République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed Bougara Boumerdes
Faculté des Sciences
Département de Mathématiques

Mémoire Présenté

Pour L'Obtention Du Diplôme De Master

En Recherche Opérationnelle

Option : Recherche Opérationnelle et Mathématique de Gestion (ROMAGE)

Par : **RABIA** Mohamed Akli
Et : **HAROUNI** Youcef

Approche de résolution d'un problème de sac à dos bi-objectif en variables binaires

Soutenu à l'UMBB, le 18/06/2017, devant le jury composé de :

M^{me} BENMENSOUR. M	M.A. classe/ B	Présidente	à l'UMBB - Boumerdes.
M^{me} ZOUAOUI. S	M.A. classe/ A	Encadreur	à l'UMBB - Boumerdes
M^{me} DRICI. W	M.A. classe/ B	Examinatrice	à l'UMBB - Boumerdes.
M^{lle} AZZOUT. C	Invitée		

Année Universitaire 2016 – 2017

Table des matières

Table des figures

Liste des tableaux

Remerciement

Ce mémoire est le résultat d'un travail de plusieurs mois. En préambule, On adresse tous nos remerciements aux personnes avec lesquelles on a pu échanger et qui nous ont aidés pour la rédaction de ce modeste travail.

En commençant par remercier tout d'abord Mlle Zouaoui.S, notre promotrice , pour son aide précieuse, ses conseils et pour le temps qu'elle nous a consacré.

Merci aux membres de jury pour l'honneur qu'ils nous ont fait en acceptant de siéger à notre soutenance.

Enfin, nous adressons nos plus sincères remerciements à nos familles : parents, frères et soeurs, proches et amis, qui nous ont accompagnés, aidés, soutenus et encouragés tout au long de la réalisation de ce mémoire.

Notations

y^N : *Point nadir*
$\mathbb{R}^p_{\geq}$: $\{\ y \in \mathbb{R}^p,\ y \geq 0\ \}$
$\mathbb{R}^p_{>}$: $\{\ y \in \mathbb{R}^p,\ y > 0\ \}$
X_{SE_M} : *Ensemble complet maximum de solutions supportées*

Introduction générale

Un problème d'optimisation est défini comme étant un problème de recherche, qui consiste à explorer un espace contenant l'ensemble de toutes les solutions potentielles réalisables, dans le but de trouver la solution optimale, sinon la plus proche possible de l'optimum, permettant de minimiser ou maximiser une fonction dite objectif, les variables de décision peuvent être soit continues et on parle alors de problème continu, soit discrètes et on parle donc de problème combinatoire.

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines applicatifs parmi eux on peut citer :

— Le problème de voyageur de commerce : on souhaite visiter un ensemble de villes et revenir au 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.

— Le problème de sac a dos : il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum. Ce problème sera décrit d'une manière plus accentué dans le deuxième chapitre.

La plupart de ces problèmes appartiennent à la classe des problèmes NP-difficile : classe où il n'existe pas d'algorithme qui fournit la solution optimale en temps polynomial en fonction de la taille du problème.

La plupart des travaux réalisés dans ce domaine étaient dédiés à l'optimisation d'un seul objectif, Cependant, dans la pratique, il existe une multitude d'applications pour lesquelles un modèle ne prenant en compte qu'un seul objectif d'évaluation des solutions possibles, ne répond en aucun cas à la réalité. En effet, de nombreux problèmes rencontrés quotidiennement dans divers domaines (transport, télécommunication, économie ...etc) nécessitent la considération de plusieurs objectifs contradictoires simultanément. Optimiser de tels problèmes relève donc de l'optimisation combinatoire multi-objectif.

L'optimisation multi-objectif consiste donc à optimiser simultanément plusieurs fonctions en cherchant le meilleur compromis possible sachant que l'amélioration d'un objectif entraine la détérioration d'un autre objectif qui lui est conflictuel. La notion de solution optimale unique dans l'optimisation mono-objectif disparait pour les problèmes d'optimisation multi-objectif au profit de la notion d'ensemble de solutions Pareto optimales. Ces solutions sont optimales dans le sens qu'aucune autre solution dans l'espace de recherche n'est supérieure à elles, lorsque tous les objectifs sont considérés simultanément.

Qu'ils comportent un seul ou plusieurs objectifs, les problèmes d'optimisation sont en général difficiles à résoudre. De plus, le temps de calcul nécessaire à leur résolution peut devenir si important que l'algorithme développé devient inutilisable en pratique. Il n'existe pas d'algorithme générique capable de résoudre toutes les instances de tous les problèmes efficacement.

Un grand nombre de méthodes ont été développées pour tenter d'apporter une réponse satisfaisante à ces problèmes. Parmi celles-ci, nous distinguons les méthodes dédiées à un problème spécifique et les méthodes plus générales, pouvant s'appliquer à un ensemble de problèmes. Parmi ces méthodes, nous relevons deux grandes classes de méthodes : les méthodes

exactes et les méthodes approchées.

Les méthodes de résolution exactes permettent d'obtenir une solution dont l'optimalité est garantie, en examinant souvent de manière implicite la totalité de l'espace de recherche. On peut cependant chercher des solutions de bonne qualité, sans garantie d'optimalité, mais au profit d'un temps de calcul plus réduit. Pour cela, on applique des méthodes appelées métaheuristiques, adaptées à chaque problème traité, avec cependant l'inconvénient de ne disposer en retour d'aucune information sur la qualité des solutions obtenues.

Dans ce mémoire nous allons présenter une méthode de résolution d'un problème de sac à dos unidimensionnelle bi-objectif en variable binaire qui est la méthode en deux phases dont la première phase est basée sur la recherche dichotomique et la seconde utilise la méthode de séparation et évaluation (branch and bound).

Le premier chapitre présente l'état de l'art de l'optimisation combinatoire multi-objectif, les concepts de bases, les principales définitions ainsi que les méthodes classiques de résolutions exactes et approchées.

Le deuxième chapitre est consacré au problème du sac à dos (Knapsack problem), nous décrivons les variantes de ce problème ainsi que les méthodes de résolution.

Le troisième et quatrième chapitre sont le coeur de ce travail, ils décrivent la méthode en deux phases appliquée à un problème de sac à dos bi-objectif.

Une conclusion générale termine ce manuscrit tout en dénombrant de nouvelles perspectives sur la base des travaux effectués.

1

Etat de l'art de l'optimisation multi-objectif

1.1 Introduction

Etant confronté chaque jour à des problèmes d'optimisation dans la vie quotidienne, ces problèmes sont souvent de nature multi-objectif car plusieurs critères d'évaluation souvent contradictoires sont à considérer simultanément. Optimiser de tels problèmes relève de l'optimisation combinatoire multi-objectif.

L'optimisation multi-objectif possède ses racines dans les travaux en économie de Edgeworth [12] et Pareto [27]. Elle a ainsi été initialement utilisée en économie et dans les sciences du management, puis graduellement dans les sciences pour l'ingénieur.

Dans ce chapitre, nous présentons l'optimisation multi-objectif, nous abordons les notions de base telles que la dominance, l'éfficacité...etc. Nous décrivons aussi les différentes méthodes de résolution des problèmes multi-objectif.

1.2 Définition

L'optimisation multi-objectif cherche à optimiser simultanément plusieurs critères souvent contradictoires. Il ne s'agit plus dans ce cas de trouver une solution optimale mais un ensemble de solutions représentant un compromis acceptable entre les différents objectifs contradictoires et connus comme l'ensemble des solutions Pareto optimales.

1.3 Problème d'optimisation combinatoire multi-objectif

Un probléme d'optimisation combinatoire multi-objectif (PMO) peut étre définie comme suit :

$$\text{(PMO)} \begin{cases} \text{"}optimiser\text{"} \ F(x) = (f_1(x), f_2(x)..., f_k(x)) \\ \quad sous \ x \in X. \end{cases}$$

où k est le nombre d'objectifs ($k \geq 2$), $x = (x_1, x_2, ..., x_n)$ est le vecteur représentant les variables de décision, chacune des fonctions $f_i(x)$ est à optimiser $i = 1, ..., k$, c'est-à-dire à minimiser ou à maximiser et X représente l'ensemble des solutions réalisables.

- l'ensemble $\mathbb{R}^n$ qui contient X est dit *espace de décision.*
- l'ensemble $\mathbb{R}^k$ qui contient F est dit *espace des critéres* ou *espace des objectifs.*

Sans perte de généralité, nous supposerons par la suite que nous considérons des problèmes de minimisation sauf indication contraire.

1.4 Concepts de bases

Lors de la résolution d'un problème d'optimisation multi-objectif , nous obtenons une multitude de solutions. Seul un nombre restreint de ces solutions va nous intéresser [9].

1.4.1 Dominance de Pareto

Définition 1.4.1. *Soient deux vecteurs critères $z, z' \in F(X)$ .On dit que z domine z' si et seulement si $z \leq z'$ et $z \neq z'$ (i.e. $z_i \leq z'_i \ \forall \ i = 1, ..., k$ et $z < z'$ pour au moins un indice i).*
Cela veut dire que z est au moins aussi bon que z' dans tous les objectifs et, z est strictement meilleur que z' dans au moins un objectif.

Définition 1.4.2. ***Dominance forte***
Soient deux vecteurs critères $z, z' \in F(X)$. On dit que z domine fortement z' si et seulement si $z < z'$ et $z \neq z'$ (i.e. $z_i < z'_i \quad \forall \ i = 1, ..., k$).
Cela veut dire que z est meilleur que z' sur tous les critéres.

1.4.2 Efficacité

Définition 1.4.3. *Une solution $x^* \in X$ est une solution efficace ou pareto optimal s'il n'éxiste pas de $x \in X$ tel que $F(x)$ domine $F(x^*)$.*

Définition 1.4.4. ***Efficacité faible***
Une solution $x^* \in X$ est dite faiblement efficace s'il n'éxiste pas de $x \in X$ telle que $F(x) < F(x^*)$.
Une solution est faiblement efficace si son vecteur critère n'est pas fortement dominé.

Définition 1.4.5. ***Efficacité forte***
Une solution $x^* \in X$ est dite fortement efficace s'il n'éxiste pas de $x \in X$ telle que $x \neq x^*$ et $F(x) \leq F(x^*)$.
Une solution x^* est fortement efficace s'il n'éxiste pas une autre solution telle que le vecteur critére, qui lui est associé soit aussi bon que celui x^*.

Remarque 1 :

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées solutions optimales au sens de Pareto (où solutions non dominées), elle forment le front de pareto [9].

Définition 1.4.6. ***Front de pareto***

Le front de pareto est l'ensemble de tous les points pareto-optimaux [6].

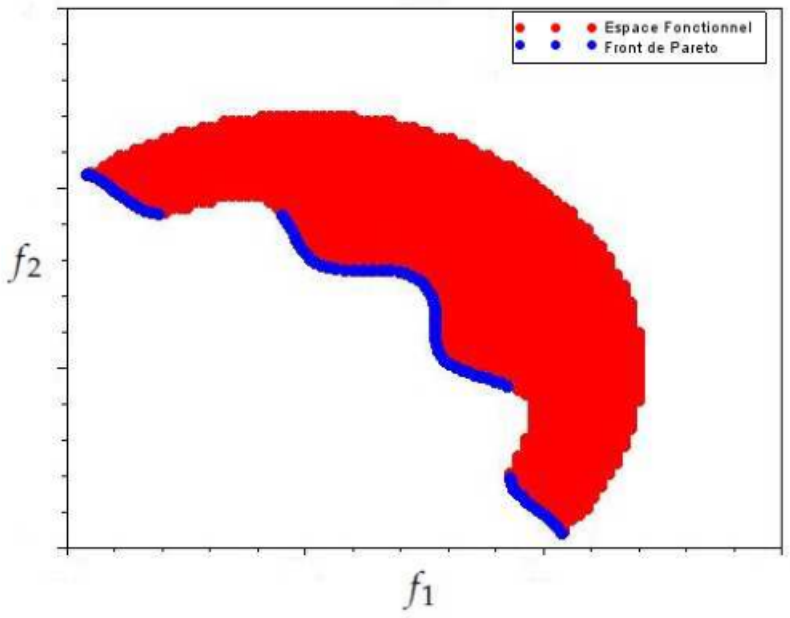


FIGURE 1.1 – Représentation du front de pareto

1.5 Optimalité lexicographique

Définition 1.5.1. *(Ordre lexicographique)*

Soient y_1 *et* $y_2 \in \mathbb{R}^k$, y_1 *est optimale au sens lexicographique ie :* $y_1 \leq_{lex} y_2$ *s'il existe* $j \in \{1, ..., n\}$ *tel que pour tout* $i < j$, $y_{1_i} = y_{2_i}$ *et* $y_{1_j} < y_{2_j}$.

Exemple : Considerons les deux points X et Y dans $\mathbb{R}^6$

$$X = (5, 4, 2, 6, 0, 8)$$
$$Y = (5, 4, 2, 6, 7, 9)$$

Pour ces deux points, nous avons $X \leq_{lex} Y$ car, jusqu'à la quatrième position, nous avons
$X_i = Y_i, \forall i = \{1, 2, 3, 4\}$ et, pour $i = 5$, on a : $X_i < Y_i$ $(0 < 7)$. On conclut donc que la solution X domine lexicographiquement la solution Y.

1.5.1 Points particuliers

En vue d'avoir certains points de références permettant de discuter de l'intérêt des solutions trouvées, des points particuliers ont été définis dans l'éspace objectif. Ces points peuvent représenter des solutions réalisables ou non [8].

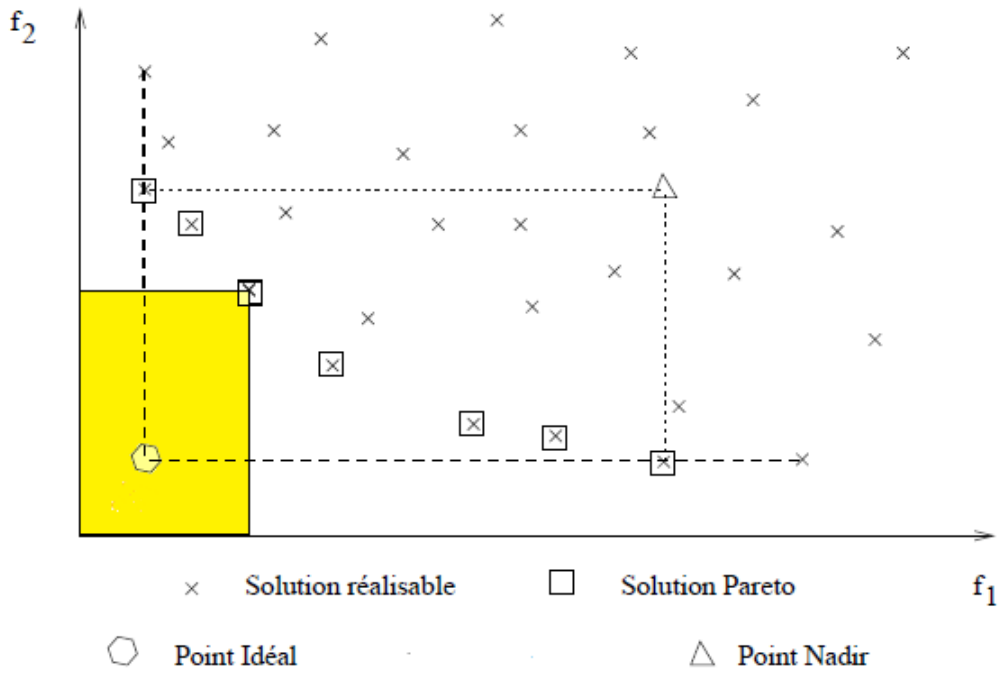


FIGURE 1.2 – Illustration des différentes définition

1. Le point **idéal** Z^I est le point qui a comme valeur pour chaque objectif, la valeur optimale de l'objectif considéré.

$$Z^I \text{ tel que } i \in [1...k], f_i\left(Z^I\right) = opt_{x \in X} f_i(x)$$

2. Le point **nadir**, Les coordonnées de ce point correspondent aux pires valeurs obtenues par chaque fonction objectif, lorsque l'on restreint l'espace des solutions à la surface de compromis.

Remarque 2 :

Le point idéal est utilisé dans beaucoup de méthodes d'optimisation comme point de référence. Le point nadir, lui, sert à restreindre l'espace de recherche ; il est utilisé dans certaines méthodes d'optimisation interactives [29].

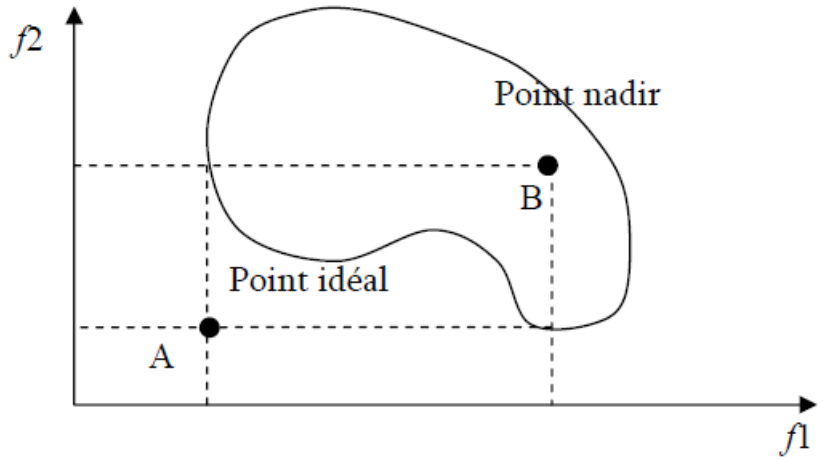


FIGURE 1.3 – Représentation du point idéal et du point nadir .

Théorème 1.5.1. *(Somme pondérée [24])*

Soit P un MOP avec k objectifs. Soit $\lambda \in \mathbb{R}^k$, on construit le problème mono-objectif (P_λ) :

$$(P_\lambda) \begin{cases} max\ F^\lambda(x) = \lambda \cdot F(x) \\ s.c \\ Ax \leq b \\ x \in \{0,1\}^n \end{cases}$$

où l'opérateur $\cdot$ représente le produit scalaire.
Si x^* est une solution optimale de P_λ, alors cette solution est pareto optimale.

Définition 1.5.2. ***Convexité***

Un ensemble D est dit convexe si tout segment joignant deux points quelconques de D est inclus dans D.

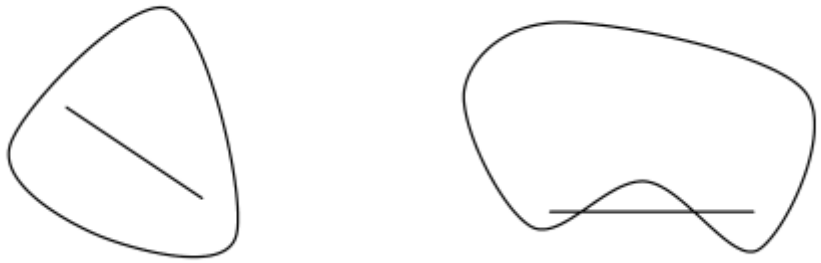


FIGURE 1.4 – Espace convexe (à gauche) et non convexe (à droite).

1.5.2 Structure du front pareto

Front minimal / front maximal complet

La définition de front se réfère à l'espace des objectifs. Une solution appartient au front si elle n'est dominée par aucune autre solution réalisable. Lorsque deux solutions ont exactement les mêmes valeurs pour l'ensemble des objectifs, elles sont équivalentes dans l'espace objectif, mais peuvent correspondre à deux solutions différentes dans l'espace décisionnel. Une question importante est de savoir s'il est intéressant de garder ces deux différentes solutions. La réponse peut dépendre du contexte (type de problème étudié) en plus de la volonté des décideurs :

— Lors de la résolution d'un problème comportant énormément de solutions Pareto, il est peut être préférable de privilégier une bonne approximation de l'ensemble de la frontière et donc favoriser la diversité (du coté objectif) des solutions retenues.

— Au contraire, lorsque la frontière Pareto comporte peu de solutions, afin d'avoir une bonne représentation de l'ensemble des solutions non dominées, il sera intéressant de rechercher les solutions de même valeur.

Nous parlerons alors de recherche du **front minimal**, dans le premier cas, et du **front maximal complet**, dans le second.

Solutions supportées / non supportées

Sur le front Pareto, deux types de solutions peuvent être différenciées : les solutions supportées et les solutions non supportées. Les premières sont celles situées sur l'enveloppe convexe de l'ensemble des solutions (voir la figure 1.5) et peuvent donc être trouvées à l'aide d'une agrégation linéaire des objectifs [24]. Elles sont donc plus simples à obtenir que les solutions non supportées qui elles, sont plus difficile à trouver.

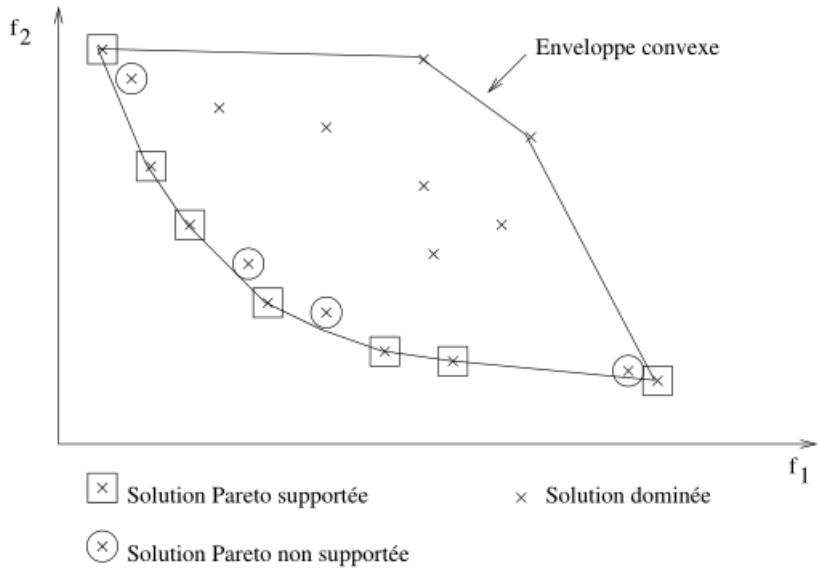


FIGURE 1.5 – Les différents types de solutions en bi-objectif.

1.6 Choix de la méthode d'aide a la décision

La résolution d'un problème multi-objectif menant à la détermination d'un ensemble de solutions Pareto, il est nécessaire de faire intervenir l'humain à travers un décideur, pour le choix final de la solution à garder. Ainsi, avant de se lancer dans la résolution d'un problème multi-objectif, il faut se poser la question du type de méthode d'optimisation à utiliser. En effet, on peut répartir les méthodes de résolution de problèmes multi-objectif en trois familles, en fonction du moment où intervient le décideur.
Ainsi nous pouvons trouver les familles suivantes :

— Les méthodes d'optimisation **a priori** : dans ce cas, le compromis que l'on désire faire entre les objectifs à été défini avant l'exécution de la méthode. Ainsi une seule exécution permettra d'obtenir la solution recherchée. Cette approche est donc rapide, mais il faut cependant prendre en compte le temps de modélisation du compromis et la possibilité pour le décideur de ne pas être satisfait de la solution trouvée et de relancer la recherche avec un autre compromis.

— Les méthodes d'optimisation **progressives** : ici, le décideur intervient dans le processus de recherche de solutions en répondant à différentes questions afin d'orienter la recherche. Cette approche permet donc de bien prendre en compte les préférences du décideur, mais nécessite sa présence tout au long du processus de recherche.

— Les méthodes d'optimisation **a posteriori** : dans cette troisième famille de méthodes, on cherche à fournir au décideur un ensemble de bonne solutions bien réparties. Il peut ensuite, au regard de l'ensemble des solutions, sélectionner celle qui lui semble la plus appropriée. Ainsi, il n'est plus nécessaire de modéliser les préférences du décideur (ce qui peut s'avérer être très difficile), mais il faut en

contre partie fournir un ensemble de solutions bien réparties, ce qui peut également être difficile et requérir un temps de calcul important (mais ne nécessite pas la présence du décideur).

1.6.1 Les méthodes exactes

Les méthodes exactes cherchent à trouver de manière certaine la solution optimale en examinant de manière explicite ou implicite la totalité de l'espace de recherche. Elles ont l'avantage de garantir la solution optimale néanmoins le temps de calcul nécessaire pour atteindre cette solution peut devenir très excessif en fonction de la taille du problème (explosion combinatoire) et le nombre d'objectifs à optimiser. Ce qui limite l'utilisation de ce type de méthode aux problèmes bi-objectifs de petites tailles.
Les références existantes et qui présentent la plupart des méthodes exactes sont Ulungu [34] et Ehrgott [13].
Ces méthodes sont basées principalement sur les procédures de Branch and Bound, Cut ou Price et la programmation dynamique. Une approche particulière pour l'optimisation multi-objectif est la programmation par but.

La recherche dichotomique

La recherche dichotomique offre un schéma d'application de l'agrégation linéaire permettant d'obtenir les solutions non supportées. Cette méthode consiste à explorer de façon dichotomique des intervalles de front de plus en plus petits.
Tout d'abord les solutions extrèmes sont recherchées. Puis une recherche est menée entre ces solutions r et s suivant une direction perpendiculaire à la droite (r,s). En interdisant de ré-obtenir les solutions r et s et en éliminant les solutions dominées par ces solutions, cette recherche trouve la meilleure solution Pareto relativement à cette direction de recherche, solution qui peut alors être non supportée. Cette nouvelle solution crée deux nouveaux intervalles qu'il faut explorer de la même façon. Cette méthode, dédiée au

bi-objectif, est intéressante mais nécessite de l'ordre de 2^n recherches, si n est le nombre de solutions du front Pareto.

Methode en deux-phases

La méthode deux-phases a initialement été proposée par Ulungu et Teghem pour la résolution d'un problème d'affectation bi-objectif. Comme son nom l'indique, cette méthode est décomposée en deux étapes : la première consiste à trouver toutes les solutions supportées du front Pareto, puis la deuxième phase cherche entre ces solutions les solutions Pareto non supportées. Cette méthode travaille donc essentiellement dans l'espace objectif.

Premiére phase

L'objectif de la première phase est d'obtenir l'ensemble des solutions Pareto supportées. Comme nous l'avons vu précédemment, ces solutions ont l'avantage d'être relativement faciles à trouver puisqu'elles optimisent une certaine combinaison linéaire des objectifs. Ainsi, durant la première phase de la méthode, les deux solutions extrêmes (solutions optimisant chacune un des deux objectifs) sont recherchées (voir figure 1.6.a). Puis, de façon récursive, dès que deux solutions supportées r et s sont trouvées, la méthode recherche d'éventuelles autres solutions supportées entre r et s, à l'aide de combinaisons linéaires bien choisies des objectifs (voir figure 1.6.b et 1.6.c). A la fin de la première phase l'ensemble des solutions supportées est donc trouvé (voir figure 1.6.d). Cette première phase rappelle la méthode dichotomique, mais ici seules les solutions supportées sont recherchées. Pour cela, lors de l'exploration entre deux solutions, on s'autorise à retrouver l'une de ces deux solutions, lorsqu'il n'existe pas d'autre solutions supportées dans l'intervalle.

Deuxiéme phase

La deuxième en phase consiste alors en la recherche des solutions non supportées appartenant au front Pareto. Ces solutions ne peuvent être obtenues par combinaisons d'objectifs. Alors on utilise les solutions supportées trouvées pour réduire l'espace de recherche puisque les solutions Pareto non supportées restantes sont forcément dans les triangles rectangles basés sur deux solutions supportées consécutives (voir figure 1.6.e). Ainsi, une recherche de type deuxième phase est exècutèe entre chaque couple de solutions supportèes adjacentes (voir figure 1.6.f et 1.6.g). La méthode de recherche au sein de ces triangles dèpend du problème ètudiè. A la fin de la deuxième phase, toutes les solutions Pareto sont trouvées.

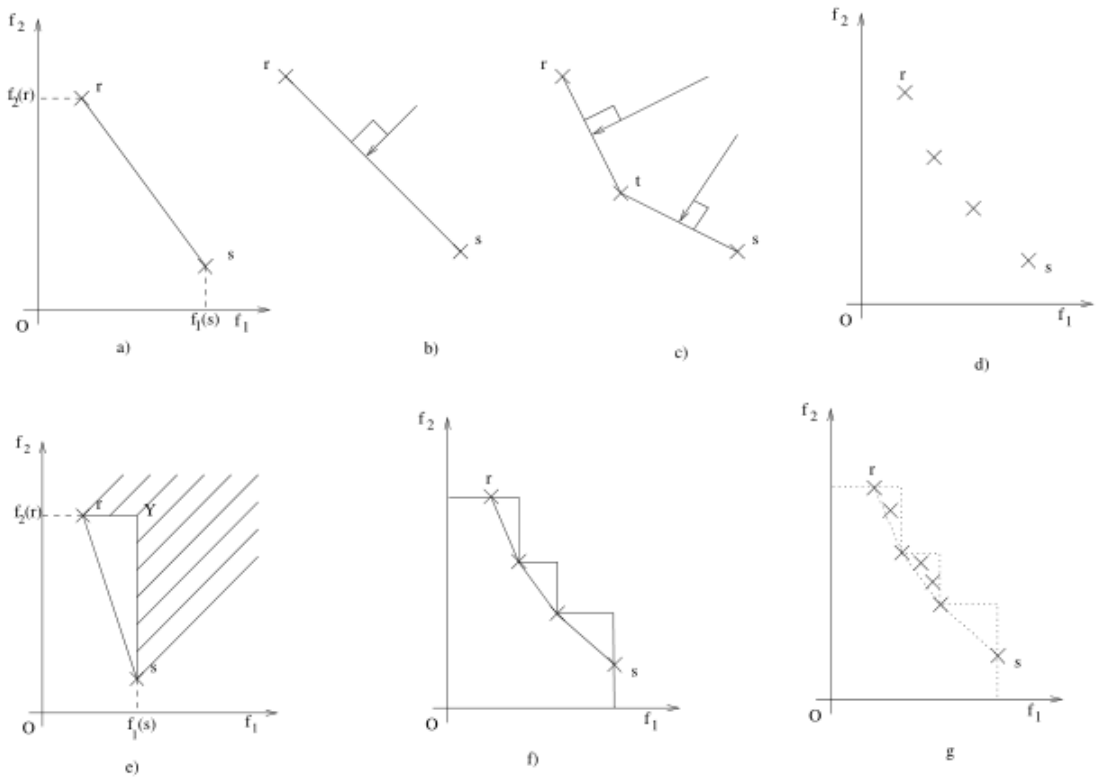


FIGURE 1.6 – Illustration des différentes étapes de la méthode en deux phases.

Méthode des sommes pondérées

Cette méthode de résolution d'un problème d'optimisation multi-objectif est la plus évidente et, probablement, la plus largement utilisée en pratique. Elle consiste a ramener le problème multi-objectif au problème de l'optimisation d'une combinaison linéaire des objectifs initiaux. Ainsi, il s'agit d'associer a chaque fonction objectif un coefficient de pondération et a faire la somme des fonctions objectifs pondérées pour obtenir une nouvelle et unique fonction objectif.
Un problème multi-objectif se transforme alors de la manière suivante :

$$\min \sum_{i=1}^{m} \lambda_i f_i(x) \quad m \geq 2$$

où les poids $\lambda_i \geq 0$ *sont tels que* : $\sum_{i=1}^{m} \lambda_i = 1.$

Les résultats obtenus avec de telles méthodes dépendent fortement des paramètres choisis pour le vecteur de poids. Les poids λ_i doivent également être choisis en fonction des préférences associées aux objectifs, ce qui est une tache délicate. Une approche généralement utilisée consiste à répéter l'exécution de l'algorithme avec des vecteurs poids différents.
Cette méthode est très efficace du point de vue algorithmique, mais son principal inconvénient est qu'elle ne permet pas de trouver les solutions enfermées dans des concavités (surface de compromis non-convexe).

La méthode de programmation par but, (Goal programming)

Cette méthode est également appelée target vector optimisation, le décideur fixe un but G_i à atteindre pour chaque fonction objectif f_i. Ces valeurs sont ensuite ajoutées au problème comme des contraintes supplémentaires. La nouvelle fonction objectif est modifiée de façon à minimiser la somme des écarts entre les résultats et les buts à atteindre :

$$\min \textstyle\sum_{i=1}^{k} |f_i(x) - G_i|$$

G_i represente le but a atteindre pour le i^{eme} objectif.
Cette méthode est facile à mettre en oeuvre mais la définition des poids et des objectifs à atteindre est une question délicate qui détermine l'efficacité de la méthode.

Méthode ϵ contrainte

Cette méthode permet de transformer le problème d'optimisation multi-objectif en un problème mono-objectif. La méthode consiste à convertir $m-1$ des m objectifs du problème en contraintes et d'optimiser séparément l'objectif restant [9].
La démarche est la suivante :

— Nous choisissons un critère à optimiser prioritairement.
— Nous transformons le problème conservant l'objectif prioritaire et nous transformons les autres objectifs en des contraintes d'inégalité.

Le problème peut être reformulé de la manière suivante :

$$\begin{cases} Min\ f_i(x) \\ tq : \\ f_1(x) \leq \epsilon_1 \\ . \\ . \\ . \\ f_{i-1}(x) \leq \epsilon_{i-1} \\ f_{i+1}(x) \leq \epsilon_{i+1} \\ . \\ . \\ . \\ f_m(x) \leq \epsilon_m \\ et\ que\ g(x) \leq 0 \\ avec \quad x \in \mathbb{R}^n, f(x) \in \mathbb{R}^m, g(x) \in \mathbb{R}^q \end{cases}$$

L'approche par ϵ-contrainte doit aussi être appliquée plusieurs fois en faisant varier le vecteur ϵ pour trouver un ensemble de points Pareto optimaux. Cette approche a l'avantage de ne pas être trompée par les problèmes

non convexes. Ainsi la figure 1.7 illustre, en dimension 2, le cas où un point $(\epsilon; f_{1min})$, de la partie non convexe, est trouvé. La figure 1.7 montre aussi comment cette approche procède.
En transformant des fonctions objectifs en contraintes, elle diminue la zone réalisable par paliers. Ensuite, le processus d'optimisation trouve le point optimal sur l'objectif restant.
L'inconvénient de cette approche réside dans le fait qu'il faille lancer un grand nombre de fois le processus de résolution. De plus, pour obtenir des points intéressants et bien répartis sur la surface de compromis, le vecteur ϵ doit être choisi judicieusement.

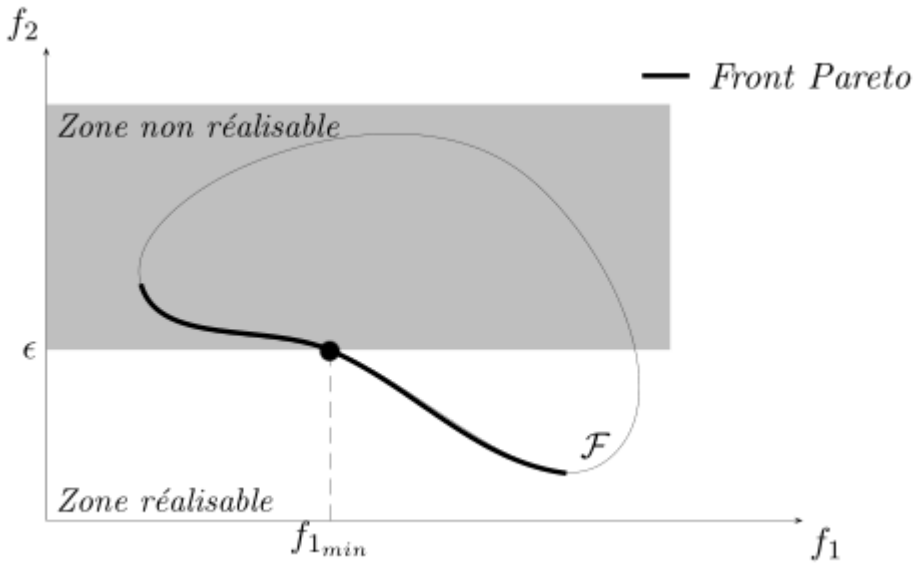


FIGURE 1.7 – Interprétation graphique de la méthode .

1.6.2 Les méthodes approchées

Méthodes souvent inspirées de mécanismes d'optimisation rencontrés dans la nature. Elles sont utilisées pour les problèmes où on ne connaît pas d'algorithmes de résolution en temps polynomial et pour lesquels on espère trouver une solution approchée de l'optimum global. Elles cherchent à produire une solution de meilleure qualité possible dictée par des heu-

ristiques avec un temps de calcul raisonnable en examinant seulement une partie de l'espace de recherche. Dans ce cas l'optimalité de la solution n'est pas garanti ni l'écart avec la valeur optimal. Parmi ces heuristiques, on trouve les métaheuristiques qui fournissent des schémas de résolution généraux permettant de les appliquer potentiellement à tous les problèmes. Plusieurs classifications des métaheuristiques ont été proposées, la plupart distinguent globalement deux catégories : celles se basant sur une solution unique (algorithmes d'exploration de voisinage) et celles se basant sur une population de solution (algorithmes évolutionnaires) [14].

Les algorithmes d'exploration de voisinage (à solution unique)

Le principe général le plus utilisé dans les méthodes métaheuristiques est celui de voisinage ; cette dernière est un impact important sur le comportement des métaheuristiques pour les problèmes combinatoires ; tel que à chaque solution d'un problème, nous associons un sous ensemble de solutions. Ce sous ensemble qu'il est possible d'atteindre par une série de transformations de données . Par extension nous désignons par le terme " voisinage" l'ensemble des transformations considérées [20].
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 données 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 .

Le récuit simulé

Le recuit simulé a été Introduit par Kirkpatrick en 1982. Il tire son origine de la mécanique statistique en utilisant les critères de la méthode

de Métropolis 1953. Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux basé sur les principes d'équilibre énergétique lors de la cristallisation des métaux. L'analogie avec une méthode d'optimisation est trouvée en associant une solution à un état du métal, on équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine. Le recuit simulé est une technique stochastique de type Monte-Carlo généralisé pour la résolution approchée de problèmes NP-difficile de l'optimisation combinatoire basé sur un paramètre appelé température, qui sera ajusté au cours de la recherche. A partir d'une solution initiale il recherche dans son voisinage une autre solution de façon aléatoire qui peut être de moins bonne qualité permettant d'échapper aux optima locaux en acceptant temporairement une dégradation de la fonction objectif. Initialement, la température est élevée autorisant une forte dégradation de qualité puis décroît pour diminuer la probabilité des dégradations importante [25].

La recherche tabou

La recherche tabou (TS) est une méthode de recherche locale combinée avec un ensemble de techniques permettant d'éviter d'être piégé dans un minimum local ou la répétition d'un cycle. La recherche tabou est introduite principalement par Glover (Glover 1986), Glover et Laguna dans (Glover et Laguna1997). Cette méthode a montré une grande efficacité pour la résolution des problèmes d'optimisation difficiles. En effet, à partir d'une solution initiales dans un ensemble de solutions local S, des sous-ensembles de solution $N(s)$ appartenant au voisinage S sont générés. Par l'intermédiaire de la fonction d'évaluation nous retenons la solution qui améliore la valeur de F , choisie parmi l'ensemble de solutions voisines $N(s)$. L'algorithme accepte parfois des solutions qui n'améliorent pas toujours la solution courante. Nous mettons en oeuvre une liste tabou (tabulist) T de longueur k contenant les k dernières solutions visitées, ce qui ne donne pas la possibilité a une solution déjà trouvée d'être acceptée

et stockée dans la liste tabou. Alors le choix de la prochaine solution est effectué sur un ensemble des solutions voisines en dehors des éléments de cette liste tabou. Quand le nombre k est atteint, chaque nouvelle solution sélectionnée remplace la plus ancienne dans la liste. La construction de la liste tabou est basée sur le principe FIFO, c'est-à-dire le premier entré est le premier sorti. Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de s^*, ou bien fixer un temps limite après lequel la recherche doit s'arrêter.

Grandibleux et al [16] ont proposé une version multi-objectif de la recherche taboue. La méthode utilise des fonctions pondérées scalaires dont les poids sont changés périodiquement. La modification du vecteur poids dégrade les poids des objectifs qui ont été nettement améliorés. Deux listes taboues sont utilisées. La première est une liste taboue régulière qui empêche de revenir aux solutions déjà visitées. La deuxième contient les vecteurs poids.

Hansen [17] a proposé une recherche taboue basée sur l'idée de la méthode "Pareto simulated annealing". La méthode utilise une population de solutions explorant chacune d'elles différentes régions de l'ensemble Pareto. Le vecteur poids est utilisé dans une fonction scalaire. De plus, à chaque solution est associée une liste taboue. La dispersion des solutions est assurée par la modification de leurs vecteurs poids. Pour chaque solution, la modification du vecteur poids vise à la déplacer des autres solutions proches dans l'espace des objectifs. Hansen a appliqué cette méthode au problème du sac à dos multi-objectif [1].

Algorithmes évolutionnaires (population de solution)

On peut distinguer trois grandes classes d'algorithmes évolutionnaires : les algorithmes génétiques [Holland, 1975 ; Goldberg, 1989], les stratégies d'évolution [Schwefel, 1981] et la programmation évolutive [Fogel, 2000]. Ces méthodes se distinguent par la manière de représenter l'information

et par la façon de faire évoluer la population d'une génération à l'autre. Un algorithme évolutionnaire est typiquement composé de trois éléments fondamentaux :

- une **population** constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné,
- un **mécanisme d'évaluation des individus** permettant de mesurer l'adaptation de l'individu à son environnement,
- un **mécanisme d'évolution de la population** permettant, grâce à des opérateurs predéfinés, d'éliminer certains individus et d'en créer de nouveaux.

Parmi les composants d'un algorithme évolutionnaire, **l'individu** et **la fonction d'évaluation** correspondent respectivement à la notion de configuration et à la fonction d'évaluation dans les méthodes de voisinage. Le mécanisme d'évolution est composé de plusieurs opérateurs tels que la sélection, la mutation et le croisement. La **sélection** a pour objectif de sélectionner des individus qui vont pouvoir se reproduire pour transmettre leurs caractéristiques à la génération suivante. **Le croisement** ou **recombinaison** est un opérateur permettant de construire de nouveaux individus enfants à partir des caractéristiques d'individus parents sélectionnés. **La mutation** effectue des légères modifications de certains individus [4].

Les algorithmes génétiques

Les algorithmes génétiques ont été largement utilisés dans la communauté multi-objectif. Ils sont trés appropriés pour résoudre des (PMO) grâce à l'utilisation d'une population de solutions. Ils peuvent chercher plusieurs solutions Pareto-optimales dans la même exécution.
Les algorithmes génétiques (AGs) ont été introduits par Holland [18] comme un modèle de méthode adaptative. Ils s'appuient sur un codage de l'information sous forme de chaines binaires de longueur fixe et d'un ensemble d'opérateurs génétiques : la sélection, la mutation, le croisement. Un individu sous ce codage, appelé un chromosome, représente une configuration du problème.

Des exemples d'algorithmes évolutionnaires sont donnés par VEGA (Vector Evaluated Genetic Algorithm [31], MOGA (Multiple Objective Genetic Algorithm [19]), (NSGA) (non dominated sorting genetic algorithm [32]), SPEA (Strength Pareto evolutionary algorithm [35]) ou encore M-PAES (memetic Pareto archived evolution strategy algorithm [22]).

1.7 Conclusion

Ce chapitre avait pour objectif de présenter dans un premier temps les principales définitions nécessaires à la présentation des problèmes d'optimisation combinatoire multi-objectif et quelques méthodes de resolution de ces problèmes, dans le chapitre suivant nous allons focaliser notre étude sur un cas particulier de ces problèmes, qui est le problème de sac à dos multi-objectif.

2

Problèmes de type sac à dos

2.1 Introduction

Le problème du sac à dos (ou knapsack problem (KP)) fait partie des problèmes d'optimisation combinatoire les plus étudiés ces cinquante dernières années, en raison de ces nombreuses applications dans le monde réel. En effet, ce problème intervient souvent comme sous-problème à résoudre dans plusieurs domaines : la logistique comme le chargement d'avions ou de bateaux, l'économie comme la gestion de portefeuille ou dans l'industrie comme la découpe de matériaux.

C'est un probléme, dans lequel l'utilisateur dispose d'un ensemble d'objets auxquels sont associés un profit et un poids. L'objectif est de sélectionner un sous-ensemble de ces objets tel que la somme des profits soit maximale et qu'ils tiennent tous ensemble dans un sac d'une capacité donnée.

La question qui se pose est quels objets doit-on mettre dans le sac ?

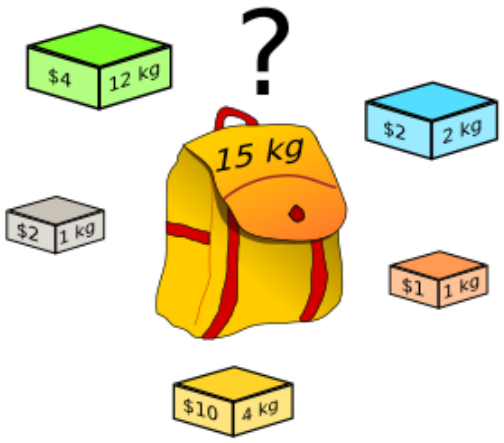


FIGURE 2.1 – Problème de sac à dos .

Dans ce chapitre, nous rappelons les diverses formes de problèmes, à savoir sac à dos mono-objectif unidimensionnel, multidimensionnel, sac à dos multi-objectif et sac à dos multi-objectif multidimensionnel, ainsi que les méthodes de résolutions (exactes et approchées).

2.2 Problème de sac à dos

Le problème de sac à dos est un problème classique d'optimisation combinatoire appartenant à la classe des problèmes NP-complets. L'énoncé de ce problème est simple : étant donné un ensemble de n objets, où chaque objet i est caractérisé par un poids w_i et un profit c_i on cherche le sous-ensemble d'objets à charger dans un sac de capacité w afin de maximiser la somme des profits. Dans sa forme la plus simple, dénomée « sac à dos unidimensionnel en variables binaires », le problème se formule ainsi :

$$\text{(01-KP)} \begin{cases} Max\ z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c: & \\ \sum_{i=1}^{n} w_i x_i \leq w & w, w_i \in \mathbb{N}^* \\ x_i \in \{0,1\} & \forall i \in \{1,...,n\} \end{cases}$$

Les poids w_i et les profits c_i ainsi que la capacité w sont des entiers

positifs $i \in \{1, ..., n\}$.

La variable x_i est la variable de décision ; elle prend la valeur 1 si l'objet i est chargé dans le sac, sinon elle prend la valeur 0.

2.2.1 Variantes autour du probléme

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. On présente dans cette section quelques unes d'entre elles.

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

Définition 2.2.2. *(**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*

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

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.

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

Le problème (LKP) est un problème relaxé de (01-KP). Il s'obtient en remplaçant la contrainte $x_i \in \{0,1\}$, par $x_i \in [0;1]$, $\forall i \in \{1,...,n\}$.
Il est définit ainsi :

$$\text{(LKP)} \begin{cases} Max\ z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c: & \\ \sum_{i=1}^{n} w_i x_i \leq w & w, w_i \in \mathbb{N}^* \\ x_i \in [0,1] & \forall i \in \{1,...,n\} \end{cases}$$

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. Son champ d'application est large, ce qui a grandement contribué à sa popularité.

$$\text{(d-KP)} \begin{cases} Max\ z(x) = \sum_{i=1}^{n} c_i x_i & c_i \in \mathbb{N}^* \\ s.c: & \\ \sum_{i=1}^{n} w_i^j x_i \leq w_j & w_j, w_i \in \mathbb{N}_{>}^d, j \in \{1,...,d\} \\ x_i \in \{0,1\}, & \forall i \in \{1,...,n\} \end{cases}$$

Sac à dos multiple

Le problème du sac à dos multiple (MKP) est une généralisation du problème standard du sac à dos en variable (0-1), où on essaie de remplir plusieurs sacs à dos de différentes capacités au lieu de considérer un seul sac à dos [23], Le (MKP) peut être formulé de la manière suivante :

$$\text{(MKP)} \begin{cases} Max\ z(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_i x_{ij} & c_i \in \mathbb{N}^* \\ s.c & \\ \sum_{i=1}^{n} w_i x_{ij} \leq w_j & j \in \{1,...,m\} \\ \sum_{j=1}^{m} x_{ij} \leq 1 & \forall i \in \{1,...,n\} \\ x_{ij} \in \{0,1\} & \forall i \in \{1,...,n\}, j \in \{1,...,m\} \end{cases} .$$

On peut citer le chargement de conteneurs sur les navires comme application du monde réel du MKP. Le problème est de choisir certains conteneurs dans un ensemble de n conteneurs à charger dans m navires de différentes capacités de chargement.

Sac à dos multi-objectif

Le problème du sac à dos multi-objectif (MOKP) en varibale binaire (0-1), est une variante du probléme ou plusieurs objectifs sont à maximiser simultanément

$$\text{(MOKP)} \begin{cases} Max\ z_j(x) = \sum_{i=1}^{n} c_i^j x_i \quad j \in \{1,...,p\},\ c_i \in \mathbb{N}_{>}^{p} \\ s.c: \\ \sum_{i=1}^{n} w_i x_i \leq w \qquad\qquad w, w_i \in \mathbb{N}^* \\ x_i \in \{0,1\} \qquad\qquad \forall i \in \{1,...,n\} \end{cases}$$

Dans ce cas la notion d'optimalité laisse place à la notion d'efficacité, comme on la déja vu dans le chapitre 1, l'objectif est donc de trouver l'ensemble des solutions du problème non dominées.

Sac à dos multidimensionnel multi-objectif

Le problème de sac à dos multi-objectif multidimensionnel (MMOKP) consiste à sélectionner un sous-ensemble d'objets maximisant une fonction multi-objectifs exprimée en fonction des valeurs de profit, tout en respectant un ensemble de contraintes de type sac à dos. Le problème du (MMOKP) peut être défini plus formellement de la manière suivante :

$$\text{(MMOKP)} \begin{cases} Max\ z_j(x) = \sum_{i=1}^{n} c_i^j x_i \quad j \in \{1,...,p\},\ c_i \in \mathbb{N}_{>}^{p} \\ s.c: \\ \sum_{i=1}^{n} w_i^l x_i \leq w_l \qquad\qquad l \in \{1,...,q\}, w, w_i \in \mathbb{N}^* \\ x_i \in \{0,1\} \qquad\qquad \forall i \in \{1,...,n\} \end{cases}$$

2.3 Méthodes de résolution

2.3.1 Méthodes éxactes

Branch and Bound

La méthode de Branch and Bound est l'une des méthodes les plus connues pour la résolution de problèmes d'optimisation combinatoire NP-difficiles. Cette méthode est basée sur une recherche arborescente d'une solution optimale par séparation et évaluation. La méthode de Branch and Bound tente d'explorer intelligemment l'ensemble des solutions admissibles en éliminant de l'espace de recherche les sous-ensembles de solutions qui ne peuvent pas fournir une solution optimale.

Présentation de l'algorithme

La recherche par décomposition de l'ensemble des solutions peut être représentée graphiquement par un arbre (voir la Figure 2.2). C'est de cette représentation que vient le nom de "méthode de recherche arborescente".

— Chaque sous-problème créé au cours de l'exploration est symbolisé par un noeud de l'arbre (ou sommet), le noeud racine représentant le problème initial.
— Les branches de l'arbre symbolisent le processus de séparation. Elles représentent la relation entre les noeuds.
— Lors de la séparation, un noeud "père" crée un ensemble de noeuds "fils".

Le Branch and Bound est basé sur trois principes :

Principe d'évaluation

Le principe d'évaluation a pour objectif de connaître la qualité des noeuds à traiter. La méthode de Branch and Bound utilise deux types de bornes

— une borne inférieure de la fonction d'utilité du problème initial,
— une borne supérieure de la fonction d'utilité

La connaissance d'une borne inférieure du problème et d'une borne supérieure de la fonction d'utilité de chaque sous-problème permet de stop-

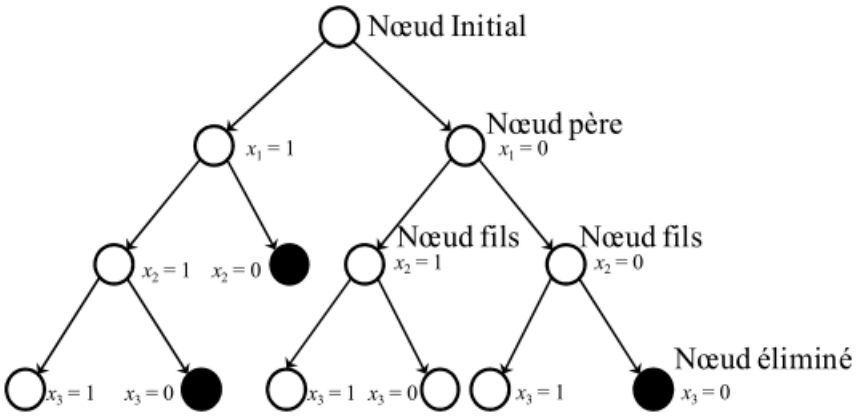


FIGURE 2.2 – Arbre engendré par décomposition d'un problème.

per l'exploration d'un sous-ensemble de solutions qui ne sont pas candidates à l'optimalité : si pour un sous-problème la borne supérieure est plus petite que la borne inférieure du problème, l'exploration du sous-ensemble correspondant est inutile. D'autre part, lorsque le sous-ensemble est suffisamment « petit », on procède à une énumération explicite : on résout alors le sous-problème correspondant.

Principe de séparation :

Le principe de séparation consiste à diviser le problème en un certain nombre de sous problèmes qui ont chacun leur ensemble de solutions réalisables. En résolvant tous les sous problèmes et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Ce principe de séparation est appliqué de manière récursive à chacun des sous ensembles tant que celui-ci contient plusieurs solutions.

Stratégie de parcours :

La stratégie de parcours est la règle qui permet de choisir le prochain sommet à séparer parmi l'ensemble des sommets de l'arborescence. Parmi les stratégies de parcours les plus connues, on peut citer :

— La profondeur d'abord :
L'exploration privilégie les sous-problèmes obtenus par le plus grand nombre de séparations appliquées au problème de départ, c'est-à-dire

aux sommets les plus éloignés de la racine (de profondeur la plus élevée).
L'obtention rapide d'une solution admissible (pour les problèmes où il est difficile d'obtenir une heuristique de bonne qualité) et le peu de place mémoire nécessaire en sont les avantages. L'inconvénient est l'exploration de sous-ensembles qui peuvent s'avérer peu prometteurs à l'obtention d'une solution optimale.

— La largeur d'abord :
Cette stratégie favorise les sous-problèmes obtenus par le moins de séparations du problème de départ, c'est-à-dire les sommets les plus proches de la racine (de profondeur la moins élevée).

— Le meilleur d'abord :
Cette stratégie favorise l'exploration des sous-problèmes possédant les plus petites bornes inferieures. Elle dirige la recherche là où la probabilité de trouver une meilleure solution est la plus grande.

Calcul des bornes :

Le calcule des bornes nous permet d'encadrer la valeur de la solution optimale. La plupart des bornes supérieures pour le problème (KP) se basent sur le tri des objets par ordre décroissant d'efficacité (voir définition 2.2.1).

La borne supérieure la plus simple peut être obtenue en prenant la solution optimale de la relaxation continue du problème du sac à dos (LKP) [23].
Ainsi on aboutit à la borne supérieure présentée par Dantzig [11] qui est exprimée comme suit :

$$U_1 = z^*(LKP)$$

Si nous considérons séparément le cas ou l'objet bloquant s est ou n'est pas ajouté à la solution, nous obtenons une deuxième borne supérieure définie par Martello et Toth.

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

Algorithme 1 Algorithme de branch and bound ;

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

Branch and Cut

La méthode du Branch and cut est aussi très intéressante pour la résolution de problèmes d'optimisation combinatoire d'une manière générale. La méthode branch and cut est le résultat de l'intégration entre méthode de coupe et la méthode de branch and bound (voir., Resende et al.[28]). La méthode de coupe est un outil efficace qui conduit à une forte réduction de la taille de l'arbre de recherche pour l'approche de branch and bound. Ainsi, la méthode de branch and bound peut être accélérée par l'emploi d'un système de plan de coupe, (pour plus de details, le lecteur peut se référer à : Crowder et al. [10] ; Balas et al.[3]).

La Programmation dynamique

La technique de programmation dynamique est une approche générale qui apparait comme un outil utile pour résoudre divers problèmes en optimisation combinatoire. L'idée de base qui se trouve derrière cette technique a été introduite par Bellman (voir., Bellman [5]). Cette approche consiste à : (i) décomposer un problème en sous-problèmes plus simples, (ii) résoudre ces sous-problèmes et (iii) combiner leurs solutions afin de trouver une solution globale. Pour plus de détails, nous recommandons le livre de Bellman [5] qui donne une introduction générale claire concernant la programmation dynamique.

2.3.2 Méthodes approchées

Le probléme KP est un probléme d'optimisation combinatoire NP-complet (voir Garey et Johnson [15]), il est donc intéressant d'avoir des algorithmes polynomiaux faciles à mettre en oeuvre, permettant de calculer une solution approchée. Parmi les méthodes heuristiques les plus utilisées pour la recherche d'une solution pour le KP, on trouve la méthode gloutonne.

Les méthodes gloutonnes sont une classe de méthodes heuristiques qui construisent une solution en se concentrant entièrement sur une amélioration immédiate sans tenir compte de l'avant. En d'autres termes, l'algorithme choisit toujours un optimum local dans l'espoir de parvenir à une solution optimum global. Les algorithmes gloutons ne donnent pas forcement des solutions optimales. Néanmoins, ils sont très puissants et fonctionnent correctement pour plusieurs problèmes d'optimisation combinatoire. Depuis des décennies, ils sont la façon la plus immédiate pour déterminer une solution réalisable [30]. Dans ce qui suit, nous présentons, une méthode gloutonne basée sur les principes proposés par Dantzig [11] pour résoudre le 0-1 KP. Premièrement, tous les éléments sont triés par ordre décroissant en fonction de leur rapport du profit sur le poids (c_i/w_i), tel que :

$$\frac{c_1}{w1} \leq \frac{c_2}{w2} \cdots \leq \frac{c_n}{w_n}$$

Ainsi, à chaque étape, sélectionner de façon gloutonne un article selon l'ordre défini précédemment. Si l'élément est recevable, cela veut dire si son poids ne dépasse pas la capacité du sac après fixation des autres éléments, alors, il est placé dans le sac à dos. Sinon, nous sélectionnons l'élément suivant qui peut être recevable, et ainsi de suite, jusqu'à épuisement des éléments qui pourraient être placés dans le sac à dos.
Il convient de noter que, cette méthode n'est pas la seule méthode gloutonne pour le problème de sac à dos. Ainsi, il existe plusieurs autres versions et améliorations par rapport à cette simple méthode .

Algorithme 2 Glouton

Entrée :une instance de KP ;
Sortie :une solution realisable $\overline{x}$;

```
1 :  Début
2 :    w̄ ← w
3 :     pour i = 1 jusqu'a n faire
4 :         Si w_i ≤ w̄ alors
5 :              x̄ ← 1 ;
6 :              w̄ ← w̄ − w̄_i ;
7 :           Sinon ;
8 :              x̄ ← 0 ;
9 :          Fin si
10 :     Fin pour
11 :  Fin
```

Exemple :

Soit un sac de poid maximal w =30 kg, on veut mettre 4 objets dans le sac, le poid et la valeur de chaque objet est donné dans le tableau suivant :

objet	1	2	3	4
c_i	7	4	3	3
w_i	13	12	8	10

TABLE 2.1 – Exemple d'application.

Déroulons l'algorithme glouton sur notre exemple :
Première étape :

— Calculer l'éfficacité (c_i/w_i) de chaque objet i ;

objet	1	2	3	4
c_i/w_i	0,54	0,33	0,37	0,30

TABLE 2.2 – Première étape.

objet	1	3	2	4
c_i	7	3	4	3
w_i	13	8	12	10
c_i/w_i	0,54	0,37	0,33	0,30

TABLE 2.3 – Deuxième étape.

Deuxième étape :

— Trier tous les objets par ordre décroissant de cette valeur ;

Troisième étape :

— Sélectionner les objets un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac, si le poids maximal reste respecté.

Objet 1 : on le met dans le sac vide, le poids du sac est alors de 13 et inférieur à 30 ;
Objet 3 : on le met dans le sac car 13 + 8 = 21 est inférieur à 30 ;
Objet 2 : on ne le met pas dans le sac car le poids total (33) dépasserait 30 ;
Objet 4 : on ne le met pas dans le sac (poids total de 31).

La solution trouvée est donc de mettre les objets 1 et 3 dans le sac, ce qui donne une valeur de 10.

Cet algorithme ne donne forcement pas de solution optimal, mais il reste assez rapide et efficace.

2.4 Conclusion

Dans ce chapitre nous avons évoqué le problème du sac à dos ainsi que ses variantes, nous avons aussi présenté les différentes méthodes de résolution de ce problème.

3

Résolution du problème de sac à dos bi-objectif unidimensionnel en variables binaires

3.1 Introduction

Ulungo et Teghem[28] ont proposé la méthode en deux phases pour la résolution exacte des problèmes d'optimisation combinatoire multi-objectif. Dans ce chapitre nous appliquons cette méthode au probléme du sac à dos bi-objectif unidimensionnel en variables binaires.

3.2 La méthode en deux phases

La méthode en deux phases a été utilisée pour résoudre de multiples problèmes bi-objectifs (sac à dos, affectation, ...etc)[7], Comme son nom l'indique cette méthode est composé de deux phases, la phase 1 varie très

peu en général et la phase 2 doit être spécifique au problème.
Soit le problème de sac à dos bi-objectif (bi-KP) :

$$\text{(01-biKP)} \begin{cases} Max\ z_j(x) = \sum_{i=1}^{n} c_i^j x_i\ \ j \in \{1,2\}, c_i \in \mathbb{N}^p \\ tq : \\ \sum_{i=1}^{n} w_i x_i \leqslant w \qquad\qquad w, w_i \in \mathbb{N}^* \\ x_i \in \{0,1\}, \qquad\qquad \forall i \in \{1,...,n\} \end{cases}$$

3.2.1 Phase1 : détermination des solutions supportées

La premiére phase est une variante de la méthode dichotomique de Aneja et Nair [2]. Elle consiste à calculer les solutions efficaces supportées X_{SE_M} [26].

On note S l'ensemble des solutions efficaces supportées initialisé par les deux solutions lexicographique optimales correspondantes à $(z_1(x), z_2(x))$ et $(z_2(x), z_1(x))$ respectivement.
Le calcule des ses solutions revient à l'optimisation à un seul objectif, nous résolvons donc les deux problème mono-objectif p_1 et p_2 pour $j = 1$ et $j = 2$:

$$\begin{cases} Max\ z_j(x) \\ tq : \\ \sum_{i=1}^{n} w_i x_i \leqslant w \quad w, w_i \in \mathbb{N}^* \\ x_i \in \{0,1\}, \forall i \in \{1,...,n\} \end{cases}$$

Les solutions trouvées x^1 et x^2 sont les solutions optimales des problèmes p_1 et p_2 respectivement .
Après la génération des deux solutions lexicographique optimales, nous appliquons la recherche dichotomique .

La recherche dichotomique est initialisée par les deux solutions lexicographique optimales x^1 et x^2 qu'on notera x^r et x^s respectivement. Par la suite on considère deux points consécutifs y^r et y^s tel que $y^r = (z_1(x^r), z_2(x^r))$ et $y^s = (z_1(x^s), z_2(x^s))$ et le segment reliant ces deux points est noté $\overline{y^r y^s}$.

On associe alors un problème (p_λ) dont la pondération :
$\lambda = (y_2^r - y_2^s, y_1^s - y_1^r)$, un point y^t est obtenu lors de la résolution de (p_λ) et deux cas sont alors à considérer :

— Si $y^t \cap \overline{y^r y^s} = \emptyset$ alors y^t n'est pas sur le segmant reliant les deux points y^r et y^s (voir figure 3.1.b), ce qui signifie que deux nouveaux intervalles qu'il faut explorer sont crée, nous avons donc deux problémes pondéré à résoudre.

— Sinon, y^t se trouve sur le segment, donc aucun nouveau problème pondéré n'est généré, par conséquent la recherche dichotomique est terminé (voir figure 3.1.a).

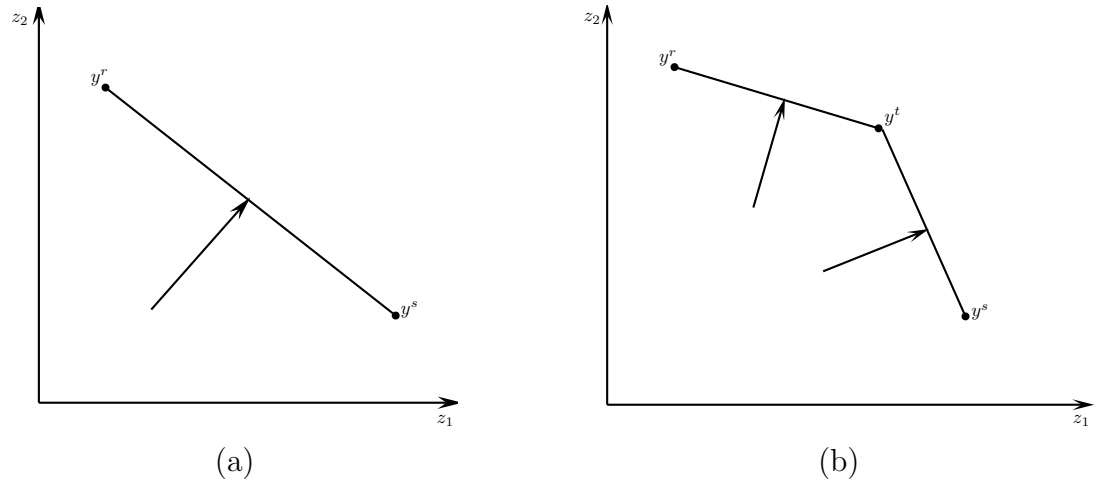


FIGURE 3.1 – Les deux cas de la recherche dichotomique.

Algorithme 3 Phase 1

Entrée : w, w_i, c_i

Sortie : Les solutions supportées y^r, y^s, y^t

1 : **Début**
2 : Résoudre p_1, p_2 , obtenir x^r, x^s ;
3 : $y^r = (z_1(x^r), z_2(x^r))$, $y^s = (z_1(x^s), z_2(x^s))$;
4 : $S \leftarrow \{y^r, y^s\}$;
5 : Choisir 2 point supportée adjacent y^r, y^s ;
6 : Résoudre p_λ avec $\lambda = (y_2^r - y_2^s, y_1^s - y_1^r)$, obtenir y^t ;
7 : Si $\lambda \cdot y^t \neq \lambda \cdot y^r$,$S \leftarrow S \cup \{y^t\}$;
8 : S'il reste des adjacences non traiter revenir à (5) ;
9 : Si $\lambda \cdot y^t = \lambda \cdot y^r$ et les adjacences sont toutes traitées aller à phase 2 ;
10 : **Fin**

3.2.2 Phase2 : détermination des solutions non supportées

Lors de la deuxième phase, on détermine les solutions efficaces non supportées. Les solutions obtenues dans la première phase sont utilisées pour diminuer l'espace de recherche où se trouvent potentiellement d'autres solutions efficaces. L'espace de recherche est ainsi découpé en plusieurs zones de recherches. Chaque zone de recherche est un triangle délimité par deux solutions efficaces adjacentes, qui contient tous les points qui ne sont dominés par aucunes des deux solutions [33].

Donc, chaque couple (y^r, y^s) de points consécutifs est utilisé pour définir un triangle rectangle dont ils forment l'hypoténuse qu'on note $\triangle(y^r, y^s)$. L'autre sommet de ce triangle est (y_1^r, y_2^s) et est appelé point nadir local de y^r et y^s . Ensuite, chaque triangle est visité un à un à la recherche de nouvelles solutions. L'exploration de chaque triangle se fait au moyen d'un algorithme d'énumération, nous utiliserons l'algorithme de séparation et evalution (Branch and Bound).

La partie **évaluation** consiste à définir les bornes qui décrivent chaque triangle.

1. **Les bornes $\underline{z}_1, \underline{z}_2$:**
   Comme le triangle $\triangle(y^r, y^s)$ est définit par les deux points y^r et y^s alors $\underline{z}_1 = y_1^r$ est une borne inférieur par rapport au premier objectif et $\underline{z}_2 = y_2^s$ pour le second.

2. **La borne inférieur pour (P_λ) :**
   La borne inférieure du problème (P_λ) est donnée par $\underline{z}^\lambda = \lambda y^N$, où y^N est le point nadir local défini par $y^N = (y_1^r, y_2^s)$.
   Cette borne est mise à jour au fur et à mesure que de nouvelles solutions potentiellement efficaces sont trouvées dans le triangle. Soit $\{y^1, ..., y^m\}$ les points réalisables potentiellement non dominés connus dans le triangle, tels que $y_1^i < y_1^{i+1}$ pour tout $i \in \{1, ..., m-1\}$. En renommant $y^0 = y^r, y^{m+1} = y^s$, la valeur de cette borne inférieure est $\underline{z}^\lambda = min_{i \in \{0,...,m-1\}}(\lambda_1 y_1^i + \lambda_2 y_2^{i+1})$.
   L'évaluation des noeuds 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 noeud est fermé [21].

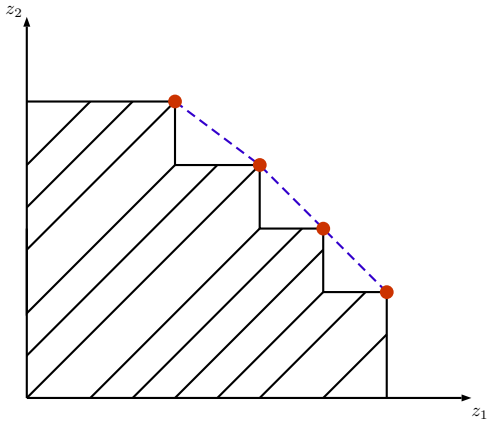


FIGURE 3.2 – Représentation des triangles.

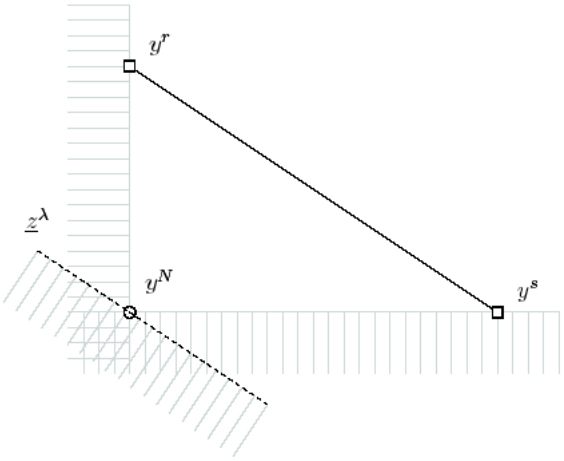


FIGURE 3.3 – Représentation des bornes du triangle.

Dans la partie **séparation** on suit les étapes suivantes :

1. Le noeud S_0 de l'arbre est initialisé par l'une des solutions x^r où x^s ,supposons x^s.
2. Soit $\{j/x_j^s = 1\ \}$ l'ensemble des indices des variables égale à 1 dans cette solution.
3. On crée alors q sous noeud S_1,...,S_q qui sont caractérisés par 1 jusqu'à q variables fixes.
4. Toute les solutions non supportée obtenue sont affectées a la liste S ,qui au début est vide.

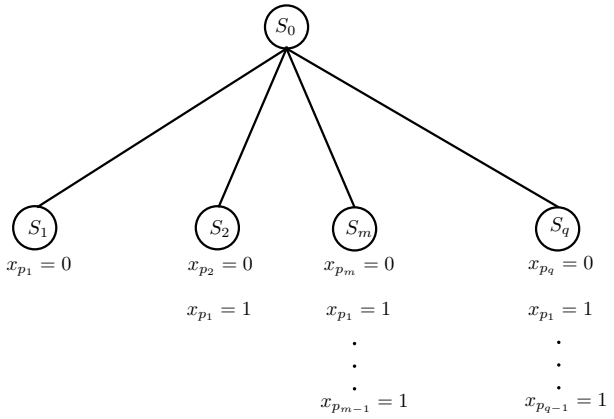


FIGURE 3.4 – Principe de séparation.

Algorithme 4 Phase 2

Entrée : x^r

Sortie : L'ensemble de solutions supportées et non supportées S

Début
Initialiser $x = x^r$;
Calculer $\underline{z_1} = y_1^r$, $\underline{z_2} = y_2^s$, $\underline{z}^\lambda = \lambda \cdot (y_1^r, y_2^s)$;
Tant que $z_1(x) \geq \underline{z}$ et $z_2(x) \geq \underline{z_2}$ et $\underline{z}^\lambda(x) \geq \underline{z}^\lambda$ **faire** ;
Déterminer l'ensemble $J^1 = \{j / x_j^r = 1\} = \{p_1, ..., p_q\}$;
Si $J_1 = \emptyset$ **terminer** ;
Sinon
Pour i allant de p_1 à p_q **tel que** $i \in J^1$ **faire**
$x_i = 0$;
$x_j = 1$, $j \in J^1$ et $j < i$ et $j > p_1$ résoudre (p_λ) ;
Si p_λ admet une solution x alors
Calculer z_1, z_2, z_λ ;
$S = S \cup \{(z_1(x), z_2(x))\} = \{y^1, ..., y^m\}$;
$y^0 = y^r, y^{m+1} = y^s$;
$\underline{z}^\lambda = min_{i \in \{0,...,m\}} \lambda_1 y_1^i + \lambda_2 y_2^{i+1}$, **aller en** 3
Fin si
Fin pour
Fin si
Fin tant que
Fin

3.3 Conclusion

Dans ce chapitre nous avons présenté la méthode en deux phases appliquée à un problème de sac à dos bi-objectif unidimensionel en variables binaires.

4

Implémentation logiciel

L'application à été développée en utilisant le logiciel MATLAB 2013, qui est un environnement puissant, complet et facile à utiliser, il est destinée au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant le calcul numérique.

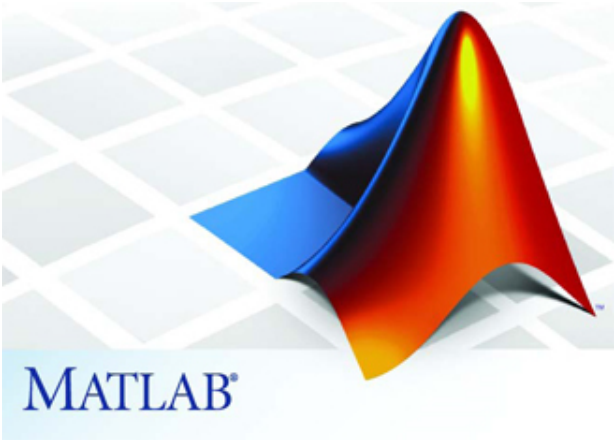


FIGURE 4.1 – Logiciel Matlab.

4.1 Le logiciel Matlab

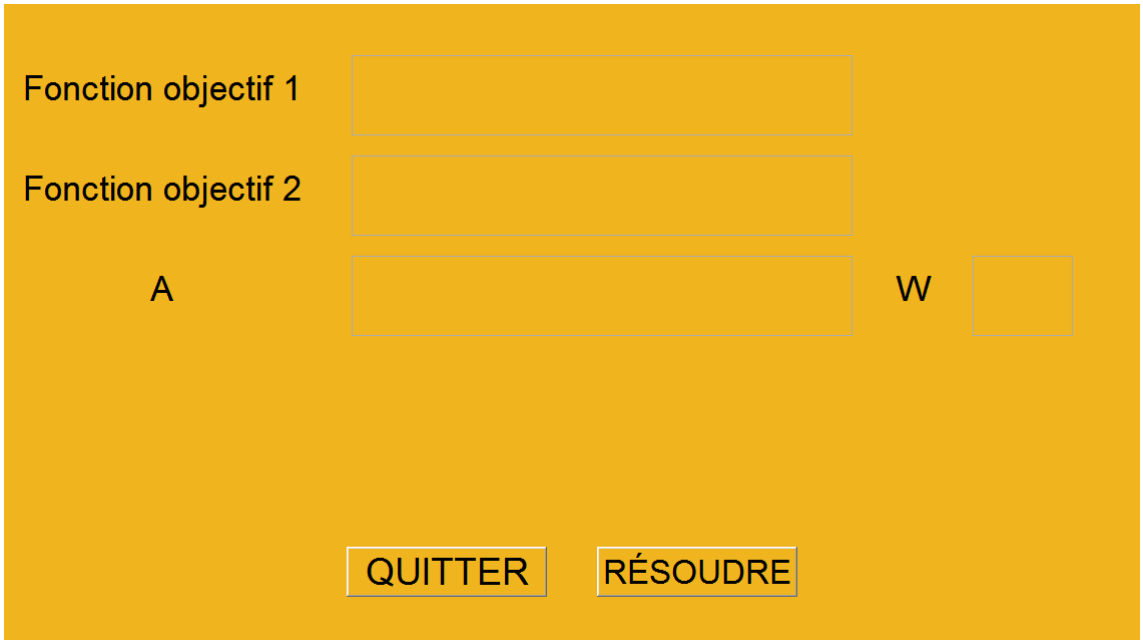
MATLAB (matrix laboratory) est un langage de programmation et un environnement de développement ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en oeuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++ et Java. Les utilisateurs de MATLAB sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche.

4.2 Description de l'application

L'application qu'on a développée est destinée à la résolution du problème de sac à dos bi-objectif en variables binaires en utilisant la méthode en deux phases.
Lorsque l'application est lancée, cette fenêtre s'affiche ;

L'utilisateur a le choix soit de commencer l'opération de l'optimisation en cliquant sur le bouton " DÉMMARER " ou bien de quitter définitivement l'application en cliquant sur le bouton " QUITTER ".

On cliquant sur " DÉMMARER " la fenêtre suivante s'affiche ;



Cette fenêtre est composée de trois champs :

1. Champ d'entré : ici l'utilisateur fait entrer les paramètres relatifs au problème :
   - Le vecteur associé à z_1 (la première fonction objectif).
   - Le vecteur associé à z_2 (la deuxième fonction objectif).
   - le vecteur A : représente la contrainte du problème.
   - W : représente le poid maximum.
2. Champ de contrôle : il est constitué de deux boutons :
   - RÉSOUDRE : lancer la résolution du problème.
   - QUITTER : pour quitter définitivement l'application.
3. Champ de sortie : dans cette partie les résultats de l'exécution de la méthode s'affichent directement :
   - Les solutions efficaces sont affichées ainsi que le temps d'éxécution.

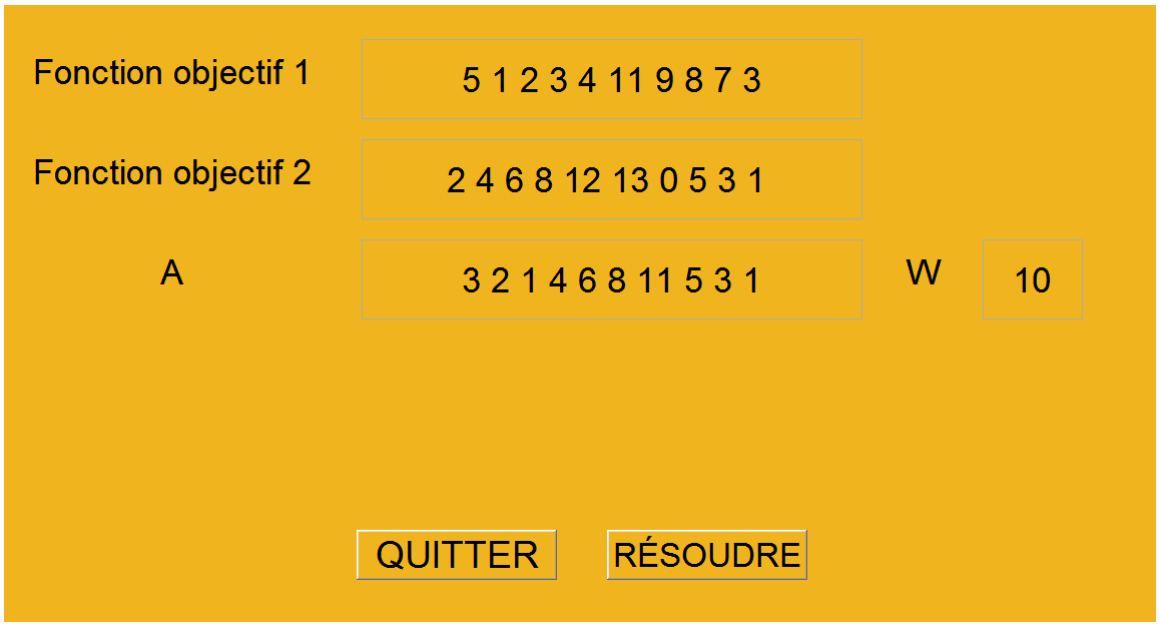
4.3 Résultat et discussion

Environnement du test

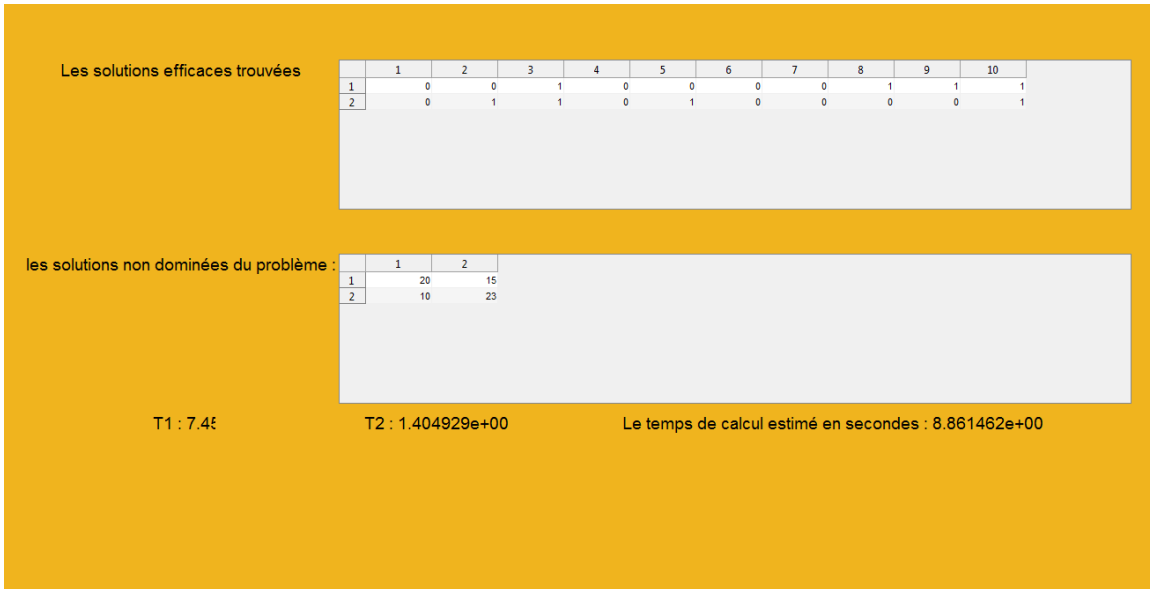
L'algorithme a été implémenté sur machine dont les caractéristiques (systéme d'exploitation windows 7, processeur i3, RAM 4GB).

Exemple d'application

Afin de tester notre application, on donne l'exemple suivant en initialisant le nombre de variables à 10 ;



On obtient le résultat suivant ;



Les solutions efficaces sont affichées, ainsi que la durée d'exécution, afin de mieux comprendre le fonctionnement de la méthode, on va lancer plusieurs tests tout en augmentant le nombre de variables à chaque exécutions, les résultats sont détaillés dans le tableau suivant :

Instance	T_1	T_2	T	nombre de solutions efficaces
50	0.9892	16.9133	17.9024	4
100	5.6443	73.0412	78.6855	5
200	5.9518	172.6235	178.5753	8
250	9.5281	797.7924	807.3206	6
300	11.1160	2.4609e+003	2.4726e+003	7
350	20.0354	2.6235e+003	2.643e+003	10
400	31.5476	2.8555e+003	2.8870e+003	8
450	45.3942	3.1542e+003	3.158e+003	6
500	51.3845	4.4573e+003	4.4624e+003	11
550	60.5606	1.0570e+004	10639.4175e+004	15

TABLE 4.1 – Résultats des tests.

T_1 : Temps d'exécution de la premiére phase ;
T_2 : Temps d'exécution de la seconde phase ;
T : Temps total d'exécution ;

Discussion

Aprés l'exécution de plusieurs instances, on observe que le temps d'exécution croit assez rapidement à chaque fois que le nombre de variables augmente, nous remarquons aussi que le temps consommé dans la seconde phase et plus grand que celui de la première, vu que la deuxième utilise la méthode de branch and bound qui est une méthode éxacte oú l'espace de recherche est totalement exploré.

4.4 Conclusion

Dans ce chapitre nous avons programmé et implémenté la méthode en deux phases appliquée à un problème de sac à dos bi-objectif, les éxpérimentations numérique nous ont montré l'efficacité de la méthode en terme de recherche de solutions efficaces, malgré le temps d'éxecution qui reste important.

Conclusion générale

Constatant la difficulté de résolution des problèmes combinatoire multi-objectif, Nous nous sommes intéressés dans ce mémoire à la résolution exacte de sac à dos bi-objectif 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. Notre travail porte sur l'application de la méthode en deux phases pour sa résolution.

Pour cela, nous avons donné les outils de bases de l'optimisation multi-objectif en premier lieu, puis nous avons présenté le problème de sac à dos et ses différentes variantes et enfin la méthode en deux phases a été appliquée pour la résolution du problème de sac à dos bi-objectif unidimensionnel en variables binaires.

Cette méthode à été programmée et implémentée sous le logiciel Matlab 2013, une étude comparative a été faite sur un ensemble de problème afin de montrer l'efficacité de cette approche.

Les problèmes d'optimisation combinatoire multi-objectif restent toujours d'actualité et les méthodes de résolution sont en développement progressive. En guise de perspectives dans ce domaine on peut citer :

— Améliorer les bornes pour restreindre encore plus l'espace de recherche.
— Améliorer les méthodes déjà existantes, et allez au-delà des deux objectifs.

Bibliographie

[1] Alaya, I. Optimisation multi-objectif par colonies de fourmis : cas des problèmes de sac à dos, 2009.

[2] Aneja, Yash P. et Nair, K. P. Bicriteria transportation problem. 73–78.

[3] Balas, E., and Xue, J. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. 397–412.

[4] Barichard, V. Approches hybrides pour les problémes multiobjectifs, 2003.

[5] Bellman, R. Dynamic programming princeton university press princeton.

[6] Benki, A. Méthodes efficaces de capture de front de pareto en conception mécanique multicritère : applications industrielles, 2014.

[7] Charles, D. Prétraitement dans les méthodes en deux phases pour la résolution de problèmes bi-objectifs.

[8] Clarisse, D. F. Optimisation combinatoire multi-objectif : Apport des méthodes coopératives et contributions à l'éxtraction de connaissances, 2005.

[9] Collette, Yann et Siarry, P. *Optimisation multiobjectif*. Eyrolles, 2002.

[10] Crowder, H., Johnson, E. L., and Padberg, M. Solving large-scale zero-one linear programming problems. 803–834.

[11] Dantzing, G. B. Discrete-variable extremum problems. 266277. Operations Research.

[12] Edgeworth, F. Y. *Mathematical psychics : An essay on the application of mathematics to the moral sciences*, vol. 10. Kegan Paul.

[13] Ehrgott, M. *Multicriteria optimization.* Springer, 2005. OCLC : 605688994.

[14] Ehrgott, Matthias et Getibleux, X. Approximative solution methods for multiobjective combinatorial optimization. 1–63.

[15] Gary, Michael R. et Johnson, D. S. *Computers et Intractability : A Guide to the Theory of NP-completeness.* WH Freeman et Company, New York, 1979.

[16] Getibleux, Xavier et Mezdaoui, N. e. F. A. A tabu search procedure to solve multiobjective combinatorial optimization problems. In *Advances in multiple objective et goal programming.* Springer, 1997, pp. 291–300.

[17] Hansen, M. P. Metaheuristics for multiple objective combinatorial optimization.

[18] Holland, J. H. Adaption in natural and artificial systems.

[19] Ishibuchi, H., and Murata, T. Multi-objective genetic local search algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* (1996), IEEE, pp. 119–124.

[20] J.K. Hao, P. Galinier, M. H. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes.

[21] Jorge, J. Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires, 2010.

[22] Knowles, J. D., and Corne, D. W. M-PAES : A memetic algorithm for multiobjective optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (2000), vol. 1, IEEE, pp. 325–332.

[23] Lalami, M. E. Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes heuristiques et parallèles.

[24] M. Geoffrion, A. Proper efficiency et the theory of vector maximization.

[25] Mahdi, S. Optimisation multiobjectif par un nouveau schéma de coopération méta/exacte.

[26] Olivier, S. Optimisation multi-objectifs : un tutoriel.

[27] Pareto, V. Cours d'économie politique.

[28] Resende, M. G., and Pardalos, P. M. *Handbook of applied optimization*. Oxford University Press, 2002.

[29] Saadi, L. Optimisation multiobjectifs par programmation génétique, 2007.

[30] Sagvan, A. S. Contribution à la résolution des problèmes combinatoires :optimisation séquentielle et parallèle, 2015.

[31] Schaffer, J. D. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st international Conference on Genetic Algorithms* (1985), L. Erlbaum Associates Inc., pp. 93–100.

[32] Srinivas, N., and Deb, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. 221–248.

[33] Thomas, V. Caractérisation des solutions efficaces et algorithmes d'énumération exacts pour l'optimisation multiobjectif en variables mixtes binaires, 2013.

[34] Ulungu, E. L. Optimisation combinatoire multicritére : détermination de l'ensemble des solutions efficaces et méthodes interactives.

[35] Zitzler, E., Thiele, L., and others. *An evolutionary algorithm for multiobjective optimization : The strength pareto approach*. Citeseer, 1998.

Résumé

Dans ce mémoire, on s'est intéressé à la résolution d'un problème d'optimisation combinatoire avec une méthode éxacte qui est la méthode en deux phase cas : sac à dos bi-objectif unidimensionnel en variables binaires. On a rappelé d'abord les notions essentièlles de l'optimisation multi-objectif (définitions et les notions de bases), de la même façon, on a donné les notions de bases du problème de sac à dos ainsi que ces différentes variantes.
Enfin, nous décrivons la méthode en deux phases appliquées à nôtre problème, tout en développant une application pour sa résolution.
Mots-clés : *optimisation multi-objectif, sac à dos, méthode en deux phases.*

Abstract

In this dissertation, we are aiming to solve combinatorial optimsation issue, the case of unidimensional bi-creteria knapsack problem with binary variable, using a methode called methode in two phases. We first mentioned the very essential terms and notions of multi-objective optimization by focusing on the definitions and basic notions. In the same context, we provided the basic terminology of Knapsack problem and it's distinctive variants.
Lastely, to archive our mentioned aim and in order to describe the methode in two phases, we have adopted a specific application.
Keywords : *Multi-objective optimization, Knapsack, method two phases.*