (s_{3,0}, s_{4,0}), (s_{4,0}, s_{5,6}), (s_{5,6}, s_{6,12}))$;
2. $\phi^2_{6,12} = ((s_{1,0}, s_{2,4}), (s_{2,4}, s_{3,8}), (s_{3,8}, s_{4,12}), (s_{4,12}, s_{5,12}), (s_{5,12}, s_{6,12}))$.

Ces chemins sont 5-suffixe-commun. Néanmoins, dans le cas général, si $s \in S$*, les chemins de* $\varphi(s)$ *ne sont pas nécessairement* k*-suffixe-commun deux à deux, pour un unique* k *donné.*
Ainsi, pour les chemins de $\varphi(s_{6,8})$ *ci-dessous, le premier est 3-suffixe-commun avec le second, mais est 2-suffixe-commun avec le troisième.*

1. $\phi^1_{6,8} = ((s_{1,0}, s_{2,0}), (s_{2,0}, s_{3,4}), (s_{3,4}, s_{4,8}), (s_{4,8}, s_{5,8}), (s_{5,8}, s_{6,8}))$;
2. $\phi^2_{6,8} = ((s_{1,0}, s_{2,4}), (s_{2,4}, s_{3,8}), (s_{3,8}, s_{4,8}), (s_{4,8}, s_{5,8}), (s_{5,8}, s_{6,8}))$;
3. $\phi^3_{6,8} = ((s_{1,0}, s_{2,4}), (s_{2,4}, s_{3,4}), (s_{3,4}, s_{4,8}), (s_{4,8}, s_{5,8}), (s_{5,8}, s_{6,8}))$.

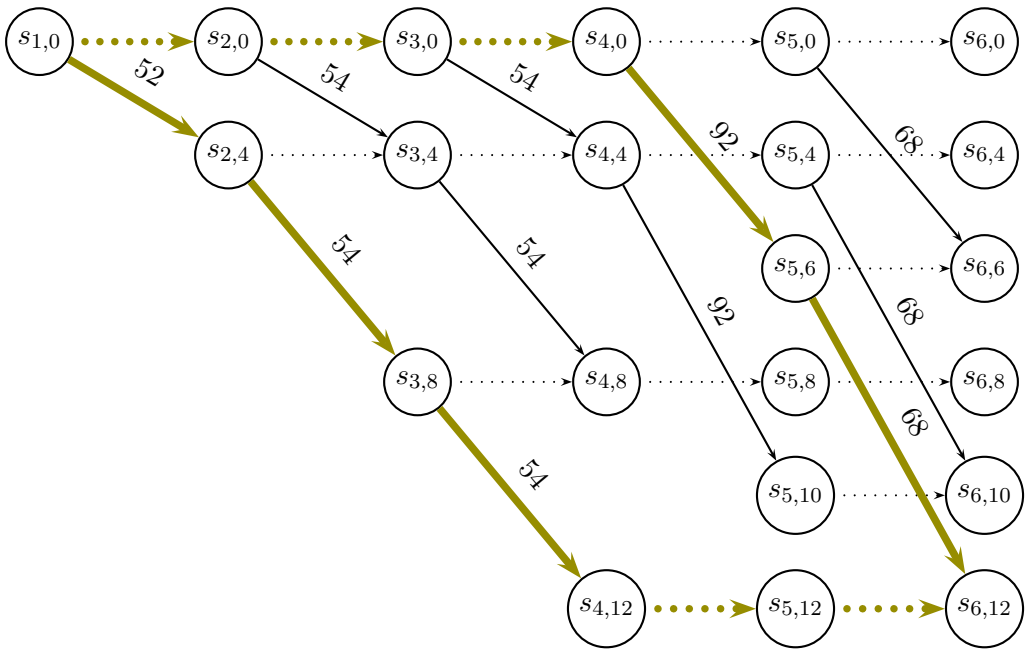

Figure 4.5 – Graphe associé à l'instance de l'exemple 4.14. La valeur indiquée au dessus des arcs est leur poids. Les arcs en pointillés ont un poids nul, non indiqué pour plus de lisibilité. Les chemins de l'ensemble $\varphi(s_{6,12})$ sont mis en évidence.

Étant donné un chemin ϕ de G_{KP} reliant les sommets $s_{j,\varpi}$ et $s_{k,\varpi'}$, les solutions x qui lui correspondent dans le problème de sac à dos initial vérifient :

$$\forall i \in \{j, \dots, k\} : x_i = \begin{cases} 1 \text{ si } \exists \varpi : (s_{i,\varpi}, s_{i+1,\varpi+w_i}) \in \phi \\ 0 \text{ sinon} \end{cases} \tag{4.2}$$

Si $|\phi| = n$, alors la solution associée est une solution du problème de sac à dos initial. Symétriquement, si x est une solution réalisable pour $01MOKP$, alors elle est associée à exactement un chemin de longueur n dans G_{KP}.

En accord avec cette correspondance, nous ne ferons aucune distinction par la suite entre une solution et le chemin qui lui est associé.

En utilisant les équations classiques de la programmation dynamique [59, 70], nous associons à chaque sommet $s_{i,\varpi}$ la longueur des plus longs chemins de $\varphi(s_{i,\varpi})$:

$$z^\lambda(s_{i,\varpi}) = \max_{x \in X} \{z^\lambda(x) : w(x) = \varpi : x_j = 0, \forall j \geq i\}$$

Selon cette équation, nous avons $z^\lambda(s_{1,0}) = 0$. Quand $i \geq 2$, $z^\lambda(s_{i,\varpi})$ est calculée comme suit :

$$z^\lambda(s_{i,\varpi}) = \begin{cases} \max\{z^\lambda(s_{i-1,\varpi}), z^\lambda(s_{i-1,\varpi-w_{i-1}}) + c^\lambda_{i-1}\} \text{ si } s_{i,\varpi} \text{ a deux arcs entrants} \\ z^\lambda(s) + l(s, s_{i,\varpi}) \text{ si } s \text{ est le sommet à l'origine du seul arc entrant} \end{cases} \tag{4.3}$$

Dans le premier cas, $s_{i,\varpi}$ ne peut être atteint que par deux parents, $s_{i-1,\varpi}$ et $s_{i-1,\varpi-w_{i-1}}$. Par construction, le premier est lié via un arc de poids nul et le second par un arc de poids c^λ_{i-1}. La valeur du plus long chemin pour rejoindre $s_{i,\varpi}$ est donc la plus grande selon que l'on passe par $s_{i-1,\varpi}$, suivi par l'arc de poids nul, ou que l'on passe par $s_{i-1,\varpi-w_{i-1}}$, suivi par l'arc non nul.

Le second cas concerne la situation où un seul sommet permet de rejoindre $s_{i,\varpi}$. Par conséquent, le chemin le plus long pour l'atteindre est le plus long amenant au parent, suivi par l'unique arc entrant.

Définition 4.5 (Arc optimal)**.** *Un arc (s,t) est dit* optimal *si $z^\lambda(t) = z^\lambda(s) + l(s,t)$.*
Dans le cas où les deux arcs entrants vérifient cette égalité, seul celui de poids non nul est optimal.

Remarque. Par définition, tout sommet autre que $s_{1,0}$ possède un arc entrant optimal.

Définition 4.6 (Arc secondaire)**.** *Si un arc $a = (s,t)$ est optimal, alors l'arc $(s',t) \in A, s' \neq s$, s'il existe, est dit* secondaire *et est noté $\bar{a}$.*

Exemple 4.15 (Longueurs, arcs optimaux et secondaires). *La figure 4.6 reprend le graphe de l'exemple 4.14 en associant à chaque sommet la longueur des chemins y menant. Les arcs secondaires y sont représentés par un trait épais et coloré. Les autres arcs sont tous optimaux.*
Dans chaque sommet, la longueur du plus long chemin pour les atteindre depuis $s_{1,0}$ est indiquée.

Une particularité de G_{KP} est que la longueur du plus long chemin utilisant un arc secondaire $\bar{a}$ se déduit immédiatement d'un chemin ne passant pas par $\bar{a}$ et passant par les mêmes arcs après celui-ci (c'est-à-dire que ce chemin passe par a). En d'autres termes, si $\phi = (a_1, \dots, a_i, \dots, a_j, \dots, a_k, \dots, a_n)$ est un chemin où a_j est un arc optimal, la longueur du plus long chemin $\phi' = (b_1, \dots, b_i, \dots, \bar{a}_j, \dots, a_k, \dots, a_n)$, s'il existe, se déduit de ϕ selon l'équation ci-dessous [45] :

$$z^\lambda(\phi') = z^\lambda(\phi) - z^\lambda(\text{tête}(a_j)) + z^\lambda(\text{queue}(\bar{a}_j)) + l(\bar{a}_j) \tag{4.4}$$

Par définition, nous avons $z^\lambda(\phi') \leq z^\lambda(\phi)$. Cette propriété nous sera utile par la suite pour construire des solutions de manière ordonnée dans le *ranking*.

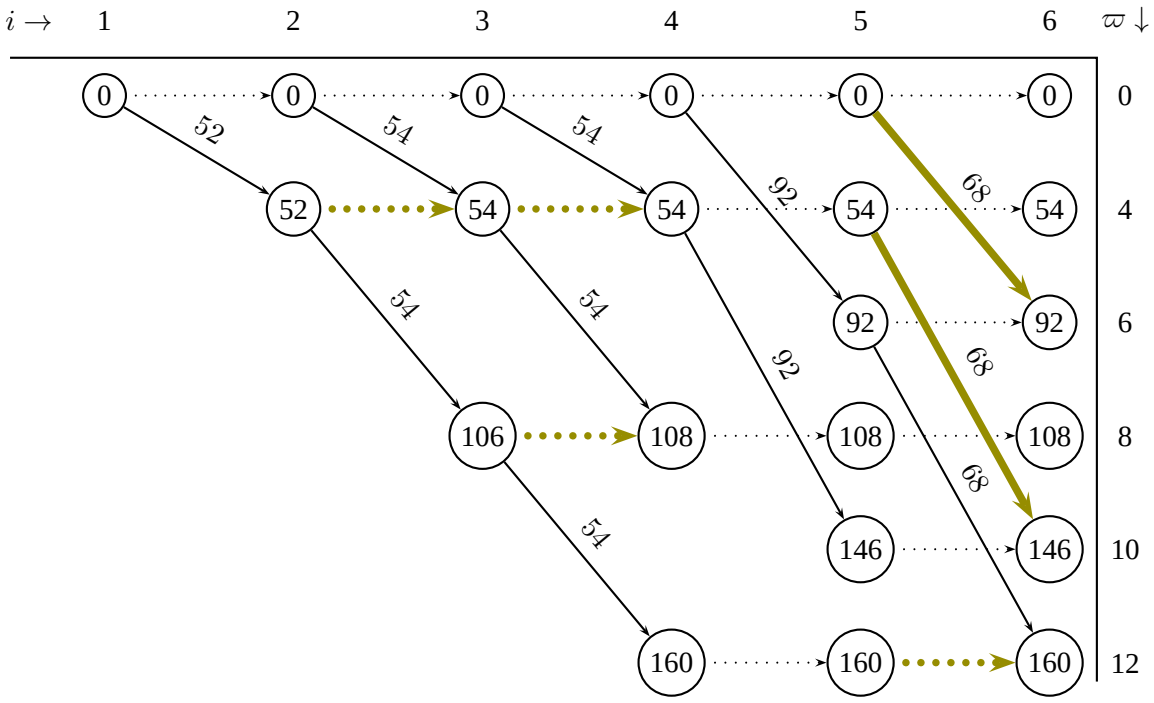

Figure 4.6 – Plus longs chemins et arcs secondaires dans l'exemple 4.14. La valeur indiquée dans les nœuds est la longueur des plus longs chemins pour les atteindre depuis $s_{1,0}$. Les arcs secondaires sont représentés d'un trait épais et coloré.

4.3.1.2 Éléments multi-objectifs dans la structure du graphe

Dans le cadre de la résolution de $01MOKP$, plusieurs éléments du problème initial sont associés au graphe et à ses chemins. Ainsi, nous associons à chaque arc $a = (s_{i,\varpi}, s_{i+1,\varpi'}) \in A$ le coût multi-objectif de la variable considérée par l'utilisation de cet arc de la manière suivante :

$$c(s_{i,\varpi}, s_{i+1,\varpi'}) = \begin{cases} 0 & \text{si } l(s_{i,\varpi}, s_{i+1,\varpi'}) = 0 \\ c_i & \text{sinon} \end{cases}$$

La longueur multi-objectif d'un chemin $\phi = (a_1, \ldots, a_n)$ est évidemment $z(\phi) = \sum_{i=1}^{n} c(a_i)$.

De plus, nous associons à chaque sommet $s_{i,\varpi}$ la valeur $z(\phi^*)$ où ϕ^* est le chemin optimal reliant $s_{1,0}$ et $s_{i,\varpi}$.

Le calcul de $z(s_{i+1,\varpi})$ peut se faire en même temps que $z^\lambda(s_{i+1,\varpi})$ en utilisant la définition 4.5, qui découle de l'équation 4.3. Cette valeur est calculée comme ci-dessous :

$$z(s_{i+1,\varpi}) = z(s_{i,\varpi'}) + \begin{cases} c_i & \text{si } (s_{i,\varpi'}, s_{i+1,\varpi}) \text{ est optimal} \\ 0 & \text{sinon} \end{cases}$$

Enfin, par construction de $z(s_{i,\varpi})$, l'équation 4.4 se généralise immédiatement au cas multi-objectif. Si $\phi = (a_1, \ldots, a_i, \ldots, a_j, \ldots, a_k, \ldots, a_n)$ est un chemin où a_j est un arc optimal, la longueur du plus long chemin ϕ', selon z^λ, avec $\phi' = (b_1, \ldots, b_i, \ldots, \bar{a}_j, \ldots, a_k, \ldots, a_n)$, se déduit de ϕ selon l'équation ci-dessous :

$$z(\phi') = z(\phi) - z(\text{tête}(a_j)) + z(\text{queue}(\bar{a}_j)) + c(\bar{a}_j)$$

Remarquons qu'aucune relation d'ordre ni de dominance n'existe entre $z(\phi')$ et $z(\phi)$ dans ce cas.

4.3.1.3 Reconstruction d'un chemin à partir d'un sommet

La valeur associée à un sommet du graphe correspond à la longueur des plus longs chemins arrivant à ce sommet. Cependant, les arcs utilisés pour ce chemin ne sont pas immédiatement disponibles. Nous décrivons ici comment les retrouver.

Étant donné un sommet $s_{i,\varpi}, i \geq 2$, on peut déduire de la manière suivante le sommet de la couche $i-1$ utilisé pour calculer $z^\lambda(s_{i,\varpi})$:

- si $s_{i,\varpi}$ n'a qu'un seul arc entrant, le sommet à l'autre extrémité est le précédent ;
- si $s_{i,\varpi}$ a deux arcs entrants, le sommet s' tel que $(s', s_{i,\varpi}) \in A$ et tel que $z^\lambda(s') + l(s', s_{i,\varpi}) = z^\lambda(s_{i,\varpi})$ est le précédent dans le chemin.

En répétant itérativement ce procédé sur le sommet précédent jusqu'à arriver au sommet $s_{1,0}$, il est possible de construire un chemin $\phi^* \in \varphi(s_{i,\varpi})$ tel que $z^\lambda(\phi^*) = z^\lambda(s_{i,\varpi})$.

Puisque chaque sommet n'a, au plus, que deux arcs entrants ou sortants, alors il suffit d'imposer un arc secondaire durant le parcours pour obtenir un chemin de longueur inférieure ou égale aux plus longs chemins se terminant par $s_{i,\varpi}$. Imposer un arc secondaire supplémentaire permet d'obtenir un chemin à nouveau inférieur ou égal, et ainsi de suite. Tout chemin du $\varphi(s_{i,\varpi})$ peut ainsi être obtenu de ϕ^* en imposant un ensemble adéquat d'arcs secondaires.

Plus généralement, tout chemin ϕ du graphe peut être implicitement représenté, de manière unique, par son dernier sommet t et un ensemble U d'arcs secondaires (Eppstein [45]). Un tel chemin sera noté

$< t, U >$ par la suite. Si $U = \emptyset$, alors ϕ sera appelé *chemin optimal*. Inversement, si $U \neq \emptyset$, ϕ sera nommé *chemin secondaire*.

Exemple 4.16 (Représentation implicite d'un chemin). *Dans le graphe des exemples 4.14 et 4.15, nous considérons les six chemins de $\varphi(s_{6,10})$. Un plus long chemin parmi eux est*

$$\phi^1_{6,10} = ((s_{1,0}, s_{2,0}), (s_{2,0}, s_{3,0}), (s_{3,0}, s_{4,4}), (s_{4,4}, s_{5,10}), (s_{5,10}, s_{6,10}))$$

Sa longueur est $z^\lambda(\phi^1_{6,10}) = 146$. Ce chemin ne contient pas d'arc secondaire ; ainsi, il sera représenté par $< s_{6,10}, \emptyset >$. Les représentations implicites des cinq autres chemins et leurs longueurs sont :

1. $\phi^2_{6,10} =< s_{6,10}, \{(s_{3,4}, s_{4,4})\} >$, *avec* $z^\lambda(\phi^2_{6,10}) = 146$;
2. $\phi^3_{6,10} =< s_{6,10}, \{(s_{3,4}, s_{4,4}), (s_{2,4}, s_{3,4})\} >$, *avec* $z^\lambda(\phi^3_{6,10}) = 144$;
3. $\phi^4_{6,10} =< s_{6,10}, \{(s_{5,4}, s_{6,10})\} >$, *avec* $z^\lambda(\phi^4_{6,10}) = 122$;
4. $\phi^5_{6,10} =< s_{6,10}, \{(s_{5,4}, s_{6,10}), (s_{3,4}, s_{4,4})\} >$, *avec* $z^\lambda(\phi^5_{6,10}) = 122$;
5. $\phi^6_{6,10} =< s_{6,10}, \{(s_{5,4}, s_{6,10}), (s_{3,4}, s_{4,4}), (s_{2,4}, s_{3,4})\} >$, *avec* $z^\lambda(\phi^6_{6,10}) = 120$.

La reconstruction d'un chemin s'effectue donc à l'envers, de son dernier sommet vers $s_{1,0}$. Ce procédé est décrit par l'algorithme 4.1. Il suffit ensuite d'utiliser l'équation 4.2 pour retrouver la solution équivalente pour l'instance du problème de sac à dos.

Procédure `reconstruire_chemin` ($\downarrow t, \downarrow U, \uparrow \phi$)
Paramètre $\downarrow t$: le sommet sur lequel se termine le chemin.
Paramètre $\downarrow U$: un ensemble d'arcs imposés.
Paramètre $\uparrow \phi$: un plus long chemin de $\varphi(t)$.

tant que $t \neq s_{1,0}$ **faire**
  si $\exists (s', t) \in U$ **alors**
    $s \leftarrow s'$ — *- - Si le chemin a un arc secondaire arrivant au sommet t, nous l'utilisons. - -*
  sinon si `degré_entrant`(t) = 1 **alors** — *- - Sinon, nous déduisons le sommet parent. - -*
    $s \leftarrow$ `parent`(t) — *- - L'unique parent de t. - -*
  sinon
    $\{s^1, s^2\}$ = `parents`(t) — *- - Les deux parents de t. - -*
    si $z^\lambda(s^1) + l(s^1, t) = z^\lambda(t)$ **alors**
      $s \leftarrow s^1$
    sinon
      $s \leftarrow s^2$
    fin si
  fin si
  $\phi \leftarrow ((s, t), a_1, \ldots, a_{|\phi|}) : a_i \in \phi$
  $t \leftarrow s$
fin tant que

ALG. 4.1 – Reconstruction d'un chemin de $\varphi(t)$.

4.3.2 Utilisation d'un *ranking* dans la seconde phase

Nous présentons ici l'intégration d'un algorithme de *ranking* dans une méthode en deux phases pour le problème de sac à dos bi-objectif puis nous détaillons les étapes amenant à une implémentation efficace. Cet algorithme est inspiré de la littérature sur les algorithmes de *ranking* pour les problèmes des plus courts chemins, en particulier des travaux de Eppstein [45].

4.3.2.1 Généralités sur l'application d'un *ranking* dans la seconde phase

Soit un triangle $\Delta(y^r, y^s)$ dont l'hypoténuse est défini par les points supportés extrêmes y^r et y^s. Nous considérons l'objectif de la somme pondérée $z^\lambda = \lambda_1 z_1(x) + \lambda_2 z_2(x)$ pour laquelle x^r et x^s sont optimales. Un algorithme de *ranking* va chercher les solutions selon leur valeur z^λ décroissante, jusqu'à ce qu'une borne inférieure soit atteinte.

L'algorithme 4.2 illustre une telle procédure d'exploration. Initialement, l'ensemble X^Δ des solutions trouvées dans le triangle est vide. Pour commencer le *ranking*, une solution optimale pour P_λ doit être calculée (ligne 2). La boucle principale (lignes 5 à 12) consiste à calculer la k^{e} meilleure solution x^k en utilisant la procédure `k_opt`. Si cette solution n'est pas dominée, nous l'ajoutons à X^Δ et mettons la borne inférieure à jour via la procédure `màj_borne_inf`, comme décrit dans la section 4.2.1. La procédure s'arrête dès que $z^\lambda(x^k)$ est inférieure à la borne inférieure.

Procédure `explorer_triangle` $(\downarrow y^r, \downarrow y^s, \uparrow X^\Delta)$
Paramètre $\downarrow y^r, y^s$: points supportés extrêmes adjacents, définissants un triangle $\Delta(y^r, y^s)$.
Paramètre $\uparrow X^\Delta$: un ensemble complet de solutions dont les images sont dans le triangle.

1: $\lambda \leftarrow (y^r_2 - y^s_2, y^s_1 - y^r_1)$
2: $k \leftarrow 1$; $x^1 \leftarrow$ `mono_opt`(λ)
3: $X^\Delta \leftarrow \{x^1\}$
4: $\underline{z}^\lambda \leftarrow$ `màj_borne_inf`(X^Δ)
5: **tant que** $z^\lambda(x^k) \geq \underline{z}^\lambda$ **faire**
6: $\quad k \leftarrow k + 1$
7: $\quad x^k \leftarrow$ `k_opt`(k, λ)
8: $\quad$ **si** $\nexists x \in X^\Delta : z(x) \geq z(x^k)$ **alors**
9: $\quad\quad$ `nouvelle_solution` $(\downarrow x^k, \updownarrow X^\Delta)$
10: $\quad\quad \underline{z}^\lambda \leftarrow$ `màj_borne_inf`(X^Δ)
11: $\quad$ **fin si**
12: **fin tant que**

ALG. 4.2 – Exploration ordonnée d'un triangle.

Remarquons que les solutions équivalentes sont énumérées par un tel algorithme. Elles sont ajoutées dans X^Δ à la ligne 9 mais pourraient simplement être ignorées.

L'utilisation d'un *ranking* implique nécessairement que des solutions correspondant à des points hors du triangle, dans un triangle adjacent, peuvent être visitées. Elles peuvent aussi être ignorées, comme dans [113], ou conservées, comme dans [105]. Dans tous les cas, ces solutions seront énumérées à nouveau lors de l'exploration du triangle dans lequel se situe leur image.

4.3.2.2 Une procédure de *ranking* pour trouver les k plus longs chemins

La procédure est basée sur une stratégie de partitionnement général itératif proposé par Lawler [84] et parfois nommée *multiple search tree algorithm*. Une solution initiale $x^1 \in X$ est calculée puis $X \setminus \{x^1\}$ est divisé en, au plus, n partitions X_i ; c'est-à-dire en des ensembles disjoints tels que $\bigcup_{i=1}^{n} X_i = X \setminus \{x^1\}$. La seconde meilleure solution x^2 est la meilleure parmi celles des ensembles X_i. Supposons que x^2 soit trouvée dans l'ensemble X_j alors X_j est renommé $\tilde{X}$ et $\tilde{X} \setminus \{x^2\}$ est partitionné en, au plus, n sous-ensembles $\tilde{X}_i$. La troisième meilleure solution est la meilleure parmi celles des sous-ensembles $X_i, i \neq j$ et $\tilde{X}_i$. Lors de la $(k+1)^{\text{e}}$ itération de cet algorithme, l'ensemble $X \setminus \{x^1, \ldots, x^k\}$ est partitionné en un certain nombre de partitions. Nous savons que x^{k+1} est dans une de ces partitions $\tilde{X}$ et $\tilde{X} \setminus \{x^{k+1}\}$ doit être partitionné pour calculer x^{k+2}...

Un point essentiel dans l'efficacité d'un algorithme de *ranking* est le calcul de la meilleure solution de chaque partition. De même, un autre point important est la gestion de l'ensemble des partitions. En effet, avec une telle stratégie de partitionnement, leur nombre peut être important et beaucoup d'entre elles ne seront pas considérées.

Nous dénoterons par la suite l'ensemble des chemins de longueur n de G_{KP} par $X(G_{KP})$. Nous rappelons qu'il y a une bijection entre les solutions $x \in X$ et les chemins de $X(G_{KP})$. Ces chemins relient le sommet $s_{1,0}$ à un sommet puits de la dernière couche. Nous savons quelle est la longueur du plus long chemin arrivant à chacun de ces sommets par l'équation 4.3. Par conséquent, nous pouvons effectuer un premier partitionnement de $X(G_{KP})$ selon ces puits, tel que chaque sous-ensemble contienne tous les chemins se terminant en un puits donné et uniquement ceux-là. Ceci est utile pour ne considérer par la suite que des partitions où nous savons que les chemins relient deux sommets particuliers, $s_{1,0}$ et un puits connu.

La proposition 4.1 formalise cette idée.

Proposition 4.1. *Soit $X_\varpi = \varphi(s_{n+1,\varpi})$. L'ensemble $\mathcal{X}_0 = \{X_\varpi : \varpi \in \{0, \ldots, \omega\}\}$ est un partitionnement de $X(G_{KP})$.*

Une meilleure solution $\phi^1 = (a_1, \ldots, a_n)$ est facilement obtenue en calculant le plus long chemin parmi chaque ensemble de $\mathcal{X}_0$, ce qui est déjà effectué lors de la construction de G_{KP}. Pour calculer la deuxième meilleure solution, l'ensemble $X(G_{KP}) \setminus \{\phi^1\}$ est partitionné. Pour cela, X_ϖ est renommé en $\tilde{X}$, où ϖ est le poids de la solution ϕ^1. Puis, le partitionnement de $\tilde{X} \setminus \{\phi^1\}$ est effectué.

Dans le cadre du problème des plus longs chemins, le partitionnement est classiquement obtenu en imposant et en interdisant des arcs dans les partitions, de manière à exclure le plus long chemin de celles-ci [45]. Nous pouvons remarquer que dans le cas de G_{KP}, imposer de ne pas prendre un arc revient à imposer de prendre l'arc secondaire. Par conséquent, nous pouvons effectuer le partitionnement de la manière décrite ci-après.

Le partitionnement initial de $\tilde{X} \setminus \{\phi^1\}$ s'obtient en observant les arcs de ϕ^1 un à un, depuis le dernier vers le premier. Pour chacun de ces arcs, une partition $\tilde{X}_i$ est obtenue en imposant d'utiliser les arcs $a_j, j > i$ et l'arc secondaire $\bar{a}_i$. Si ce dernier n'existe pas, alors la partition est vide. La seconde meilleure solution est ensuite calculée dans l'ensemble des partitions, puis le processus est répété. D'une manière générale, lors de la k^{e} itération, la k^{e} meilleure solution ϕ^k est calculée dans toutes les partitions disponibles, y compris les partitions X_ϖ dans lesquelles aucune solution n'a été choisie pour l'instant. Nous notons $\phi^k = (b_1, \ldots, b_l, \ldots, b_n)$, où les arcs $b_l, \ldots, b_n$ sont ceux imposés dans la partition où est trouvée ϕ^k, notée $\tilde{X}_k$. Notons que ces arcs peuvent indifféremment être optimaux ou secondaires et que, si $\tilde{X}_k \in \mathcal{X}_0$, alors ϕ^k n'a pas d'arc imposé. $X(G_{KP}) \setminus \{\phi^1, \ldots, \phi^k\}$ est alors partitionné d'une manière

similaire, en considérant un à un les arcs $b_1, \ldots, b_{l-1}$ depuis le dernier jusqu'au premier. Pour chaque arc b_i considéré, une nouvelle partition est crée, en imposant d'utiliser les arcs $b_j, j \in \{i+1, \ldots, n\}$, ainsi que l'arc secondaire $\bar{b}_i$, s'il existe.

Remarquons que les partitions obtenues sont définies par des chemins secondaires ϕ^k du plus long chemin $\phi^* = (a_1, \ldots, a_n)$ de $\tilde{X}_k$. Ainsi, la plus grande longueur au sein de $\tilde{X}_k$ est connue d'après l'équation 4.4 et est égale à $z^\lambda(\phi^k)$. En conservant cette information lors de la construction de $\tilde{X}_k$, nous savons à tout moment dans quelle partition chercher la meilleure solution suivante. Ces plus longs chemins secondaires sont ensuite utilisés pour partitionner $\tilde{X}_k$ de la même manière.

Ainsi, le plus long chemin d'une partition est suffisant pour l'identifier : si $\phi^* =< s_{n+1,\varpi}, U >$ représente $\tilde{X}$, alors $\tilde{X}_k$ est identifiée par le chemin

$$\phi^k = \begin{cases} < s_{n+1,\varpi}, U \cup \{\bar{a}_k\} >, & \text{si } \exists \bar{a}_k \\ \emptyset & \text{sinon} \end{cases}$$

4.3.3 Algorithme et implémentation

Nous présentons ici l'algorithme de *ranking* tel que nous l'appliquons au problème de sac à dos, ainsi que les points importants pour obtenir une implémentation efficace.

L'application de cet algorithme se fait pour chaque triangle durant la seconde phase et est précédée, à chaque fois, d'un appel à une procédure de réduction appliquée à P_λ, telle que présentée dans la section 2.3.1 page 42. Remarquons que cette procédure requiert que les objets soient triés par ordre d'efficacité décroissante, ordre pouvant varier d'un triangle à l'autre. Cet ordre nous sera utile pendant la construction du graphe, ci-après.

Lors de l'application du *ranking* à $2-01KP$, nous conserverons les performances des solutions trouvées hors du triangle. Ainsi, avant d'explorer un triangle adjacent, les performances qui y ont déjà été trouvées permettront de calculer une borne inférieure initiale $\underline{z}^\lambda$ plus serrée. Cette borne sera utilisée pour fixer un plus grand nombre de variables lors de l'application de la procédure de réduction avant de commencer l'exploration du triangle.

Les triangles sont traités dans l'ordre décroissant de la distance entre l'hypoténuse et la borne inférieure associée. Nous supposons en effet que ceux pour lesquels cette distance est la plus faible seront explorés plus rapidement et que les performances trouvées en dehors permettront de réduire cette distance dans les triangle adjacents.

4.3.3.1 Construction du graphe

Le graphe G_{KP} décrit par les équations 4.1 et 4.3 se construit de couche en couche, en partant du sommet initial $s_{1,0}$. De nombreux sommets n'interviennent pas dans les chemins construits par le *ranking*. En particulier, si $\underline{z}^\lambda$ est une borne inférieure de la valeur des solutions efficaces du triangle exploré (voir la section 4.2.1), alors les sommets $s_{n+1,\varpi}$ tels que $z^\lambda(s_{n+1,\varpi}) < \underline{z}^\lambda$ ne seront pas visités ; ils peuvent donc être ignorés durant la construction.

D'une manière générale, tout sommet n'intervenant que dans des chemins ϕ tels que $z^\lambda(\phi) < \underline{z}^\lambda$ peut être ignoré. L'exclusion de ces sommets entraîne une réduction de la consommation de mémoire et, si leur suppression est efficace, peuvent aussi apporter une économie sur le coût de construction du graphe.

Néanmoins, déterminer lors de sa construction si un sommet n'interviendra pas dans un chemin visité par le *ranking* n'est pas une chose évidente. De plus, il n'est bien entendu pas utile de supprimer

ces sommets *a posteriori*. En effet, le coût de construction aura déjà été consommé et la quantité de mémoire utilisée sera alors déjà à son maximum.

La solution que nous avons adopté est la suivante. Lorsqu'un nouveau sommet $s_{i,\varpi}$ est ajouté au graphe, nous connaissons $z^\lambda(s_{i,\varpi})$ par l'équation 4.3. Nous calculons alors une borne supérieure $\overline{z}^\lambda$ en appliquant la relaxation U_2 de la section 2.2.1.1, page 36, sur les objets dont l'indice est supérieur ou égal à i. Si $z^\lambda(s_{i,\varpi})\overline{z}^\lambda < \underline{z}^\lambda$, alors ce nouveau sommet est ignoré.

Remarquons que ce procédé requiert que les objets considérés dans le calcul de U_2 soient triés par ordre décroissant d'efficacité. Or, cet ordre a déjà été appliqué pour effectuer la réduction de la taille du problème à l'initialisation de l'exploration.

4.3.3.2 Génération des partitions

La procédure de génération des chemins secondaires identifiant les partitions est présentée par l'algorithme 4.3. Elle prend en paramètre un chemin $\phi =< s_{n+1,\varpi}, U >$ à partir duquel de nouveaux chemins sont dérivés. Ces nouveaux chemins sont ajoutés à un ensemble T.

Cette procédure est similaire à l'algorithme 4.1 pour le parcours du chemin. Lorsque tous les arcs de U ont été visités, un nouveau chemin $< s, U \cup \{\bar{a}\} >$ est créé (ligne 18), ceci pour chaque arc secondaire $\bar{a}$ arrivant sur les sommets rencontrés jusqu'à $s_{1,0}$.

Nous pouvons remarquer que l'ensemble U définissant ϕ ne sera pas modifié dans la suite du traitement. Nous pouvons donc adopter une implémentation où U n'est pas copié d'un chemin à l'autre mais où les nouveaux chemins contiennent une référence vers U, ainsi que l'arc secondaire. Une structure hiérarchique comme celle-ci permet d'éviter de nombreuses copies et ainsi d'économiser de la mémoire. Un autre avantage est que la construction du nouveau chemin en ligne 19 se fait en temps constant.

Exemple 4.17 (Partitionnement). *Soit l'instance de l'exemple 4.14 page 71, pour lequel les arcs secondaires ont été illustrés par la figure 4.6 page 74. Nous allons décrire le partitionnement de X_ϖ pour $\varpi = 10$.*
Le plus long chemin de X_ϖ est $\phi^1 =< s_{n+1,10}, U^1 = \emptyset >$. Durant le parcours de ce chemin par l'algorithme 4.3 les arcs secondaires $(s_{5,4}, s_{5,10})$ et $(s_{3,4}, s_{4,4})$ ne sont pas traversés. Ils sont utilisés pour le partitionnement, duquel résulte deux nouveaux chemins

$$\phi^2 =< s_{n+1,10}, U^2 = \{(s_{5,4}, s_{5,10})\} \cup U^1 >$$
$$\phi^3 =< s_{n+1,10}, U^3 = \{(s_{3,4}, s_{4,4})\} \cup U^1 >$$

Ces chemins seront ensuite parcourus et un nouveau chemin sera obtenu pour chaque, respectivement

$$\phi^4 =< s_{n+1,10}, U^4 = \{(s_{3,4}, s_{4,4})\} \cup U^2 >$$
$$\phi^5 =< s_{n+1,10}, U^5 = \{(s_{2,4}, s_{3,4})\} \cup U^3 >$$

Le parcours de ϕ^5 n'entraînera pas d'autres chemins, par contre celui de ϕ^4 donnera le dernier chemin

$$\phi^6 =< s_{n+1,10}, U^6 = \{(s_{2,4}, s_{3,4})\} \cup U^4 >$$

Dans tous les cas, l'union des ensembles d'arcs consiste en pratique à partager les sous-ensembles.

Remarquons que les arcs obtenus en remontant dans sa hiérarchie sont ordonnés en fonction de l'indice de leur sommets, de manière croissante. De plus, l'ordre est aisément maintenu lors de l'ajout de l'arc secondaire à la ligne 19 en considérant celui-ci en dernière position dans $U \cup \{(s', t)\}$.

Procédure `construire_chemins_secondaires` ($\downarrow< s_{n+1,\varpi}, U >, \uparrow T$)
Paramètre $\downarrow< s_{n+1,\varpi}, U >$: le chemin à partir duquel les chemins secondaires sont créés.
Paramètre $\uparrow T$: ensemble de chemins secondaires.

1: $t \leftarrow s_{n+1,\varpi}$
2: $U' \leftarrow U$
3: **tant que** $t \neq s_{1,0}$ **faire**
4: **si** $\exists (s', t) \in U'$ **alors**
5: $s \leftarrow s'$
6: $U' \leftarrow U' \setminus \{(s', t)\}$
7: **sinon si** `degré_entrant`(t) = 1 **alors**
8: $s \leftarrow$ `parent`(t) *- - L'unique parent de t. - -*
9: **sinon**
10: $\{s^1, s^2\}$ = `parents`(t) *- - Les deux parents de t. - -*
11: **si** $z^\lambda(s^1) + l(s^1, t) = z^\lambda(t)$ **alors**
12: $s \leftarrow s^1$ *- - s est le sommet précédent dans le chemin. - -*
13: $s' \leftarrow s^2$ *- - (s', t) est l'arc secondaire, que nous ne traversons donc pas. - -*
14: **sinon**
15: $s \leftarrow s^2$
16: $s' \leftarrow s^1$
17: **fin si**
18: **si** $U' = \emptyset$ **alors**
19: $T \leftarrow T \cup \{< s_{n+1,\varpi}, U \cup \{(s', t)\} >\}$
20: **fin si**
21: **fin si**
22: $t \leftarrow s$
23: **fin tant que**

ALG. 4.3 – Partitionnement de X_ϖ : construction des chemins secondaires.

Si nous ordonnons initialement les sommets de U', en ligne 2, selon l'ordre décroissant des indices des sommets de ses arcs (en inversant par exemple la liste récupérée dans U) le premier test en ligne 4 se fait alors en temps constant. En effet, si un arc vérifie la condition, alors il est en première position dans U. Il suffit alors d'implémenter l'affectation en ligne 6 comme le remplacement de U' par une référence vers son sous-ensemble, une opération qui se fait aussi en temps constant, et la boucle des lignes 3–21 se fait en $O(n)$. À ceci s'ajoute la complexité de $O(n)$ pour la récupération ordonnée des arcs de U ; la totalité de l'algorithme a alors une complexité temporelle de $O(n)$.

4.3.3.3 Exploration d'un triangle

Étant donnés y^r et y^s deux points décrivant un triangle $\Delta(y^r, y^s)$ à explorer, l'algorithme 4.4 construit un graphe G_{KP} associé à l'instance du problème mono-objectif obtenu avec la pondération $\lambda = (y_2^r - y_2^s, y_1^s - y_1^r)$ (ligne 3). Une borne inférieure $\underline{z}^\lambda$ est calculée initialement, en ligne 2, selon la procédure présentée dans la section 4.2.1, puis les solutions dont les images sont dans le triangle $\Delta(y^r, y^s)$ sont obtenues de manière ordonnée, aux lignes 7–22, en utilisant le *ranking* présenté par la procédure 4.2, avant d'être ajoutées dans un ensemble X^Δ. Les images des solutions trouvées hors du triangle sont conservées dans un ensemble noté O^Δ.

L'ensemble T des chemins à explorer est initialisé avec les plus longs chemins arrivant sur les sommets de la dernière couche de G_{KP}, puis le plus long chemin de T est sélectionné pour être traité (lignes 4–5).

La boucle des lignes 7–22 procède au traitement d'un chemin ϕ. Ce chemin est initialement supprimé de T, puis son image $y = z(\phi)$ est calculée. Si cette image est dans le triangle $\Delta(y^r, y^s)$ et qu'il n'existe pas de solution dont l'image domine y, alors la solution associée à ϕ est construite avec une procédure similaire à 4.1 et ajoutée à X^Δ ; puis la borne $\underline{z}^\lambda$ est mise à jour (lignes 10–14). Dans le cas où y est situé hors de $\Delta(y^r, y^s)$, il est ajouté à O^Δ, uniquement s'il n'est pas dominé par un point de cet ensemble (ligne 17).

Enfin, les chemins secondaires de ϕ sont construits par la procédure présentée dans la section 4.3.3.2 et ajoutés à T. Un plus long chemin de cet ensemble est ensuite sélectionné pour être traité (lignes 19–21).

La boucle de traitement des chemins (lignes 7–22) s'arrête sur le premier chemin ϕ tel que $z^\lambda(\phi) < \underline{z}^\lambda$. Si k est le nombre de chemins traités par cette boucle, l'algorithme s'exécute avec une complexité de $O(n^2\omega + kn)$.

4.3.3.4 Réduction de la consommation de mémoire et incrémentation des bornes

Chaque chemin traité par la ligne 19 de l'algorithme 4.4 peut ajouter jusqu'à n chemins dans T. Aussi, il est important de maintenir la taille de T à une dimension raisonnable pour éviter la saturation de la mémoire.

Plusieurs actions sont effectuées dans ce sens. Tout d'abord, lorsque la borne $\underline{z}^\lambda$ est améliorée, en ligne 14, les chemins $\phi \in T$ tels que $z^\lambda(\phi) < \underline{z}^\lambda$ deviennent inutiles ; ils ne seront jamais à nouveau considérés. Ainsi, nous supprimons ces chemins à chaque amélioration de la borne, de manière à économiser la mémoire qui leur est attribuée.

Ensuite, nous pouvons remarquer que la génération des chemins secondaires en ligne 19 peut ajouter à T des chemins ϕ tels que $z(\phi) \notin \Delta(y^r, y^s)$. Ces chemins auront leur image éventuellement ajoutée à O^Δ, mais leur traitement se limitera à en générer les chemins secondaires. En attendant ce traitement, ces

Procédure `explorer_triangle` ($\downarrow y^r, \downarrow y^s, \updownarrow X^\Delta, \uparrow O^\Delta$)
Paramètre $\downarrow y^r, \downarrow y^s$: points supportés extrêmes adjacents, définissant un triangle $\Delta(y^r, y^s)$.
Paramètre $\updownarrow X^\Delta$: les solutions dont les images sont dans $\Delta(y^r, y^s)$.
Paramètre $\uparrow O^\Delta$: les images non dominées des solutions trouvées hors de $\Delta(y^r, y^s)$.

1: $\lambda \leftarrow (y_2^r - y_2^s, y_1^s - y_1^r)$
2: $\underline{z}^\lambda \leftarrow$ `màj_borne_inf`(X^Δ)
3: `construire_graphe` $(\downarrow \lambda, \downarrow \underline{z}^\lambda, \uparrow G_{KP})$ -- $O(n^2\omega)$ --
4: $T \leftarrow \{< s_{n+1,\varpi}, \emptyset >: \varpi \in \{0, \ldots, \omega\} : s_{n+1,\varpi} \in S\}$
5: $\phi^1 \leftarrow \text{argmax}_{\phi \in T}\{z^\lambda(\phi)\}$
6: $k \leftarrow 1$
7: **tant que** $z^\lambda(\phi^k) \geq \underline{z}^\lambda$ **faire**
8: $T \leftarrow T \setminus \{\phi^k\}$
9: $y^k \leftarrow z(\phi^k)$
10: **si** $y^k \in \Delta(y^r, y^s)$ **alors**
11: **si** $\nexists x \in X^\Delta : z(x) \geq y^k$ **alors**
12: $x^k \leftarrow$ `construire_solution`(ϕ^k) -- $O(n)$ --
13: $X^\Delta \leftarrow X^\Delta \cup \{x^k\}$
14: $\underline{z}^\lambda \leftarrow$ `màj_borne_inf`(X^Δ)
15: **fin si**
16: **sinon si** $\nexists y \in O^\Delta : y \geq y^k$ **alors**
17: $O^\Delta \leftarrow O^\Delta \cup \{y^k\}$
18: **fin si**
19: `construire_chemins_secondaires`(ϕ^k, T) -- $O(n)$ --
20: $k \leftarrow k + 1$
21: $\phi^k \leftarrow \arg\max_{\phi \in T}\{z^\lambda(\phi)\}$
22: **fin tant que**

ALG. 4.4 – Exploration d'un triangle à l'aide d'un *ranking*.

chemins consomment de la mémoire. De plus, il n'est pas évident, *a priori*, que leurs chemins secondaires permettent de revenir dans le triangle pour y trouver une solution efficace.

Puisque $z(\phi)$ est connu avant l'ajout de ϕ à T, nous proposons, dans l'optique de réduire l'empreinte mémoire de la procédure, d'effectuer la vérification $z(\phi) \in \Delta(y^r, y^s)$ et l'ajout éventuel à O^Δ avant de considérer l'ajout dans T. Si $z(\phi) \notin \Delta(y^r, y^s)$, alors nous générons immédiatement les chemins secondaires de ϕ.

Ainsi, cette génération en profondeur s'arrête soit parce que la longueur des chemins devient inférieure à $\underline{z}^\lambda$, soit parce qu'elle correspond à un point $y = z(\phi)$ non dominé dans le triangle. Dans ce dernier cas, nous pouvons utiliser ce point pour augmenter la valeur de $\underline{z}^\lambda$. Nous maintenons donc un ensemble Y^Δ de points non dominés trouvés dans le triangle durant la génération des chemins secondaires. Cet ensemble est utilisé à la place de X^Δ dans le calcul de la borne, en particulier à la ligne 14, mais aussi pendant la génération des chemins.

Remarquons que nous pourrions immédiatement construire la solution associée au point y. Cependant, nous perdrions alors la garantie de l'ordre dans leur construction et par conséquent le fait que les solutions construites ne soient pas remises en question dans la suite de l'exploration.

4.4 Expérimentations numériques

Cette section présente une analyse des résultats obtenus avec l'algorithme en deux phases utilisant un *ranking*. La consommation de mémoire est analysée dans la première partie, puis les temps de résolution sont comparés avec ceux obtenus avec l'approche PSE de la section 4.2, ainsi qu'avec la programmation dynamique de Bazgan *et al.* [10], la meilleure procédure de résolution à ce jour pour $01MOKP$. L'implémentation de cet algorithme nous a été fournie par les auteurs de la méthode, que nous remercions.

Les algorithmes ont été implémentés en C++ et exécutés sur un Pentium 4 cadencé à 3,73 GHz et accompagné de 3 Go de mémoire vive. Le temps de résolution a été limité à une heure par instance.

4.4.1 Consommation de mémoire

Le point faible reconnu de la programmation dynamique est son énorme consommation de mémoire. Aussi, étant donné que l'approche *ranking* s'appuie fortement sur les équations de la programmation dynamique, c'est tout naturellement que nous nous intéressons à sa consommation.

La figure 4.7 présente, pour chaque instance, la taille du plus grand graphe obtenu parmi tous les triangles explorés et, pour chacun de ces graphes, le nombre de sommets inaccessibles depuis sa dernière couche. Ces sommets apparaissent à cause des coupes présentées dans la section 4.3.3.1. Dans la mesure où nous ne les supprimons pas, il est intéressant de se demander quelle part de la consommation de mémoire leur est due. La figure 4.8 présente le pourcentage de ces sommets inaccessibles par rapport à la taille du graphe, ainsi que la consommation de mémoire en méga-octets. Un sommet pèse 40 octets dans notre implémentation.

Avec un maximum autour de 10 méga-octets, la quantité de mémoire nécessaire pour représenter le graphe est plus qu'acceptable. Nous pouvons remarquer que les sommets inaccessibles sont présents dans une proportion d'autant plus faible que le graphe est grand, avec un taux à 20% pour le plus grand. Dans ces conditions, et sachant que ces sommets sont implicitement ignorés par la procédure de *ranking*, il nous semble contre productif d'utiliser du temps processeur pour les supprimer.

L'autre élément important de la consommation de mémoire est l'ensemble T des chemins en attente de traitement. La figure 4.9 présente quelques statistiques au sujet des chemins générés durant le parcours

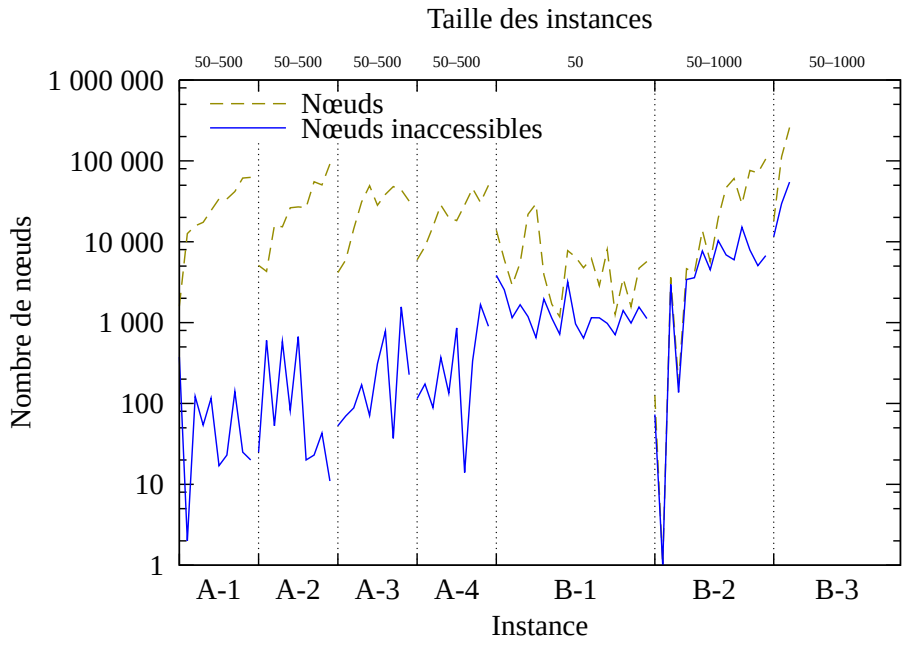

Figure 4.7 – Taille du plus grand graphe obtenu parmi tous les triangles explorés, en termes de nœuds, pour chaque instance des groupes A et B. Le nombre de nœuds non reliés à un sommet de la dernière couche, « inaccessibles », est aussi indiqué.

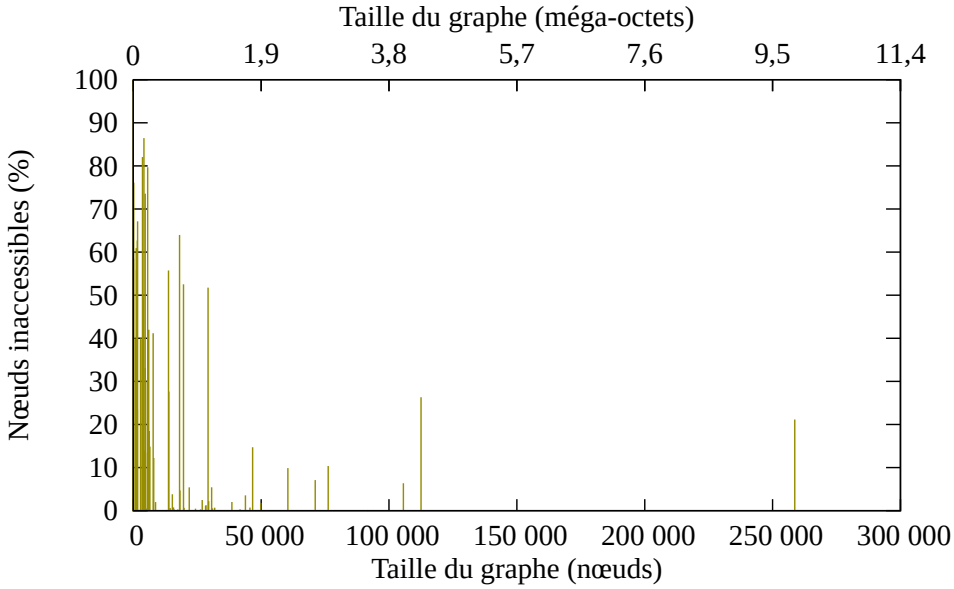

Figure 4.8 – Pourcentage de l'espace mémoire utilisé par les nœuds inaccessibles, par rapport à l'espace utilisé par le graphe entier, pour les instances des groupes A et B. L'axe des abscisses situé en haut de l'image indique la taille du graphe en méga-octets. Seul le plus grand graphe obtenu sur chaque instance est considéré.

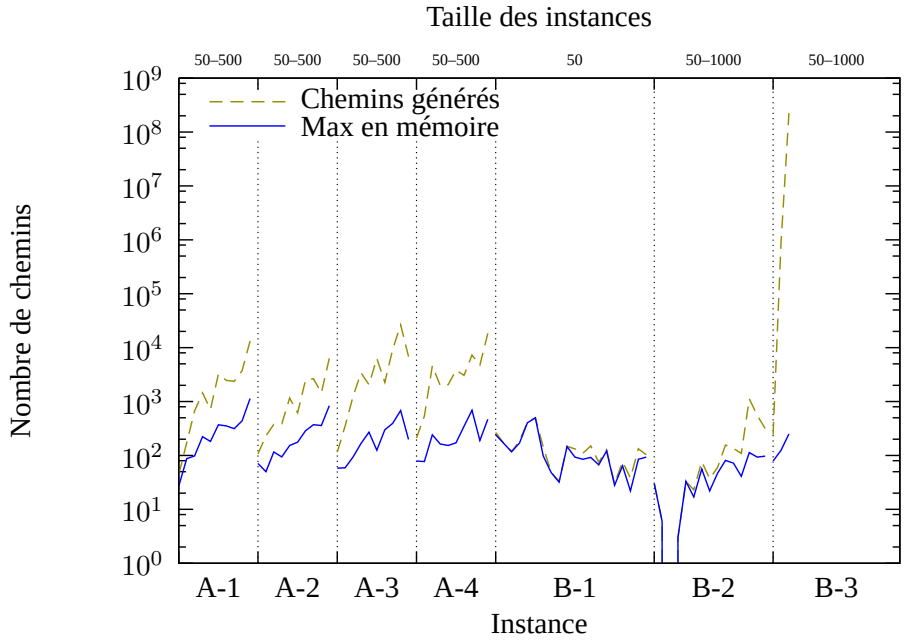

Figure 4.9 – Pour le plus grand graphe de chaque instance, le nombre total de chemins construits (courbe « chemins générés ») et le plus grand nombre de chemins simultanément en mémoire (courbe « max en mémoire »).

du graphe. Sur celle-ci, la courbe « chemins générés » indique le nombre total de chemins construits, y compris ceux non ajoutés à T car amenant à une solution hors du triangle, et pour lesquels les chemins secondaires sont immédiatement construits (section 4.3.3.4). La courbe « max en mémoire » représente quant à elle le nombre maximum de chemins en mémoire durant l'exploration du triangle, y compris pendant la génération des chemins secondaires (c'est-à-dire des chemins ne faisant pas partie de T). Nous pouvons remarquer que cette courbe est très en dessous du nombre de chemins traités, avec un maximum proche du millier pour environ 500 variables.

4.4.2 Comparaison des approches PSE et *ranking*

La figure 4.10 présente le temps nécessaire pour calculer X_{E_M} sur les instances des groupes A et B avec deux algorithmes en deux phases, utilisant respectivement une approche PSE et une approche *ranking*. Le calcul des solutions supportées pendant la première phase de cette dernière utilise une implémentation de la programmation dynamique présentée par Martello et Toth dans [92].

Le *ranking* est clairement la solution la plus performante en général, et en particulier pour les instances du groupe A. Sur ce groupe, notre algorithme est 7 fois plus rapide en moyenne, environ 100 fois au mieux. De plus, il est au moins 10 fois plus rapide dès 200 variables pour chacun de ses sous-groupes.

Concernant les instances du groupe B, nous pouvons observer que les deux approches sont équivalentes en terme de temps processeur, le *ranking* prend toutefois un avantage quand la taille de l'instance augmente.

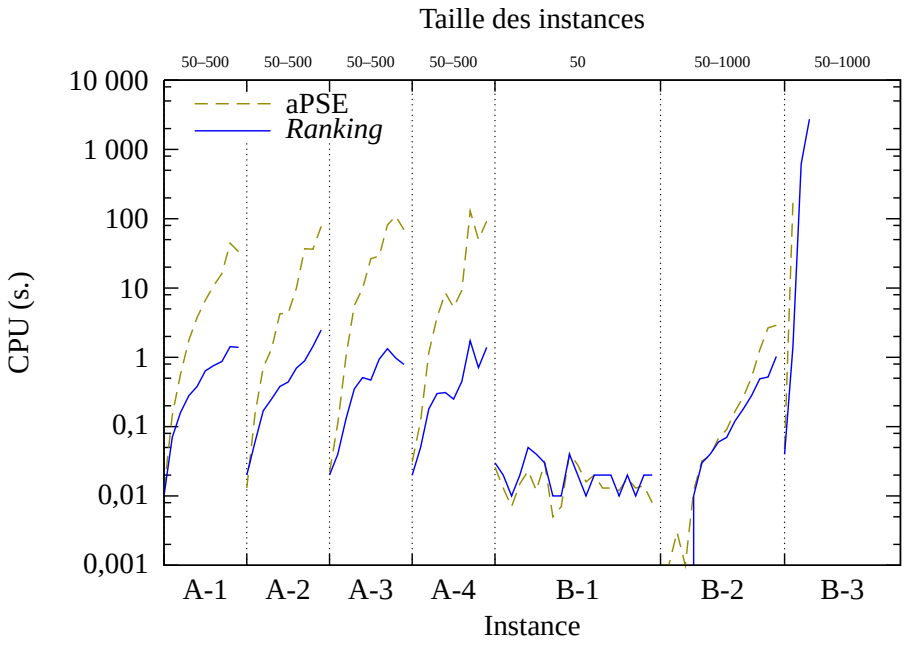

Figure 4.10 – Temps nécessaires pour résoudre les instances des groupes A et B avec deux algorithmes en deux phases, utilisant respectivement une approche PSE et une approche *ranking*.

Enfin, le résultat est mitigé pour les instances du groupe C. Notre approche résout deux instances de plus que la version PSE, mais les temps augmentent rapidement et le *ranking* dépasse le délai de l'heure dès 150 variables.

4.4.3 Comparaison entre la deux phases avec *ranking* et la programmation dynamique

L'algorithme de programmation dynamique de Bazgan *et al.* [10] a été présenté dans la section 2.4.1.3 page 47. Cette section compare les performances de leur algorithme avec la deux phases utilisant un *ranking*. Nous rappelons que la procédure de programmation dynamique calcule l'ensemble des point non dominés Y_N.

Les figures 4.11 et 4.12 présentent le temps nécessaire pour résoudre les instances des groupes A et B, pour la première, et celles du groupe C pour la seconde. Pour ce dernier groupe, nous rappelons que dix instances ont été générées pour chaque taille. La courbe présente le temps minimal, maximal et moyen obtenu pour chaque taille. L'algorithme de *ranking* apparaît comme plus efficace que la programmation dynamique en général, sauf pour le sous-groupe B-3. Nous pouvons remarquer sur la figure 4.12 que le comportement de la programmation dynamique est très homogène et que le rapport entre les temps minimaux, maximaux et moyens reste constant avec l'augmentation de la taille des instances. Pour l'algorithme de *ranking*, par contre, ce rapport est plus irrégulier lorsque la taille de l'instance varie.

Pour certaines tailles d'instances, le temps maximum de la programmation dynamique n'est pas indiqué car celle-ci a dépassé le délai de l'heure pour au moins une instance parmi les dix. Dans ce cas, le temps moyen est calculé selon les temps obtenus sur les instances résolues dans ce délai. Le nombre de ces instances est alors indiqué sur le graphique.

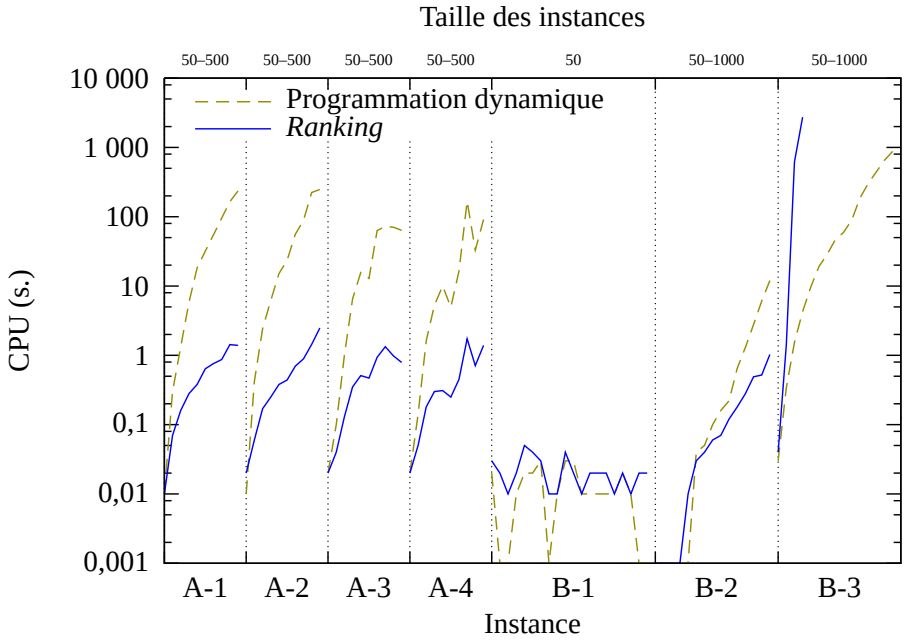

Figure 4.11 – Comparaison des temps de résolution des instances des groupes A et B obtenus avec la programmation dynamique de [10] et avec la procédure en deux phases utilisant un *ranking*.

Il est étonnant de constater que l'algorithme en deux phases avec *ranking* éprouve de grandes difficultés à résoudre les instances du groupe B-3 alors qu'il résout sans problèmes celles des autres groupes, même les instances fortement corrélées des groupes C-3 et C-4.

Nous avons observé expérimentalement que le temps de résolution des instances du groupe B-3 est principalement consommé par la génération des chemins secondaires, en particulier dans les derniers triangles, où les solutions ont une grande valeur sur z_1 et une petite sur z_2. Nous avons remarqué que ces triangles sont vides de solutions et que les quelques solutions efficaces qu'ils contiennent ont leur image proche de ses bords. Par contre, de nombreuses solutions réalisables mais non efficaces ont leur image entre les triangles. La figure 4.13 illustre la répartition de ces solutions dans l'espace des objectifs pour l'instance de taille 150.

Une conséquence du vide des triangles est que la borne $\underline{z}^{\lambda}$, utilisée pour stopper la procédure, est très peu incrémentée. L'exploration s'effectue donc quasiment sur la totalité du triangle. Or, nous avons relevé dans la section 4.3.2.1 que le *ranking* allait nécessairement explorer des points situés hors du triangle. Ainsi, pour ces instances, l'exploration de ces nombreux points représente quasiment la totalité du coût de la résolution.

En pratique, pour les instances à 100 et 150 variables, le nombre de chemins générés pendant l'exploration du dernier triangle est respectivement de 184 119 et de 27 090 799 ; pour 41 935 et 38 664 points dans l'espace des objectifs.

Dans l'état actuel de nos connaissances, la construction de ces chemins ne peut être évitée. En effet, nous ne savons pas, lors de la génération d'un chemin, si la suite des chemins secondaires qui en seront dérivés aboutira ou non à une nouvelle solution efficace. Pour cette même raison, nous ne pouvons ignorer les chemins ayant les même performances. Or, le nombre de chemins équivalents est immense pour ces instances.

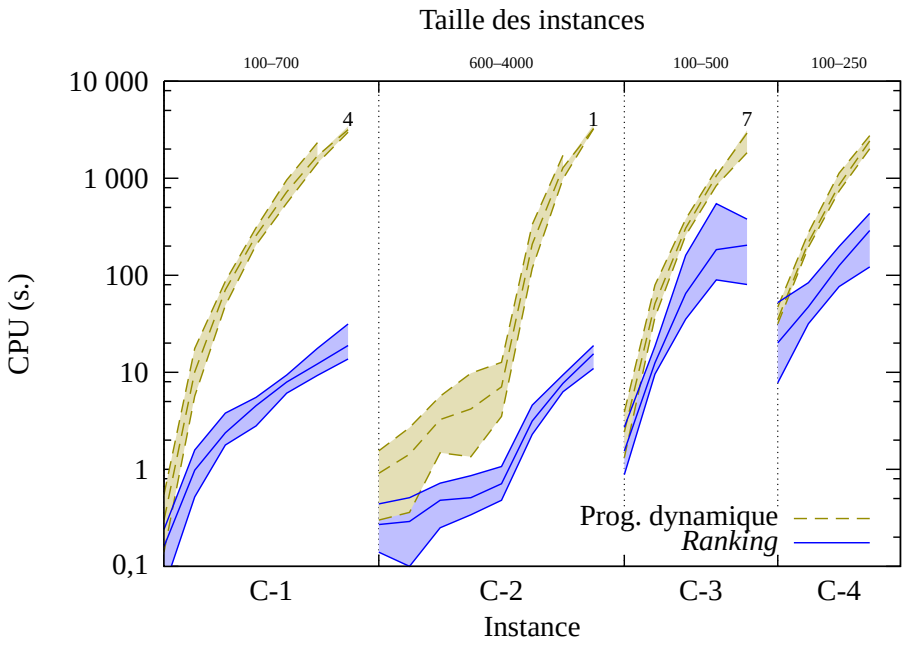

Figure 4.12 – Comparaison des temps de résolution minimaux, moyens et maximaux pour les instances du groupe C, obtenus avec la programmation dynamique de [10] et avec la procédure en deux phases utilisant un *ranking*. La cassure dans le groupe C-2 est due au changement du pas dans la taille des instances, passant de 100 à 1000 variables d'écart. Le nombre d'instances résolues est indiqué lorsque certaines instances n'ont pas été résolues dans le délai d'une heure, pour une taille donnée.

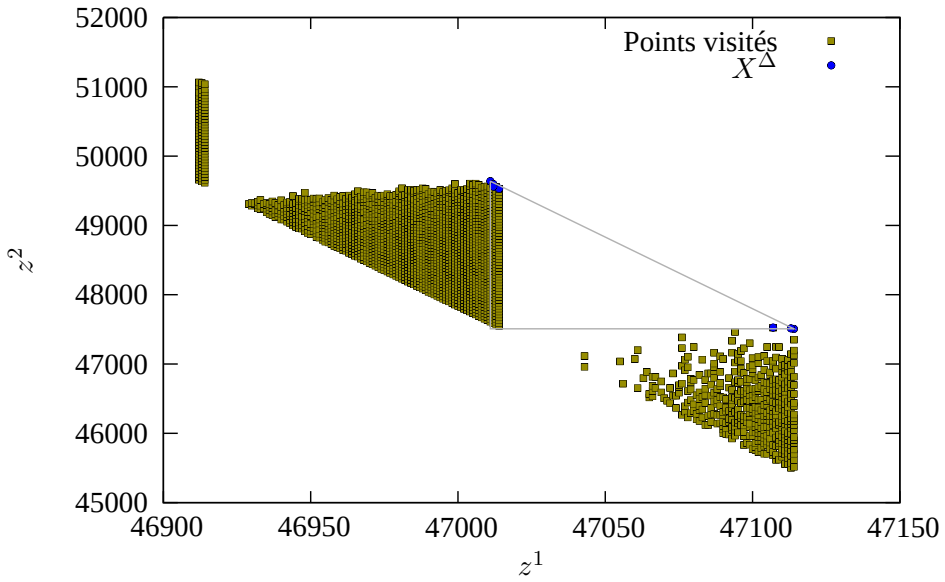

Figure 4.13 – Le temps de résolution des instances du groupe B-3 est principalement consommé par la construction des chemins correspondant à des solutions dont les images sont entre les triangles. Pour le dernier de l'instance à 150 variables, représenté ici, 27 090 799 chemins sont construits pour ces solutions, correspondant à 38 664 points dans l'espace des objectifs (« Points visités »). Le triangle est quasiment vide de solutions et celles qu'il contient sont si près des bords que $\underline{z}^\lambda$ ne peut s'éloigner de l'angle droit du triangle.

Une solution à explorer pour améliorer les performances de l'algorithme sur ces instances consisterait à appliquer le *ranking* sur plusieurs triangles adjacents simultanément, sous réserve que leurs hypoténuses soient parallèles ou presque, dans une mesure à déterminer. En effet, nous pouvons constater sur la figure 4.13 que le triangle adjacent à celui considéré dans la résolution est aussi quasiment totalement exploré. Ainsi, en appliquant la procédure sur les deux triangles simultanément, les nombreux points situés entre eux ne seront visités qu'une fois pour les deux, plutôt qu'une fois pour chaque.

4.5 Conclusion

Dans ce chapitre, nous avons présenté des améliorations sur les bornes utilisées pour l'exploration d'un triangle dans l'algorithme en deux phases de référence pour $2-01KP$, tel que décrit par Visée *et al.* [136]. Nous avons ensuite proposé une nouvelle procédure en deux phases pour ce problème, utilisant un *ranking* pour l'exploration des triangles.

Les expérimentations numériques montrent une claire amélioration de la procédure en deux phases utilisant des PSE lorsque les bornes améliorées sont utilisées. Quant à l'approche *ranking*, les expérimentations indiquent qu'elle est la plus performante comparée à l'algorithme utilisant des PSE et comparée à la programmation dynamique de Bazgan *et al.*, en dehors d'une classe d'instances particulières où cette dernière semble être à privilégier. Par une observation du comportement du *ranking* sur ces instances, nous avons extrait ce qui pourrait être le point faible de cette approche.

Finalement, notre procédure est restreinte au cas bi-objectif, en particulier par la première phase, qui utilise le schéma dichotomique d'Aneja et Nair [3], mais aussi du fait de la description de la zone de recherche dans la seconde phase, basée sur une représentation par triangles et s'appuyant par conséquent sur l'ordre implicite des solutions dans l'espace des objectifs. Il est nécessaire, pour étendre notre procédure à plus de deux objectifs, d'utiliser un autre algorithme pour le calcul des solutions supportées et utiliser une autre description de la zone de recherche, par exemple telle que décrite dans [111].

CHAPITRE 5

Algorithme en deux phases pour le problème de sac à dos multi-objectif

Bien que générales dans le principe, les procédures en deux phases ont été reistreintes pendant plus d'une décénie aux problèmes bi-objectifs. En général, ces procédure prennent en compte des particularités de ce cas, à savoir l'ordre implicite des points non dominés pour le calcul des solutions supportées, dans la première phase, et la description de l'espace de recherche par des triangles, dans la seconde (voir [113, 134, 136], entre autres). Ces particularités impliquent que ces procédures ne peuvent être immédiatement utilisées pour des problèmes à trois objectifs où plus. Cependant, Przybylski [111] a récemment proposé une procédure en deux phases pour des problèmes ayant plus de deux objectifs, avec une application au problème d'affectation.

Ce chapitre présente une contribution originale pour le problème de sac à dos, inspirée des travaux de Przybylski [111]. Nous y décrivons une adaptation de sa méthode dans le cas d'une maximisation, avec une application au problème de sac à dos multi-objectif. La première section décrit le déroulement de la première phase, durant laquelle les solutions supportées sont déterminées. Une description de l'espace de recherche pour le calcul des solutions non supportées est effectuée dans la deuxième section, puis nous présentons l'algorithme de calcul des solutions non supportées utilisant l'algorithme de *ranking* du chapitre précédent. Les résultats d'expérimentations numériques sont présentés dans la dernière section, où notre procédure est comparées avec l'algorithme de programmation dynamique de Bazgan *et al.* [10].

5.1 Calcul des solutions supportées

Les solutions supportées sont, par définition, optimales pour au moins un problème P_λ, avec $\lambda \in \mathbb{R}^p_>$ (voir la section 1.2.2.2 page 16). Toute la difficulté consiste à déterminer ces poids $\lambda \in \mathbb{R}^p_>$.

Si $\bar{Y}$ est un ensemble de points supportés connus à une étape de la résolution, nous allons essayer de déterminer les solutions supportées restantes d'une manière similaire au cas bi-objectif. Un problème P_λ est construit en fonction de plusieurs points $\{y^1, \ldots, y^q\} \in \bar{Y}$ potentiellement adjacents, de telle manière que, si y^* est le point correspondant à une solution optimale de P_λ, nous ayons :

- soit $\exists i \in \{1, \ldots, q\}$ tel que $y^* = y^i$, auquel cas les points $\{y^1, \ldots, y^q\}$ sont effectivement adjacents et décrivent une facette de $\mathrm{conv}(Y)$;
- ou alors y^* est un nouveau point supporté, auquel cas il est ajouté à $\bar{Y}$ avant de répéter la procédure.

Dans le cas bi-objectif, nous avons $q = 2$ et deux points y, y' supportés sont dits adjacents, avec $y_1 < y'_1$, s'il n'existe pas d'autre point supporté y'' tel que $y_1 < y''_1 < y'_1$.

Une généralisation immédiate de la méthode de Aneja et Nair [3], utilisée pour le cas bi-objectif, consisterait à choisir $\{y^1, \dots, y^q\}$ de manière à décrire une facette de $\mathrm{conv}(\bar{Y})$ et de calculer ensuite une pondération normale à cette facette. Cependant, il n'est pas exclu qu'un tel poids ait une composante négative, comme illustré par l'exemple 5.18. Dans une telle situation, l'utilisation de ce poids ne garantit pas que la solution trouvée par la résolution de P_λ soit efficace, d'après le théorème 1.1 page 16.

Exemple 5.18 (Une difficulté de la généralisation de la méthode de Aneja et Nair). *Soit l'instance du problème de sac à dos tri-objectif ci-dessous :*

$$\left.\begin{array}{lrcl} \max & 23x_1 + 8x_2 + 24x_3 + 23x_4 + 15x_5 + 20x_6 & & \\ & 9x_1 + 4x_2 + 7x_3 + 28x_4 + 8x_5 + 8x_6 & & \\ & 8x_1 + 29x_2 + 6x_3 + 22x_4 + 11x_5 + 10x_6 & & \\ \text{s.c} & 1x_1 + 1x_2 + 1x_3 + 1x_4 + 1x_5 + 1x_6 & \leq & 2 \end{array}\right\} 3-01KP$$

Essayons d'appliquer la dichotomie sur cette instance. À l'initialisation, nous obtenons les solutions lexicographiquement optimales :

- $x^1 = (0,0,1,1,0,0)$*, dont la performance est* $y^1 = (47,35,28)$*.* y^1 *est le seul point qui maximise le premier objectif ;*
- $x^2 = (1,0,0,1,0,0)$*, dont la performance est* $y^2 = (46,37,30)$*.* y^2 *est le seul point qui maximise le deuxième objectif ;*
- $x^3 = (0,1,0,1,0,0)$*, dont la performance est* $y^3 = (31,32,51)$*.* y^3 *est le seul point qui maximise le troisième objectif.*

Il n'y a qu'un seul choix possible pour le poids initial. En calculant la normale au plan défini par y^1, y^2 *et* y^3*, nous obtenons le poids* $\lambda = (52,-9,35)$ *ou* $\lambda = (-52,9,-35)$*. Dans les deux cas, au moins une de ses composantes est négative, l'hypothèse* $\lambda \in \mathbb{R}^p_>$ *du théorème 1.1 n'est donc pas vérifiée et la résolution du problème mono-objectif ne garantit donc pas l'obtention d'une solution efficace.*

Si nous essayons néanmoins de calculer les solutions des sommes pondérées obtenues avec ces poids, nous trouvons que :

- *les trois solutions optimales pour le problème* $P_{(52,-9,35)}$ *sont* x^1, x^2 *et* x^3 *;*
- *l'unique solution optimale pour* $P_{(-52,9,-35)}$ *est la solution vide* $(0,0,0,0,0,0)$*.*

Aucun de ces poids ne permet donc de trouver une nouvelle solution optimale. Pourtant, la solution $x^4 = (0,0,0,1,1,0)$ *dont le point correspondant est* $y^4 = (38,36,37)$ *est supportée, nous pouvons en effet l'obtenir par résolution du problème* $P_{(1,14,4)}$*.*

Ainsi, la détermination des poids λ pour le calcul des solutions supportées n'est pas triviale. La suite de cette section décrit en détail le choix des pondérations et le calcul de ces solutions. L'initialisation de $\bar{Y}$ par le calcul des solutions lexicographiquement optimales est présenté ensuite.

5.1.1 Espace des poids

L'idée principale du calcul des solutions supportées consiste en un partitionnement de l'ensemble des poids $\lambda \in \mathbb{R}^p_>$. Nous savons que pour chacun d'eux, la résolution de P_λ donne un ensemble de points supportés et que chaque point supporté peut être obtenu avec au moins une de ces pondérations. Ainsi, à tout $y \in Y_{SN}$, nous pouvons associer un ensemble de poids, nommé $W^0(y)$, tel que $\forall \lambda \in W^0(y)$, y

corresponde à l'image d'une solution optimale de P_λ. Le calcul des solutions supportées revient alors à déterminer les ensembles $W^0(y)$, ce que nous le décrivons dans la suite de cette section.

L'espace des poids, dans lequel seront choisies les pondérations pour construire les problèmes P_λ, est défini comme suit :

Définition 5.1 (Espace des poids)**.** *Soit W^0 l'ensemble défini par :*

$$W^0 = \left\{ \lambda \in \mathbb{R}^p_> : \lambda_p = 1 - \sum_{i=1}^{p-1} \lambda_i \right\}$$

La propriété ci-dessous indique que l'utilisation de W^0 pour choisir des poids λ, afin de définir des problèmes P_λ, est suffisante pour obtenir toutes les solutions supportées d'un MOP. Remarquons que W^0 est un polytope de dimension $p-1$ et, en particulier, l'égalité $W^0 =]0,1[$ est vérifiée dans le cas bi-objectif.

Propriété 5.1 (Ensemble minimal des poids [111])**.**

(i) Pour tout $\lambda \in \mathbb{R}^p_>$, il existe $\alpha \in \mathbb{R}^*, \lambda' \in W^0$ tels que $\lambda = \alpha\lambda'$;

(ii) $\nexists \lambda, \lambda' \in W^0, \alpha \in \mathbb{R}^* \setminus \{1\}$ tel que $\lambda = \alpha\lambda'$.

Les deux points ci-dessus réunis impliquent que W^0 est une description minimale de l'ensemble des poids.

Nous pouvons remarquer que les composantes de $\lambda \in W^0$ sont toutes strictement positives. Ceci a pour effet de garantir qu'une solution optimale pour P_λ sera efficace et non pas faiblement efficace (voir le théorème 1.1 page 16).

Définition 5.2 (Ensemble des poids propres à un point)**.** *Pour tout point supporté y, nous posons :*

$$W^0(y) = \left\{ \lambda \in W^0 : \lambda \cdot y = \max_{y' \in Y_{SN}} \{\lambda \cdot y'\} \right\}$$

$W^0(y)$ est l'ensemble des poids pour lesquels y est optimal pour P_λ.

Si y^1 et y^2 sont deux points supportés extrêmes, voisins dans une facette de $\mathrm{conv}(Y_{SN})$, alors $W^0(y^1)$ et $W^0(y^2)$ sont séparés par un hyperplan ($\dim(W^0(y^1) \cap W^0(y^2)) = \dim(W^0(y^1)) - 1$). De plus, si y est un point supporté non extrême, alors $W^0(y)$ est l'intersection des $W^0(y')$ pour tout $y' \in Y_{SN1}$ décrivant la facette de $\mathrm{conv}(Y_{SN})$ contenant y. Przybylski [111] énonce les propriétés suivantes.

Propriété 5.2 (Propriétés de $W^0(y)$)**.** *Soit y un point supporté. Nous avons les trois propriétés ci-dessous :*

(i) $W^0(y)$ est un polytope non vide ;

(ii) y est un point supporté extrême si et seulement si $\dim(W^0(y)) = p - 1$;

(iii) soit y' tel que $W^0(y) \cap W^0(y') \neq \emptyset$, alors $W^0(y) \cap W^0(y')$ est la face commune de dimension maximale de $W^0(y)$ et $W^0(y')$.

Le premier point ci-dessous peut sembler évident. En effet, si y est un point supporté, alors la propriété 5.1 nous indique qu'il existe au moins une pondération $\lambda \in W^0$ pour laquelle y correspond à une solution optimale de P_λ. Pour le second point, nous pouvons intuitivement visualiser la situation dans le cas tri-objectif par un plan posé sur $\mathrm{conv}(Y)$, sur un point supporté y. Son inclinaison représente un poids de $W^0(y)$ et nous pouvons l'orienter selon $2\ (= p-1)$ axes sans rencontrer d'autre point supporté si y est extrême. Par contre, si y est dans une facette de $\mathrm{conv}(Y)$, alors le plan posé dessus passe nécessairement par les autres sommets de la facette. Nous avons alors $|W^0(y)| = 1$ et donc $\dim(W^0(y)) = 0$. Le plan n'a aucun axe de liberté. Enfin, si y est sur une arête de $\mathrm{conv}(Y)$, alors le plan repose sur les sommets définissant l'arête. Il peut être incliné autour de l'axe passant par ces sommets, ce qui correspond à un seul axe de liberté et $\dim(W^0(y)) = 1$ dans ce cas.

Ce dernier cas permet d'illustrer aussi le troisième point. On imagine ainsi que si le plan peut être incliné autour de deux points y et y', alors les poids correspondant à ces inclinaisons sont situés à la frontière entre $W^0(y)$ et $W^0(y')$ et couvrent l'intégralité de cette frontière.

L'exemple ci-dessous illustre les ensembles de poids propres pour un ensemble de points supportés, ainsi que les remarques énoncées ci-dessus.

Exemple 5.19 (Points supportés et espace des poids). *Nous considérons ici la situation d'un MOP tri-objectif où $Y_{SN} = \{y^1, \dots, y^6\}$, tel que*
- *$y^1 = (10, 1, 1)$ est l'image des solutions lexicographiquement optimales selon (1, 2, 3) et (1, 3, 2) ;*
- *$y^2 = (4, 9, 3)$ est l'image des solutions lexicographiquement optimales selon (2, 1, 3) et (2, 3, 1) ;*
- *$y^3 = (2, 6, 9)$ est l'image de la solution lexicographiquement optimales selon (3, 2, 1) ;*
- *$y^4 = (6, 3, 9)$ est l'image de la solution lexicographiquement optimales selon (3, 1, 2) ;*
- *$y^5 = (7, 5, 2)$ est situé sur l'arête de $\mathrm{conv}(Y_{SN})$ reliant y^1 à y^2 ;*
- *$y^6 = (4, 6, 7)$ est situé sur la facette de $\mathrm{conv}(Y_{SN})$ délimitée par y^2, y^3 et y^4.*

La figure 5.1 illustre la position de ces points dans l'espace des objectifs ainsi que les facettes de $\mathrm{conv}(Y_{SN})$ qu'ils délimitent.

Nous nous intéressons à la facette délimitée par les points y^2, y^3 et y^4. Nous pouvons remarquer que plusieurs pondérations permettent de trouver ces points en résolvant un problème P_λ. Il n'en existe par contre qu'une seule permettant de trouver y^6. Celle-ci est un vecteur perpendiculaire à la facette et est l'unique pondération à l'intersection des ensemble des poids permettant de trouver y^2, y^3 ou y^4.

Nous allons maintenant déterminer les ensembles $W^0(y)$. Pour cela, nous commençons par décrire l'ensemble $W^0(y^1)$ selon les autres points supportés :

$$W^0(y^1) = \{\lambda \in W^0 : \lambda \cdot y^1 \geq \lambda \cdot y^j : j \in \{2, \dots, 6\}\} \tag{5.1}$$

ou, de manière équivalente, sachant que $\lambda_3 = 1 - \lambda_1 - \lambda_2$,

$$W^0(y^1) = \left\{ \lambda \in W^0 : \begin{array}{l} 8\lambda_1 - 6\lambda_2 \geq 2 \\ 16\lambda_1 + 3\lambda_2 \geq 8 \\ 12\lambda_1 + 6\lambda_2 \geq 8 \\ 4\lambda_1 - 3\lambda_2 \geq 1 \\ 12\lambda_1 + \lambda_2 \geq 3 \\ \lambda_1 + \lambda_2 \leq 1 \\ \lambda_1 \geq 0 \\ \lambda_2 \geq 0 \end{array} \right\} \tag{5.2}$$

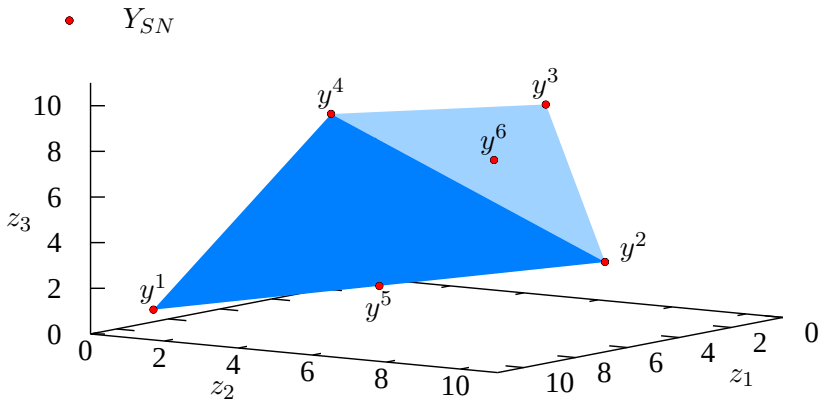

Figure 5.1 – Représentation, dans l'espace des objectifs, des six points supportés de l'exemple 5.19 et des faces de l'enveloppe convexe qu'ils décrivent.

Certaines inégalités sont redondantes et peuvent être supprimées. Le système se réduit alors à

$$W^0(y^1) = \left\{ \lambda \in W^0 : \begin{array}{l} 8\lambda_1 - 6\lambda_2 \geq 2 \\ 12\lambda_1 + 6\lambda_2 \geq 8 \\ \lambda_2 \geq 0 \\ \lambda_1 + \lambda_2 \leq 1 \end{array} \right\} \tag{5.3}$$

La figure 5.2 illustre l'ensemble $W^0(y^1)$. En appliquant le même procédé pour chacun des autres points supportés, nous obtenons finalement le partitionnement de W^0 illustré par la figure 5.3.
En imaginant un plan posé sur un point supporté, sur $\mathrm{conv}(Y)$, et en observant les inclinaisons qu'il peut prendre avant de toucher un autre point, nous pouvons retrouver les éléments du partitionnement de W^0. Pour un point supporté extrême y (de y^1 à y^4), nous avons $\dim(W^0(y)) = 2 = p - 1$. Le plan posé sur ce point peut être librement incliné selon deux axes. Nous pouvons observer sur la figure 5.3 que les partitions de W^0 pour ces points se représentent par des polygones dans un plan.
Si le plan repose sur deux points supportés extrêmes y et y', et s'il peut être incliné autour d'eux sans toucher un autre point extrême, alors les poids correspondants à ces inclinaisons sont ceux de $W^0(y) \cap W^0(y')$. Ils sont représentés par un segment sur la figure 5.3. Le segment séparant $W^0(y^1)$ et $W^0(y^2)$ est particulièrement intéressant. En effet, y^5 est situé sur l'arête définie par y^1 et y^2 et ce segment représente aussi $W^0(y^5)$.
Enfin, y^6 est aussi intéressant pour des raisons identiques. Étant situé dans une facette de $\mathrm{conv}(Y)$, si nous posons le plan dessus, alors il repose nécessairement sur les sommets définissant la facette. Nous ne pouvons l'incliner d'aucune autre manière et nous avons $\dim(W^0(y^6)) = 0$. $W^0(y^6)$ est représenté par un point dans la figure 5.3.

Les dernières observations de l'exemple précédent évoquent une propriété intéressante sur l'adjacence des points supportés extrêmes. Classiquement, deux points $y^r, y^s \in Y_{SN1}$ sont dits adjacents si

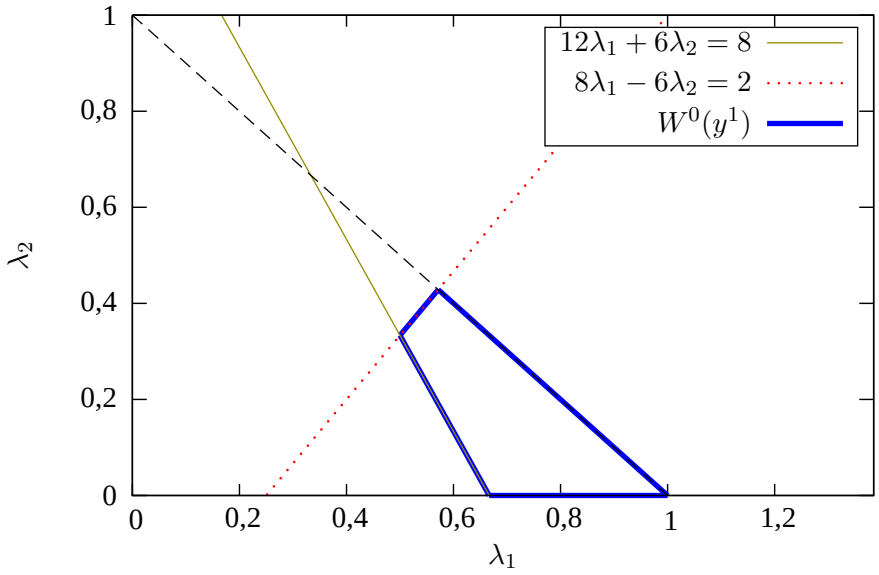

Figure 5.2 – L'ensemble $W^0(y^1)$ obtenu dans l'exemple 5.19 est délimité par le trait épais. L'ensemble des poids, W^0, est représenté par le triangle délimité par les axes et par la ligne en tirets.

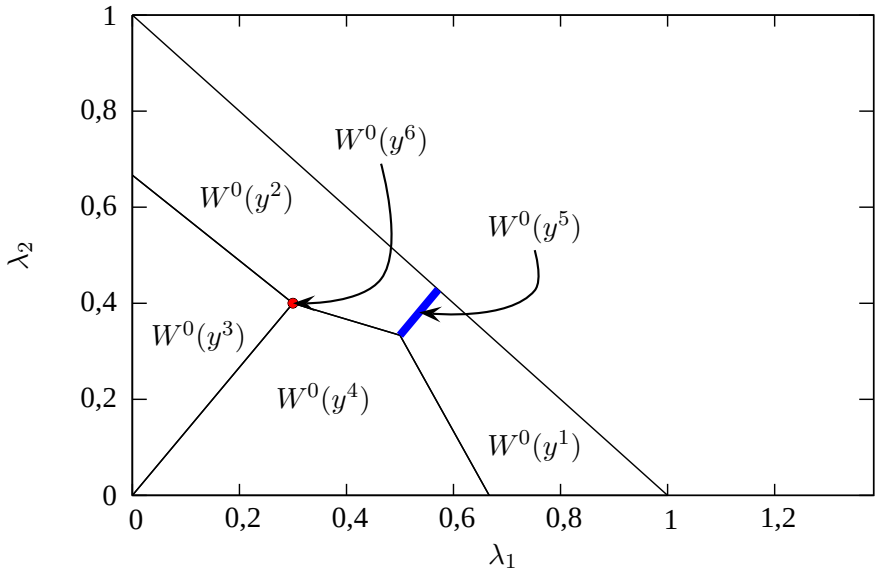

Figure 5.3 – Partitionnement de W^0 obtenu avec les points de l'exemple 5.19. Pour tout λ choisi dans une partie $W^0(y^i)$, la résolution de P_λ retournera une solution dont l'image est y^i.

ils sont consécutifs par rapport à un objectif. Du point de vue de W^0, cette adjacence se traduit effectivement par le fait que les partitions $W^0(y^r)$ et $W^0(y^s)$ ont une intersection non vide. Przybylski [111] complète cette notion d'adjacence géométrique par la définition suivante :

Définition 5.3 (Adjacence)**.** *Deux points $y^1, y^2 \in Y_{SN1}$ sont adjacents si $W^0(y^1) \cap W^0(y^2)$ est un polytope de dimension $p-2$.*

Nous pouvons remarquer que la méthode de Aneja et Nair [3], utilisée dans le cas bi-objectif pour calculer les solutions supportées, n'est, dans le fond, que le calcul d'une partition de W^0 aboutissant à l'ensemble des $W^0(y), y \in Y_{SN}$. Néanmoins, ce partitionnement est grandement simplifié du fait de l'ordre implicite des points de Y_N. L'exemple 5.20 ci-dessous illustre cette analogie.

Exemple 5.20 (Partitionnement de W^0 dans le cas bi-objectif avec la méthode de Aneja et Nair). *Pour cet exemple, nous notons $\lambda(y, y') \in \mathbb{R}^2$ le poids défini par $\lambda(y, y')_1 = \frac{y'_1 - y_1}{y_2 - y'_2 + y'_1 - y_1}$ et $\lambda(y, y')_2 = 1 - \lambda(y, y')_1$, avec $y_1 < y'_1$ et $y'_2 < y_2$.*
Soit x^r et x^s deux solutions lexicographiquement optimales respectivement selon (2, 1) et (1, 2), d'images respectives y^r et y^s. Sous réserve qu'il n'y ait pas d'autre point supporté extrême réalisable, on suppose que $W^0(y^s) =]0, \lambda(y^r, y^s)_1]$ et $W^0(y^r) = [\lambda(y^r, y^s)_1, 1[$.
Pour confirmer ou infirmer l'inexistence d'un autre point supporté réalisable entre y^r et y^s, il suffit de résoudre le problème P_λ où λ est choisi à la frontière séparant $W^0(y^r)$ et $W^0(y^s)$; soit ici $\lambda = \lambda(y^r, y^s)$. La résolution de ce problème donne une solution x^t d'image y^t. Deux situations peuvent alors se présenter :

1. *Si $\lambda \cdot y^t = \lambda \cdot y^r = \lambda \cdot y^s$, alors l'inexistence d'un point supporté extrême réalisable entre y^r et y^s est confirmée et $W^0(y^r) =]0, \lambda(y^r, y^s)_1], W^0(y^s) = [\lambda(y^r, y^s)_1, 1[$.*
2. *Autrement, y^t est un nouveau point supporté. $W^0(y)$ est mis à jour pour tout y adjacent à y^t et la procédure est appliquée à nouveau pour $\lambda(y^r, y^t)$ et $\lambda(y^t, y^s)$.*

À l'issue de la procédure, l'ensemble des $W^0(y), y \in Y_{SN1}$, est obtenu. Si on note $Y_{SN1} = \{y^1, \ldots, y^m\}$, avec $y^i_1 < y^{i+1}_1, \forall i \in \{1, \ldots, m-1\}$, alors
– $W^0(y^1) = \left]0, \lambda(y^1, y^2)_1\right]$;
– $W^0(y^m) = \left[\lambda(y^{m-1}, y^m)_1, 1\right[$;
– $W^0(y^i) = \left[\lambda(y^{i-1}, y^i)_1, \lambda(y^i, y^{i+1})_1\right], \forall i \in \{2, \ldots, m-1\}$;

5.1.2 Procédure de calcul des solution supportées

La procédure de calcul des solutions supportées consiste à déterminer $W^0(y)$ pour tout $y \in Y_{SN}$. Bien sûr, une méthode de résolution n'a pas une connaissance *a priori* des points supportés. W^0 devra donc être partitionné itérativement, au fur et à mesure que de nouveaux points seront trouvés.

5.1.2.1 Description générale

Soit $\bar{Y} \subseteq Y_{SN}$ un ensemble de points supportés. Nous supposons pour l'instant que $\bar{Y}$ est initialisé avec les points lexicographiquement optimaux. Nous noterons

$$W^0_p(y) = \left\{ \lambda \in W^0 : \lambda \cdot y = \max_{y' \in \bar{Y}} \{\lambda \cdot y'\} \right\}$$

l'ensemble potentiel des poids propres à y. Les facettes de $W_p^0(y)$ sont potentiellement des facettes de $W^0(y)$. L'algorithme va les explorer une à une pour confirmer ou infirmer cette hypothèse. Dans ce dernier cas, $W_p^0(y)$ sera ajusté et de nouvelles facettes pourront être considérées.

Pour chaque point $y \in \bar{Y}$, nous définissons l'ensemble $A(y)$ des points adjacents à y et l'ensemble $P(y)$ des points potentiellement adjacents à y. Nous avons donc :

$$W_p^0(y) = \left\{\lambda \in W^0 : \forall y' \in A(y) \cup P(y), \lambda \cdot y \geq \lambda \cdot y'\right\}$$

De plus, nous nous donnons un ordre de traitement des points de $\bar{Y}$, par exemple, l'ordre de découverte de ces points. Nous notons alors $\bar{Y} = \{y^1, \ldots, y^q\}$ et nous initialisons :

$$A(y^i) = \emptyset, \forall i \in \{1, \ldots, q\}$$
$$P(y^i) = \{y^{i+1}, \ldots, y^q\}, \forall i \in \{1, \ldots, q\}$$

Généralement, nous aurons toujours, pour y^i et y^j potentiellement adjacents avec $i < j : y^j \in P(y^i)$ et $y^i \notin P(y^j)$. Ce choix est justifié par le fait que nous considérons les points supportés un par un. Donc, si deux points supportés sont adjacents, nous en aurons la confirmation en considérant le premier par rapport à l'ordre dans $\bar{Y}$. Sinon, ces deux points ne sont pas adjacents. Dans les deux cas il ne sera pas nécessaire de considérer le premier dans le traitement du second.

Si nous obtenons l'information que deux points sont potentiellement adjacents alors que nous savons déjà qu'ils sont adjacents, aucune actualisation des ensembles $P(.)$ ne sera nécessaire. De manière générale, si $y' \in A(y)$, alors $y' \notin P(y)$.

Nous considérons ensuite itérativement les points $y \in \bar{Y}$. Pour chacun, nous déterminons $W_p^0(y)$ en fonction de $A(y)$ et $P(y)$. Il est possible que les contraintes définies par certains points de $P(y)$ soient redondantes dans la description de $W_p^0(y)$. Dans ce cas, nous supprimons ces points de $P(y)$. Tant que $P(y) \neq \emptyset$, il existe au moins une facette de $W_p^0(y)$ dont nous n'avons pas encore la confirmation qu'il s'agit également d'une facette de $W^0(y)$. Supposons que $y' \in P(y)$, nous devons explorer la facette définie par $\lambda \cdot y = \lambda \cdot y'$. Nous obtenons alors un ensemble de points supportés que nous utilisons pour mettre à jour les ensembles $\bar{Y}, A(y), P(y)$ et $W_p^0(y)$.

Nous répétons ce processus pour chaque point de $\bar{Y}$, jusqu'au dernier point de $\bar{Y}$. Nous aurons alors confirmé chaque facette de chaque $W_p^0(y)$, et nous aurons donc $W_p^0(y) = W^0(y)$, pour tout $y \in \bar{Y}$, avec $Y_{SN1} \subseteq \bar{Y}$. Cette procédure est décrite par l'algorithme 5.1. Son exécution est illustée par l'exemple 5.21. Les éléments intervenant dans les sous-routines de l'algorithme 5.1 seront exposés dans la suite de cette section.

Exemple 5.21 (Illustration du déroulement de la première phase). *Soit l'instance ci-dessous du problème tri-objectif de sac à dos.*

$$\left.\begin{array}{lll} \max & 5x_1 + 2x_2 + 8x_3 + 8x_4 + 1x_5 & \\ & 6x_1 + 2x_2 + 3x_3 + 2x_4 + 2x_5 & \\ & 4x_1 + 7x_2 + 6x_3 + 2x_4 + 8x_5 & \\ s.c & 8x_1 + 3x_2 + 7x_3 + 8x_4 + 7x_5 & \leq 16 \end{array}\right\} 3 - 01KP$$

Les points lexicographiquement optimaux obtenus initialement sont $y^1 = (16, 5, 8), y^2 = (13, 9, 10)$ *et* $y^3 = (3, 4, 15)$. *Nous initialisons* $\bar{Y} = \{y^1, y^2, y^3\}, P(y^1) = \{y^2, y^3\}, P(y^2) = \{y^3\}, P(y^3) = A(y^1) = A(y^2) = A(y^3) = \emptyset$, *puis nous procédons au traitement de* y^1.
L'ensemble $W_p^0(y^1)$ *est illustré par la figure 5.4. La seule facette qui n'est pas sur la frontière de* W^0 *est définie par* $\lambda \cdot y^1 \geq \lambda \cdot y^2$. *Appelons-la* F. *En résolvant le problème* P_λ *pour tout* $\lambda \in F$, *nous trouvons*

Procédure `phase_1` ($\downarrow P, \uparrow \bar{Y}$)
Paramètre $\downarrow P$: l'instance du problème multi-objectif de sac à dos.
Paramètre $\uparrow \bar{Y} \supseteq Y_{SN1}$: l'ensemble des points supportés extrêmes de P.

- - Calcul des points lexicographiquement optimaux $y^1, \ldots, y^r$. - -

1: $\bar{Y} \leftarrow$ `lex_opt`(P)
2: **pour tout** $i \in \{1, \ldots, r\}$ **faire**
3: $A(y^i) \leftarrow \emptyset$
4: $P(y^i) \leftarrow \{y^{i+1}, \ldots, y^r\}$
5: **fin pour**

6: $i \leftarrow 1$
7: **tant que** $i \neq |S| + 1$ **faire**
8: `calculer_poids_propres` ($\downarrow A(y^i), \downarrow P(y^i), \uparrow W_p^0(y^i)$)
9: **tant que** $P(y^i) \neq \emptyset$ **et** $\dim W_p^0(y^i) = \dim W^0$ **faire**
- - Sélection d'une facette F de $W_p^0(y^i)$ à explorer, et du point $y^* \in P(y^i)$ définissant cette facette. - -
10: `choisir_facette` ($\downarrow W_p^0(y^i), \downarrow P(y^i), \uparrow y^*, \uparrow F$)
11: $Y' \leftarrow$ `explorer_facette` ($\downarrow P, \downarrow F$)
12: `màj_adjacence` ($\downarrow Y', \downarrow F, \downarrow y^*, \updownarrow \bar{Y}, \updownarrow A(.), \updownarrow P(.)$)
13: **fin tant que**
14: $i \leftarrow i + 1$
15: **fin tant que**

ALG. 5.1 – Calcul des solutions supportées.

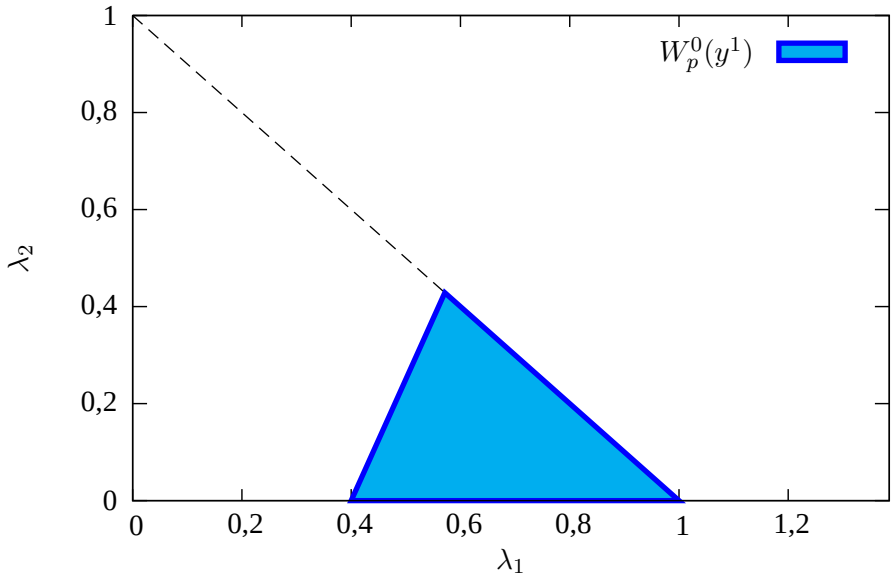

Figure 5.4 – L'ensemble $W_p^0(y^1)$ obtenu après l'initialisation.

le point supporté $y^4 = (9, 5, 14)$, ainsi que y^1 et y^2, à nouveau. Il suffit que y^1 ou y^2 soit présent, pour affirmer que y^1 et y^2 sont adjacents. Le segment de F pour lequel l'égalité $\lambda \cdot y^1 = \lambda \cdot y^2$ est vérifiée est donc une facette de $W^0(y^1)$ et de $W^0(y^2)$.
Nous mettons maintenant à jour les ensembles suivants :

$$\begin{aligned}
\bar{Y} &= \bar{Y} \cup \{y^4\},\\
A(y^1) &= \{y^2\},\\
A(y^2) &= \{y^1\},\\
P(y^1) &= P(y^1) \setminus \{y^2\} \cup \{y^4\},\\
P(y^2) &= P(y^2) \cup \{y^4\},\\
P(y^3) &= P(y^3) \cup \{y^4\},\\
P(y^4) &= A(y^4) = \emptyset.
\end{aligned}$$

Puis nous mettons à jour $W_p^0(y^1)$. Ce dernier est illustré par la figure 5.5. Nous procédons ensuite au traitement de la facette définie par $\lambda \cdot y^1 \geq \lambda \cdot y^4$. L'exploration de cette facette donne les points y^1 ou y^4. Aucun nouveau point n'est trouvé, nous confirmons juste l'adjacence entre y^1 et y^4 en mettant à jour les ensembles

$$\begin{aligned}
A(y^1) &= \{y^2, y^4\},\\
A(y^4) &= \{y^1\},\\
P(y^1) &= P(y^1) \setminus \{y^4\}
\end{aligned}$$

L'ensemble $P(y^1)$ est maintenant vide. Nous avons obtenu que $W_p^0(y^1) = W^0(y^1)$. La procédure continue avec le point suivant dans $\bar{Y}$.

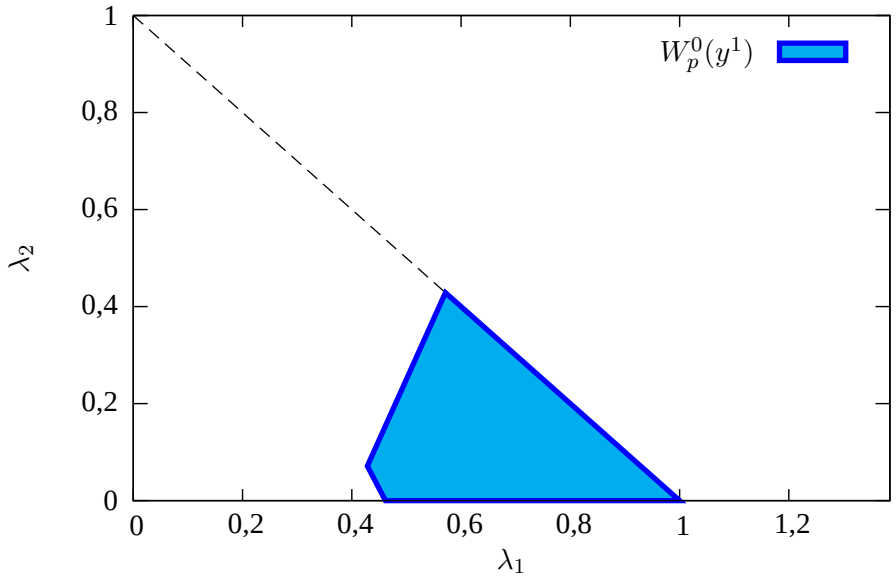

Figure 5.5 – L'ensemble $W_p^0(y^1)$ mis à jour après la découverte de y^4.

Nous avons $A(y^2) = \{y^1\}, P(y^2) = \{y^3, y^4\}$ *et nous pouvons remarquer que la contrainte* $\lambda \cdot y^2 \geq \lambda \cdot y^3$ *est redondante avec* $\lambda \cdot y^2 \geq \lambda \cdot y^4$*. Par conséquent, nous supprimons* y^3 *de* $P(y^2)$*. L'ensemble* $W_p^0(y^2)$ *est illustré par la figure 5.6. Nous allons maintenant explorer la seule facette non confirmée, définie par* $\lambda \cdot y^2 \geq \lambda \cdot y^4$*. La résolution de* P_λ *pour chacun des poids de cette facette nous donne le nouveau point supporté* $y^5 = (6, 8, 12)$*, ainsi que* y^2 *ou* y^4*, à nouveau.*
La présence de ce dernier point confirme l'adjacence de y^2 *et* y^4*. Le segment de la facette pour lequel l'égalité* $\lambda \cdot y^2 = \lambda \cdot y^4$ *est vérifiée est donc une facette de* $W^0(y^2)$ *et de* $W^0(y^4)$*.*
Nous mettons maintenant à jour les ensembles :

$$\begin{aligned}
\bar{Y} &= \bar{Y} \cup \{y^5\},\\
A(y^2) &= A(y^2) \cup \{y^4\},\\
A(y^4) &= A(y^4) \cup \{y^2\},\\
P(y^2) &= P(y^2) \setminus \{y^4\} \cup \{y^5\},\\
P(y^3) &= P(y^3) \cup \{y^5\},\\
P(y^4) &= P(y^4) \cup \{y^5\},\\
P(y^5) &= A(y^5) = \emptyset.
\end{aligned}$$

Puis nous mettons à jour $W_p^0(y^2)$*. Ce dernier est illustré par la figure 5.7. Nous procédons ensuite à l'exploration de la facette définie par* $\lambda \cdot y^2 \geq \lambda \cdot y^5$*, qui nous donne le point* y^5*. Aucun nouveau point n'est trouvé, nous confirmons juste l'adjacence entre* y^2 *et* y^5 *en mettant à jour les ensembles*

$$\begin{aligned}
A(y^2) &= A(y^2) \cup \{y^5\},\\
A(y^5) &= \{y^2\},\\
P(y^2) &= P(y^2) \setminus \{y^5\}.
\end{aligned}$$

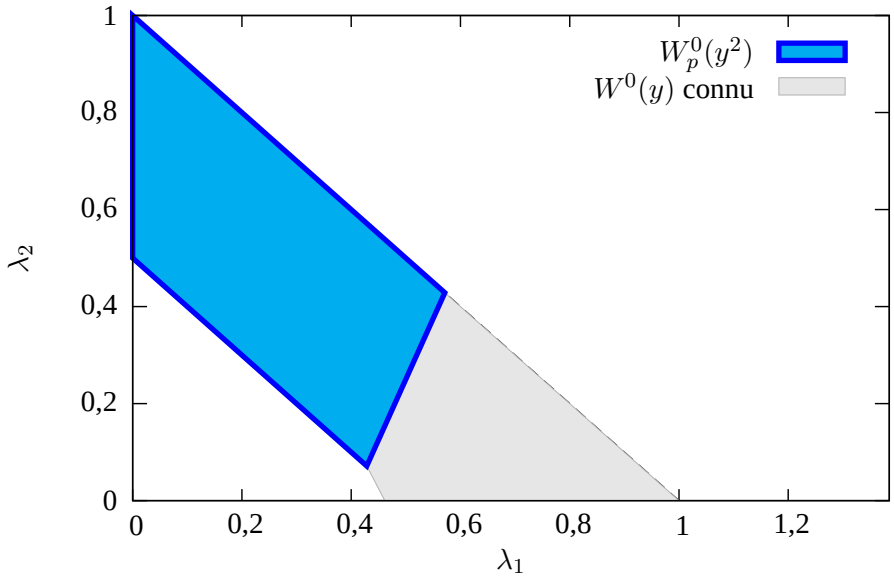

Figure 5.6 – L'ensemble $W_p^0(y^2)$ obtenu après l'initialisation.

L'ensemble $P(y^2)$ est maintenant vide. Nous avons obtenu que $W_p^0(y^2) = W^0(y^2)$. La procédure continue avec le point suivant dans $\bar{Y}$.

Nous avons $A(y^3) = \emptyset, P(y^3) = \{y^4, y^5\}$. L'ensemble $W_p^0(y^3)$ est illustré par la figure 5.8. Il ne reste qu'à explorer la facette définie par $\lambda \cdot y^3 \geq \lambda \cdot y^4$ pour obtenir un des points supportés, déjà connus, y^3 ou y^4. Nous mettons à jour les ensembles E

$$\begin{aligned} A(y^3) &= A(y^3) \cup \{y^4\}, \\ A(y^4) &= A(y^4) \cup \{y^3\}, \\ P(y^3) &= P(y^3) \setminus \{y^4\}. \end{aligned}$$

$W_p^0(y^3)$ n'est pas modifié par cette mise à jour. Nous procédons à l'exploration de l'autre facette, définie par $\lambda \cdot y^3 \geq \lambda \cdot y^5$. Cette exploration aboutit à nouveau à un des points connus y^3 ou y^5. Nous mettons à jour les ensembles

$$\begin{aligned} A(y^3) &= A(y^3) \cup \{y^5\}, \\ A(y^5) &= A(y^5) \cup \{y^3\}, \\ P(y^3) &= P(y^3) \setminus \{y^5\}. \end{aligned}$$

$P(y^3)$ est maintenant vide, nous avons donc confirmé toutes les facettes de $W^0(y^3)$ et nous procédons maintenant au point suivant de $\bar{Y}$.

Nous avons $A(y^4) = \{y^1, y^2, y^3\}, P(y^4) = \{y^5\}$. L'ensemble $W_p^0(y^4)$ est illustré par la figure 5.9. L'exploration des facettes définies par $\lambda \cdot y^4 \geq \lambda \cdot y^5$ donne un des points, déjà connus, y^4 ou y^5. Comme

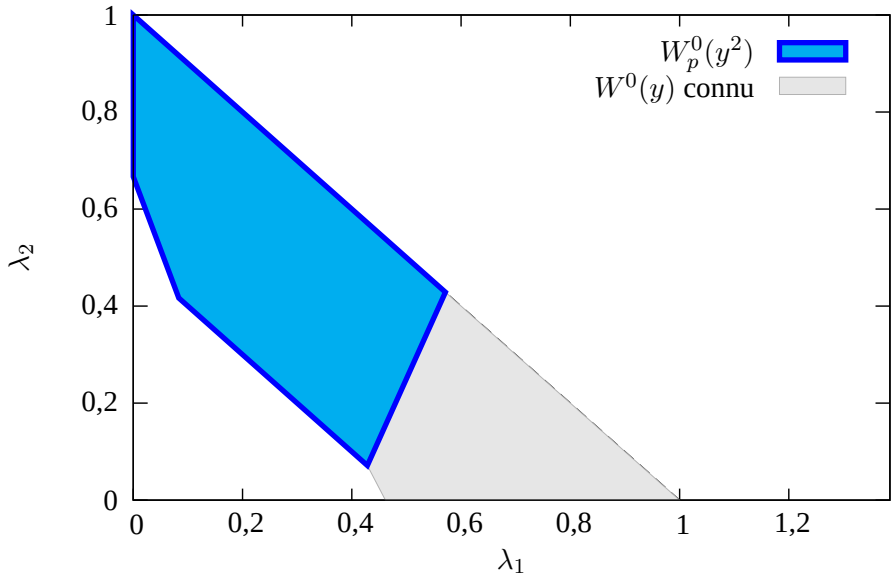

Figure 5.7 – L'ensemble $W_p^0(y^2)$ mis à jour après la découverte de y^5.

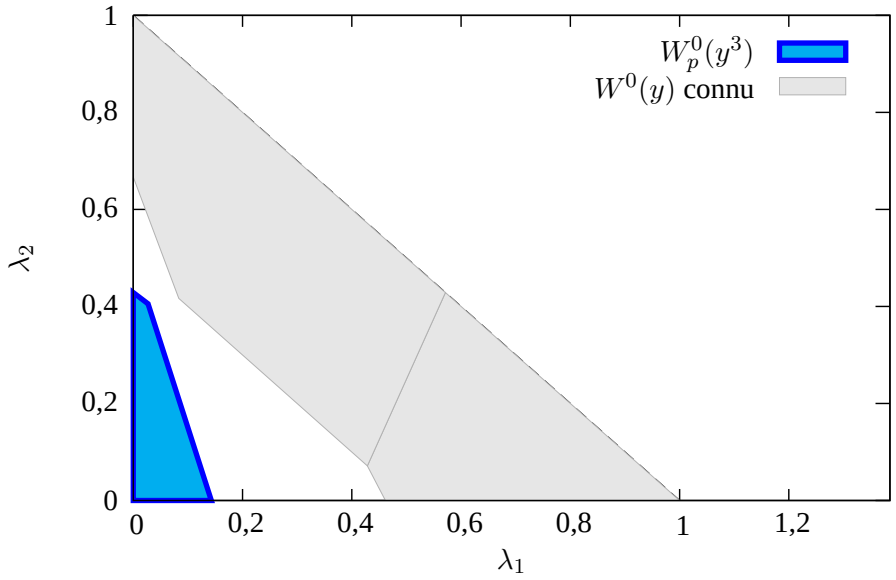

Figure 5.8 – L'ensemble $W_p^0(y^3)$ obtenu après l'initialisation.

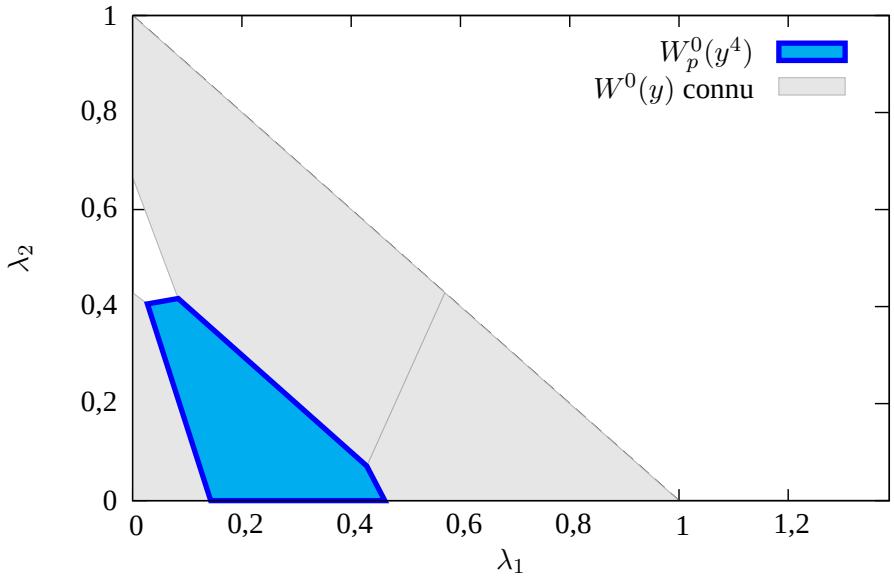

Figure 5.9 – L'ensemble $W_p^0(y^4)$ obtenu après l'initialisation.

y^4 est présent, nous pouvons confirmer que y^4 et y^5 sont adjacents et nous mettons à jour les ensembles :

$$A(y^4) = A(y^4) \cup \{y^5\},$$
$$A(y^5) = A(y^5) \cup \{y^4\},$$
$$P(y^4) = P(y^4) \setminus \{y^5\}.$$

$P(y^4)$ est maintenant vide, nous avons donc confirmé toutes les facettes de $W^0(y^4)$ et nous procédons maintenant au point suivant de $\bar{Y}$.

Le seul point restant est y^5. Nous avons $A(y^5) = \{y^2, y^3, y^4\}, P(y^5) = \emptyset$. Par conséquent, nous avons la confirmation que $W_p^0(y^5) = W^0(y^5)$.
Tous les points ont été considérés, l'algorithme se termine et nous avons $\bar{Y} = Y_{SN1}$.

5.1.2.2 Initialisation de $\bar{Y}$ par les solutions lexicographiquement optimales

La procédure initialise $\bar{Y}$ en calculant les solutions lexicographiquement optimales. Pour cela, nous utilisons l'hypothèse d'intégrité des coûts.

Nous obtenons tout d'abord le point idéal y^I en calculant une solution optimale x^i pour chaque problème mono-objectif P_i, obtenu en ne considérant que l'objectif i (voir la section 1.2.4 page 19). Soit y^i le point correspondant à x^i pour le problème tri-objectif. Puis nous résolvons un problème P_λ, avec un poids λ adéquat, pour chaque permutation de $\{1, \ldots, p\}$, de manière à obtenir une solution lexicographiquement optimale selon chaque permutation. Le poids λ employé est posé par la proposition ci-dessous.

Proposition 5.1 (Poids pour obtenir une solution lexicographiquement optimale)**.** *Soit π une permutation de $\{1,\ldots,p\}$ et y^I le point idéal. Soit le poids λ défini par :*

$$\lambda_{\pi^i} = \prod_{j=i+1}^{p} (y^I_{\pi^j} + 1)$$

Alors une solution optimale pour P_λ est lexicographiquement optimale selon la permutation π pour le problème multi-objectif.

L'idée du choix de cette pondération est de préférer améliorer une solution sur l'objectif π^i d'une unité plutôt que d'améliorer l'objectif π^{i+1} au maximum. Supposons, sans perte de généralité, que $\pi = (1,\ldots,p)$. Pour expliquer le choix de la valeur de λ, nous ne considérons pas, en premier lieu, les objectifs $j > 2$. Nous sommes alors dans une situation bi-objectif dans laquelle le poids $\lambda = ((y^I_2 + 1), 1, 0, \ldots)$ convient clairement. En effet, avec un tel poids, le solveur préfèrera toujours une solution améliorant le premier objectif, même d'une unité, qu'une solution améliorant le second objectif, même au maximum. Dans le cas tri-objectif, nous ajoutons l'objectif 3 en posant $\lambda_3 = 1$. Il s'agit alors d'exprimer la préférence de l'objectif 2 sur l'objectif 3 en posant $\lambda_2 = y^I_3 + 1$ et de reporter cette préférence sur l'objectif 1 en multipliant le poids associé par cette même valeur. Le poids obtenu est alors $\lambda = ((y^I_3 + 1) \times (y^I_2 + 1), y^I_3 + 1, 1, 0, \ldots)$. Le processus se répète alors itérativement jusqu'à poser $\lambda_p = 1$.

5.1.2.3 Exploration d'une facette

Dans le cas tri-objectif, une facette F de $W^0_p(y)$ est une arête. Nous pouvons facilement l'explorer de la manière détaillée ci-après [111].

Supposons que F soit définie par $\lambda \cdot y = \lambda \cdot y'$, où $y' \in \bar{Y}$. Nous notons ses points extrêmes λ^1 et λ^2. Nous pouvons explorer F en utilisant un problème bi-objectif dont les vecteurs de profits sont définis par :

$$c'^1 = \lambda^1_1 c^1 + \lambda^1_2 c^2 + \lambda^1_3 c^3$$
$$c'^2 = \lambda^2_1 c^1 + \lambda^2_2 c^2 + \lambda^2_3 c^3$$

et en déterminant un ensemble $X_{SE1_m} = \{x^1,\ldots,x^q\}$ pour ce problème, ce qui peut se faire aisément avec la méthode de Aneja et Nair [3]. Nous supposons que l'ensemble X_{SE1_m} ainsi obtenu est trié par valeur croissante de l'objectif défini par c'^1. Soit $\{y^1,\ldots,y^q\}$ l'ensemble des points correspondant aux solutions $\{x^1,\ldots,x^q\}$ pour le problème tri-objectif. Nous notons F_i les segments dans F tels que pour tout $\lambda \in F_i, \max_{y' \in \bar{Y}}\{\lambda \cdot y'\} = \lambda \cdot y^i$. Nous avons alors le résultat suivant :

Propriété 5.3 (Exploration d'une facette et adjacence [111])**.**

(i) Si pour tout $\lambda \in F_i, \lambda \cdot y = \lambda \cdot y^i$, alors $W^0(y) \cap W^0(y') = F_i$ et y et y' sont adjacents.

(ii) Si pour tout $\lambda \in \mathrm{rint}(F_i), \lambda \cdot y > \lambda \cdot y^i$, alors $W^0_p(y) \neq W^0(y)$.

Où $\mathrm{rint}(E)$ désigne l'intérieur relatif d'un ensemble E (c'est-à-dire E privé de ses bornes).

Dans le cas multi-objectif général, $W^0(y)$ est un polytope de dimension $p-1$ et ses facettes ne sont plus des arêtes, mais un polytope de dimension $p-2$. En particulier, si $p = 4$, une facette est un polytope de dimension 2, ce qui n'est pas nécessairement un triangle. Cette facette ne définit donc

pas trois vecteurs de profits utilisables pour déterminer les points supportés extrêmes d'un problème tri-objectif.

Przybylski [111] propose une généralisation de la procédure en remarquant que l'exploration d'une face de dimension $p-2$ consiste à déterminer les points supportés extrêmes pour un problème avec $p-1$ objectifs. L'algorithme est alors appliqué récursivement jusqu'à obtenir un problème tri-objectif pouvant être résolu avec la procédure décrite ici.

5.1.2.4 Mise à jour des ensembles $W_p^0(.), A(.), P(.)$

Nous décrivons ici comment les ensembles $W_p^0(.), A(.), P(.)$ sont mis à jour avec les points $\{y^1, \ldots, y^q\}$ obtenus à l'issue de l'exploration d'une facette.

Soit y le point pour lequel nous cherchons $W^0(y)$, soit la facette F définie par $\lambda \cdot y = \lambda \cdot y', y' \in \bar{Y}$.

Si pour tout $\lambda \in F_i, \lambda \cdot y^i = \lambda \cdot y$, alors d'après le point (i) de la propriété 5.3, y et y' sont adjacents et $F_i = W^0(y) \cap W^0(y')$.

Autrement, si pour tout $\lambda \in F_i, \lambda \cdot y^i > \lambda \cdot y$, alors d'après le point (ii) de la propriété 5.3, la contrainte définie par $\lambda \cdot y \geq \lambda \cdot y^i$ peut être utilisée afin de réduire $W_p^0(y)$. Nous actualisons $\bar{Y} \leftarrow \bar{Y} \cup \{y^i\}$ et dans le cas où y^i est un nouveau point supporté, nous initialisons $A(y^i) = P(y^i) = \emptyset$. Afin de modifier $W_p^0(y)$, nous posons que y et y^i sont potentiellement adjacents. Comme y^i est situé « entre » y et y', nous posons aussi que y' et y^i sont potentiellement adjacents.

Ensuite, pour tout $i \in \{1, \ldots, q-1\}$, y^i et y^{i+1} sont localement adjacents par rapport à F et $W^0(y^i) \cap W^0(y^{i+1}) \neq \emptyset$. Donc si $y^i, y^{i+1} \in \bar{Y}$, nous posons que y^i et y^{i+1} sont potentiellement adjacents.

Enfin, nous considérons le point extrême de l'arête F situé du côté de F_1, le segment correspondant au point y^1. Si ce point extrême n'est pas situé dans la frontière de W^0, alors il est aussi un point extrême d'une autre arête définie par $\lambda \cdot y = \lambda \cdot y^*$, où $y^* \in A(y) \cup P(y)$. Nous posons alors que y' et y^* sont potentiellement adjacents. De plus, si $y^1 \in \bar{Y}$, alors nous posons aussi que y^1 et y^* sont potentiellement adjacents. Nous procédons de même pour l'autre point extrême de F_q et y^q.

5.2 Calcul des solutions non supportées

Si $\bar{Y} \supseteq Y_{SN1}$ est l'ensemble des images des solutions obtenues à l'issue de la première phase, l'espace de recherche de la seconde est restreint aux solutions dont les images sont dans $\mathrm{conv}(\bar{Y} - \mathbb{R}^p_{\geqq}) \setminus (\bar{Y} - \mathbb{R}^p_{>})$. Dans le cas bi-objectif, cet espace est décrit par un ensemble de triangles. On peut alors légitimement supposer que la situation du cas tri-objectif se représente comme un ensemble de pyramides. Cependant, il n'en est rien. En effet, l'espace non dominé est délimité par l'union des cônes de dominances. Le résultat de cet union est illustré pour l'exemple 5.19 par la figure 5.10. Les solutions non dominées ont nécessairement leurs images dans l'espace hors des cônes, qui n'est pas décrit par une forme régulière.

Nous présentons dans cette section une procédure d'exploration de cet espace basée sur les travaux de [111] et utilisant l'algorithme de *ranking* présenté dans le chapitre 4 pour calculer l'ensemble complet X_{E_M} des solutions efficaces.

5.2.1 Description générale de la procédure

Comme dans le cas bi-objectif, l'exploration se fait en résolvant des problèmes mono-objectifs, construit de manière à maximiser selon une direction perpendiculaire à une facette de $\mathrm{conv}(Y_{SN})_N$.

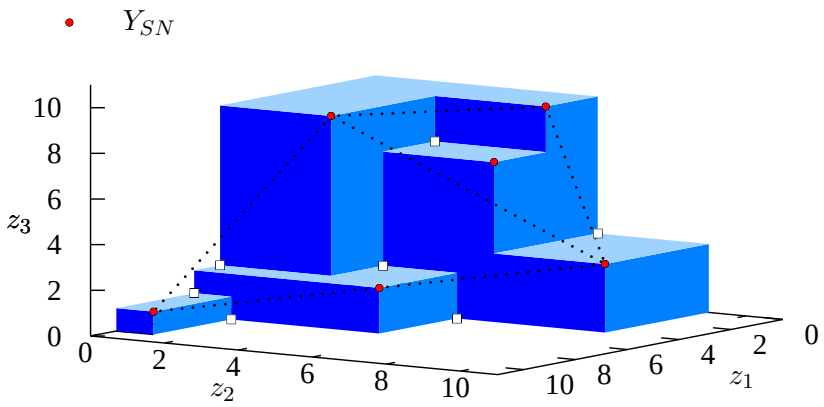

Figure 5.10 – Cônes de dominance des six points supportés de l'exemple 5.19. Les solutions non supportées ont leurs images en dehors de ces cônes, dans une région de l'espace décrite de manière irrégulière. Les □ marquent les creux dans les intersections des cônes de dominance.

Dans la continuité de l'exemple 5.20, nous pouvons observer que dans le cas bi-objectif, les poids utilisés dans la seconde phase sont ceux bornant les ensembles $W^0(y)$.

Dans le cas multi-objectif, le choix des poids utilisés dans la seconde phase est similaire au cas bi-objectif. Les facettes de $\mathrm{conv}(Y_{SN})_N$, définies par les points supportés extrêmes, ont été obtenues à l'issue de la première phase, à l'aide du partitionnement de W^0. Si $Y' \subseteq Y_{SN1}$ est un ensemble de points adjacents définissant une face maximale de $\mathrm{conv}(Y_{SN})_N$ (maximale dans le sens où il n'existe pas d'autre face de $\mathrm{conv}(Y_{SN})_N$ définie par les points supportés extrêmes et contenant cette face), alors le poids utilisé pour construire le problème mono-objectif vérifie $\lambda \in \bigcap_{y \in Y'} W^0(y)$. Cependant, cette intersection n'est pas nécessairement de dimension 1, nous pouvons en effet obtenir des faces de dimension 1 à $p-1$.

Chaque face sera traitée individuellement pour trouver les solutions non supportées situées « au dessous », à l'aide d'une procédure de *ranking*. Przybylski [111] a montré que, contrairement au cas bi-objectif, nous ne pouvons plus utiliser les nadirs locaux pour délimiter l'espace de recherche et stopper le *ranking*. L'auteur propose alors une autre description, inspirée du concept d'ensemble bornant [41] (voir la définition 1.13 page 22). Nous décrivons dans la section suivante l'adaptation de cette description au cas d'une maximisation.

5.2.2 Description de l'espace de recherche

La zone de recherche est intuitivement décrite par $(\mathrm{conv}(\bar{Y}) - \mathbb{R}^p_{\leqq}) \setminus (\bar{Y} - \mathbb{R}^p_{>})$. Cependant, il est difficile de l'explorer en utilisant cette description. En effet, dans le cas bi-objectif, la zone de recherche n'est pas explorée globalement, mais en plusieurs explorations locales. Or, aucun partitionnement clair de la zone de recherche n'apparaît dans cette formulation.

En notant $d^1, \dots, d^q$ les extrémités des segments aux creux des intersections des cônes de dominance, illustrés par des $\square$ sur la figure 5.10, nous pouvons remarquer que l'espace complémentaire à l'espace dominé dans $\mathbb{R}^p_{\geqq}$ est décrit par $\bigcup_{i=1}^{q}(d^i + \mathbb{R}^p_{\geqq})$, au moins pour cet exemple. C'est sur cette idée que Przybylski [111] définit un ensemble de points $D(U)$ pour décrire * l'espace non dominé par un ensemble U de points non dominés, et vérifiant l'égalité $(\text{conv}(U) - \mathbb{R}^p_{\geq}) \setminus (U - \mathbb{R}^p_{>}) = (\text{conv}(U) - \mathbb{R}^p_{\geqq}) \cap \bigcup_{d \in D(U)}(d + \mathbb{R}^p_{\geqq})$. Cet ensemble de points est défini de la manière suivante :

Définition 5.4 (Description de la zone de recherche [111])**.** *Soit U un ensemble de points réalisables tel que $U = U_N$. Soit $D(U) \in \mathbb{R}^p$ l'ensemble de points de cardinalité maximale tel que $d \in D(U)$ si et seulement si les deux conditions ci-dessous sont vérifiées :*

(i) *d n'est pas strictement dominé par un point de U ;*

(ii) *$\nexists d' \in D(U)$ vérifiant le point (i) avec $d \geq d'$.*

L'ensemble $D(U)$ permet une description de l'espace de recherche équivalente à celle utilisant des ensembles bornant. En particulier, nous pouvons remarquer que, dans le cas bi-objectif, $D(Y_{SN})$ est l'ensemble des points nadirs locaux définis par les points supportés. La section suivante présente une procédure de construction de cet ensemble $D(U)$.

5.2.2.1 Construction et mise à jour de la description de la zone de recherche

La construction de $D(U)$ se fait itérativement, au fur et à mesure que de nouveaux points sont ajoutés à U. Initialement, nous avons $D(\emptyset) = (0, \dots, 0)$ (ou $\{y^N\}$ si nous connaissons le point nadir). Tous les points de U sont considérés un par un et une procédure de mise à jour est appliquée pour trouver $D(U)$.

Supposons que nous ajoutions un point y à l'ensemble U, tel que pour tout $y' \in U, y \ngeqq y'$ et $y' \ngeqq y$. Nous comparons alors y avec chaque point d de $D(U)$. Deux cas se présentent. Soit $y \not> d$, auquel cas l'espace non dominé $d + \mathbb{R}^p_{\geq}$ n'est en rien modifié. Nous conservons le point d dans ce cas. Autrement, si $y > d$, alors $d + \mathbb{R}^p_{\geq}$ n'est plus un espace non dominé. Il s'agit alors de remplacer le point d par d'autres points, vérifiant la définition 5.4. La propriété ci-dessous formalise la mise à jour de $D(U)$ et la construction de ces nouveaux points.

Propriété 5.4 (Mise à jour de $D(U)$ [111])**.** *Soit $Q \subset U$, où U est un ensemble de points réalisables qui ne contient pas deux points y^1, y^2 avec $y^1 \geq y^2$ et soit $y \in U \setminus Q$.*

Nous posons $S = \{d \in D(Q) : y > d\}$ et $d(i) = (d_1, \dots, d_{i-1}, y_i, d_{i+1}, \dots, d_p)$ pour y donné. Nous avons alors :

$$D(Q \cup \{y\}) = (D(Q) \setminus S) \cup \bigcup_{d \in S} \bigcup_{i=1}^{p} \{d(i) : \nexists d' \neq d \in D(Q) \text{ tel que } d(i) \geqq d'\}$$

L'algorithme 5.2 initialise l'ensemble $D(U)$ pour un U passé en paramètre. Pour cela, il initialise $D(Q = \emptyset)$ avec le point nadir puis appelle la procédure de mise à jour de $D(Q)$ pour chaque point de U. Cette dernière procédure est une implémentation immédiate de la formule de la propriété 5.4 et est décrite par l'algorithme 5.3. Nous pouvons noter que la vérification « $\nexists d' \neq d \in D(Q)$ tel que $d(i) \geqq d'$ » a pour but de limiter le nombre de points dans $D(Q)$. En effet, s'il existe un $d' \neq d \in D(Q)$ tel que $d(i) \geqq d'$, alors l'espace non dominé représenté par $d(i) + \mathbb{R}^p_{\geq}$ est inclus dans $d' + \mathbb{R}^p_{\geq}$. Dans ce cas, $d(i)$ n'apporte aucune information supplémentaire et peut donc être ignoré.

À l'issue de la première phase, nous connaissons Y_{SN}. Nous sommes donc en mesure de calculer $D(Y_{SN})$. Il s'agit alors d'explorer $D(Y_{SN}) + \mathbb{R}^p_{\geq}$ de manière efficace.

∗. $D(U)$ se lit comme « la description obtenue depuis l'ensemble U ».

Procédure `init_description` ($\downarrow U, \downarrow y^N, \uparrow D(U)$)
Paramètre $\downarrow U$: les points pour lesquels on construit la description.
Paramètre $\downarrow y^N$: le point nadir.
Paramètre $\uparrow D(U)$: la description de l'espace non dominé par U.

1: $Q \leftarrow \emptyset$
2: $D(Q) \leftarrow \{y^N\}$
3: **pour tout** $y \in U$ **faire**
4: `màj_description`($\downarrow y, \downarrow D(Q), \uparrow D(Q \cup \{y\})$)
5: $Q \leftarrow Q \cup \{y\}$
6: **fin pour**

ALG. 5.2 – Initialisation de l'ensemble $D(U)$.

Procédure `màj_description` ($\downarrow y, \downarrow Q, \uparrow D(Q \cup \{y\})$)
Paramètre $\downarrow y$: un point $y \in U \setminus Q$.
Paramètre $\downarrow D(Q)$: la description précédente.
Paramètre $\uparrow D(Q \cup \{y\})$: la description quand le point y est considéré.

1: $S \leftarrow \emptyset$ *- - Les points supprimés de la description. - -*
2: $N \leftarrow \emptyset$ *- - Les points ajoutés à la description. - -*
3: **pour tout** $d \in D(Q)$ **faire**
4: **si** $y > d$ **alors**
5: $S \leftarrow S \cup \{d\}$
6: **pour** i **de** 1 **à** p **faire**
7: **si** $\nexists d' \neq d \in D(Q)$ tel que $d(i) \geqq d'$ **alors**
8: $N \leftarrow N \cup \{d(i)\}$
9: **fin si**
10: **fin pour**
11: **fin si**
12: **fin pour**
13: $D(Q \cup \{y\}) \leftarrow (D(Q) \setminus S) \cup N$

ALG. 5.3 – Mise à jour de l'ensemble $D(U)$.

5.2.3 Choix du poids définissant P_λ avant l'application du *ranking*

Les solutions supportées définissent les faces de $(\text{conv}(Y_{SN}))_N$. Elles sont, par définition, toutes des solutions optimales de problèmes P_λ, où λ se situe dans l'intersection d'ensembles $W^0(y^i)$. Nous devons donc considérer les faces de $W^0(y^i)$ pour identifier les poids appropriés λ.

Dans le cas tri-objectif, le choix des poids est réalisé en considérant chaque arête A de $W^0(y)$, où $y \in Y_{SN1}$. Si un des sommets de A n'est pas situé dans la frontière de W^0, nous l'ajoutons dans la liste des poids à utiliser pour une énumération. Au moins trois points supportés extrêmes définissant une facette de $(\text{conv}(Y_{SN}))_N$ correspondent aux solutions optimales du problème P_λ pour un tel poids λ.

Dans le cas où les deux sommets de A sont situés dans la frontière de W^0, alors n'importe quel point de A peut convenir pour le *ranking*. Nous choisissons arbitrairement le poids situé en son centre.

5.2.4 Exploration de la zone de recherche

Contrairement au cas bi-objectif, l'espace décrit par $(\text{conv}(Y_{SN})_N - \mathbb{R}^p_{\leqq}) \setminus (d + \mathbb{R}^p_{>})$, avec $d \in D(Y_{SN})$, n'est pas nécessairement situé sous une unique facette de Y_{SN}. Cela peut être observé sur la figure 5.10, où le point à l'intersection des cônes de dominance des points y^5 et y^6, par exemple, décrit une région qui est sous chacune des faces (y^1, y^2, y^4) et (y^2, y^3, y^4). Il conviendra donc d'éviter les redondances dans l'exploration.

Par la suite, nous noterons E l'ensemble des points délimitant ce qu'il reste à explorer dans la zone de recherche et $\bar{Y}$ l'ensemble des points réalisables non dominés connus. Nous avons initialement $E = D(Y_{SN})$ et $\bar{Y} = Y_{SN}$. Pour chaque point $d \in E$, nous devons explorer $d + \mathbb{R}^p_{>}$. Pour cette exploration, du fait de l'utilisation d'un algorithme de *ranking*, nous allons explorer une bande définie par l'hyperplan engendré par une facette de $(\text{conv}(Y_{SN}))_N$ et l'hyperplan parallèle contenant le point d. Naturellement, comme dans le cas bi-objectif, les points obtenus durant cette exploration seront utilisés pour réduire cette bande. Pour les mêmes raisons que celles détaillées dans la section 4.3.3, nous traiterons en premier lieu les bandes les plus fines. Pour cela, nous devons utiliser l'hyperplan engendré par une facette de $(\text{conv}(Y_{SN}))_N$ le plus proche du point d. Cet hyperplan est obtenu par l'algorithme 5.4.

La distance entre un point $d = (d_1, \ldots, d_p)$ et un hyperplan h d'équation $\lambda \cdot z + \alpha = 0, \alpha \in \mathbb{R}$, peut être calculé à l'aide de la formule suivante :

$$dist(d, h) = \frac{\lambda \cdot d + \alpha}{\sqrt{\lambda \cdot \lambda}}$$

Nous notons H l'ensemble des hyperplans engendrés par les facettes de $(\text{conv}(Y_{SN}))_N$. Pour tout $d \in E$, nous déterminons l'hyperplan le plus proche $h(d)$. Nous notons $H_p = \{h \in H : \exists d \in E \text{ tel que } h = h(d)\}$ l'ensemble des hyperplans potentiellement sélectionnés pour une exploration. Nous pouvons avoir $H_p \neq H$, de même que nous pouvons avoir $h(d) = h(d')$ pour $d \neq d' \in E$. Pour tout $h \in H_p$, nous notons $U(h) = \{d \in E : h(d) = h\}$ l'ensemble des points de E pour lesquels h est l'hyperplan le plus proche, et $Val(h) = \max_{d \in U(h)} dist(d, h)$ la distance séparant l'hyperplan h du plus éloigné des points de $U(h)$. Cette mesure représente la largeur de la bande explorée lorsque l'hyperplan h est choisi. Ainsi, afin d'explorer la plus petite bande possible, nous choisissons l'hyperplan $h^* \in H_p$ minimisant cette valeur.

Une fois l'exploration réalisée, nous actualisons $E \leftarrow E \setminus U(h^*)$ et $H \leftarrow H \setminus \{h^*\}$. Si Y' est l'ensemble des nouveaux points non dominés obtenus à l'issue de l'exploration, nous actualisons $\bar{Y} \leftarrow \bar{Y} \cup Y'$, ainsi que l'ensemble E en utilisant l'algorithme 5.3 et en considérant les points de Y' un par un.

Nous réitérons le processus jusqu'à ce que $E = \emptyset$ pour pouvoir conclure que $\bar{Y} = Y_N$.

Procédure `choisir_poids_et_description` ($\downarrow H, \downarrow E, \uparrow h^*, \uparrow U(h^*)$)
Paramètre $\downarrow H$: les hyperplans définissant les faces de $(\text{conv}(Y_{SN}))_N$ restant à explorer.
Paramètre $\downarrow E$: les points décrivant la zone de recherche restante.
Paramètre $\uparrow h^*$: l'hyperplan sélectionné pour l'exploration.
Paramètre $\uparrow U(h^*)$: le sous ensemble de E décrivant la zone explorée en utilisant h^*.

1: $H_p \leftarrow \emptyset$
2: **pour tout** $h \in H$ **faire**
3: $\quad U(h) \leftarrow \emptyset$
4: **fin pour**
5: **pour tout** $d \in E$ **faire**
6: $\quad h' \leftarrow \text{argmin}_{h \in H}\, dist(d, h)$
7: $\quad U(h') \leftarrow U(h') \cup \{d\}$
8: $\quad H_p \leftarrow H_p \cup \{h'\}$
9: **fin pour**
10: $h^* \leftarrow \text{argmin}_{h \in H_p} \max_{d \in U(h)} dist(d, h)$

ALG. 5.4 – Sélection de l'hyperplan et des points décrivant la zone de recherche, utilisés pour le *ranking*.

5.2.5 Bornes utilisés dans une exploration

Dans le cas bi-objectif, la borne inférieure permettant de stopper le *ranking* est obtenue en calculant la plus petite valeur $\underline{z}^\lambda = \lambda \cdot y'$, où y' est un point non dominé, mais pas nécessairement réalisable (voir la section 4.2.1 page 66).

Dans cette section, nous présentons le calcul de la borne $\underline{z}^\lambda$ dans le cas tri-objectif, en nous appuyant sur les travaux de Przybylski [111] sur le problème d'affectation. Bien entendu, l'ensemble $U(h)$ défini précédemment pour décrire la zone de recherche à explorer sera utilisé pour obtenir cette borne.

Comme dans le cas bi-objectif, une première borne inférieure peut être obtenue de la manière suivante :

$$\underline{z}_o^\lambda = \min_{d \in U(h)} \{\lambda \cdot d\}$$

Cette valeur correspond à une évaluation sur un point dominé.

Contrairement au cas bi-objectif, nous ne pouvons pas utiliser directement un décalage de (1,1,1) sur chaque point $d \in U(h)$, nous devons aussi considérer les facettes de $d + \mathbb{R}^p_{\geqq}$ pour vérifier quelle partie de chaque facette est dominée. Pour cela, pour toutes les facettes de $d + \mathbb{R}^p_{\geqq}$, nous calculons une borne inférieure comme nous le faisons dans le cas bi-objectif.

Nous notons $F_j(d) = \{y \in (d + \mathbb{R}^p_{\geqq}) : y_j = d_j\}$ les facettes de $d + \mathbb{R}^p_{\geqq}$, pour $j \in \{1, \ldots, p\}$. Nous notons $\bar{Y}_j(d) = \bar{Y} \cap F_j(d)$ l'ensemble des points potentiellement non dominés situés dans une de ces facettes. Sans perte de généralités, nous supposons par la suite que la facette $F_3(d)$ est considérée. Pour calculer la borne inférieure locale à celle-ci, nous supposons que $\bar{Y}_j(d)$ est trié par ordre croissant du premier objectif (le troisième objectif est le même pour tous les points de $\bar{Y}_j(d)$, donc le second est nécessairement décroissant). Nous notons $\bar{Y}_3(d) = \{y^1, \ldots, y^m\}$ et nous calculons les points nadir

locaux à la facette, définis par :

$$\begin{aligned} n^0 &= (d_1, y_2^1, d_3) \\ n^i &= (y_1^i, y_2^{i+1}, d_3), \quad \forall i \in \{1, \ldots, m-1\} \\ n^m &= (y_1^m, d_2, d_3) \end{aligned}$$

Une borne inférieure locale à cette facette est obtenue de la manière suivante :

$$\underline{z}_3^{\lambda}(d) = \min\left(\min_{i\in\{1,\ldots,m\}}\left\{\lambda \cdot y^i\right\}, \min_{i\in\{0,\ldots,m\}}\left\{\lambda \cdot (n^i + (1,1,0))\right\}\right)$$

Remarque. Si nous ne connaissons pas de point réalisable dans la facette $F_3(d)$, alors $\underline{z}_3^{\lambda}(d) = \lambda \cdot (d + (1,1,0))$.

Pour trouver la distance du point potentiellement non dominé le plus éloigné de h dans $d + \mathbb{R}^p_{\geqq}$, nous devons calculer $\underline{z}_j^{\lambda}(d)$ pour tout $j \in \{1,2,3\}$ et utiliser la formule suivante :

$$\underline{z}^{\lambda}(d) = \min\left\{\min_{j\in\{1,2,3\}}\left\{\underline{z}_j^{\lambda}(d)\right\}, \lambda \cdot (d + (1,1,1))\right\}$$

Enfin, pour calculer la borne inférieure correspondant au point potentiellement non dominé le plus éloigné de h dans $\bigcup_{d\in U(h)}(d + \mathbb{R}^p_{\geqq})$, il suffit de calculer

$$\underline{z}^{\lambda} = \min_{d\in U(h)} \underline{z}^{\lambda}(d)$$

Toutes les solutions $x \in X$ telles que $z(x) \in \bigcup_{d\in U(h)}(d + \mathbb{R}^p_{\geqq})$ avec $\lambda \cdot z(x) < \underline{z}^{\lambda}$ sont dominées. L'énumération de tout $x \in X$ avec $\lambda \cdot z(x) \geq \underline{z}^{\lambda}$ permet l'obtention de toutes les solutions non supportées dont le point correspondant est situé dans $\bigcup_{d\in U(h)}(d + \mathbb{R}^p_{\geqq})$, y compris les solutions équivalentes. Donc, après toutes les explorations, en utilisant cette borne inférieure, nous obtenons l'ensemble complet maximal X_{E_M}.

5.3 Description de l'algorithme

La procédure permettant d'obtenir l'ensemble des solutions efficaces pour un problème de sac à dos multi-objectif est présentée par l'algorithme 5.5. Elle débute par le calcul de l'ensemble X_{SE1} des solutions supportées. À l'issue de cette étape, l'ensemble W^0 a été partitionné et le calcul de l'ensemble H des facettes de $(\mathrm{conv}(Y_{SN}))_N$ a été effectué. La description de l'espace de recherche est ensuite initialisée.

À chaque itération de la boucle principale, un hyperplan de H et un sous ensemble de E sont sélectionnés pour une exploration. L'hyperplan est utilisé dans l'application du *ranking* (voir la section 4.3 page 69) pour construire le problème mono-objectif P_λ, tandis que le sous ensemble de E permet d'obtenir une borne inférieure pour l'arrêter. À l'issue du *ranking*, un ensemble S' de nouvelles solutions efficaces a été découvert. Il est utilisé pour mettre à jour l'ensemble E. L'algorithme s'arrête lorsque E est vide, c'est à dire que tout l'espace de recherche a été exploré.

Procédure `deux_phases_mokp`($\downarrow P, \uparrow S$)
Paramètre $\downarrow P$: l'instance du problème multi-objectif de sac à dos.
Paramètre $\uparrow S$: l'ensemble complet maximal des solutions efficaces de P.
- - Calcul des solutions supportées. - -
`phase_1`($\downarrow P, \uparrow S$)
- - Calcul des hyperplans à utiliser pour les poids de la phase 2, à partir des arêtes obtenues par le partitionnement de W^0 (voir la section 5.2.3). - -
$H \leftarrow$ `hyperplans_depuis_arêtes`($\downarrow W^0(y) : y \in S$)
`init_description` ($\downarrow S, \downarrow (0, \ldots, 0), \uparrow E$)
tant que $E \neq \emptyset$ **faire**
  `choisir_poids_et_description` ($\downarrow H, \downarrow E, \uparrow h^*, \uparrow U(h^*)$)
  - - Application d'un ranking *à P_λ pour calculer un ensemble S' de solutions efficaces, où λ est donné par h^*, et la borne inférieure est obtenue avec $U(h^*)$. - -*
  `ranking` ($\downarrow P, \downarrow h^*, \downarrow U(h^*), \uparrow S'$)
  pour tout $y \in \{z(x) : x \in S'\}$ **faire**
    `màj_description`($\downarrow y, \downarrow E, \uparrow E$)
  fin pour
  $S \leftarrow S \cup S'$
  $H \leftarrow H \setminus \{h^*\}$
  $E \leftarrow E \setminus U(h^*)$
fin tant que

ALG. 5.5 – Algorithme en deux phases pour le problème de sac à dos multi-objectif.

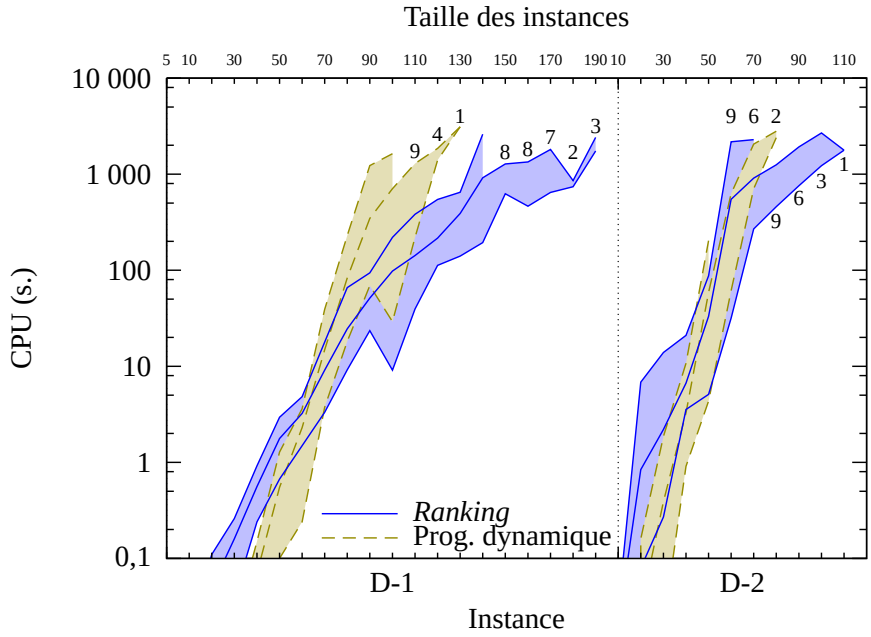

Figure 5.11 – Comparaison des temps de résolution minimaux, moyens et maximaux pour les instances du groupe D, obtenus avec la programmation dynamique de [10] et avec la procédure en deux phases utilisant un *ranking*. Le nombre d'instances résolues est indiqué lorsque certaines instances n'ont pas été résolues dans le délai d'une heure, pour une taille donnée. Les chiffres 9, 6 et 2 en haut à droite concernent la programmation dynamique.

5.4 Expérimentations numériques

L'algorithme 5.5 a été implémenté en C++ et appliqué sur les instances de sac à dos tri-objectif décrites dans la section 3.4.1.2. L'ordinateur exécutant le programme dispose d'un Pentium 4 cadencé à 3,73 GHz et de 3 Go de mémoire vive. Le temps de résolution est limité à une heure par instance. Comme dans le cas bi-objectif, un algorithme de réduction est appliqué avant la résolution de chaque problème P_λ traité dans l'algorithme en deux phases.

La figure 5.11 présente le temps minimal, maximal et moyen obtenu pour les dix instances de chaque taille. L'algorithme de *ranking* apparaît globalement comme plus efficace que la programmation dynamique, sauf pour les plus petites instances (de taille inférieure ou égale à 60 variables pour le sous-groupe D-1 et inférieure ou égale à 50 variables pour le groupe D-2). À nouveau, les algorithmes éprouvent plus de difficultés à résoudre les instances corrélées que les instances non corrélées, principalement du fait du nombre important de solutions efficaces sur ces premières.

Les courbes 5.12 et 5.13 reportent la taille minimale, moyenne et maximale de X_{E_M}, pour toutes les tailles d'instances pour lesquelles les dix problèmes ont été résolus. Nous pouvons remarquer que le nombre de solutions efficaces pour une taille d'instance donnée est bien plus important pour les instances du groupe D-2. De plus, la moyenne est, pour les deux groupes, plus proche du minimum que du maximum, ce qui traduit un comportement singulier de certaines instances. Le calcul de la valeur médiane pour chaque taille donne des valeurs plus faible que la moyenne et suggère donc que certaines instances acceptent en effet un nombre exceptionnellement grand de solutions efficaces.

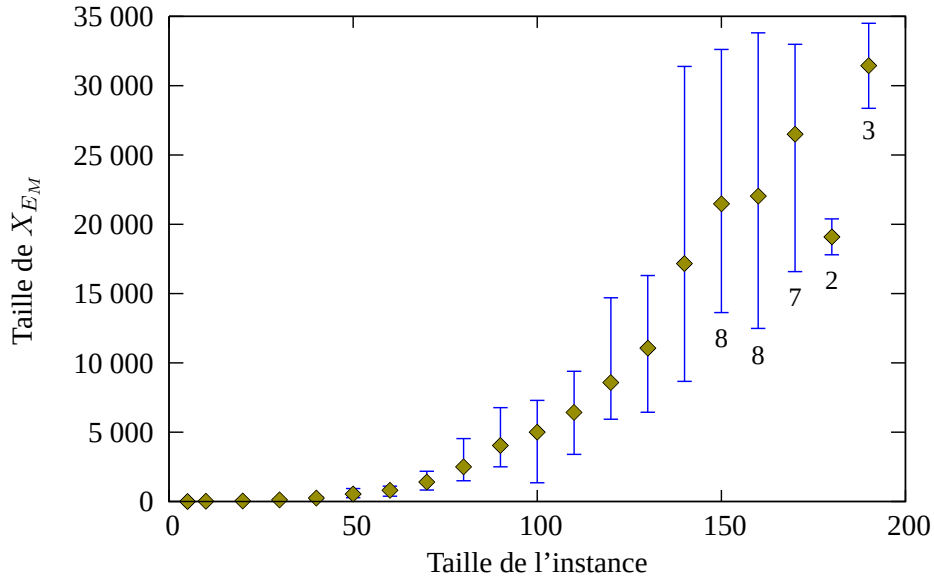

Figure 5.12 – Taille de X_{E_M} pour chaque taille d'instance du groupe D-1. Lorsque certaines instances n'ont pas été résolues dans le temps imparti, le nombre d'instances sur lequel porte le calcul est indiqué sous la barre.

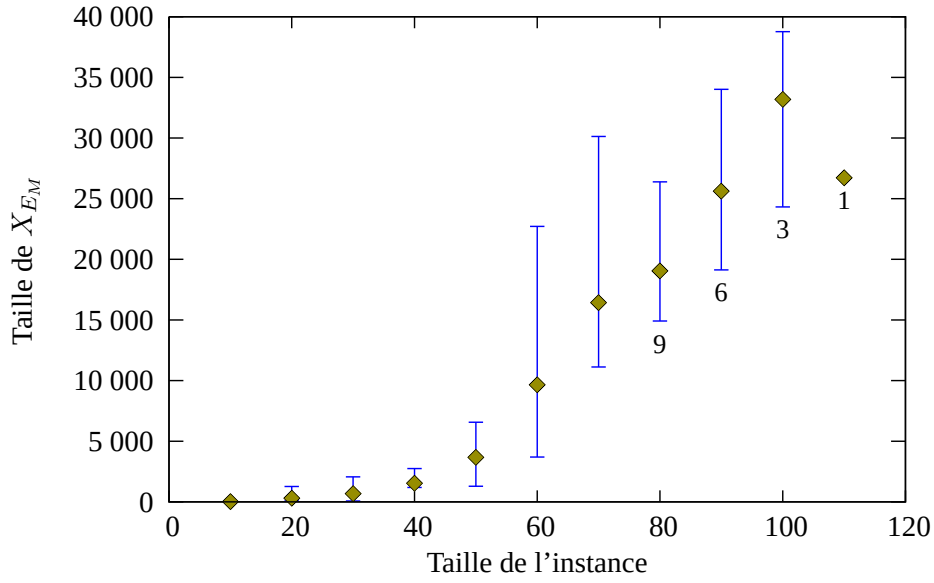

Figure 5.13 – Taille de X_{E_M} pour chaque taille d'instance du groupe D-2. Lorsque certaines instances n'ont pas été résolues dans le temps imparti, le nombre d'instances sur lequel porte le calcul est indiqué sous la barre.

n	$\lvert X_{E_M}\rvert$ (D-1)	$\lvert X_{E_M}\rvert$ (D-2)	facteur D-2/D-1	facteur croissance D-1	facteur croissance D-2
10	9,2	18,6	2,0		
20	36,4	301,9	8,3	4,0	16,2
30	114,3	670,1	5,9	3,1	2,2
40	254,0	1 537,2	6,0	2,2	2,3
50	544,0	3 664,9	6,7	2,1	2,1
60	804,3	9 647,9	12,0	1,5	2,6
70	1 402,4	16 423,7	11,7	1,7	1,7

Table 5.1 – Taille moyenne de X_{E_M} pour les instances des groupes D-1 et D-2 (colonnes 2 et 3). La colonne « facteur D-2/D-1 » reporte le rapport entre le nombre de solutions efficaces obtenues dans le groupe D-2 et celui obtenu dans le groupe D-1. Enfin, les deux dernières colonnes contiennent le rapport d'augmentation du nombre de solutions efficaces quand la taille de l'instance augmente de 10 variables.

Le facteur d'augmentation du nombre de solutions avec la taille de l'instance, reporté dans la table 5.1, est sensiblement le même pour les groupes D-1 et D-2, avec en moyenne une valeur de 2,15 pour le premier et 2,25 pour le second, pour une augmentation de la taille de 10 variables [†]. En moyenne, pour une taille d'instance, celles du groupe D-2 ont 7,5 fois plus de solutions efficaces que celles du groupe D-1.

Étonnamment, aucune instance du groupe D-1 ne possède de solutions efficaces équivalentes. Seulement trois instances du groupe D-2 acceptent quant à elles des solutions efficaces équivalentes, toutes de 70 variables, mais dans une quantité très faible (l'écart de la taille de X_{E_M} et de celle de Y_N est au plus égale à trois).

5.5 Conclusion

Dans ce chapitre, nous avons présenté une généralisation au cas tri-objectif de la méthode en deux phases présentée dans le chapitre 4, en nous appuyant sur les travaux de Przybylski [111] sur le problème multi-objectif d'affectation. Nous avons ensuite comparé les performance de notre algorithme avec celles de la programmation dynamique, jusqu'alors meilleure méthode de résolution exacte [10] pour le problème considéré.

Ces expérimentations numériques suggèrent que notre méthode est la plus adaptée pour résoudre des instances du problème multi-objectif de sac à dos, dépassant dès cinquante variables les performances de la programmation dynamique. Nous pouvons néanmoins constater que, comparativement aux tailles résolues dans le cas bi-objectif, ces algorithmes résolvent des instances relativement petites. Nous expliquons cette limitation par le grand nombre de solutions efficaces acceptées par les instances multi-objectif.

L'obtention de bonnes performances avec cette procédure requiert de connaître un bon algorithme de résolution pour la version mono-objectif du problème ainsi qu'une procédure de *ranking* efficace, ce qui n'est pas évident. D'une manière générale, si un tel algorithme est inconnu, il faudra s'orienter vers une méthode plus générique telle qu'une procédure de séparation et évaluation (PSE). Dans l'optique de comparer une telle approche avec la méthode en deux phases, le chapitre suivant présente une PSE pour le problème multi-objectif de sac à dos.

†. Le rapport des tailles 10 et 20 du groupe D-2, qui se démarque par l'étonnant facteur de 18,6, est ignoré dans ce calcul

CHAPITRE 6

Procédures de séparation et évaluation pour le problème de sac à dos multi-objectif

Les procédures de séparation et évaluation (PSE) ont prouvé qu'elles pouvaient être performantes sur des problèmes d'optimisation combinatoire, pour lesquels de bonnes fonctions de calcul de bornes sont disponibles. C'est le cas pour le problème de sac à dos mono-objectif.

Cependant, comme l'ont souligné Ehrgott et Gandibleux [37], peu de ces procédures ont été conçues pour des problèmes d'optimisation combinatoire multi-objectifs (MOCO). À notre connaissance, seuls quelques articles proposent des procédures pour des problèmes multi-objectifs linéaires en nombres entiers (MOILP) [16, 17, 78, 90, 96], jusqu'à une contribution récente de Sourd et Spanjaard [126]. Ces derniers ont proposé une généralisation des PSE pour des MOCO, avec une application au problème d'arbre couvrant minimal bi-objectif.

Ainsi, bien qu'elles soient efficaces dans le cas mono-objectif, aucune PSE pour $01MOKP$ n'est disponible dans la littérature.

Ce chapitre décrit deux procédures pour ce problème. La première section rappelle le cadre global des PSE. Dans la section suivante, les questions du calcul des bornes et de l'ordre de sélection des variables sont posées. Une première procédure est proposée, puis évaluée expérimentalement. Après analyse des résultats obtenus, une seconde procédure est proposée, puis évaluée expérimentalement à son tour. Le chapitre se conclut par un récapitulatif des résultats obtenus et une discussion de l'efficacité de ces algorithmes.

6.1 Le cadre des PSE

L'espace de recherche pour un problème de sac à dos à n variables binaires est constitué des vecteurs x de taille n aux valeurs variant de $(0, \ldots, 0)$ à $(1, \ldots, 1)$; soit 2^n vecteurs. Nombreux sont ceux ne correspondant pas à une solution réalisable, les autres étant des solutions soit efficaces, soit dominées.

Le concept algorithmique derrière les procédures de séparation et évaluation consiste en une énumération complète de cet espace d'une manière intelligente, de façon à en extraire toutes les solutions efficaces. Cette exploration se fait par partitionnement, en divisant l'espace de recherche en deux

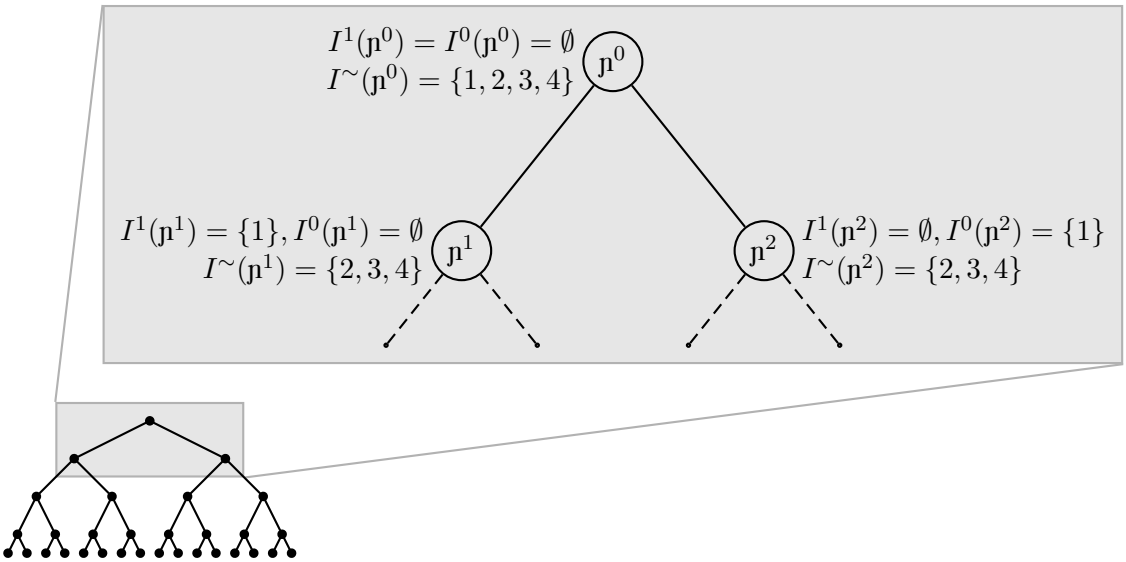

Figure 6.1 – Branchement sur l'objet numéro 1 à la racine de l'arbre pour un problème à quatre variables. L'exploration complète est illustrée en bas à gauche. Notons que les nœuds $\mathfrak{n}^1$ et $\mathfrak{n}^2$ sont indépendants. Par conséquent, ils peuvent être divisés sur des objets différents parmi $I^\sim$.

sous-espaces selon la valeur d'une variable donnée. L'un correspond aux solutions où cette variable prend la valeur 1, l'autre où elle prend la valeur 0. En appliquant ce procédé récursivement sur chaque sous-espace, l'ensemble des 2^n vecteurs sera généré. Cependant, pour des raisons de performances * il convient de limiter l'espace exploré, notamment en ignorant les solutions non réalisables, ou en stoppant l'exploration d'un sous-espace en observant qu'il ne contient aucune solution efficace. C'est ici que l'*évaluation* intervient.

L'exploration est souvent représentée par un arbre binaire où chaque nœud correspond à une affectation de valeurs pour certaines variables, une décision étant à prendre pour les autres. Un nœud de cet arbre sera noté $\mathfrak{n}$. On notera $I^1(\mathfrak{n})$ (resp. $I^0(\mathfrak{n})$) l'ensemble des objets $i \in I$ tels que $x_i = 1$ (resp. $x_i = 0$) dans $\mathfrak{n}$. L'ensemble $I^\sim(\mathfrak{n})$ des objets pour lesquels aucune décision n'a été prise en est déduit, ainsi que la capacité résiduelle notée $\bar{\omega}(\mathfrak{n}) = \omega - \sum_{i \in I^1(\mathfrak{n})} w_i$. La solution associée à un nœud sera notée $x(\mathfrak{n})$, telle que $x_i = 1, \forall i \in I^1(\mathfrak{n})$ et $x_i = 0, \forall i \notin I^1(\mathfrak{n})$.

Si cela n'introduit pas d'ambiguïté, la précision « $(\mathfrak{n})$ » sera omise, pour aboutir aux notations $I^1, I^0, I^\sim$ et $\bar{\omega}$.

La séparation d'un nœud est illustrée par la figure 6.1. Nous notons $\bar{X}$ l'ensemble des solutions potentiellement efficaces connues à une étape de la procédure et $\bar{Y}$ son image dans l'espace des objectifs.

Exemple 6.22 (Exploration de l'espace de recherche par séparation). *Cet exemple déroule la séparation de l'espace de recherche pour le problème 3.5 présenté en page 55. Les objets sont choisis dans l'ordre croissant de leur numéro. Nous avons noté dans l'exemple 3.13 que* $\forall x \in X_{E_M} : x_2 = 0$ *et* $x^6 = 1$. *Par conséquent, la racine de l'arbre, notée* $\mathfrak{n}^0$, *sera telle que* $I^1 = \{6\}, I^0 = \{2\}$. *La séparation portera sur* $I^\sim = \{1,3,4,5\}$, *avec* $\bar{\omega} = 15$. *La première s'effectue sur l'objet numéro 1 pour obtenir*

∗. voir les problèmes de combinatoire de la section 1.1.1

deux sous-espaces, l'un contenant les solutions où $x_1 = 1$, l'autre où $x_1 = 0$. Les valeurs de $I^{\sim}$ et $\bar{\omega}$ sont donnés à titre informatif.

$\mathfrak{p}^0$: *L'exploration débute par la séparation de $\mathfrak{p}^0$, avec $I^1 = \{6\}, I^0 = \{2\}, I^{\sim} = \{1,3,4,5\}$ et $\bar{\omega} = 15$. Cette séparation s'effectue sur l'objet numéro 1.*

- **$\mathfrak{p}^1$:** *$I^1 = \{\underline{1}, 6\}, I^0 = \{2\}, I^{\sim} = \{3,4,5\}$ et $\bar{\omega} = 7$. Ce nœud est maintenant divisé sur l'objet numéro 3.*
  - **$\mathfrak{p}^2$:** *$I^1 = \{1, \underline{3}, 6\}, I^0 = \{2\}, I^{\sim} = \{4,5\}$ et $\bar{\omega} = 0$. Aucun autre objet ne peut être ajouté, nous obtenons une première solution $x^1 = (1,0,1,0,0,1), z(x^1) = (13,22)$. C'est la meilleure solution connue pour l'instant, nous l'ajoutons à $\bar{X} \leftarrow \{x^1\}$.*
  - **$\mathfrak{p}^3$:** *$I^1 = \{1,6\}, I^0 = \{2,\underline{3}\}, I^{\sim} = \{4,5\}$ et $\bar{\omega} = 7$. Ce nœud est divisé sur l'objet numéro 4.*
    - **$\mathfrak{p}^4$:** *$I^1 = \{1,\underline{4},6\}, I^0 = \{2,3\}, I^{\sim} = \{5\}$ et $\bar{\omega} = 2$. L'objet 5 ne peut pas être ajouté à I^1 car $w_5 = 4 > \bar{\omega}$. Il est par conséquent directement ajouté à I^0 pour obtenir une nouvelle solution $x^2 = (1,0,0,1,0,1), z(x^2) = (17,18)$. Aucune solution de $\bar{X}$ ne la domine, elle est donc sauvegardée : $\bar{X} \leftarrow \bar{X} \cup \{x^2\} = \{x^1, x^2\}$.*
    - **$\mathfrak{p}^5$:** *$I^1 = \{1,6\}, I^0 = \{2,3,\underline{4}\}, I^{\sim} = \{5\}$ et $\bar{\omega} = 7$. L'objet 5 peut ici être ajouté à I^1 pour obtenir la solution $x^3 = (1,0,0,0,1,1), z(x^3) = (16,21)$. Aucune solution de $\bar{X}$ ne la domine, elle est donc sauvegardée : $\bar{X} \leftarrow \bar{X} \cup \{x^3\}$.*
- **$\mathfrak{p}^6$:** *$I^1 = \{6\}, I^0 = \{\underline{1},2\}, I^{\sim} = \{3,4,5\}$ et $\bar{\omega} = 15$. Ce nœud est maintenant divisé sur l'objet numéro 3.*
  - **$\mathfrak{p}^7$:** *$I^1 = \{\underline{3},6\}, I^0 = \{1,2\}, I^{\sim} = \{4,5\}$ et $\bar{\omega} = 8$. Ce nœud est divisé sur l'objet numéro 4.*
    - **$\mathfrak{p}^8$:** *$I^1 = \{3,\underline{4},6\}, I^0 = \{1,2\}, I^{\sim} = \{5\}$ et $\bar{\omega} = 3$. L'objet numéro 5 ne peut pas être ajouté à I^1 car $w_5 = 4 > \bar{\omega}$. Il est donc directement ajouté à I^0 pour donner la solution $x^4 = (0,0,1,1,0,1), z(x^4) = (20,16)$. Aucune solution de $\bar{X}$ ne la domine, elle est donc sauvegardée : $\bar{X} \leftarrow \bar{X} \cup \{x^4\}$.*
    - **$\mathfrak{p}^9$:** *$I^1 = \{3,6\}, I^0 = \{1,2,\underline{4}\}, I^{\sim} = \{5\}$ et $\bar{\omega} = 8$. L'objet numéro 5 peut ici être ajouté à I^1 pour obtenir la solution $x^5 = (0,0,1,0,1,1), z(x^5) = (19,19)$. Aucune solution de $\bar{X}$ ne la domine, elle est donc sauvegardée. Par contre, elle domine x^2. Cette dernière est donc supprimée de $\bar{X}$ qui, au final, est égal à $\{x^1, x^3, x^4, x^5\}$.*
  - **$\mathfrak{p}^{10}$:** *$I^1 = \{6\}, I^0 = \{1,2,\underline{3}\}, I^{\sim} = \{4,5\}$ et $\bar{\omega} = 15$. Remarquons que $\sum_{i \in I^{\sim}} w_i \leq \bar{\omega}$. Par conséquent, tous les objets de $I^{\sim}$ peuvent être ajoutés à I^1 pour obtenir la dernière solution, $x^6 = (0,0,0,1,1,\underline{1}), z(x^6) = (23,15)$. Aucune solution de $\bar{X}$ ne la domine, elle est donc sauvegardée : $\bar{X} \leftarrow \bar{X} \cup \{x^6\}$.*

Lorsque l'exploration est terminée, toutes les solutions efficaces $\{x^1, x^3, x^4, x^5, x^6\}$ ont été trouvées.

6.1.1 Particularités des PSE pour le problème de sac à dos mono-objectif

Les PSE pour le problème de sac à dos mono-objectif profitent de deux particularités de sa structure. Premièrement, une notion d'efficacité (définition 2.2 page 32) est associée à chaque objet et permet de les

comparer deux à deux d'une manière pertinente. Il est alors tout naturel, lors de l'exploration, de choisir les objets les plus efficaces en premier, au détriment des moins efficaces. Deuxièmement, plusieurs bonnes fonctions de calcul de bornes sont disponibles. En particulier, si les objets sont initialement triés en ordre décroissant d'efficacité, la borne U_0 présentée en page 38 est calculée en temps constant. On lui préfère néanmoins les bornes U_1 et U_2, donnant de meilleurs résultats pour un temps de calcul linéaire.

Dans le cas multi-objectif, l'efficacité d'un objet peut être calculée sur chaque objectif. On notera $e^i = \left(\frac{c_i^1}{w_i}, \ldots, \frac{c_i^p}{w_i}\right)$ celle de l'objet i. La comparaison de deux objets n'est plus immédiate. Cependant, une propriété de dominance peut être appliquée. Ainsi, si $e^i \geq e^j$, alors il est clair que l'objet i semble plus intéressant que l'objet j. Il devrait donc être choisi en premier. Par contre, si $e^i \ngeq e^j$ et $e^j \ngeq e^i$, alors l'efficacité ne permet pas de définir une priorité de choix entre i et j.

Les bornes disponibles en mono-objectif ne s'étendent pas immédiatement au cas multi-objectif. Classiquement, pour le premier, une borne inférieure est la valeur de la meilleure solution connue et une borne supérieure est obtenue en calculant la relaxation linéaire du problème. On obtient alors deux valeurs dans $\mathbb{R}$, comparables. Dans une situation multi-objectif, un ensemble de solutions se substitue à la meilleure solution et la relaxation linéaire multi-objectif est un ensemble bornant supérieurement. La comparaison n'est plus immédiate.

La section suivante discute de ces concepts d'ordre et de bornes dans une situation multi-objectif. Leur application sera faite dans deux procédures de séparation et évaluation présentées dans les sections suivantes.

6.2 Généralisation au cas multi-objectif du branchement et de l'évaluation d'un nœud

Cette section décrit les étapes d'une généralisation de la procédure de séparation et évaluation du sac à dos mono-objectif au cas multi-objectif. La base de cette généralisation est l'algorithme 6.1 ; une procédure de haut niveau pour résoudre une instance de $01MOKP$.

Nous pouvons remarquer que pour passer de cet algorithme à celui de la page 37 pour $01KP$, il suffit de détailler les points suivants

- l'ensemble $\bar{X}$ contient la meilleure solution connue, remplacée si nécessaire en ligne 6 ;
- la fonction `évaluation_prometteuse`$(\downarrow \mathfrak{n}, \downarrow \bar{X})$ à la ligne 7 équivaut à calculer une borne supérieure $\bar{z}$ et à la comparer avec la plus grande valeur de $z(x), x \in \bar{X}$;
- le choix de l'objet i en ligne 8 se réduit à sélectionner l'objet ayant la plus grande efficacité parmi les objets restants.

Ces points sont aussi les trois éléments à détailler pour le cas du problème $01MOKP$. L'écriture d'une procédure `nouvelle_solution`$(\downarrow x, \updownarrow \bar{X})$ est aisée ; il suffit de mettre à jour $\bar{X} \leftarrow \{x' \in \bar{X} \cup \{x\} : \nexists x'' \in \bar{X} \cup \{x\} : z(x') \leq z(x'')\}$. Par contre, le choix de l'objet i en ligne 8 et l'évaluation en ligne 7 ne sont pas des étapes triviales.

Dans la suite de cette section, nous présentons les notions de tris et de bornes étendues du cas mono-objectif au cas multi-objectif, destinés à être utilisés au niveau des deux lignes citées ci-dessus. Contrairement au cas mono-objectif, il n'existe pas, en général, d'ordre total sur les objets. Plusieurs tris sont ainsi proposés pour les ordonner dans l'optique de minimiser l'espace de recherche exploré. La seconde partie de cette section sera une discussion sur les bornes dans l'espace des objectifs. Une borne inférieure, utilisée pour décrire l'espace de recherche, et une borne supérieure, pour évaluer une solution incomplète.

1: **Procédure** `pse_mokp` ($\downarrow \mathfrak{n}, \updownarrow \bar{X}$)
2: Paramètre $\downarrow \mathfrak{n}$: le nœud à partir duquel se fait l'exploration.
3: Paramètre $\updownarrow \bar{X}$: ensemble des solutions potentiellement efficaces.
4: **si** $\sum_{j \in I^1(\mathfrak{n})} w_j \leq \omega$ **alors**
5: **si** $I^{\sim}(\mathfrak{n}) = \emptyset$ **alors**
6: `nouvelle_solution` ($\downarrow x(\mathfrak{n}), \updownarrow \bar{X}$)
7: **sinon si** `évaluation_prometteuse` ($\downarrow \mathfrak{n}, \downarrow \bar{X}$) **alors**
8: choisir $i \in I^{\sim}(\mathfrak{n})$
- - Construire les nœuds fils, le premier ne contenant pas l'objet i, l'autre le contenant - -
9: $\mathfrak{n}^1 \leftarrow$ `sélectionner`$(\downarrow \mathfrak{n}, \downarrow i, 1)$
10: $\mathfrak{n}^0 \leftarrow$ `sélectionner`$(\downarrow \mathfrak{n}, \downarrow i, 0)$
- - Séparer la recherche sur ces deux nœuds - -
11: `pse_mokp`$(\mathfrak{n}^1, \bar{X})$
12: `pse_mokp`$(\mathfrak{n}^0, \bar{X})$
13: **fin si**
14: **fin si**

ALG. 6.1 – Schéma d'une procédure de séparation et évaluation pour le problème $01MOKP$

L'ensemble des solutions atteignables dans le sous-arbre d'un nœud $\mathfrak{n}$ sera noté $X^T(\mathfrak{n})$, leurs images seront $Y^T(\mathfrak{n}) = \{z(x) : x \in X^T(\mathfrak{n})\}$. L'ensemble des points non dominés dans ce dernier sera noté $Y_N^T(\mathfrak{n}) = \{y \in Y^T(\mathfrak{n}) : \nexists y' \in Y^T(\mathfrak{n}), y \leq y'\}$. Les solutions associées à ces points seront notées $X_E^T(\mathfrak{n}) = \{x \in X^T(\mathfrak{n}) : z(x) \in Y_N^T(\mathfrak{n})\}$.

À nouveau, la précision « $(\mathfrak{n})$ » sera omise si cela ne provoque pas d'ambiguïté.

6.2.1 Notions de tri des variables en multi-objectif

L'intérêt de trier les variables *a priori* est de sélectionner en priorité celles pour lesquelles l'affectation d'une valeur permettra de réduire au maximum l'espace de recherche. C'est-à-dire que cette valeur amènera autant que possible vers des solutions efficaces. L'idéal, ou utopie, étant de ne faire que des choix amenant systématiquement à de telles solutions.

On notera π^j une permutation des objets obtenue en les triant dans l'ordre décroissant d'efficacité sur l'objectif j.

Dans [10], les auteurs observent l'influence de l'ordre de considération des objets sur les performances de leur algorithme. Les ordres présentés sont basés sur la position des objets dans les permutations π^j. Ainsi, si r_i^j est la position de l'objet i dans l'ordre π^j, les auteurs définissent π^{sum} une permutation obtenue selon les valeurs croissantes des sommes des positions des objets dans les p permutations π^j. De même, $\pi^{\max}$ est la permutation obtenue selon les valeurs croissantes de la plus mauvaise position des objets dans les p ordres π^j. Cette position est calculée pour l'objet i par $\max_{j=1,\ldots,p} \left\{ r_i^j \right\} + \frac{1}{pn} \sum_{j=1}^{p} r_i^j$, de manière à discriminer les objets ayant la même position maximum. D'une manière symétrique, $\pi^{\min}$ est la permutation obtenue selon les valeurs croissantes de la meilleure position des objets dans les p ordres π^j, calculée par $\min_{j=1,\ldots,p} \left\{ r_i^j \right\} + \frac{1}{pn} \sum_{j=1}^{p} r_i^j$.

i	1	2	3	4	5	6
e^1	0,25	0,25	0,71	1,8	2	3
e^2	1	0,25	0,85	0,4	1,25	4

Table 6.1 – Efficacités des objets de l'exemple 3.13 de la page 55

Nous définissons, de plus, les permutations suivantes, basées sur la relation de dominance sur l'efficacité des objets. π^{dom} est une permutation obtenue en mettant une contrainte de précédence sur un objet par rapport à ceux qu'il domine, en terme d'efficacité. $\pi^{\#}$ est obtenue en triant les objets par ordre croissant du nombre d'objets qui les dominent, en terme d'efficacité. Enfin, π^{rg} est obtenue selon le rang des objets en niveau de dominance, comme décrit par Srinivas et Deb pour l'algorithme NSGA [127]. C'est-à-dire que les objets dont l'efficacité n'est pas dominée sont placés en premiers, puis viennent ceux dont l'efficacité est non dominée parmi les objets restants, et ainsi de suite.

Enfin, π^{rnd} sera une permutation aléatoire des objets.

Exemple 6.23 (Tris des objets en multi-objectif). *Cet exemple reprend les données de l'exemple 3.13 présenté en page 55. Les efficacités des objets sont notées dans la table 6.1.*
La construction de π^{dom} se base sur la dominance entre les efficacités. D'après la définition, selon la précédence déduite de ces dominances, l'objet 6 doit être avant le 5, qui doit lui-même être avant les objets 1, 3 et 4, qui doivent eux-mêmes être avant l'objet 2. Ainsi, un ordre valide est $\pi^{dom} = (6, 5, 1, 3, 4, 2)$.
Remarquons que cette permutation est aussi valide pour $\pi^{\#}$ et π^{dom} ; mais ce n'est pas toujours le cas. De plus, plusieurs combinaisons peuvent être correctes pour un critère de tri donné. Ainsi, pour calculer π^{rg}, l'objet 6 est le seul dont l'efficacité n'est pas dominée. Il est donc placé en tête. Parmi les objets restants, seul 5 n'est pas dominé par un autre objet que le 6. Il se place donc en seconde position. Puis, après suppression de l'objet 5, les objets 1, 3 et 4 sont non dominés. Par conséquent, toute permutation de ces trois objets convient. Enfin, il ne reste que l'objet 2 à placer en fin.
Toujours selon les efficacités, une permutation vérifiant π^1 serait soit $(6, 5, 4, 3, 2, 1)$, soit $(6, 5, 4, 3, 1, 2)$; tandis que celle pour π^2 est $(6, 5, 1, 3, 4, 2)$. Dans un souci de simplification, π^1 sera choisi égal à $(6, 5, 4, 3, 2, 1)$ dans la suite de l'exemple.
Les positions des objet dans ces permutations sont alors $r^1 = (6, 5, 4, 3, 2, 1)$ et $r^2 = (3, 6, 4, 5, 2, 1)$.
La permutation $\pi^{sum} = (6, 5, 3, 4, 1, 2)$ se calcule immédiatement.
Pour calculer $\pi^{\min}$ et $\pi^{\max}$, il faut commencer par calculer les meilleures et pires positions des objets. Ces positions sont ici respectivement égales à $\left(3 + \frac{9}{12}, 5 + \frac{11}{12}, 4 + \frac{8}{12}, 3 + \frac{8}{12}, 2 + \frac{4}{12}, 1 + \frac{2}{12}\right)$ et $\left(6 + \frac{9}{12}, 6 + \frac{11}{12}, 4 + \frac{8}{12}, 5 + \frac{8}{12}, 2 + \frac{4}{12}, 1 + \frac{2}{12}\right)$. Après avoir trié ces positions en ordre croissant, les permutations résultantes sont $\pi^{\min} = (6, 5, 4, 1, 3, 2)$ et $\pi^{\max} = (6, 5, 3, 4, 1, 2)$.

6.2.2 Bornes dans un contexte multi-objectif

L'existence de bornes de bonne qualité est un élément important de toute procédure de séparation et évaluation. Classiquement, une borne supérieure est en effet calculée dans chaque nœud et comparée à une borne inférieure.

Pour que le nombre de nœuds visités soit faible et l'algorithme efficace, il faut d'une part que la borne inférieure soit au plus près des solutions efficaces et que les solutions du sous-arbres soient représentées au plus juste par la borne supérieure. D'autre part, comme leurs évaluations, ainsi que leur comparaison, sont faites dans chaque nœud, il faut que le calcul soit rapide. En effet, si le calcul de la borne est au moins aussi coûteux que l'exploration du sous-arbre, alors elle est inutile.

Malheureusement, la précision d'une borne est souvent d'autant plus faible qu'elle se calcule rapidement, en particulier sur le problème de sac à dos. De plus, le coût de comparaison des bornes doit être modéré. En effet, si cette comparaison est instantanée en mono-objectif, nous verrons qu'elle peut faire intervenir de nombreux points dans $\mathbb{R}^p$ en multi-objectif.

6.2.2.1 Borne inférieure en multi-objectif

D'après la définition 1.13 page 22, l'ensemble $\bar{Y}$ des points potentiellement non dominés connus à une étape de la procédure définit un ensemble bornant inférieurement pouvant tenir le rôle de borne inférieure.

Cette description est peu utilisable en pratique. Aussi, nous utiliserons par la suite la borne décrite dans la section 5.2.5 page 113, avec un poids λ à déterminer, ainsi que la description $D(\bar{Y})$ de la zone de recherche, sur laquelle elle s'appuie.

6.2.2.2 Borne supérieure en multi-objectif et comparaison avec la borne inférieure

Plusieurs approches sont envisageables pour calculer une borne supérieure multi-objectif, plus ou moins coûteuses en terme de calcul et de comparaison avec la borne inférieure.

Point utopique : Une première façon d'évaluer les solutions du sous-arbre est de calculer un point localement utopique y^U qui, par définition, domine tous les points de Y^T. Il suffit alors de le comparer avec les points de $\bar{Y}$. Si $\exists y \in \bar{Y} : y \geq y^U$, alors les performances de toutes les solutions du sous-arbres sont dominées et le nœud doit être fermé. Autrement, l'exploration doit continuer.

Pour calculer un point utopique, il suffit par exemple de calculer une des bornes U_0, U_1 ou U_2 sur chaque objectif pris individuellement (voir la section 4.2.2 page 68).

Enveloppe bornant supérieurement : Une généralisation stricte d'une pratique des algorithmes mono-objectifs consiste à résoudre la relaxation linéaire du problème à l'aide d'un solveur linéaire multi-objectif (voir [13, 14, 15] ou encore [43]). Le résultat est un ensemble d'hyperplans d'équations $\lambda \cdot y = \alpha, \forall y \in Y^T$, avec $\lambda \in \mathbb{R}^p, \alpha \in \mathbb{R}$. Par la suite, une telle enveloppe sera décrite comme suit :

$$H = \{(\lambda, \alpha) : \lambda \in \mathbb{R}^p, \alpha \in \mathbb{R} : \lambda \cdot y \leq \alpha : \forall y \in Y^T\}$$

Il suffit alors de comparer ces hyperplans avec $D(\bar{Y})$. Si $\exists d \in D(\bar{Y})$ tel que $\forall (\lambda, \alpha) \in H : \lambda \cdot d \leq \alpha$ alors il existe potentiellement une solution $x \in X_E^T$ telle que $z(x) \geq d$ et l'exploration doit continuer dans le sous-arbre. Autrement, tous les points $y \in Y^T$ sont dominés et le nœud doit être fermé.

Cette méthode est précise, mais aussi très coûteuse. Une façon de procéder, qui n'enlève rien au coût, est d'appliquer la première phase de la méthode en deux-phases (voir les chapitres 4 et 5) pour obtenir Y_{SN}^T et en déduire H.

Ensemble bornant adaptatif : Remarquons que l'ensemble d'hyperplans basé sur un point utopique local y^U et constitué des équations $\lambda_{|j} \cdot z = y_j^U, \forall j \in \{1, \ldots, p\}$ est une enveloppe bornant supérieuremnt Y^T, avec $\lambda_{|j}$ le vecteur de taille p où seule la composante j est à 1, les autres étant à 0.

De manière à réduire les temps de calcul, il peut être intéressant de commencer par utiliser cette enveloppe comme évaluation initiale, puis de l'affiner en ajoutant des hyperplans. Ces derniers peuvent être obtenus en exécutant une dichotomie jusqu'à un certain niveau de profondeur, ou en calculant les

relaxations linéaires des problèmes mono-objectifs sur une famille de poids Λ. En effet, si U est une fonction de relaxation pour $01KP$, alors Y^T est couvert par l'enveloppe $\{(\lambda, U(P_\lambda)) : \lambda \in \Lambda\}, \forall \Lambda \in \mathbb{R}^p_{\geq}$.

6.3 Procédure de séparation et évaluation en profondeur d'abord

La première procédure de séparation et évaluation effectue une exploration en profondeur d'abord, en affectant les valeurs 1 puis 0 aux variables, sélectionnées dans un ordre parmi ceux décrits précédemment. Cet ordre définit une priorité de choix, mais plusieurs objets non adjacents peuvent être sélectionnés simultanément dans un nœud par application des propriétés 3.2 et 3.3 pages 54. À cette fin, les ensembles $Pref(v^i)$ et $Dom(v^i)$ sont calculés à l'initialisation de la procédure.

Cette procédure est une application directe du schéma de l'algorithme 6.1.

6.3.1 Évaluation du nœud

L'évaluation d'un nœud par la fonction `évaluation_prometteuse` se fait en trois étapes, au plus. Le premier test porte sur la cardinalité de la solution (voir la section 3.2.1 page 53). Nous notons $UB(\omega, I)$ le résultat du calcul de $UB(\omega)$ sur les objets de l'ensemble I. Si $UB(\bar{\omega}, I^\sim) + |I^1| < LB(\omega)$, alors les solutions du sous-arbre n'auront pas la cardinalité minimum attendue des solutions efficaces. Son exploration n'amènera à aucune solution efficace et le nœud est fermé.

Les deux étapes suivantes dans l'évaluation consistent à tester une enveloppe supérieure adaptative du sous-arbre. Un point utopique y^U local est calculé en évaluant la borne U_2 sur chaque objectif pris individuellement. Si ce point est dominé, alors le nœud est fermé. Autrement, un unique problème P_λ est construit et une borne supérieure mono-objectif $\bar{z}$ est calculée en utilisant U_2. Les points de $D(\bar{Y})$ situés sous y^U (c'est à dire dominés par y^U) sont évalués au regard de la pondération λ. Si $\lambda \cdot u > \bar{z}, \forall u \in D(\bar{Y})$, alors le nœud est fermé, il n'amènera à aucune solution efficace. La valeur λ utilisée dans ce dernier cas est arbitrairement choisie telle que $\lambda_i = \frac{y^U_i}{||y^U||}$ de manière à maximiser vers le point utopique.

6.3.2 Choix d'un objet

Les objets sont ordonnés à l'initialisation de l'algorithme selon une des permutations présentées dans la section 6.2.1. Le choix consiste dès lors à sélectionner le premier objet disponible selon cet ordre.

6.3.3 Séparation

Lors du branchement sur $\mathfrak{n}$ pour construire $\mathfrak{n}^1$, la procédure diffuse la valeur 1 sur les variables correspondant à $Pref(v^i)$:

$$I^1(\mathfrak{n}^1) \leftarrow I^1(\mathfrak{n}) \cup \{i\} \cup Pref(v^i)$$

Les variables libres qui ne peuvent plus rentrer dans le sac suite à ces affectations sont ensuite mises à zéro :

$$I^0(\mathfrak{n}^1) \leftarrow I^0(\mathfrak{n}) \cup \{j \in I^\sim(\mathfrak{n}) \setminus \{i\} : w_j > \bar{\omega}(\mathfrak{n}) - w_i\}$$

Puis le nœud est évalué pour déterminer s'il peut potentiellement amener à une solution efficace.

De même, lors du branchement sur $\mathfrak{n}$ pour construire $\mathfrak{n}^0$, la procédure diffuse la valeur 0 sur les variables correspondant à $Dom(v^i)$:

$$I^0(\mathfrak{n}^0) \leftarrow I^0(\mathfrak{n}) \cup \{i\} \cup Dom(v^i)$$

Le nœud est ensuite évalué de la même manière que pour le branchement à 1.

6.3.4 Expérimentations numériques

L'algorithme 6.1 a été implémenté en C++ et appliqué sur les instances de sac à dos tri-objectif décrites dans la section 3.4.1.2. L'ordinateur exécutant le programme dispose d'un Pentium 4 cadencé à 3,73 GHz et de 3 Go de mémoire vive. Le temps de résolution est limité à une heure par instance.

Les figures 6.2 à 6.4 présentent respectivement les temps minimaux, moyens et maximaux obtenus sur l'ensemble des dix instances de chacune des tailles disponibles pour le groupe D-1.

Il ressort de ces résultats que l'ordre de sélection des objets influe grandement sur le temps de résolution. L'ordre π^{rnd}, correspondant à une sélection aléatoire des objets, donne régulièrement des résultats parmi les plus mauvais. Par contre, au moins un des trois ordres parmi π^1, π^2 et π^3 donne les meilleurs résultats, que ce soit en terme de temps minimal, moyen ou maximal. Ces résultats confirment la nécessité d'obtenir un ordre favorable à l'initialisation de l'algorithme. Le calcul d'un tel ordre n'est pas trivial, en témoignent les résultats obtenus avec l'ordre $\pi^{\max}$ qui, à quelques exceptions près, est moins efficace que π^{rnd}.

Les ordres π^{dom}, π^{rg} et $\pi^{\#}$, basés sur la dominance en terme d'efficacité, donnent des résultats mitigés, souvent proches et pas tellement meilleurs que ceux obtenus avec π^{rnd}.

Remarquons que, hormis π^1, π^2 et π^3, $\pi^{\min}$ est le plus souvent l'ordre donnant les meilleurs résultats. Même si un ordre parmi π^1, π^2 et π^3 correspond toujours ici aux meilleures performances, il n'est pas évident de déterminer *a priori* lequel des trois doit être choisi. En particulier, π^1 donne de mauvais résultats en moyenne pour les instances de taille 60 et 70. Au contraire, $\pi^{\min}$ a l'avantage, par rapport à ces derniers, d'être clairement défini et régulier sur les instances utilisées ici.

Les instances de taille supérieure à 70 variables n'ont pas été résolues dans le temps imparti, à l'exception d'une instance de taille 80 avec l'ordre $\pi^{\min}$.

Les figures 6.5 à 6.7 présentent les résultats obtenus avec ce même algorithme pour les instances du groupe D-2.

Les performances des ordres sont similaires à celles obtenues sur le groupe D-1. En particulier, $\pi^{\max}$ donne des temps de résolution plus longs que π^{rnd}, et $\pi^{\min}$ est souvent parmi les ordres donnant les meilleurs temps ; c'est même le meilleur en moyenne.

Les temps de résolution augmentent énormément par rapport au groupe D-1, attestant de la difficulté de résolution introduite par la corrélation des objectifs et du vecteur de poids. Ainsi, les instances de taille supérieure à 40 variables n'ont pas été résolues dans le temps imparti, à l'exception de trois instances de taille 50 avec l'ordre π^2 et une de cette même taille avec $\pi^{\min}$.

Les figures 6.8 et 6.9 présentent le nombre de nœuds visités pour chaque taille d'instance du groupe D-1, avec les ordres $\pi^{\min}$ et $\pi^{\max}$. Pour chacune de ces tailles, le nombre de nœuds fermés par les critères de la fonction évaluation (section 6.3.1) sont indiqués en terme de rapport sur le nombre total visité [†].

Nous remarquons que le nombre de nœuds vus augmente significativement avec l'ordre $\pi^{\max}$ par rapport à $\pi^{\min}$, d'un facteur pouvant dépasser 5 pour les instances les plus grandes.

Le test de dominance sur un point utopique local à chaque nœud est clairement le plus efficace de tous les tests. À l'opposé, le test portant sur la cardinalité de la solution est pratiquement inutile. L'évaluation d'une relaxation du problème P_λ ferme quelques nœuds supplémentaires. Cette quantité peut sembler

†. Attention à l'échelle logarithmique.

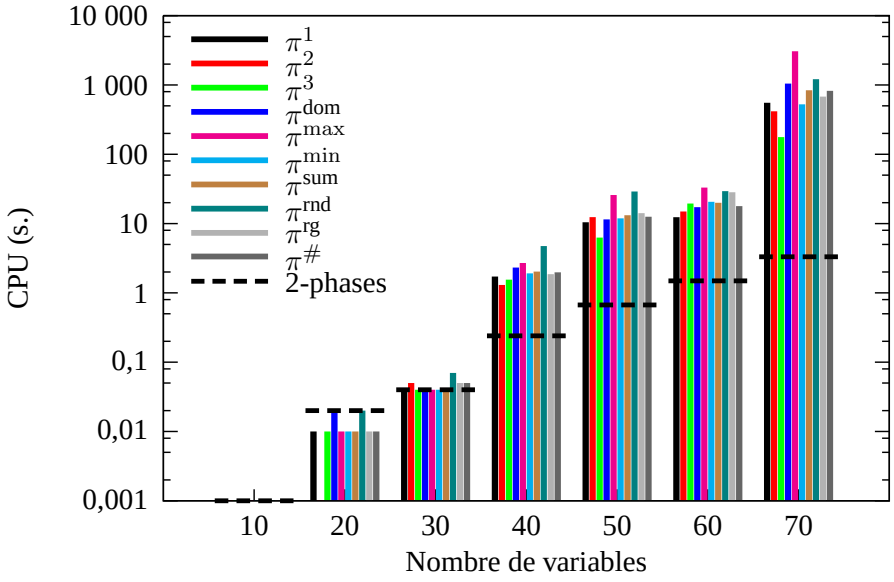

Figure 6.2 – Temps minimal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-1 avec la procédure `pse_mokp`. Si cette page est imprimée en noir et blanc : pour chaque taille, les barres observées de gauche à droite correspondent à la légende lue de haut en bas.

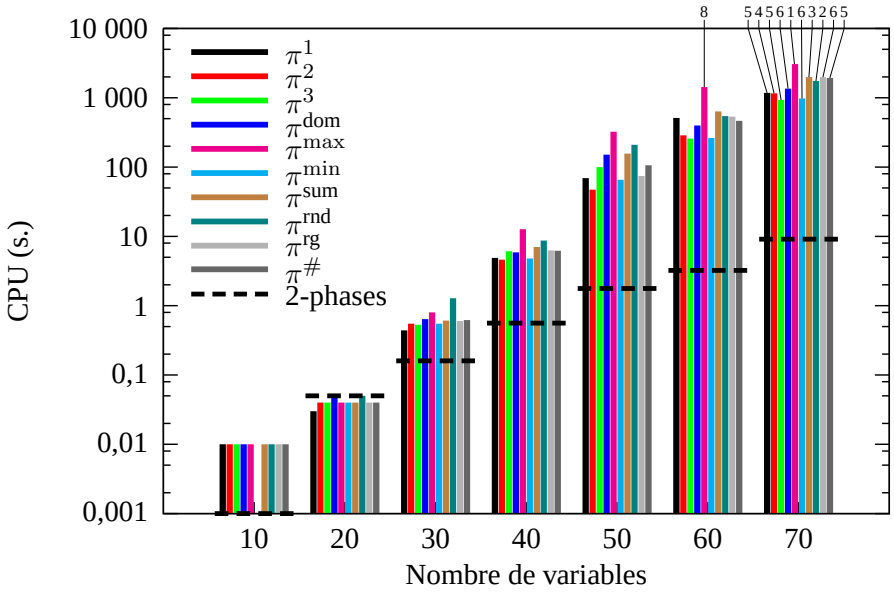

Figure 6.3 – Temps moyen obtenu sur les instances du groupe D-1 avec la procédure `pse_mokp`. Lorsque certaines instances n'ont pas été résolues dans le délai de temps, le nombre d'instances sur lequel porte le calcul est indiqué au dessus du graphique.

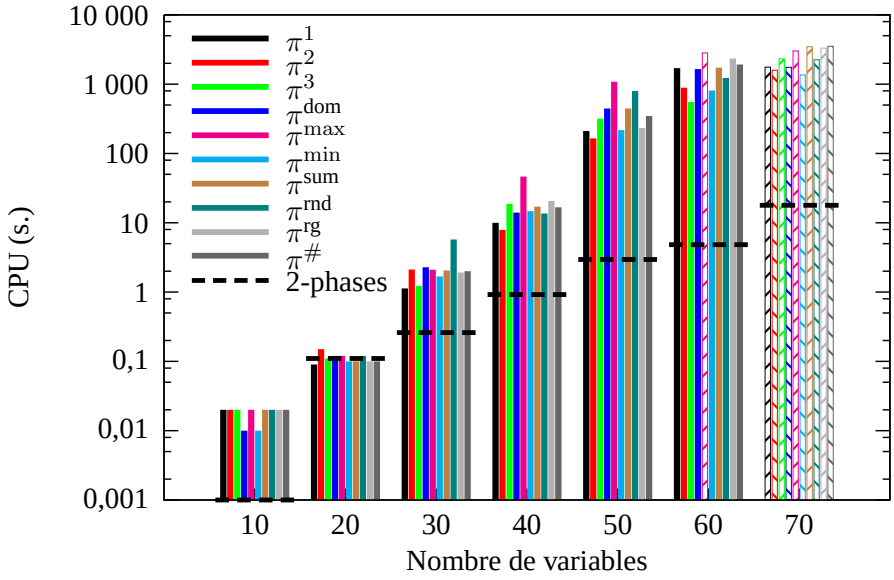

Figure 6.4 – Temps maximal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-1 avec la procédure `pse_mokp`. Lorsque les dix instances d'une taille donnée n'ont pas été résolues dans le délai de temps, le temps maximum parmi celles résolues est donné à titre indicatif sous forme d'une barre hachurée.

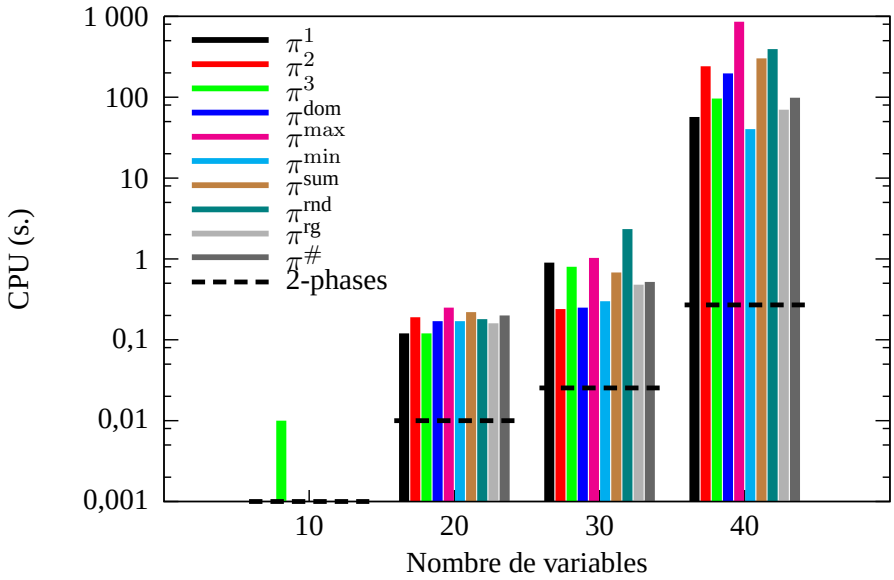

Figure 6.5 – Temps minimal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-2 avec la procédure `pse_mokp`.

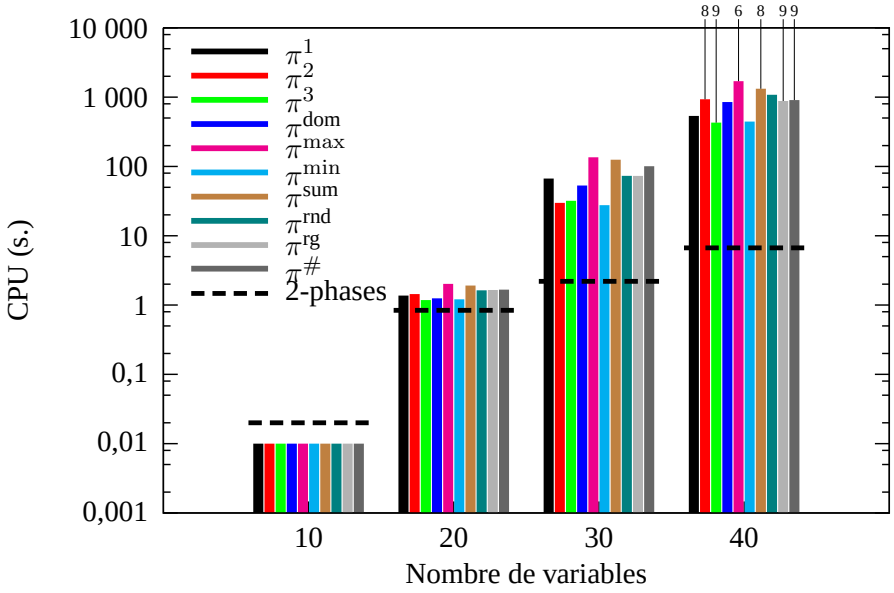

Figure 6.6 – Temps moyen obtenu sur les instances du groupe D-2 avec la procédure `pse_mokp`. Lorsque certaines instances n'ont pas été résolues dans le délai de temps, le nombre d'instances sur lequel porte le calcul est indiqué au dessus du graphique.

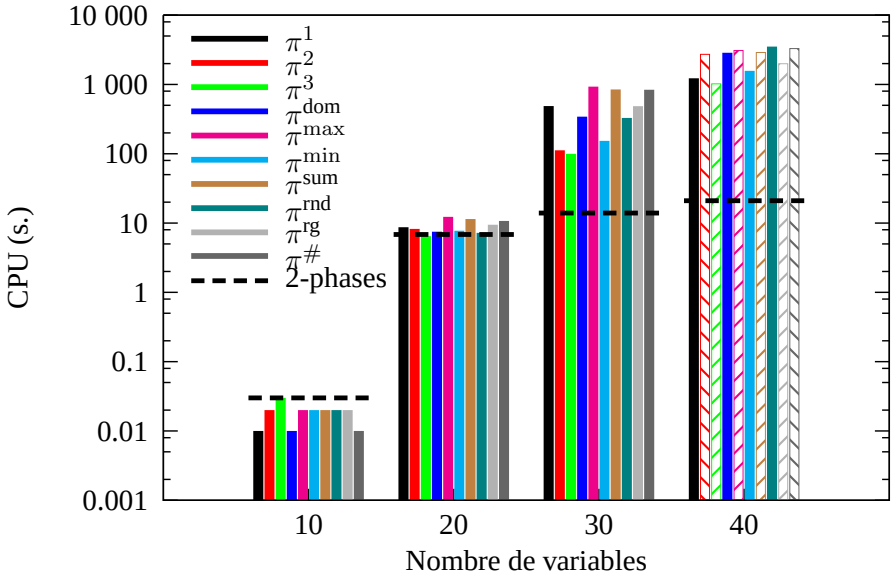

Figure 6.7 – Temps maximal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-2 avec la procédure `pse_mokp`. Lorsque les dix instances d'une taille donnée n'ont pas été résolues dans le délai de temps, le temps maximum parmi celles résolues est donné à titre indicatif sous forme d'une barre hachurée.

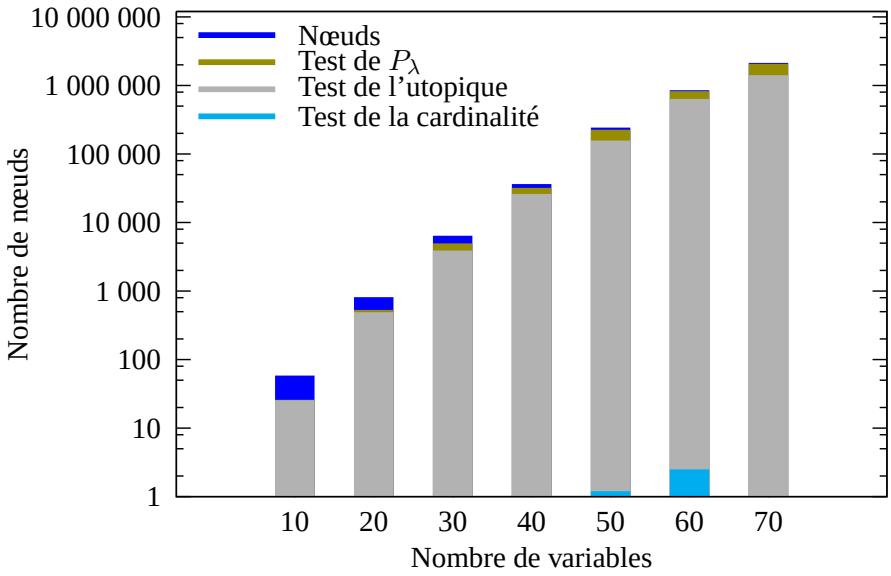

Figure 6.8 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-1, en utilisant l'ordre $\pi^{\min}$ avec la procédure `pse_mokp`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.3.1 sont aussi représentés. Si cette page est imprimée en noir et blanc : pour chaque taille, les couleurs des barres, prises de haut en bas, correspondent à la légende lue dans ce même sens.

faible, cependant il est utile de rappeler que ce test n'est effectué que sur les nœuds non fermés par le test sur le point utopique.

Globalement, il ressort de ces résultats que le nombre de nœuds amenant à une solution (c'est-à-dire ceux qui n'ont pas été fermés) est extrêmement faible par rapport au nombre total de nœuds visités. Ce résultat indique que la procédure d'exploration ne se dirige pas de manière optimale vers les solutions efficaces.

Les figures 6.10 et 6.11 présentent les mêmes données pour les instances du groupe D-2. Le rapport du nombre de nœuds visités entre $\pi^{\min}$ et $\pi^{\max}$ est du même ordre de grandeur que pour les instances du groupe D-1. Le rapport du nombre de nœuds fermés par chaque test sur le nombre de nœud visités est lui aussi semblable à celui obtenus sur les instances du groupe D-1.

Le nombre total de nœuds visités augmente grandement selon le type de l'instance, pour une taille d'instance donnée. Ainsi, en utilisant $\pi^{\min}$, ce nombre est en moyenne 16 fois plus grand pour les instances de taille 40 de D-2 par rapport à D-1. Si $\pi^{\max}$ est utilisé, ce rapport monte jusqu'à 27 pour cette même taille et jusqu'à 30 pour les instances de taille 30. Ceci est compréhensible du fait que, comme nous l'avons observé dans la section 5.4, le nombre de solutions pour les instances du groupe D-2 est bien plus grand, pour une taille donnée, que pour les instances du groupe D-1. Par conséquent, la procédure d'exploration ira plus souvent au fond de l'arbre pour ces instances corrélées, augmentant ainsi le nombre de nœuds.

De toute évidence, cette PSE est largement dominée par la procédure en deux phases.

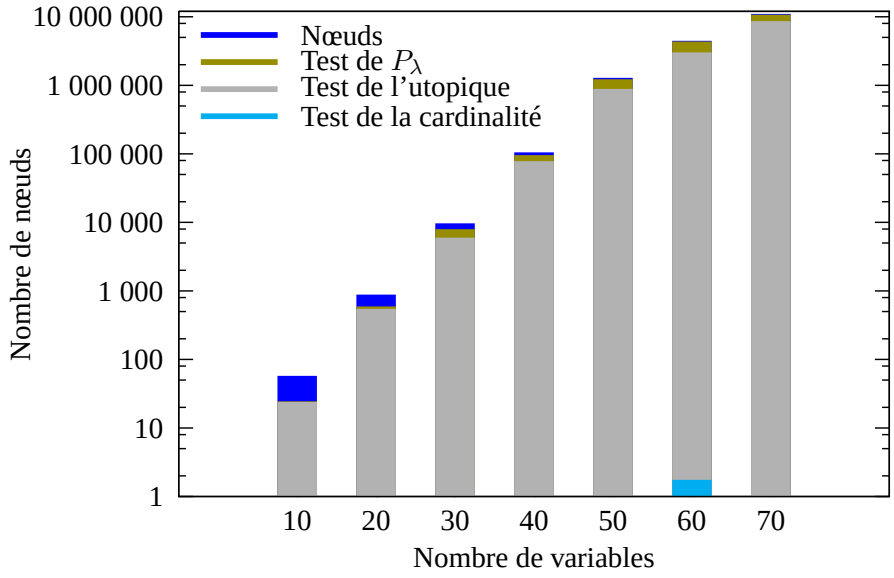

Figure 6.9 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-1, en utilisant l'ordre $\pi^{\max}$ avec la procédure `pse_mokp`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.3.1 sont aussi représentés.

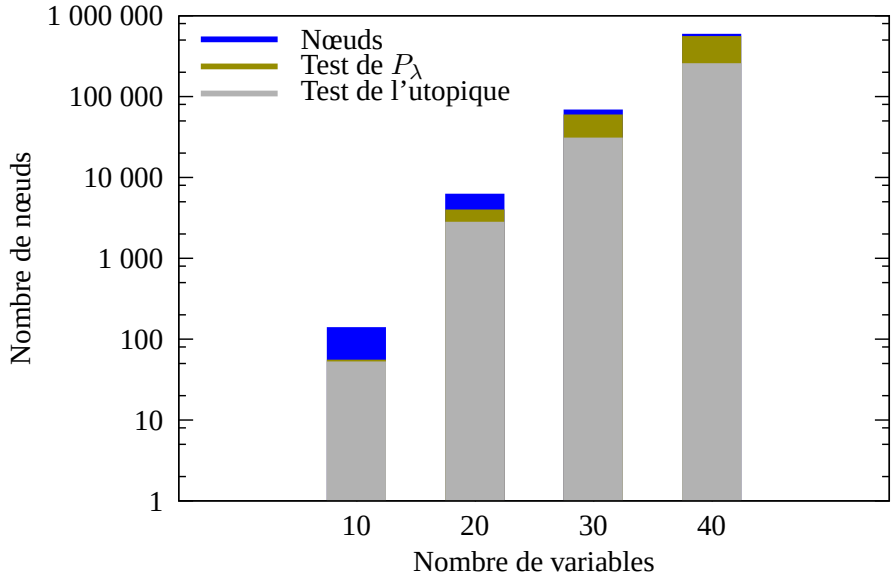

Figure 6.10 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-2, en utilisant l'ordre $\pi^{\min}$ avec la procédure `pse_mokp`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.3.1 sont aussi représentés.

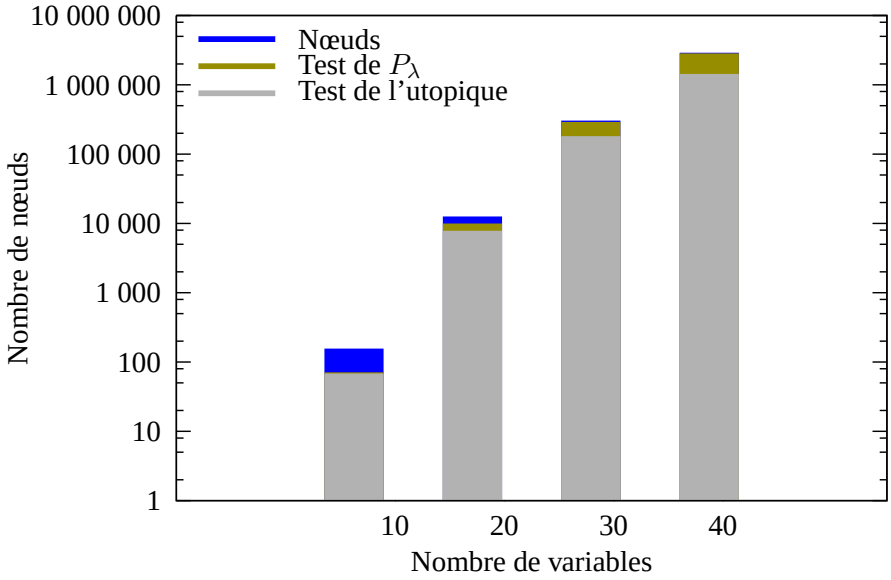

Figure 6.11 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-2, en utilisant l'ordre $\pi^{\max}$ avec la procédure `pse_mokp`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.3.1 sont aussi représentés.

6.4 Procédure de séparation et évaluation à branchement mixte

Les expérimentations numériques indiquent que la procédure 6.1 éprouve des difficultés à résoudre en un temps raisonnable des instances qui sont malgré tout d'une taille relativement petite. De plus, il ressort que l'ordre de sélection des objets a une influence importante sur les performances de l'algorithme.

Une des faiblesses de cette procédure est la séparation des nœuds, aux lignes 9 à 12. En effet, aucune évaluation des nœuds $\mathfrak{n}^1$ et $\mathfrak{n}^0$ n'est effectuée pour déterminer l'ordre des appels récursifs. Par exemple, si les solutions trouvées lors du second appel sont meilleures que celles du premier, si cela était remarqué *a priori*, alors il serait pertinent d'inverser les appels.

Dans cette section nous présentons la procédure 6.2, variante de l'algorithme 6.1 au niveau de la séparation des nœuds. Les modifications concernent l'évaluation des nœuds, le choix de l'objet sur lequel effectuer la séparation, ainsi que l'ordre dans lequel les nœuds sont traités.

L'algorithme procède à la séparation des nœuds en les traitant selon un critère de qualité décroissant. Les nœuds créés lors de la séparation sont ajoutés à une file d'attente $\mathcal{N}$ triée de telle façon que les nœuds les plus intéressants soient en tête. Ce tri se fait lexicographiquement selon 5 critères présentés plus bas.

6.4.1 Évaluation du nœud

L'évaluation des nœuds se fait d'une manière similaire à ce qui est présenté dans la section 6.3.1 à deux détails près. Tout d'abord, le test de l'enveloppe supérieure adaptative ne se fait plus en utilisant un point utopique mais en utilisant un point idéal y^I, obtenu en résolvant les problèmes P_i comme indiqué dans la section 1.2.4 page 19. Si ce point est dominé, alors le nœud est fermé. Autrement, un unique problème P_λ est construit et une borne supérieure mono-objectif $\bar{z}$ est calculée en utilisant U_2. Les points de $D(\bar{Y})$ situés sous y^I sont évalués au regard de la pondération λ. Si $\lambda \cdot d > \bar{z}, \forall d \in D(\bar{Y})$,

1: **Procédure** `pse_mokp_2` ($\uparrow \bar{X}$)
2: Paramètre $\uparrow \bar{X}$: ensemble complet maximal des solutions efficaces.

- - $\mathfrak{n}^s$ est la racine de l'arbre, avec $I^0(\mathfrak{n}^s) = I^1(\mathfrak{n}^s) = \emptyset$ - -
3: $\mathcal{N} \leftarrow \{\mathfrak{n}^s\}$
4: **tant que** $\mathcal{N} \neq \emptyset$ **faire**
5: $\mathfrak{n} \leftarrow$ `tête`($\mathcal{N}$)
6: $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\mathfrak{n}\}$
7: **si** $I^{\sim}(\mathfrak{n}) = \emptyset$ **alors**
8: `nouvelle_solution` $(\downarrow x(\mathfrak{n}), \updownarrow \bar{X})$
9: **sinon**
- - Le choix est fait selon l'ordre initial - -
10: choisir $i \in I^{\sim}(\mathfrak{n})$
- - Construire les nœuds fils, le premier ne contenant pas l'objet i, l'autre le contenant - -
11: $\mathfrak{n}^1 \leftarrow$ `sélectionner`$(\downarrow \mathfrak{n}, \downarrow i, 1)$
12: $\mathfrak{n}^0 \leftarrow$ `sélectionner`$(\downarrow \mathfrak{n}, \downarrow i, 0)$
13: **si** `évaluation_prometteuse` $(\downarrow \mathfrak{n}^1, \updownarrow \bar{X})$ **alors**
14: $\mathcal{N} \leftarrow \mathcal{N} \cup \{\mathfrak{n}^1\}$
15: **fin si**
16: **si** `évaluation_prometteuse` $(\downarrow \mathfrak{n}^0, \updownarrow \bar{X})$ **alors**
17: $\mathcal{N} \leftarrow \mathcal{N} \cup \{\mathfrak{n}^0\}$
18: **fin si**
- - Maintenir l'ordre des nœuds - -
19: `ordonner`$(\updownarrow \mathcal{N})$
20: **fin si**
21: **fin tant que**

ALG. 6.2 – Schéma d'une procédure de séparation et évaluation pour le problème $01MOKP$

alors le nœud est fermé, il n'amènera à aucune solution efficace. La valeur λ utilisée dans ce cas est arbitrairement choisie telle que $\lambda_i = \frac{y_i^I}{||y^I||}$ de manière à maximiser vers le point idéal.

Si le nœud n'a pas encore été fermé, une évaluation supplémentaire est effectuée. Les solutions supportées $X_{SE}^T(\mathfrak{n})$ du sous-arbre sont calculées à l'aide de la dichotomie utilisée dans la première phase de la méthode en deux phases décrite dans le chapitre 5 (voir la section 5.1 page 93). Puis, l'enveloppe convexe de Y_{SN} est comparé avec $D(\bar{Y})$. Soit H un ensemble d'hyperplans décrivant cette enveloppe. Si pour tout $d \in D(\bar{Y})$ tel que $d < y^I$, il existe un hyperplan $(\lambda, \alpha) \in H$ tel que $\lambda \cdot d > \alpha$, alors le nœud est fermé, il n'amènera a aucune solution efficace.

De plus, nous profitons du calcul des solutions supportées $X_{SE}^T(\mathfrak{n})$ du sous-arbre pour mettre à jour $\bar{X}$ avec de nouvelles solutions potentiellement efficaces.

6.4.2 Choix d'un objet

Tout comme pour la procédure 6.1 les objets sont triés initialement selon une des permutations présentées dans la section 6.2.1. Le choix consiste dès lors à sélectionner le premier objet disponible selon cet ordre. De plus, nous sélectionnons en priorité les objets réguliers dans les solutions supportées du sous-arbre du nœud traité (définition 3.1 page 52). C'est-à-dire, si π est la permutation des objets, alors l'objet choisi pour la séparation est π_j tel que :

$$j = \min\left\{i : \pi_i \in I^\sim : \nexists x, x' \in X_{SE}^T : x_{\pi_i} = 1 \wedge x'_{\pi_i} = 0\right\}$$

Si aucun indice ne satisfait cette condition, c'est-à-dire que pour tous les objets disponibles, il existe une solution supportée dans le sous-arbre contenant cet objet et une autre ne le contenant pas, alors la séparation se fait sur le premier objet disponible selon la permutation choisie, de la même manière que pour l'algorithme 6.1.

6.4.3 Ordre de traitement des nœuds

Les nœuds sont traités selon une notion de qualité décroissante. Soit $\mathfrak{n}, \mathfrak{n}'$ deux nœuds à comparer. La comparaison se fait lexicographiquement selon les 5 critères ci-dessous. Le nœud maximal selon ceux-ci est ensuite choisi pour être traité.

1. Les problèmes de sac à dos dont le *tightness ratio* (définition 2.1 page 32) est éloigné de 0,5 font intervenir une combinatoire faible et sont par conséquent plus faciles à résoudre. Les nœuds pour lesquels le ratio résiduel vérifie cette propriété seront donc traités en priorité :

$$\left|0,5 - \frac{\sum_{i \in I^\sim(\mathfrak{n})} w_i}{\bar{\omega}(\mathfrak{n})}\right| > \left|0,5 - \frac{\sum_{i \in I^\sim(\mathfrak{n}')} w_i}{\bar{\omega}(\mathfrak{n}')}\right| \Rightarrow \mathfrak{n} > \mathfrak{n}'$$

2. Nous considérons qu'un nœud est d'autant plus intéressant que les solutions supportées obtenues dans son sous-arbre sont non dominées :

$$\left|\left\{y \in Y^T(\mathfrak{n}) : \nexists y' \in \bar{Y} : y \leq y'\right\}\right| > \left|\left\{y \in Y^T(\mathfrak{n}') : \nexists y' \in \bar{Y} : y \leq y'\right\}\right| \Rightarrow \mathfrak{n} > \mathfrak{n}'$$

3. Ensuite, le nœud dont le sous-arbre amène au plus de solutions supportées est placé devant :

$$\left|X_{SE}^T(\mathfrak{n})\right| > \left|X_{SE}^T(\mathfrak{n}')\right| \Rightarrow \mathfrak{n} > \mathfrak{n}'$$

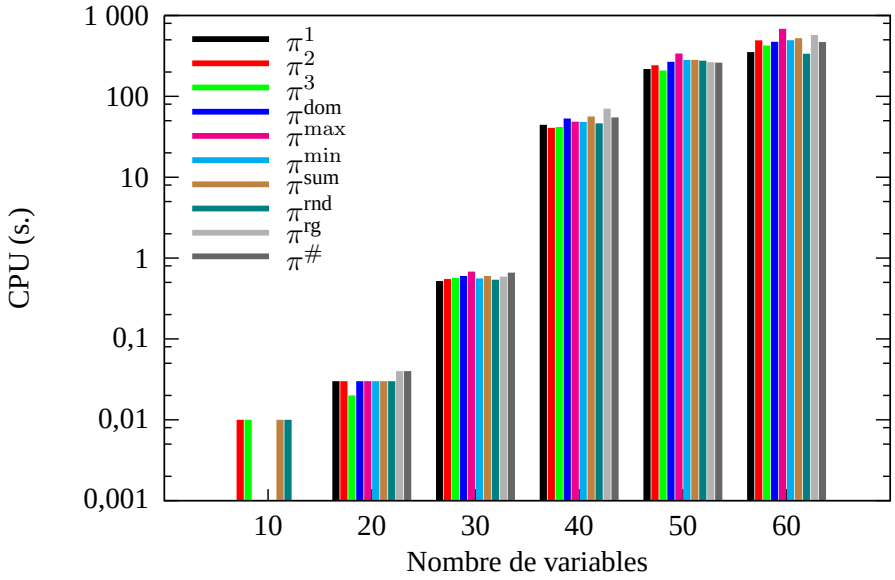

Figure 6.12 – Temps minimal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-1. Si cette page est imprimée en noir et blanc : pour chaque taille, les barres observées de gauche à droite correspondent à la légende lue de haut en bas.

4. Puis les nœuds dont la capacité résiduelle est la plus faible sont mis en avant :

$$\bar{\omega}(\mathfrak{n}) < \bar{\omega}(\mathfrak{n}') \Rightarrow \mathfrak{n} > \mathfrak{n}'$$

5. Le dernier critère place en priorité les nœuds dont la solution en cours de construction a la plus grande cardinalité :

$$\left|I^1(\mathfrak{n})\right| > \left|I^1(\mathfrak{n}')\right| \Rightarrow \mathfrak{n} > \mathfrak{n}'$$

6.4.4 Expérimentations numériques

L'algorithme 6.2 a été implémenté en C++ et appliqué sur les instances de sac à dos tri-objectif décrites dans la section 3.4.1.2. L'ordinateur exécutant le programme dispose d'un Pentium 4 cadencé à 3,73 GHz et de 3 Go de mémoire vive. Le temps de résolution est limité à une heure par instance.

Les figures 6.12 à 6.14 présentent respectivement les temps minimaux, moyens et maximaux obtenus sur l'ensemble des dix instances de chacune des tailles disponibles pour le groupe D-1.

Nous pouvons remarquer ce second algorithme peine à résoudre certaines instances du groupe D-1 dans le temps imparti, dès 40 variables. En effet, les temps de résolution ont énormément augmenté avec cet algorithme et aucune instance de 70 variables ou plus n'a été résolue. Néanmoins, il ressort de ces graphiques que les écarts de temps entre les différents ordres de sélection des objets est bien moindre que pour l'algorithme 6.1. Ainsi, les critères de branchement décrits dans la section 6.4.3 semblent palier au caractère difficilement comparable de l'efficacité des objets en multi-objectif.

Les figures 6.15 à 6.17 présentent les résultats obtenus avec ce même algorithme pour les instances du groupe D-2.

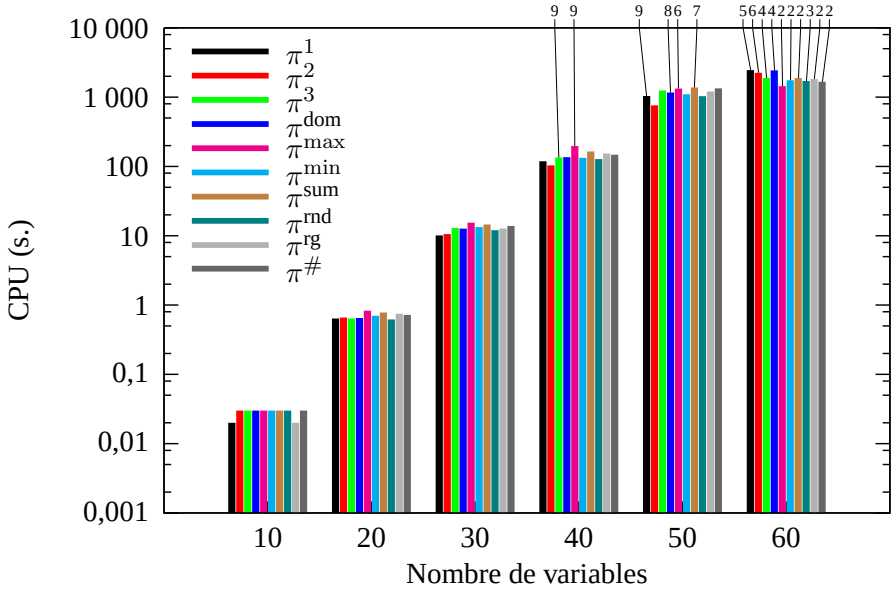

Figure 6.13 – Temps moyen obtenu sur les instances du groupe D-1 avec la procédure `pse_mokp_2`. Lorsque certaines instances n'ont pas été résolues dans le délai de temps, le nombre d'instances sur lequel porte le calcul est indiqué au dessus du graphique.

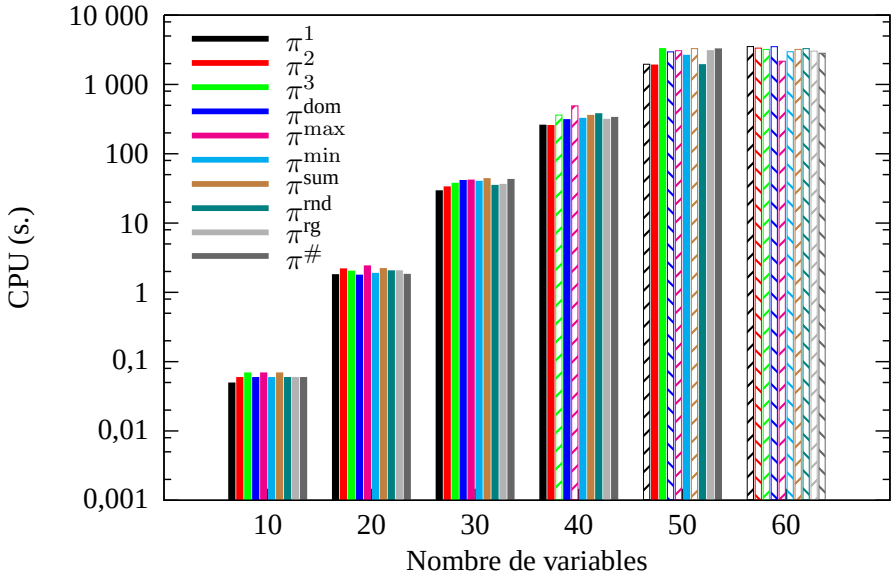

Figure 6.14 – Temps maximal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-1 avec la procédure `pse_mokp_2`. Lorsque les dix instances d'une taille donnée n'ont pas été résolues dans le délai de temps, le temps maximum parmi celles résolues est donné à titre indicatif sous forme d'une barre hachurée.

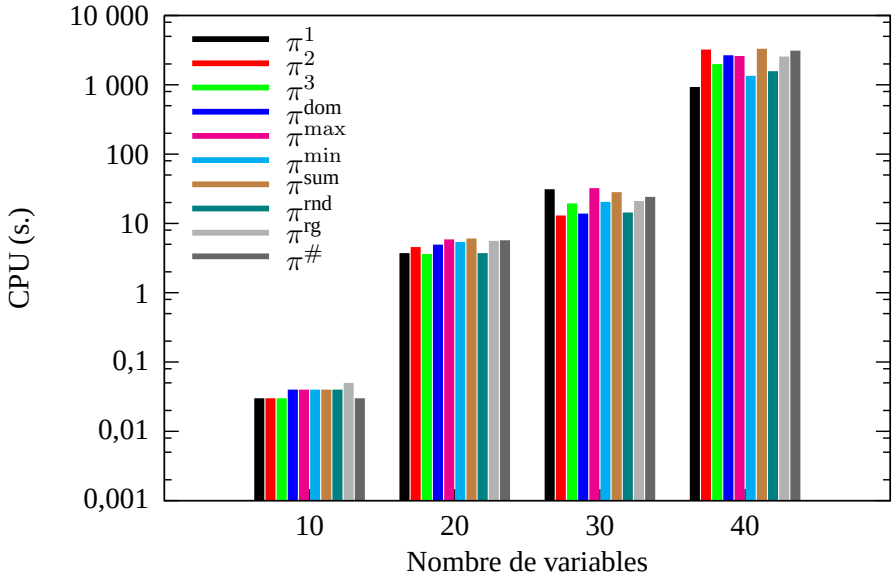

Figure 6.15 – Temps minimal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-2 avec la procédure `pse_mokp_2`.

Nous retrouvons ici des résultats similaires à ceux obtenus pour les instances du groupe D-1. Les temps de résolution ont augmenté et peu d'instances sont résolues à partir de 40 variables. Nous retrouvons aussi une homogénéisation des temps entre les différents ordres de sélection des objets par rapport aux résultats obtenus avec l'algorithme 6.1.

Les figures 6.18 et 6.19 présentent le nombre de nœuds visités pour chaque taille d'instance du groupe D-1, avec les ordres $\pi^{\min}$ et $\pi^{\max}$. Pour chacune de ces tailles, le nombre de nœuds fermés par les critères de la fonction évaluation (section 6.3.1) sont indiqués en terme de rapport sur le nombre total visité ‡.

Nous remarquons que l'écart du nombre de nœuds entre les deux ordres est beaucoup plus faible que pour la procédure `pse_mokp`. De plus, le nombre de nœud exploré est plus faible d'un facteur d'environ 10 par rapport à cette dernière. Les instances de taille 60 semblent produire moins de nœuds en moyenne que celle de taille 50, néanmoins il est utile de rappeler que peu de ces premières ont été résolues dans le temps imparti. La moyenne porte donc sur moins d'instances.

À nouveau, le test de dominance sur un point utopique local à chaque nœud est le plus efficace de tous les tests. À l'opposé, le test portant sur la cardinalité de la solution est pratiquement inutile. Enfin, l'utilisation de $\text{conv}(Y_{SN}^T)$ pour borner les solutions accessibles depuis un nœud permet de fermer, globalement, à peu près autant de nœuds que l'évaluation d'une relaxation du problème P_λ. Ces valeurs peuvent sembler proches, nous rappelons néanmoins que ce premier test n'est effectué que sur les nœuds non fermés par le second.

Les figures 6.20 et 6.21 présentent les mêmes données pour les instances du groupe D-2. Le rapport du nombre de nœuds visités comparé à la procédure `pse_mokp` est à nouveau autour de 10. De plus, Le

‡. Attention à l'échelle logarithmique.

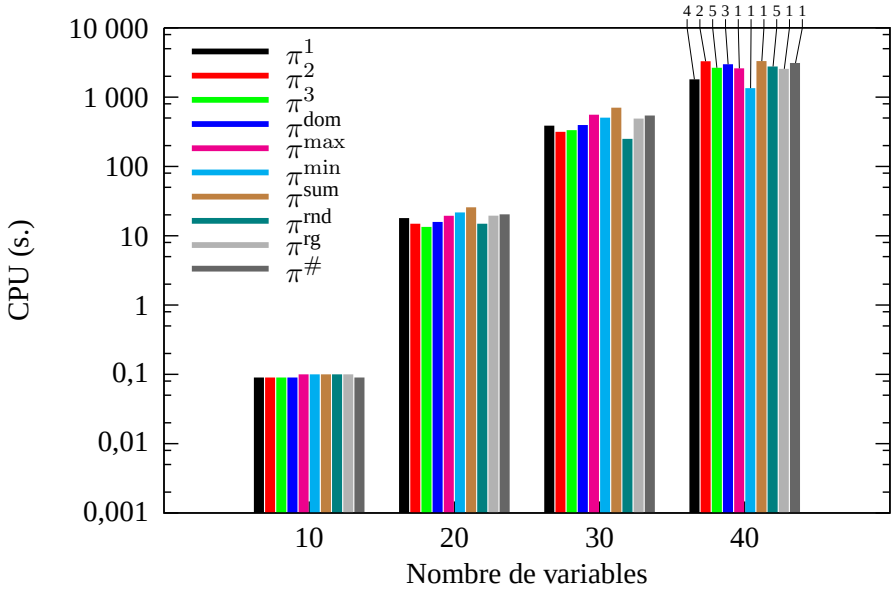

Figure 6.16 – Temps moyen obtenu sur les instances du groupe D-2 avec la procédure `pse_mokp_2`. Lorsque certaines instances n'ont pas été résolues dans le délai de temps, le nombre d'instances sur lequel porte le calcul est indiqué au dessus du graphique.

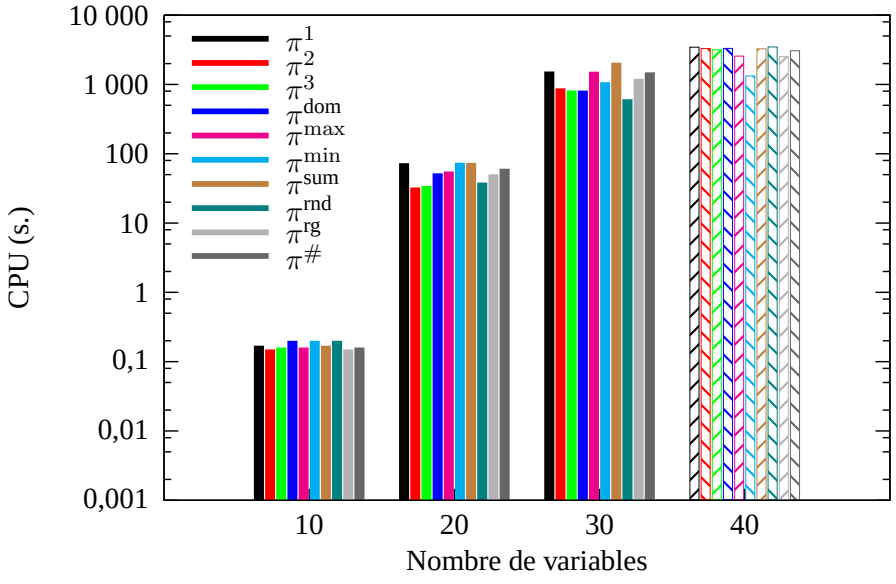

Figure 6.17 – Temps maximal obtenu sur chaque ordre pour chaque taille d'instances du groupe D-2 avec la procédure `pse_mokp_2`. Lorsque les dix instances d'une taille donnée n'ont pas été résolues dans le délai de temps, le temps maximum parmi celles résolues est donné à titre indicatif sous forme d'une barre hachurée.

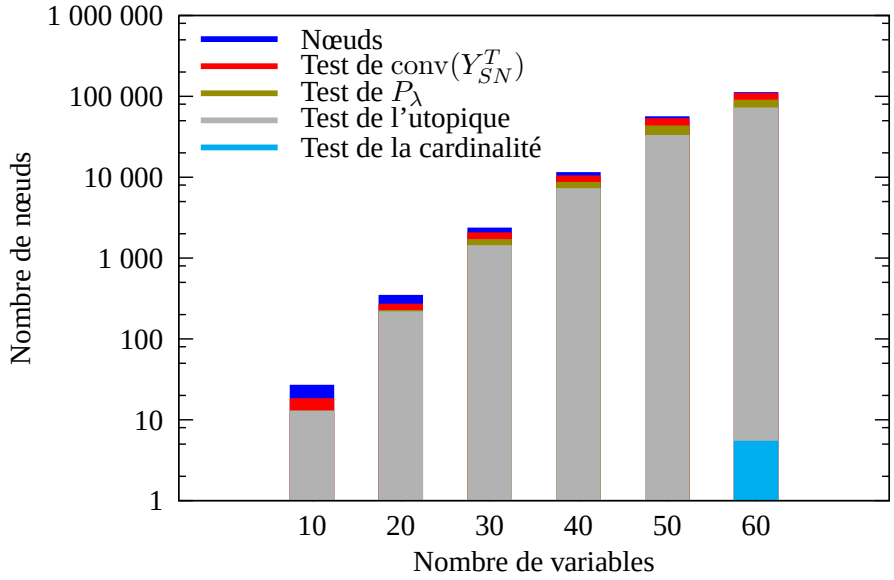

Figure 6.18 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-1, en utilisant l'ordre $\pi^{\min}$ avec la procédure `pse_mokp_2`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.4.1 sont aussi représentés. Si cette page est imprimée en noir et blanc : pour chaque taille, les couleurs des barres, prises de haut en bas, correspondent à la légende lue dans ce même sens.

rapport du nombre de nœuds fermés par chaque test sur le nombre de nœud visités est aussi semblable à celui obtenus sur les instances du groupe D-1.

À nouveau, cette PSE est largement dominée par la procédure en deux phases, ainsi que par `pse_mokp` en terme de temps de résolution. Cependant, elle domine cette dernière sur la quantité des nœuds explorés.

6.5 Conclusion

À notre connaissance, il n'existe pas à ce jour de procédures de séparation et évaluation présentées dans la littérature pour des problèmes ayant plus de deux objectifs. Nous avons présenté dans ce chapitre deux procédures de ce type pour le problème multi-objectif de sac à dos. Une première difficulté dans la conception d'une telle procédure se trouve dans les critères de sélection des objets. Pour cela, nous en avons proposé plusieurs, basés sur la notion d'efficacité de ces objets. Une seconde difficulté consiste à évaluer les nœuds de manière performante pour déterminer au plus tôt si l'exploration du sous-arbre peut apporter de nouvelles solutions efficaces. Nous avons exposé plusieurs fonctions pour cela.

La première procédure proposée est au plus proche de la PSE classique utilisée pour la version mono-objectif du problème. Les résultats numériques mettent en évidence une forte influence de l'ordre de sélection des objets sur les performances de l'algorithme. Néanmoins, aucun des ordres proposés ne domine les autres. Certains sont même parfois moins efficaces qu'une sélection aléatoire. De plus, le nombre de nœuds explorés est dans tous les cas d'une grande quantité, même si la plupart d'entre eux sont fermés par les tests évaluation.

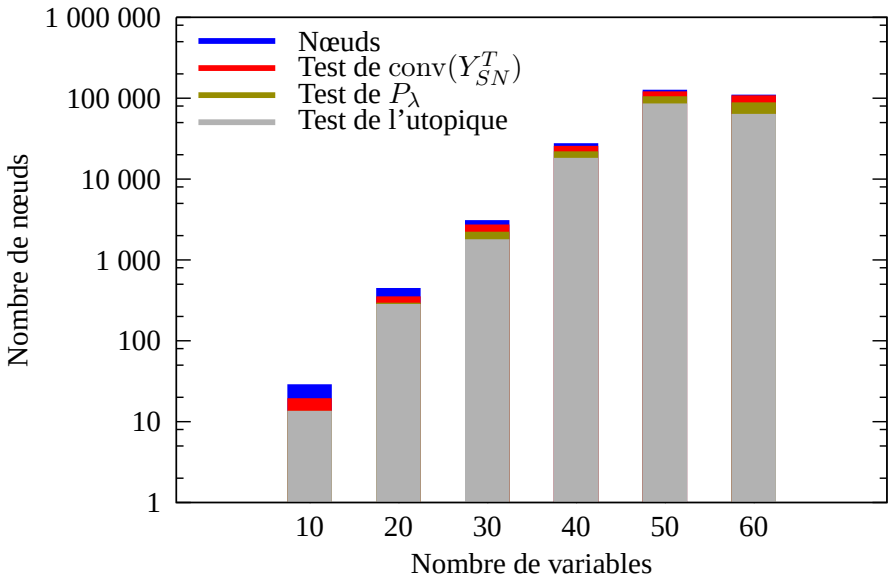

Figure 6.19 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-1, en utilisant l'ordre $\pi^{\max}$ avec la procédure `pse_mokp_2`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.4.1 sont aussi représentés.

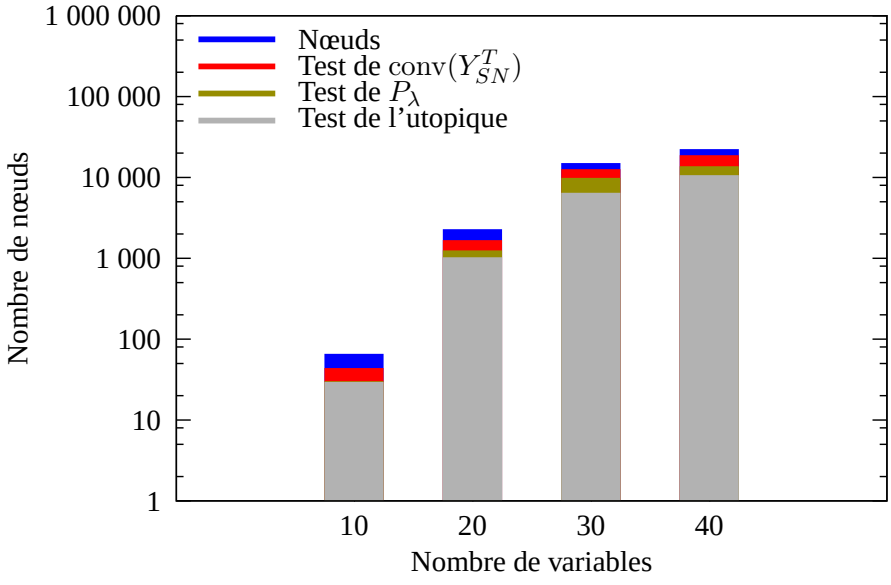

Figure 6.20 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-2, en utilisant l'ordre $\pi^{\min}$ avec la procédure `pse_mokp_2`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.4.1 sont aussi représentés.

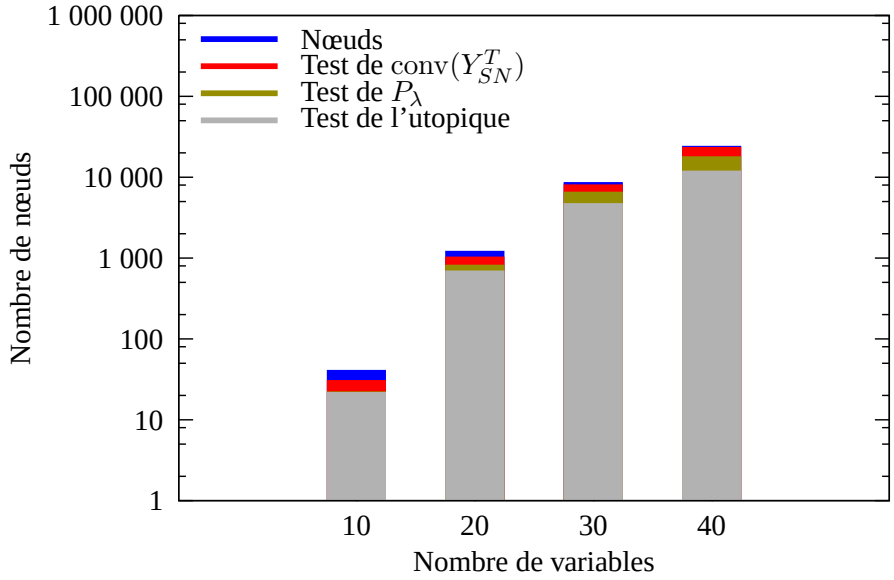

Figure 6.21 – Nombre de nœuds visités en moyenne pour chaque taille d'instance du groupe D-2, en utilisant l'ordre $\pi^{\max}$ avec la procédure `pse_mokp_2`. Le nombre de nœuds fermés avec chaque test de la procédure évaluation décrite en 6.4.1 sont aussi représentés.

Ces résultats nous ont encouragé à proposer une seconde procédure, dans l'optique de réduire le nombre de nœuds visités et de réduire, voire d'annuler, l'influence de l'ordre de sélection des objets sur les performances de l'algorithme. Les résultats numériques vont dans ce sens en cela que les temps de résolution sont relativement similaires d'une ordre à l'autre et que le nombre de nœuds visités a été réduit d'un facteur d'environ 10. Néanmoins, ces résultats ont un coût non négligeable et nous observons une forte augmentation des temps de résolution par rapport à la procédure précédente.

Sans surprise, ces deux procédures sont dans tous les cas dominés par l'approche en deux phases présentée dans le chapitre 5, spécifique au problème de sac à dos multi-objectif. Néanmoins, les procédures de séparation et évaluation restent pertinentes de par leur généricité relativement au problème à traiter, en particulier lorsqu'il n'existe pas d'algorithme permettant d'appliquer une procédure en deux phases au problème de manière efficace.

Conclusions générales et perspectives

Nous nous sommes intéressés dans ce mémoire à la résolution exacte de problèmes d'optimisation combinatoire multi-objectifs. L'objectif de notre travail portait sur l'étude de schémas de résolution dans le cas multi-objectif. En mettant en parallèle l'approche en deux phases, spécifique à cette situation, et en proposant une généralisation des procédures de type séparation et évaluation du cas mono-objectif vers le cas multi-objectif, notre objectif a été atteint.

Notre étude s'appuie sur le problème multi-objectif de sac à dos unidimensionnel en variables binaires. Ce dernier étant un problème classique de l'optimisation combinatoire, présent en tant que sous problème dans de nombreux problèmes d'optimisation, il nous permet d'obtenir un bon indicateur des performances minimales pouvant être attendues d'une méthode de résolution générale.

Notre première contribution porte sur ce problème en particulier. Dans cadre mono-objectif, les algorithmes de résolution sont grandement améliorés par la détermination *a priori*, sans effectuer une résolution exacte, de la valeur que certaines variables auront dans les solutions optimales. Aussi, il est tout naturel d'envisager d'effectuer un pré-traitement similaire dans le cas multi-objectif.

Nous avons étudié les solutions efficaces de plusieurs instances de référence issues de la littérature. Nous avons observé que, dans tous les cas, de nombreuses variables, dites régulières, n'acceptent qu'une unique valeur dans les solutions efficaces. En associant cette valeur avec les données du problème, nous avons remarqué l'apparition de motifs. Nous avons ensuite posé plusieurs propriétés qui, à partir des données du problème et sans résolution préalable, permettent de déterminer la valeur de certaines de ces variables.

Les résultats numériques obtenus à l'issue de cette étape sont contrastés. En effet, les propositions que nous avons posées permettent de déterminer la valeur de plusieurs variables, en particulier sur les instances où les coefficients sont indépendants. Cependant, cette quantité est de mesure faible au regard du nombre réel de variables régulières. De plus, nos propositions peinent avec l'augmentation du nombre d'objectifs. Enfin, elles sont, par définition, sans effet sur les instances où une corrélation apparaît entre les coefficients d'un objectif et ceux de la contrainte de capacité. Les expérimentations numériques montrent néanmoins que ces instances ne sont pas exemptes de variables régulières.

Notre deuxième contribution porte sur l'application d'une procédure en deux phases au problème bi-objectif de sac à dos. Nous avons tout d'abord amélioré la procédure existante, décrite par Visée *et al.* [136], en proposant des bornes plus précises, appliquées dans l'exploration des triangles durant la seconde phase. Les expérimentations numériques montrent une réelle réduction des temps de résolution avec cette modification.

Nous avons ensuite proposé d'utiliser une procédure de type *ranking* pour l'exploration des triangles durant la seconde phase. Pour cela, nous nous sommes appuyés sur l'analogie du problème étudié avec le problème des plus longs chemins dans un graphe acyclique. Nous avons décrit la construction du graphe puis nous avons détaillé le schéma général des procédures de type *ranking*. Nous avons ensuite

proposé un algorithme adapté au problème de sac à dos, inspiré d'une procédure de Eppstein [45] pour le problème des plus longs chemins.

Les expérimentations numériques montrent que notre procédure en deux phases avec *ranking* est nettement plus performante que les précédentes méthodes de résolution exacte sur le problème. En particulier, en dehors d'une classe d'instances particulières, notre procédure surclasse la programmation dynamique de Bazgan *et al.* [10], jusqu'alors la meilleure méthode exacte pour le problème. De plus, l'étude de la quantité de mémoire requise par notre algorithme exhibe des valeurs tout à fait raisonnables et accessibles.

Notre troisième contribution concerne la généralisation de la méthode en deux phases à trois objectifs et plus. En nous appuyant sur les travaux de Przybylski [111], nous avons généralisé le calcul des solutions supportées dans la première phase en utilisant le concept de l'ensemble des poids. Puis nous avons généralisé la seconde phase en utilisant une description de l'espace de recherche appropriée au cas multi-objectif et en adaptant le calcul des bornes dans l'exploration. Enfin, l'algorithme de *ranking* décrit dans le chapitre 4 a pu être appliqué tel quel pour le calcul des solutions non supportées.

Finalement, les expérimentations numériques montrent à nouveau que l'approche en deux phases est tout à fait adaptée à la résolution de problèmes multi-objectifs. En particulier, notre procédure surpasse à nouveau les meilleures algorithmes pour le problème. Comme l'a souligné Przybylski [111], l'existence d'un algorithme de *ranking* performant pour un problème d'optimisation combinatoire suggère que l'application d'une méthode en deux phases s'impose comme résolution efficace.

La dernière de nos contributions porte sur l'application d'un schéma général de résolution de problèmes d'optimisation combinatoire multi-objectifs. Nous avons choisi d'adapter une procédure de séparation et évaluation. En effet, ce type de procédure est largement répandue dans le cadre de l'optimisation mono-objectif et y donne de bons résultats, en particulier sur le problème de sac à dos.

Nous avons proposé une généralisation au cas multi-objectif d'une procédure classique utilisée pour le problème mono-objectif de sac à dos. Pour cela, nous avons identifié deux verrous, dans l'ordre de sélection des variables, d'une part, et dans l'évaluation des nœuds, d'autre part.

Pour lever ces verrous, nous avons proposé plusieurs ordres dans l'exploration, basés sur la notion d'efficacité des objets. Nous avons, de plus, proposé l'utilisation d'ensembles bornant inférieurement et supérieurement, tels que définis par Ehrgott et Gandibleux [41], pour effectuer l'évaluation des nœuds. Nous avons décrit plusieurs procédures pour le calcul de l'ensemble bornant supérieurement, permettant d'obtenir une évaluation précise pour un coût raisonnable.

Les expérimentations numériques effectuées à cette étape ont montré que l'ordre de sélection des objets était très influent sur les performances de la procédure. De plus, aucun des ordres proposé ne domine clairement les autres, certains étant même parfois moins efficaces qu'une sélection aléatoire des objets. D'autre part, nous avons observé que l'espace exploré était très large. Nous avons alors proposé une seconde procédure, dans laquelle la direction de la recherche est en permanence adaptée en fonction des solutions précédemment rencontrées. Nous avons aussi ajouté une étape dans l'évaluation des nœuds, basée sur un calcul plus fin, mais coûteux, d'un ensemble bornant supérieurement.

Les expérimentations numériques effectuées sur ce second algorithme montrent que l'influence de l'ordre de sélection des objets a été grandement diminuée et que l'espace exploré est largement plus petit. Néanmoins, les temps de résolution ont augmenté, en particulier du fait du nouveau calcul d'un l'ensemble bornant supérieurement.

Globalement, les résultats obtenus dans cette partie sont intéressants pour le caractère générique de la procédure de séparation et évaluation. Cependant, étant donné l'écart de performances par rapport à

une procédure en deux phases, il s'impose que les procédures de séparation et évaluation sont à réserver, en multi-objectif, aux problèmes pour lesquels le schéma de la deux phases ne peut s'appliquer.

Les perspectives de ces travaux sont nombreuses. Les plus prometteuses sont évoquées ci-dessous.

À notre connaissance, les résultats que nous avons obtenus sur le calcul des variables régulières sont les premiers de ce type pour ce problème multi-objectif de sac à dos. Ainsi plusieurs pistes restent à explorer. En particulier, tous les éléments sont présents dans ce manuscrit pour proposer un algorithme sur le principe proche de ceux produits dans le cas mono-objectif pour la réduction d'instance. Un tel algorithme peut se décrire en trois points :

1. calculer un ensemble $\bar{Y}$ de points réalisables, ainsi que $D(\bar{Y})$;
2. pour chaque variable, calculer un ensemble bornant supérieurement les solutions qui seront obtenues en forçant la variable à zéro (respectivement à un) ;
3. si cet ensemble n'est aucunement dans $D(\bar{Y}) + \mathbb{R}^p_{\geqq}$, alors cette variable a la valeur un (respectivement zéro) dans toutes les solutions efficaces.

Le calcul de $\bar{Y}$ peut, par exemple, s'effectuer en appliquant la première phase de la procédure en deux phases. Les ensembles bornant supérieurement peuvent être obtenus avec différents degrés de précision, comme détaillé dans le chapitre 6. En particulier, au regard des résultats obtenus sur l'évaluation des nœuds dans ce chapitre, le calcul d'un point utopique semble être un bon candidat.

Les résultats obtenus avec la deux phases bi-objectif sur des instances faisant intervenir une forte corrélation entre les coefficients d'un objectif et ceux de la contrainte de capacité exhibent une difficulté inhérente à l'utilisation d'un algorithme de type *ranking*. En effet, ce type de procédure implique l'exploration nécessaire d'une partie de l'espace dominé. Or, dans des cas particuliers tels qu'obtenus avec ces instances, cet espace peut contenir un nombre particulièrement grand de solutions réalisables. La procédure se perd alors dans l'énumération de ces solutions et les performances chutent en conséquence. Dans une telle situation, il sera préférable d'appliquer un algorithme spécifique moins sensible à cette difficulté. En particulier, la programmation dynamique de Bazgan *et al.* [10] est la plus efficace sur ces instances. Néanmoins, le choix de l'algorithme à appliquer dans ce cas doit se faire *a priori*, par déduction depuis les données du problème. Une bonne piste pour décider du type d'algorithme à appliquer peut se trouver dans le travail de classification entamé par Przybylski *et al.* [112] sur des instances dites difficiles pour le problème multi-objectif d'affectation.

Bien que la procédure en deux phases donne de très bon résultats en termes de performances, nous observons que la taille des instances résolues diminue fortement lorsque le nombre d'objectifs augmente. De plus, les expérimentations numériques montrent que le nombre de solutions efficaces atteint rapidement plusieurs dizaines de milliers dès trois objectifs. Intuitivement, nous pouvons envisager que les solutions réalisables tendent à être toutes efficaces lorsque le nombre d'objectifs augmente. De plus, il est inconcevable de présenter une telle quantité de solutions à un décideur. Par conséquent, la résolution exacte de telles instances dans le cadre d'un système d'aide à la décision devra passer par la prise en compte des préférences du décideur durant la recherche, de manière à réduire l'espace de recherche et ainsi lui présenter des solutions à la fois pertinentes et en nombre réduit.

Enfin, les résultats portants sur les procédures de séparation et évaluation sont encourageants. C'est en effet la seconde fois, avec les travaux de Sourd et Spanjaard [126], qu'une telle procédure permet de

résoudre dans des temps raisonnables des instances d'un problème d'optimisation combinatoire multi-objectif, et la première fois que cette approche s'attache à un problème comportant plus de deux objectifs. Évidemment, une approche en deux phases est préférable lorsqu'elle est applicable, ce qui n'est pas immédiat lorsqu'un problème quelconque se présente. Des pistes dans la continuité de ces travaux sont à explorer dans l'indépendance entre la méthode et le problème résolu, ainsi que dans l'évaluation des nœuds. Pour ce dernier point, l'utilisation d'un solveur linéaire multi-objectif (obtenu par exemple depuis les travaux de Benson *et al.* [13, 14, 15]), à la place du calcul des solutions supportées du sous-arbre permettrait d'obtenir un ensemble bornant supérieurement, avec une précision certes moindre mais peut-être compensée par un calcul plus rapide.

Bibliographie

[1] E. AARTS et J. K. LENSTRA, réds. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[2] F. B. ABDELAZIZ, J. CHAOUACHI et S. KRICHEN. A hybrid heuristic for multiobjective knapsack problems. Dans S. VOSS, S. MARTELLO, I. OSMAN et C. ROUCAIROL, réds., *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 205–212. Kluwer Academic Publishers, Dordrecht, 1999.

[3] Y. P. ANEJA et K. P. K. NAIR. Bicriteria transportation problem. *Management Science*, 25:73–78, 1979.

[4] E. BALAS et E. ZEMEL. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.

[5] S. BALEV, N. YANEV, A. FRÉVILLE et R. ANDONOV. A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem. *European Journal of Operational Research*, 186(1):63–76, April 2008.

[6] V. BARICHARD et J. K. HAO. Un algorithme hybride pour le problème de sac à dos multi-objectifs. Dans *JNPC'2002 : Huitièmes Journées nationales sur la Résolution Pratique de Problèmes NP-Complets*, pages 19–30, Nice, France, may 2002.

[7] V. BARICHARD et J. K. HAO. Genetic Tabu Search for the Multi-objective Knapsack Problem. *Journal of Tsinghua Science and Technology*, 8(1):8–13, 2003.

[8] C. BAZGAN, H. HUGOT et D. VANDERPOOTEN. *A Practical Efficient FPTAS for the 0-1 Multi-objective Knapsack Problem*, volume 4698 de *Lecture Notes in Computer Science*, pages 717–728. Springer Berlin / Heidelberg, septembre 2007.

[9] C. BAZGAN, H. HUGOT et D. VANDERPOOTEN. Implementing an efficient FPTAS for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47–56, 2009.

[10] C. BAZGAN, H. HUGOT et D. VANDERPOOTEN. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009.

[11] R. E. BELLMAN. *Dynamic Programming*. Princeton University Press, 1957.

[12] H. P. BENSON. Existence of Efficient Solutions for Vector Maximization Problems. *Journal of Optimization Theory and Applications*, 26(4):569–580, 1978.

[13] H. P. BENSON. Further analysis of an outcome set-based algorithm for multiple-objective linear programming. *Journal of Optimization Theory and Applications*, 97(1):1–10, 1998.

[14] H. P. BENSON et E. SUN. Outcome space partition of the weight set in multiobjective linear programming. *Journal of Optimization Theory and Applications*, 105(1):17–36, 2000.

[15] H. P. BENSON et E. SUN. A weight set decomposition algorithm for finding all efficient extreme points in the outcome set of a multiple objective linear program. *European Journal of Operational Research*, 139(1):26–41, May 2002.

[16] G. R. BITRAN et J. M. RIVERA. A Combined Approach to Solve Binary Multicriteria Problems. *Naval Research Logistics Quarterly*, 29(2):181–200, 1982.

[17] K. Bouibede HOCINE. *Énumération en ordonnancement multi-critère : application à un problème bicritère à machines parallèles*. Thèse de Doctorat, Université de Tours, Mai 2007.

[18] V. BOYER. *Contribution à la programmation en nombre entier*. Thèse de Doctorat, Institut national des sciences appliquées, Toulouse, décembre 2007.

[19] K. M. BRETTHAUER et B. SHETTY. The nonlinear knapsack problem - algorithms and applications. *European Journal of Operational Research*, 138(3):459–472, May 2002.

[20] D. BROCKHOFF. *Many-Objective Optimization and Hypervolume Based Search*. Shaker Verlag, Aachen, Germany, 2009. PhD thesis at ETH Zurich.

[21] P. CAPPANERA et M. TRUBIAN. A Local-Search-Based Heuristic for the Demand-Constrained Multidimensional Knapsack Problem. *INFORMS Journal on Computing*, 17(1):82–98, 2005.

[22] E. CAPTIVO, J. CLÍMACO, J. FIGUEIRA, E. MARTINS et J. L. SANTOS. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers & Operational Research*, 30:1865–1886, 2003.

[23] C. R. CHEGIREDDY et H. W. HAMACHER. Algorithms for Finding k-best Perfect Matchings. *Discrete Applied Mathematics*, 18(2):155–165, 1987.

[24] G. CHRISTIAN, R. CHRISTIAN et S. HEINZ. *Multiobjective Disk Cover Admits a PTAS*, volume 5369 de *Lecture Notes in Computer Science*, pages 40–51. Springer Berlin / Heidelberg, décembre 2008.

[25] G. CORNUÉJOLS. *Combinatorial optimization: packing and covering*, volume 74 de *CBMS-NSF Regional Conference Series In Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[26] P. CZYZAK et A. JASZKIEWICZ. Pareto simulated annealing — a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

[27] G. B. DANTZIG. Discrete-Variable Extremum Problems. *Operations Research*, 5(2):266–277, avril 1957.

[28] F. DEGOUTIN et X. GANDIBLEUX. Un retour d'expériences sur la résolution de problèmes combinatoires bi-objectifs, may 2002. Journée Programmation Mathémathique Multiobjectifs. Angers.

[29] X. DELORME, X. GANDIBLEUX et F. DEGOUTIN. Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research*, 204(2):206–217, Juillet 2010.

[30] C. DHAENENS, J. LEMESRE et E. G. TALBI. "k"-PPM: A new exact method to solve multi-objective combinatorial optimization problems. *European Journal of Operational Research*, 200(1):45–53, 2010.

[31] M. DORIGO, G. DI CARO et L. M. GAMBARDELLA. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999.

[32] K. DUDZIŃSKI et S. WALUKIEWICZ. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28(1):3–21, January 1987.

[33] M. E. DYER, W. O. RIHA et J. WALKER. A hybrid dynamic programming/branch-and-bound algorithm for the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics*, 58(1):43–54, 1995.

[34] E. 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, Novembre 2003.

[35] M. EHRGOTT. *Multicriteria Optimization*, volume 491 de *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, Berlin, 2000.

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

[37] M. EHRGOTT et X. GANDIBLEUX. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[38] M. EHRGOTT et X. GANDIBLEUX. *Bounds and bound Sets for Biobjective Combinatorial Optimization Problems*, volume 507 de *Lectures Notes in Economics and Mathematical Systems*. Springer, 2001.

[39] M. EHRGOTT et X. GANDIBLEUX. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, volume 52 de *Kluwer's International Series in Operations Research and Management Science*, pages 369–444. Kluwer Academic Publishers, 2002.

[40] M. EHRGOTT et X. GANDIBLEUX. Approximative solution methods for multiobjective combinatorial optimization. *TOP*, 12(1):63, 2004.

[41] M. EHRGOTT et X. GANDIBLEUX. Bound Sets for Biobjective Combinatorial Optimization Problems. *Computers & Operations Research*, 34(9):2674–2694, 2007.

[42] M. EHRGOTT, K. KLAMROTH et C. SCHWEHM. An MCDM approach to portfolio optimization. *European Journal of Operational Research*, 155(3):752–770, Juin 2004.

[43] M. EHRGOTT, J. PUERTO et A. RODRIGUEZ-CHIA. Primal-Dual Simplex Method for Multiobjective Linear Programming. *Journal of Optimization Theory and Application*, 134(3):483–497, September 2007.

[44] D. EL BAZ et M. ELKIHEL. Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0-1 knapsack problem. *Journal of Parallel and Distributed Computing*, 65(1):74–84, 2005.

[45] D. EPPSTEIN. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[46] D. FAYARD et G. PLATEAU. An algorithm for the solution of the 0-1 knapsack problem. *Computing*, 28:269–287, 1982.

[47] J. R. FIGUEIRA, G TAVARES et M. M. WIECEK. Labeling algorithms for multiple objective integer knapsack problems. *Computers & Operations Research*, 37(4):700–711, 2009.

[48] M. FLEISCHER et M. FLEISCHER. The Measure of Pareto Optima. Applications to Multi-objective Metaheuristics. Dans *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, volume 2632 de *Lecture Notes in Computer Science*, pages 519–533. Springer, 2003.

[49] K. FLESZAR et K. S. HINDI. Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. *Computers & Operations Research*, 36(5):1602–1607, 2009.

[50] K. FLORIOS, G. MAVROTAS et D. DIAKOULAKI. Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. *European Journal of Operational Research*, 203(1):14–21, 2010.

[51] A. FRÉVILLE. The multidimensional 0-1 knapsack problem : An overview. *European Journal of Operational Research*, 155:1–21, 2004.

[52] A. FRÉVILLE et G. PLATEAU. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *discrete applied mathematics*, 49(1-3):189–212, 1994.

[53] X. GANDIBLEUX et A. FRÉVILLE. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: the two objectives case. *Journal of Heuristics*, 6:361–383, 2000.

[54] X. GANDIBLEUX, J. JORGE, X. DELORME et J. RODRIGUEZ. *Algorithme de fourmis pour mesurer et optimiser la capacité d'un réseau ferroviaire*, volume 1 de *Fourmis artificielles: des bases de l'optimisation aux applications industrielles*, chapitre 9, pages 215–244. N. Monmarché, F. Guinand, et P. Siarry, 2009.

[55] X. GANDIBLEUX, J. JORGE, S. MARTINEZ et N. SAUER. Premier retour d'expérience sur le flowshop biobjectif et hybride à deux étages avec une contrainte de blocage particulière. Avril 2006. 6ème Conférence Francophone de Modélisation et Simulation MOSIM'06.

[56] X. GANDIBLEUX, N. MEZDAOUI et A. FRÉVILLE. A tabu search procedure to solve multiobjective combinatorial optimization problems. Dans R. CABALLERO, F. RUIZ et R. STEUER, réds., *Advances in Multiple Objective and Goal Programming*, volume 455 de *Lecture Notes in Economics and Mathematical Systems*, pages 291–300. Springer Verlag, Berlin, 1997.

[57] X. GANDIBLEUX, H. MORITA et N. KATOH. *The Supported Solutions Used as a Genetic Information in a Population Heuristic*, volume 1993 de *Lecture Notes in Computer Science*, pages 429–442. Springer-Verlag, 2001. First International Conference on Evolutionary Multi-Criterion Optimization.

[58] X. GANDIBLEUX, D. VANCOPPENOLLE et D. TUYTTENS. A first making use of GRASP for solving MOCO problems. Rapport technique, 1998. Paper presented at MCDM 14, June 8-12 1998, Charlottesville, VA.

[59] R. S. GARFINKEL et G. L. NEMHAUSER. *Integer Programming*. John Wiley & Sons, New York, 1972.

[60] G. V. GENS et E. V. LEVNER. *Computational Complexity of Approximation Algorithms for Combinatorial Problems*, volume 74 de *Lecture Notes in Computer Science*, pages 292–300. Springer, 1979.

[61] A. M. GEOFFRION. Proper Efficiency and the theory of Vector Maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

[62] F. GLOVER. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6):879–919, 1965.

[63] A. V. GOLDBERG et A. MARCHETTI-SPACCAMELA. On finding the exact solution of a zero-one knapsack problem. Dans *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 359–368, New York, NY, USA, 1984. ACM Press.

[64] C. Gomes da SILVA, J. CLÍMACO et J. R. FIGUEIRA. Core problems in $0,1$ Bi-criteria Knapsack Problems. Dans *Proceedings of the 7^th international conference on MultiObjective Programming and Goal Programming (MOPGP' 06)*, June 2006. 12-14 Juin 2006.

[65] Y. Y. HAIMES, L. S. LASDON et D. A. WISMER. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man and Cybernetics* , SMC-1(3):296–297, 1971.

[66] H. W. HAMACHER et G. RUHE. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[67] P. HANSEN. Bicriterion path problems. Dans G. FANDEL et T. GAL, réds., *Multiple Criteria Decision Making Theory and Application*, volume 177 de *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Verlag, Berlin, 1979.

[68] E. HOROWITZ et S. SAHNI. Computing partitions with applications to the knapsack problem. *Journal of ACM*, 23:277–292, 1974.

[69] H. HUGOT. *Approximation et énumération des solutions efficaces dans les problèmes d'optimisation combinatoire multi-objectifs*. Thèse de Doctorat, Université Paris-Dauphine, octobre 2007.

[70] T. IBARAKI. Enumerative approaches to combinatorial optimization, part II. *Annals of Operations Research*, 11:343–602, 1987.

[71] G. P. INGAGORLIA et J. F. KORSH. Reduction algorithm for zero-one single knapsack problem. *Management Science*, 20:460–463, 1973.

[72] A. JASZKIEWICZ. Multiple objective genetic local search algorithm. Dans *Multiple Criteria Decision Making in the New Millenium*, volume 507, chapitre Lecture Notes in Economics and Mathematical Systems, pages 231–240. Köksalan, M. and Zionts, S., 2001.

[73] J. JORGE et X. GANDIBLEUX. Self-Adaptive Stopping Condition for an Ant Colony Optimization Inspired Algorithm Designed for Set Packing Problems. Màlaga, Espagne, Novembre 2006. 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics.

[74] R. M. KARP. Reducibility among combinatorial problems. Dans R. E. MILLER et J. W. THATCHER, réds., *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[75] H. KELLERER et U. PFERSCHY. A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem. *Journal of Combinatorial Optimization*, 3(1):59–71, 1999.

[76] H. KELLERER et U. PFERSCHY. Improved Dynamic Programming in Connection with an FPTAS for the Knapsack Problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.

[77] H. KELLERER, U. PFERSCHY et D. PISINGER. *Knapsack Problems*, chapitre 1–7, 9, 13–14. Springer, 2004.

[78] G. KIZILTAN et E. YUCAOGLU. An algorithm for multiobjective zero-one linear programming. *Management Science*, 29(12):1444–1453, 1983.

[79] K. KLAMROTH et M. WIECEK. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, 47:57–76, 2000.

[80] M. KONG, P. TIAN et Y. KAO. A new ant colony optimization algorithm for the multidimensional Knapsack problem. *Computers & Operations Research*, 35(8):2672–2683, 2008.

[81] B. KORTE et R. SCHRADER. *On the existence of fast approximation schemes*, volume 4 de *Nonlinear Programming*, pages 415–437. Academic Press, 1981.

[82] P. A. KROKHMAL et P. M. PARDALOS. Random assignment problems. *European Journal of Operational Research*, 194(1):1–17, 2009.

[83] J. L. LAURIÈRE. *Éléments de programmation dynamique*. Recherche opérationnelle appliquée 3. Gauthier-Villars New York, 1979.

[84] E. L. LAWLER. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[85] E. L. LAWLER. Fast approximation algorithms for knapsack problems. Dans *SFCS '77: Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 206–213, Washington, DC, USA, 1977. IEEE Computer Society.

[86] H. LEE et P. SIMIN PULAT. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, 66(1):148–157, April 1993.

[87] J. LEMESRE, C. DHAENENS et E. G. TALBI. An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research*, 177(3):1641–1655, 2007.

[88] J. LEMESRE, C. DHAENENS et E. G. TALBI. Parallel partitioning method (PPM): A new exact method to solve bi-objective problems. *Computers & Operations Research*, 34(8):2450–2462, 2007.

[89] M. J. MAGAZINE et O. OGUZ. A fully polynomial approximation algorithm for the 0-1 knapsack problem. *European Journal of Operational Research*, 8(3):270–273, November 1981.

[90] O. MARCOTTE et R. M. SOLAND. An interactive branch-and-bound algorithm for multiple criteria optimization. *Management Science*, 32(1):61–75, 1986.

[91] S. MARTELLO et P. TOTH. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34(5):633–644, 1988.

[92] S. MARTELLO et P. TOTH. *Knapsack Problems : Algorithms and Computer Implementations*, chapitre 1-4. John Wiley & sons, 1990.

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

[94] G. B. MATHEWS. On the partition of numbers. Dans *Proceedings of the London Mathematical Society*, volume 28, pages 486–490, 1897.

[95] G. MAVROTAS, J. R. FIGUEIRA et K. FLORIOS. Solving the bi-objective multi-dimensional knapsack problem exploiting the concept of core. *Applied Mathematics and Computation*, 215(7):2502–2514, Décembre 2009.

[96] S. MAVROTAS et D. DIAKOULAKI. A branch and bound algorithm for mixed zero-one multiple objective linear programming . *European Journal of Operational Research* , 107(3):530–541, 1998.

[97] R. C. MERKLE et M. E. HELLMAN. Hiding Information and Receipts in Trap Door Knapsacks. Cornell University, Ithaca, New York, Octobre 1977. Internal Symposium on Information Theory.

[98] S. MICHEL, N. PERROT et F. VANDERBECK. Knapsack problems with setups. *European Journal of Operational Research*, 196(3):909–918, August 2009.

[99] R. J. MORAGA, G. W. DEPUY et G. E. WHITEHOUSE. Meta-RaPs approach for the 0-1 multidimensional Knapsack problem. *Computers & Industrial Engineering*, 48(1):83–96, 2005.

[100] H. MULLER-MERBACH. An improved upper bound for the zero-one knapsack problem : A note on the paper by Martello and Toth. *European Journal of Operational Research*, 2(3):212–213, May 1978.

[101] T. MURATA et H. ISHIBUCHI. MOGA: Multi-objective genetic algorithms. Dans *Proceedings of the 2nd IEEE International Conference on Evolutionary Computing, Perth, Australia*, pages 289–294. IEEE Service Center, Piscataway, NJ, 1995.

[102] P. NEUMAYER. Complexity of optimization on vectorweighted graphs. Dans *Operations Research 93*, pages 359–361, Heidelberg, 1994. Physica Verlag.

[103] M. PADBERG. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[104] C. H. PAPADIMITRIOU et M. YANNAKAKIS. On the approximability of trade-offs and optimal access of web sources. Dans *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 86–92, 2000.

[105] C. R. PEDERSEN, L. R. NIELSEN et K. A. ANDERSEN. The Bicriterion Multimodal Assignment Problem: Introduction, Analysis, and Experimental Results. *Informs Journal on Computing*, 20(3):400–411, 2008.

[106] U. PFERSCHY. Dynamic programming revisited: improving knapsack algorithms. *Computing*, 63(4):419–430, 1999.

[107] D. PISINGER. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.

[108] D. PISINGER. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[109] D. PISINGER. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[110] G PLATEAU et M. ELKIHEL. A hybrid method for the 0-1 knapsack problem. *Methods of Operations Research*, 49:277–293, 1985.

[111] A. PRZYBYLSKI. *Méthode en deux phases pour la résolution exacte de problèmes d'optimisation combinatoire comportant plusieurs objectifs : nouveaux développements et application au problème d'affectation linéaire*. Thèse de Doctorat, Université de Nantes, décembre 2006.

[112] A. PRZYBYLSKI, J. BOURDON et X. GANDIBLEUX. Distribution of Solutions of Multi-objective Assignment Problem and Links with the efficiency of Solving Methods, September, 5–7 2007. Gesellschaft für Operations Research 2007.

[113] A. PRZYBYLSKI, X. GANDIBLEUX et M. EHRGOTT. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509–533, oct 2008.

[114] J. PUCHINGER, G. R. RAIDL et U. PFERSCHY. The Core Concept for the Multidimensional Knapsack Problem. Dans *Evolutionary Computation in Combinatorial Optimization (EVOCOP 2006)*, pages 195–208. Springer, 2006.

[115] R. M. RAMOS, S. ALONSO, J. SICILIA et C. GONZALEZ. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111(3):617–628, December 1998.

[116] J. RODRIGUEZ, X. DELORME, X. GANDIBLEUX, G. MARLIERE, R. BARTUSIAK, F. DEGOUTIN et S. SOBIERAJ. RECIFE : modèles et outils pour l'analyse de la capacité ferroviaire. *Recherche Transports Sécurité* , 95 :129–146, 2007. G2I-MSGI-2007 .

[117] G. RUHE. Complexity results for multicriteria and parametric network flows using a pathological graph of Zadeh. *Zeitschrift für Operations Research*, 32:9–27, 1988.

[118] H. M. SAFER et J. B. ORLIN. Fast approximation schemes for multi-criteria combinatorial optimization. Rapport technique 9756-95, MIT Sloan School of Management, 1995.

[119] H. M. SAFER et J. B. ORLIN. Fast approximation schemes for multi-criteria flow, knapsack and scheduling problems. Rapport technique 3757-95, MIT Sloan School of Management, 1995.

[120] J. David SCHAFFER. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. Dans *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[121] A. SEDEÑO-NODA et C. GONZÁLEZ-MARTÍN. An algorithm for the biobjective integer minimum cost flow problem. *Computers & Operations Research*, 28(2):139–156, 2001.

[122] P. SERAFINI. Some considerations about computational complexity for multiobjective combinatorial problems. Dans J. JAHN et W. KRABS, réds., *Recent advances and historical development of vector optimization*, volume 294 de *Lecture Notes in Economics and Mathematical Systems*, Berlin, 1986. Springer-Verlag.

[123] M. S. S. SHAHRAKI. A computationally efficient algorithm for determination of efficient max-ordering optimal solutions in multiple objective programming. *Applied Mathematics and Computation*, 196(2):720–723, 2008.

[124] A. SHAMIR. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. Dans *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 145–152, Washington, DC, USA, 1982. IEEE Computer Society.

[125] M. SNIEDOVICH. *Dynamic Programming*. Marcel Dekker, Inc., New York, NY, USA, 1991.

[126] 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.

[127] N. SRINIVAS et K. DEB. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[128] S. STEINER et T. RADZIK. Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers & Operations Research*, 35(1):198–211, 2008.

[129] S. S. SYAM. A dual ascent method for the portfolio selection problem with multiple constraints and linked proposals. *European Journal of Operational Research*, 108(1):196–207, July 1998.

[130] D. TENFELDE-PODEHL. A recursive algorithm for multiobjective combinatorial optimization problems with q criteria. Rapport technique, Institut für Mathematik, Technische Universität Graz, 2003.

[131] P. TOTH. Dynamic programming algorithms for the Zero-One Knapsack Problem. *Computing*, 25(1):29–45, 1980.

[132] G. TSAGGOURIS et C. ZAROLIAGIS. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications. *Theory of Computing Systems*, 45(1):162–186, 2009.

[133] E. L. ULUNGU. *Optimisation combinatoire multicritère : détermination de l'ensemble des solutions efficaces et méthodes interactives*. Thèse de Doctorat, Université de Mons-Hainaut, Faculté des sciences, oct 1993.

[134] E. L. ULUNGU et J. TEGHEM. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

[135] S. VAUDENAY. Cryptanalysis of the Chor-Rivest cryptosystem. Dans *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference*, volume 1462 de *Lecture Notes in Computer Science*, pages 243–256, Santa Barbara, California, USA, 1998.

[136] J. VISÉE, J. TEGHEM, M. PIRLOT et E. L. ULUNGU. Two-phases Method and Branch and Bound Procedures to Solve the Bi-objective Knapsack Problem. *Journal of Global Optimization*, 12:139–155, 1998.

[137] C. WILBAUT. *Heuristiques hybrides pour la résolution de problèmes en variables 0-1 mixtes*. Thèse de Doctorat, Université de Valenciennes et du Hainaut-Cambresis, Septembre 2006.

[138] C. WILBAUT, S. HANAFI et S. SALHI. A survey of effective heuristics and their application to a variety of knapsack problems. *IMA Journal of Management Mathematics*, 19:227–244, 27 *July* 2008.

[139] L. A. WOLSEY et G. L. NEMHAUSER. *Integer and Combinatorial Optimization*. Wiley-Interscience, November 1999.

[140] T. ZHONG et R. YOUNG. Multiple Choice Knapsack Problem: Example of planning choice in transportation. *Evaluation and Program Planning*, 33(2):128–137, Mai 2010.

[141] E. ZITZLER, D. BROCKHOFF et L. THIELE. *The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted integration*, volume 4403 de *Lecture Notes in Computer Science*, pages 962–876. Springer Berlin / Heidelberg, mai 2007. EMO 2007.

[142] E. ZITZLER, L. THIELE, M. LAUMANNS, C. M. FONSECA et V. G. DA FONSECA. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.

Liste des tableaux

Liste des figures

Crédit photo

Les armoiries de la ville de Nantes utilisées sur la page de titre proviennent de *Wikimedia Commons* et sont disponibles à l'adresse `http://commons.wikimedia.org/w/index.php?title=File:Blason_Nantes_Orne.svg&oldid=25670809`. Elles y ont été reproduites par l'utilisateur Moktoipas et mises à disposition sous une licence *Creative Commons Attribution ShareAlike 3.0* [a]

a. `http://creativecommons.org/licenses/by-sa/3.0/`

Liste des exemples

Table des matières

Annexes

ANNEXE A

Réponses aux exemples de la section 1.1.2

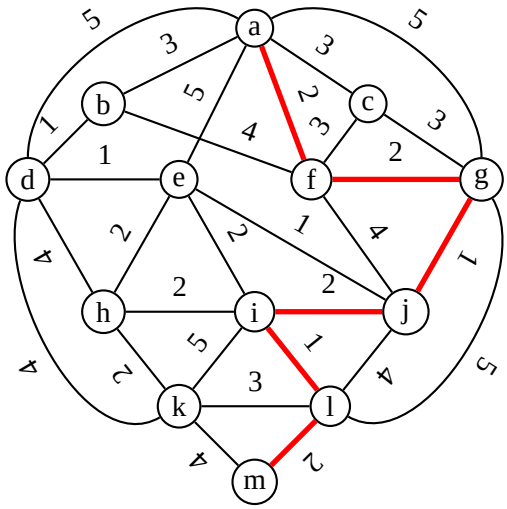

Figure A.1 – Une solution optimale du problème de l'exemple 1.1. La longueur du chemin indiqué est égale à 10. Il existe deux autres solutions optimales équivalentes pour ce problème. La première passe par les villes (a, b, d, e, i, l, m) et la seconde par (a, e, i, l, m).

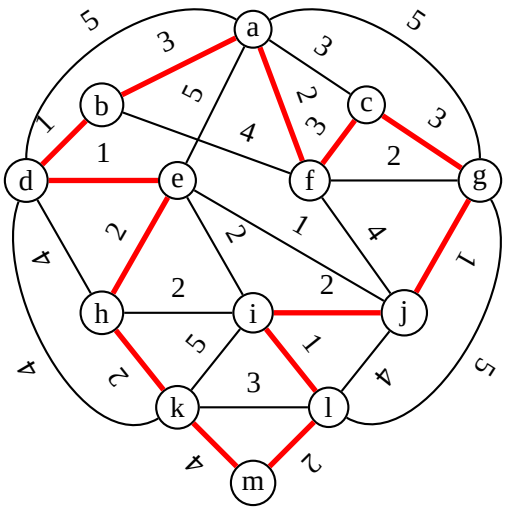

Figure A.2 – Une solution optimale du problème de l'exemple 1.2. La longueur du chemin indiqué est égale à 27.

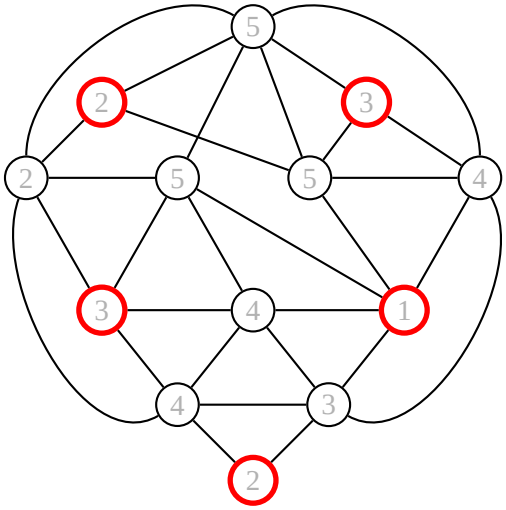

Figure A.3 – Une solution optimale du premier problème de *set packing* de l'exemple 1.3. Les invités sélectionnés sont indiqués ici, soit 5 personnes.

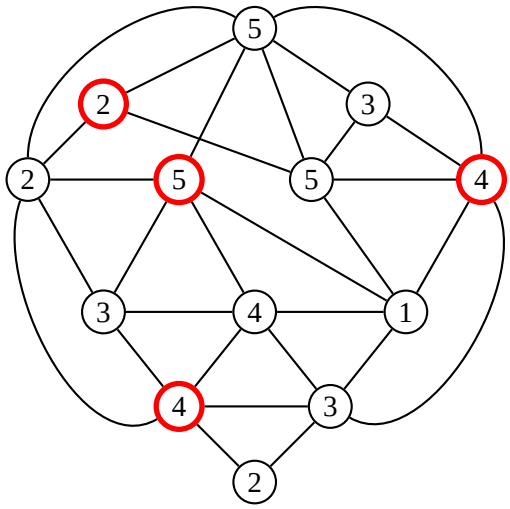

Figure A.4 – Une solution optimale du second problème de *set packing* de l'exemple 1.3. Les invités sélectionnés sont indiqués ici, soit 4 personnes. Cette solution a une valeur de 15 tandis que celle de l'exemple précédent aurait eut une valeur de 11.

ANNEXE B

Publications et communications

B.1 Communications avec actes

D. SALAZAR, X. GANDIBLEUX, J. JORGE et M. SEVAUX. A Robust-Solution-Based Methodology to Solve Multiple-Objective Problems with Uncertainty. Dans *Multiobjective Programming and Goal Programming Theoretical Results and Practical Applications Multiobjective Programming and Goal Programming*. Volume 618 de *Lecture Notes in Economics and Mathematical Systems*, pages 197–207. Springer. V. Barichard, M. Ehrgott, X. Gandibleux et Vincent T'Kindt réds., 2008.

X. GANDIBLEUX, J. JORGE, S. MARTINEZ et N. SAUER. Premier retour d'expérience sur le flow-shop biobjectif et hybride à deux étages avec une contrainte de blocage particulière. *MOSIM'06 : 6ème Conférence Francophone de Modélisation et Simulation MOSIM'06 : 6ème Conférence Francophone de Modélisation et Simulation*, volume II, pages 1355–1362. Lavoisier, M. Gourgand et F. Riane réds. 2006.

B.2 Communications sans actes

J. JORGE, X. GANDIBLEUX et M. WIECEK. A Priori Reduction of the Size of the Binary Multiobjective Knapsack Problem, 24 septembre 2008. *MOPGP'08 Multi-Objective Programming and Goal Programming*.

J. JORGE et X. GANDIBLEUX. Branch and bound algorithm for the 0-1 knapsack problem with multiple objectives, 13 juillet 2008. *IFORS'08 International Federation of Operational Research Societies Conference*.

J. JORGE, X. GANDIBLEUX et M. WIECEK. Fixation de variables par dominance en utilisant des bornes sur la cardinalité des solutions, appliquée au problème de sac à dos multi-objectif, 25 février 2008. *ROADEF'08*.

J. JORGE, X. GANDIBLEUX et A. PRYBYLSKI. Ranking algorithm for the 01 unidimensional multi-objective knapsack problem, *GOR'07 : Operations Research*. 5 septembre 2007.

J. JORGE et X. GANDIBLEUX. Nouvelles propositions pour la résolution exacte du problème de sac à dos bi-objectif unidimensionnel en variables binaires, 20 février 2007. *FRANCORO V / ROADEF'07*.

M. SEVAUX, D. SALAZAR, X. GANDIBLEUX et J. JORGE. A robust-solution-based methodology to solve multiple-objective problems with uncertainty, 12 juin 2006. *7th International Conference on Multi-Objective Programming and Goal Programming, MOPGP'06.*

X. GANDIBLEUX, J. JORGE, S. ANGIBAUD, X. DELORME et J. RODRIGUEZ. An ant colony optimization inspired algorithm for the set packing problem with application to railway infrastructure, 22 août 2005. *6th Metaheuristics International Conference (MIC).*

B.3 Chapitres d'ouvrages scientifiques

X. GANDIBLEUX, J. JORGE, X. DELORME et J. RODRIGUEZ. *Algorithme de fourmis pour mesurer et optimiser la capacité d'un réseau ferroviaire*, volume 1 de *Fourmis articielles, des bases algorithmiques aux concepts et réalisations avancés*, chapitre 9, pages 215–244. N. Monmarché, F. Guinand, et P. Siarry, 2009.

B.4 Communications dans des groupes de travail

J. JORGE et X. GANDIBLEUX. Algorithmes pour la résolution exacte du problème 01KP unidimensionnel bi-objectif, 9 mars 2007. *1ère journée du groupe de travail de la ROADEF « KnapSack et Optimisation »*.

J. JORGE et X. GANDIBLEUX. Self-adaptive stopping condition for an ant colony optimization inspired algorithm designed for set packing problems, 16 novembre 2006. *7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*.

B.5 Autres publications

J. JORGE. Algorithme pour la résolution exacte d'un 01-IP bi-objectif, août 2006. Rapport de stage de master recherche, université de Nantes.

Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires

Julien JORGE

Résumé

Ce travail porte sur la résolution exacte d'un problème d'optimisation combinatoire multi-objectif. Nous cherchons d'une part à confirmer l'efficacité de l'algorithme dit en deux phases, et d'autre part à poser une généralisation des procédures de séparation et évaluation, populaires dans le cadre mono-objectif mais presque absentes en multi-objectif. Notre étude s'appuie sur le problème multi-objectif de sac à dos unidimensionnel en variables binaires. Ce dernier est un classique de l'optimisation combinatoire, présent comme sous problème dans de nombreux problèmes d'optimisation.
La première partie de nos travaux porte sur un pré-traitement permettant de réduire la taille d'instances de ce problème. Nous mettons en évidence plusieurs propriétés permettant de déterminer *a priori* une partie de la structure de toutes les solutions efficaces. Nous nous attachons ensuite à décrire une procédure performante de type deux phases pour ce problème, tout d'abord dans le cas bi-objectif, où nous améliorons la procédure décrite par Visée *et al.* en 1998. Puis nous proposons un nouvel algorithme permettant de trouver plus efficacement les solutions recherchées durant la seconde phase. Nous étendons ensuite cette procédure pour des instances ayant trois objectifs ou plus. Les résultats obtenus sont comparés aux meilleurs algorithmes existants pour ce problème et confirment l'efficacité de l'approche en deux phases. La dernière partie de notre travail concerne la généralisation au cas multi-objectif d'une procédure de séparation et évaluation. Nous identifions plusieurs difficultés auxquelles nous répondons en proposant deux nouvelles procédures. Les expérimentations numériques indiquent que ces dernières permettent de résoudre des instances en des temps raisonnables, bien qu'elles n'atteignent pas les performances d'une procédure de type deux phases.

Mots-clés : optimisation combinatoire multi-objectif, problème de sac à dos unidimensionnel en variables binaires, résolution exacte, pré-traitement, méthode en deux phases, procédure de séparation et évaluation

Abstract

The purpose of this work is the exact solution of a problem from the field of multi-criteria combinatorial optimisation. Our goal his twofold. First, we aim at confirming the efficiency of the so-named two-phases algorithms. Then, we set a generalisation of the branch and bound procedures, popular in the mono-criteria case but almost non-existent in the multi-criteria case. Our work is based on the unidimensional multi-criteria knapsack problem with binary variables, a classic from combinatorial optimisation, found as a sub problem in many optimisation problems.
The first part concerns the reduction of the instances of the problem. We expose several properties allowing to *a priori* find some parts of the structure of all efficient solutions. Then, we describe an efficient two-phases procedure for this problem. Initially in the bi-criteria case, we improve the original procedure from Visée *et al.* (1998) before defining a new procedure to efficiently find the solutions in the second phase. This algorithm is extended to the tri- and multi-criteria case in the next part. Finally, the generalisation of the branch and bound procedure is the last part of our work. We focus on several difficulties, to which we answer with two new procedures. Numerical experiments show that these procedures can solve instances in acceptable time. Nevertheless, the two-phases algorithms outperform these procedures, just like the best known procedures for this problem.

Keywords: multi-objective combinatorial optimisation, unidimensional binary knapsack problem, exact resolution, preprocessing, two phase method, branch and bound