N° D'ORDRE : 11/2010/M/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene
Faculté des Mathématiques

MÉMOIRE
PRÉSENTÉ POUR L'OBTENTION DU DIPLÔME DE MAGISTER

EN RECHERCHE OPÉRATIONNELLE
OPTION : MATHÉMATIQUE DISCRETE ET OPTIMISATION

Par : Larbi ASLI

Approche hybride pour les problèmes d'optimisation combinatoire multiobjectif : cas des problèmes de type sac-à-dos

Soutenu publiquement à l'USTHB, le 28/06/2010, devant le jury composé de :

M^{r}M. MOULAÏ	Professeur	Président	à l'USTHB - Alger.
M^{r} M. AÏDER	Professeur	Directeur de thèse	à l'USTHB - Alger.
M^{r} D. CHAABANE	M.C. classe/ A	Examinateur	à l'USTHB - Alger.
M^{me}C. ADICHE	M.C. classe/ B	Invitée	à l' USTHB - Alger.

Année Universitaire 2009 – 2010

Louange A Dieu, le miséricordieux, sans Lui rien de tout cela n'aurait pu être.

***J**e tiens tout d'abord à remercier le Professeur M^r^ M. AIDER pour l'honneur qu'il ma fait en acceptant de m'encadrer. Ses conseils précieux ont permis une bonne orientation dans la réalisation de ce modeste travail.*

***J**e teins également à remercier M^me^ BOUCHEMAKH, pour tout son soutien et aide pour les étudiants de l'école doctorale.*

***J**e teins également à remercier M^r^ M. MOULAI d'avoir accepté de présider le jury de ce mémoire.*

***J**e remercie M^r^ D. CHAABANE et M^me^ C. ADICHE d'avoir accepter de faire partie du jury et consacrer leurs temps à la lecture et à la correction de ce mémoire.*

***M**es remerciements les plus vifs vont tout particulièrement à mes parents.*

***E**nfin, merci à tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.*

Je dédie ce modeste travail :

A *la mémoire de mon très chère père .*

A *ma très chère mère.*

A *mes frères et soeurs.*

A *ma promotion Magister $\mathcal{MDO}$.*

A*u groupe scout Assirem du bejaia .*

A *tous mes amis*

S*ans oublier la composante fortement connexe "****SCOR****" le Club Scientifique de la Recherche Opérationnelle.*

A. Larbi

Table des matières

Table des figures

Liste des Algorithmes

Liste des tableaux

Notations

Symbole	signification
$\mathcal{SE}$	Ensemble du solutions efficaces.
ND	Solutions non dominées, ou Front pareto.
χ	Espace réalisable.
$\mathcal{N}$	Voisinage.
$\mathcal{F}$	Espace réalisable dans l'espace des objectifs.
S	Espace de recherche.
$\overline{x}$	Solution réalisable.
Article, Items	Objets.
$v_j, \quad c_j$	Coût de l'objet j.
w_j	Poids de l'objet j .
C	Capacité du sac.

Introduction générale

"Une mesure exacte vaut l'avis d'un millier d'experts."
Grace Hopper

La vie est pleine de décisions qui impliquent habituellement plusieurs objectifs contradictoires. Le fait que les problèmes du monde réel doivent être résolus de façon optimale selon les critères qui interdisent une solution «idéale» "optimale pour chaque décideur pour chacun des critères considérés", a mené au développement de l'optimisation multiobjectif.

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines applicatifs, tels que le traitement d'images, la conception de systèmes, la conception d'emplois du temps, ... La plupart, sont qualifiés de difficiles, car leur résolution nécessite l'utilisation d'algorithmes évolués, et il n'est en général pas possible de fournir dans tous les cas une solution optimale dans un temps raisonnable.

Un problème d'optimisation combinatoire est défini par un ensemble fini de solutions discrètes S et une ou plusieurs fonction(s) objectif(s) f, associant à chaque solution une valeur. Ainsi, un problème d'optimisation combinatoire consiste en l'optimisation (minimisation ou maximisation) d'un certain critère sous différentes contraintes, permettant de délimiter l'ensemble des solutions réalisables (ou solutions admissibles).

Qu'ils comportent un seul ou plusieurs objectifs, les problèmes d'optimisation combinatoires 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 génériques, pouvant s'appliquer à un ensemble de problèmes. Parmi ces méthodes génériques, nous relevons deux grandes classes de méthodes : les méthodes exactes et les méthodes approchées.

Les méthodes exactes examinent, souvent de manière implicite, la totalité de l'espace de recherche. Ainsi, elles ont l'avantage de produire une solution optimale lorsqu'aucune

contrainte de temps n'est donnée. Néanmoins, le temps de calcul nécessaire pour atteindre une solution optimale peut devenir vite prohibitif, et ce, malgré les diverses techniques et heuristiques qui ont été développées pour accélérer l'énumération des solutions. Dans de telles situations (et dans beaucoup d'autres), les méthodes approchées constituent une alternative indispensable et complémentaire. Le but d'une telle méthode n'est plus de fournir une solution optimale au problème donné, mais elle cherche avant tout à produire une solution de meilleure qualité possible avec un temps de calcul raisonnable. En général, une méthode approchée examine seulement une partie de l'espace de recherche. Le choix des solutions examinées est souvent dicté par des heuristiques.

À partir de deux méthodes de résolution différentes, il est possible de concevoir des méthodes hybrides dans le but de fournir des résultats meilleures que ceux des méthodes qui les composent.

Les caractéristiques d'un algorithme d'optimisation voué à l'optimisation d'un problème multiobjectif (PMO), dépassent la simple recherche de la meilleure solution possible. Il s'agit ici de chercher un ensemble de points correspondant aux meilleurs compromis possibles pour résoudre notre problème.

Dans l'univers de l'optimisation multiobjectif, il existe un nombre important de méthodes, que certains auteurs classifient en cinq groupes :[16]

- Méthodes scalaires,
- Méthodes interactives,
- Méthodes floues,
- Méthodes exploitant une metaheuristique,
- Méthodes d'aide à la décision.

Les méthodes de ces cinq groupes, peuvent aussi être rangées en trois familles de méthodes d'optimisation multiobjectif [62] :

- **Méthodes à préférence a priori :**
  Dans ces méthodes, l'utilisateur définit le compromis qu'il désire réaliser (il fait part de ses préférences) avant de lancer la méthode d'optimisation. On retrouve dans cette famille la plupart des méthodes par agrégation (où les fonctions objectifs sont fusionnées en une seule).
- **Méthodes à préférence progressive :**
  Dans ces méthodes, l'utilisateur affine son choix de compromis au fur et à mesure du déroulement de l'optimisation. On retrouve dans cette famille les méthodes interactives.
- **Méthodes à préférence a posteriori :**
  Dans ces méthodes, l'utilisateur choisit une solution de compromis en examinant toutes les solutions extraites par la méthode d'optimisation.

Le problème du sac-à-dos (ou knapsack problem) est un problème classique d'optimisation appartenant à la classe des problèmes NP-difficiles. Il modélise de nombreux problèmes concrets tels que les problèmes des investissements,. . . .

En outre il intervient comme un sous problème dans plusieurs problèmes, par exemple : les problèmes de découpe (voir Glomre et Gomory [4]) lors de la génération d'un pivot du simplexe. Sous le terme de "sac-à-dos" sont regroupées différentes variantes qui, bien qu'elles semblent très proches, ne font pas du tout appel aux mêmes méthodes de résolution exactes ou approchées pour une variante donnée. Il existe diverses formes de ce problème, à savoir Sac-à-dos monobjectif uni-dimensionnel, borné et non borné multi-dimensionnel, sac-à-dos multiobjectif et sac-à-dos multiobjectif multi-dimensionnel.

Dans ce mémoire, nous étudions la combinaison de méthodes pour la résolution des problèmes multiobjectif. Nous nous penchons plus particulièrement sur des méthodes combinant des métaheuristiques et des méthodes exactes. En premier lieu, nous allons proposer une méthode hybride pour résoudre le problème de sac-à-dos multiobjectif unidimensionnel, qui sera une cooperation entre une méthode exacte et une métaheuristique. Cette méthode combine particulièrement la recherche tabou avec le branch and bound. Puis une discussion sera menée sur les résultats expérimentaux.

Organisation du document

Ce mémoire est structuré de la façon suivante :

• Le premier chapitre comprend les difinitions et les déférents concepts de base de l'optimisation combinatoire ainsi que celles de l'optimisation multiobjectif.

• Au deuxième chapitre, nous présentons un état de l'art sur les méthodes de résolution des problèmes multiobjectif. Ces méthodes sont classifiées en différentes catégories, nous présentons pour chacune de celles-ci les méthodes les plus courantes.

• Dans le troisième chapitre, nous abordons le problème de sac-à-dos ou knapsack problem qui sera le cadre de travail de ce mémoire. Nous présentons le problème basique, ainsi que différentes extensions.

• Le quatrième chapitre contient une présentation d'une méthode de résolution, qui compose deux algorithmes bien connus : le branch and bound et la recherche tabou pour obtenir le $BBTS^{MOKP}$, qui est un algorithme hybride pour le problème de Sac-à-dos multiobjectif.

Le travail s'achève par une conclusion mettant l'accent sur les perspectives de recherche induites par les résultats obtenus.

1

L'optimisation multiobjectif

Introduction

Résoudre un problème d'optimisation consiste à trouver la ou les meilleures solutions vérifiant un ensemble de contraintes et d'objectifs définis par l'utilisateur. Pour déterminer si une solution est meilleure qu'une autre, il est nécessaire que le problème introduise un critère de comparaison. Ainsi, la meilleure solution, appelée aussi solution optimale, est la solution ayant obtenu la meilleure valuation au regard du critère défini.

Contents

L'optimisation permet de modéliser de nombreux problèmes appliqués : le traitement d'images, la conception de systèmes, la conception d'emplois du temps, ... Ces problèmes sont pour la plupart qualifies de difficiles, car leur résolution nécessite l'utilisation d'algorithmes évolués, et il n'est en général pas possible de fournir dans tous les cas une solution optimale en un temps raisonnable. Lorsqu'un seul critère est donné, par exemple un critère de minimisation de coût, la solution optimale est clairement définie, c'est celle qui a le coût minimal. Mais dans de nombreuses situations, un seul critère peut être insuffisant. En effet, la plupart des applications traitées intègrent simultanément plusieurs critères, souvent contradictoires. Tenir compte de critères contradictoires pose un réel problème. Considérons les actions suivantes :

- Louer un appartement bien situé et à un prix raisonnable.
- Établir un planning pour les vacances satisfaisant toute la famille.
- Choisir entre plusieurs itinéraires.
- Acheter une voiture.

Plusieurs critères liés à ces tâches peuvent être contradictoires. La solution idéale n'existe pas, et il faut donc trouver un compromis. En effet, en considérant deux critères contradictoires x et y, améliorer x détériore forcément y et inversement. Le concept de solution optimale devient alors plus difficile à définir. Dans ce cas, la solution optimale cherchée n'est plus un point unique, mais un ensemble de compromis. Résoudre un problème comprenant plusieurs critères, appelé communément problème multiobjectif, consiste donc à calculer le meilleur ensemble de solutions de compromis : le front Pareto. Dans ce travail, nous traiterons principalement de la résolution de problèmes multiobjectif.

Dans ce chapitre, nous présentons l'optimisation multiobjectif. Nous introduisons les concepts fondamentaux tels que la dominance, la surface de compromis. Nous décrivons aussi les principales approches de résolution de ces problèmes.

1.1 Problèmes d'optimisation monobjectif

Résoudre un problème d'optimisation consiste à trouver une solution qui minimise ou maximise un critère particulier. Dans la plupart des cas, l'optimum découvert n'est pas unique. Ainsi il peut exister un ensemble de solutions minimisant ou maximisant le critère considéré. Nous pouvons décrire formellement un problème d'optimisation comme suit :
Considérons un problème de minimisation :

$$\left\| \begin{array}{l} \textit{Soit } f : \mathbb{R}^n \to \mathbb{R}; \\ \textit{et soit } \chi \textit{ un ensemble fermé de } \mathbb{R}^n; \\ \textit{alors l'ensemble des minimiseurs est } : \widehat{\chi} = \{x \in \chi | \forall x' \in \chi, f(x') \geq f(x)\}; \\ \textit{et le minimum du problème est } : f(\widehat{\chi}). \end{array} \right.$$

L'ensemble χ représente l'ensemble des solutions réalisables du problème. Il est déterminé à l'aide de contraintes (souvent analytiques) données dans l'énoncé du problème. Le formalisme précédent ne faisant pas explicitement intervenir les contraintes du problème, un problème d'optimisation monobjectif est plus souvent donné sous la forme suivante :

$$\left\| \begin{array}{ll} Minimiser \;\; f(x), & \text{(fonction à optimiser)}\,; \\ tel \;\; que \;\; g(x) \leq 0, & \text{(q contraintes à satisfaire)}\,; \\ avec \;\; x \in \mathbb{R}^n; g(x) \in \mathbb{R}^q. & \end{array} \right.$$

Ces deux écritures sont équivalentes, la deuxième étant le plus souvent rencontrée dans les domaines de l'Intelligence Artificielle et de l'optimisation. Il est clair que pour minimiser (resp. maximiser) la fonction objectif d'un problème, il faut déjà être capable de déterminer l'ensemble des solutions réalisables. On distingue ainsi deux notions : la notion d'espace de recherche représentant l'ensemble des valeurs pouvant être prises par les variables, et la notion d'espace réalisable représentant l'ensemble des valeurs des variables satisfaisant les contraintes. Dans le cadre de ce travail, nous traiterons des problèmes appartenant à la classe de problèmes NP-difficiles. Nous reviendrons sur cette notion plus tard.

Sur la figure 1.1, $\underline{x_i}$ (resp. $\overline{x}_i$) représente la borne inférieure (resp. supérieure) donnée pour x_i, $\mathcal{S}$ représente l'espace de recherche égal au produit cartésien des domaines des variables, et χ l'espace réalisable délimité par les contraintes. Cette figure, illustre en dimension 2 le fait que $\chi = \{x | g(x) \leq 0; x \in \mathcal{S}\}$ l'espace réalisable (délimité par des contraintes) est un sous-ensemble de $\mathcal{S}$ (ici $\mathcal{S} = \mathbb{R}^2$) l'espace de recherche.

Sauf mention contraire explicite, tous les énoncés et définitions seront donnés dans le cadre de problèmes de minimisation. En effet un problème de maximisation peut être aisément transformé en un problème de minimisation en considérant l'équivalence suivante :

$$\text{maximiser } f(x) \Leftrightarrow \text{-minimiser } f(x)\,;$$

1.1.1 Vocabulaire et définitions

Nous donnons ici les définitions des principaux termes fréquemment utilisés dans ce mémoire :

Définition 1.1 [4] *Un problème d'optimisation est défini par un espace d'état, une ou plusieurs fonction(s) objectif(s) et un ensemble de contraintes.*

Définition 1.2 [4] *L'espace d'état est défini par l'ensemble des domaines de définition des variables de problème.*

Remarque 1 Dans la plupart des problèmes, cet espace est fini car la méthode de résolution utilisée a besoin de travailler sur un espace restreint (Exemple : la méthode de Monté-Carlo, les algorithmes génétiques,...) cette limitation n'est pas problématique car lorsqu'un

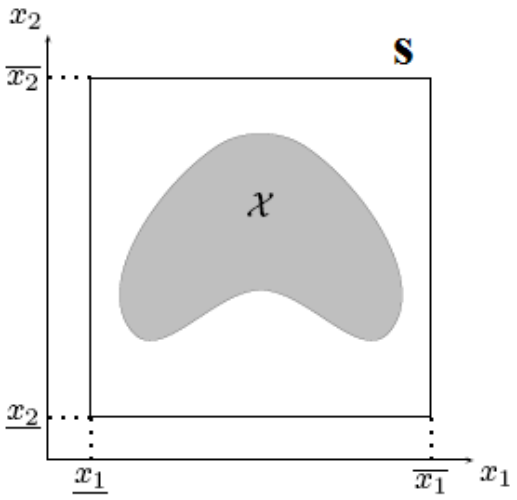


FIG. 1.1 – *Espace de recherche et espace réalisable[4]*

problème est posé, le décideur précise un domaine de valeurs envisageables à chac'une des variables. De plus, pour des raisons opératoires et de temps de calcul, il est préférable de travailler sur des domaines finis.

Définition 1.3 [4]*Les variables du problème peuvent être de natures diverses (réelle, entière, booléenne, etc.) et peuvent exprimer des données quantitatives ou qualitatives.*

Définition 1.4 *L'ensemble des contraintes définit des conditions sur l'espace d'etat que les variables doivent satisfaire. Ces contraintes sont souvent des contraintes d'égalité ou d'inégalité et permettent en général de limiter l'espace de recherche.*

Définition 1.5 [4]*Une méthode d'optimisation recherche le point ou un ensemble de points de l'espace des états qui satisfais au mieux un ou plusieurs critères. Le résultat est appelé* ***valeur efficace****.*

Définition 1.6 *Une boîte ou pavé est le nom donné au produit cartésien d'intervalles fermés. L'ensemble $\mathcal{S}$ du Figure 1.1 est le pavé résultant du produit cartésien des intervalles $[x_1]$ et $[x_2]$.*

Définition 1.7 [4]*Vecteur de décision est le nom donné au vecteur x. Il correspond à l'ensemble des variables du problème.*

Définition 1.8 [4]*La Fonction objectif représente la fonction f (appelé aussi fonction de coût ou critère d'optimisation).*

Définition 1.9 *Minimum global Un point $x \in \chi$ est un minimum global du problème si et seulement si : $\forall x' \in \chi; f(x) \leq f(x')$ (il appartient alors à l'ensemble des minimiseurs).*

Définition 1.10 [4]**Minimum local :** *Un point $x \in \chi$ est un minimum local du problème si et seulement si : $\forall x' \in \mathcal{V}(\chi); f(x) \leq f(x')$, où $\mathcal{V}(\chi)$ définit un voisinage de x.*

1.2 Classification des problèmes d'optimisation

On peut classer les différents problèmes d'optimisation que l'on rencontre dans la vie courante en fonction de leurs caractéristiques[18] :

1. Nombre de variables de décision :
   - une ⇒ mono variable.
   - Plusieurs ⇒ multi variable.
2. Type des variables de décision :
   - Nombres réels continu ⇒ continu.
   - Nombres entiers ⇒ entier ou discret.
   - Permutations sur un ensemble fini de nombres ⇒ combinatoire.
3. Type de la fonction objectif :
   - Fonction linéaire des variables de decision ⇒ linéaire.
   - Fonction quadratique des variables de decision ⇒ quadratique
   - Fonction non linéaire des variables de decision ⇒ non linéaire.
4. Formulation du problème :
   - Avec contraintes ⇒ contraint.
   - Sans contraintes ⇒ non contraint.

1.3 Choix d'une méthode d'optimisation

La nature des variables, des domaines de définition et des critères à optimiser va influencer le choix de la méthode d'optimisation à utiliser. Il y a deux grandes familles de méthodes d'optimisation : les méthodes déterministes et les méthodes stochastiques.

1.4 Les différentes méthodes

La manière la plus naturelle et la plus ancienne de résoudre un problème d'optimisation est la méthode par essai/erreur. Le décideur corrige ses résultats jusqu'à obtenir une solution satisfaisante. Cette méthode apparemment simpliste est à la base d'un très grand nombre de méthodes d'optimisation.

1.4.1 Méthodes déterministes

Définition 1.11 [18]*Les méthodes déterministes se caractérisent par l'exploration systématique (méthodique) de l'espace de recherche.*

Il existe de nombreuses méthodes d'optimisation déterministes, les méthodes locales qui assurent la convergence vers l'optimum local de la fonction en explorant le voisinage, et les méthodes globales qui s'attachent à faire converger la solution vers l'optimum global de la fonction.

Nous allons présenter les méthodes de gradients, la méthodes multistart et la méthode de simplexe qui sont des méthodes de recherche locale. D'une manière générale, ces méthodes obéissent à l'algorithme suivant :

a)- Choix d'une première solution courante i admissible.

b)- Génération d'une solution j dans le voisinage de i.

c)- Si $f(j)$ est meilleure que $f(i)$, alors j devient la solution courante puis retour en b.

d)- L'algorithme se termine lorsque il n'y a plus d'amélioration de la solution courante.

Ensuite, nous évoquerons des méthodes de recherche globales qui sont les méthodes par séparation et évaluation (branch and bound) et la méthode de tunneling.

Les méthodes de gradient

Historiquement, les méthodes de gradient sont les plus anciennes. Elles permettent de résoudre des problèmes non linéaires (Minoux 1983), et sont basées sur une hypothèse forte : la connaissance de la dérivée de la fonction objectif en chacun des points de l'espace. Cette famille de méthodes procède de la façon suivante :

On choisit un point du départ x_0, et on calcule le gradient $\nabla f(x_0)$ en x_0. Comme le gradient indique la direction de plus grande augmentation de f, on se déplace d'une quantité λ_0 dans le sens opposé au gradient, et on définit le point x_1 :

$$x_1 = x_0 - \lambda_0 \frac{\nabla f(x_0)}{\| \nabla f(x_0) \|};$$

Cette procedure est répétée et engendre les points $x_0, x_1, \ldots, x_n$. Ainsi, pas à pas, la distance entre le point d'indice k et l'optimum diminue.

$$x_{k+1} = x_k - \lambda_k \frac{\nabla f(x_k)}{\| \nabla f(x_k) \|}; \qquad \forall k, \lambda_k > 0;$$

λ_k est le pas de déplacement à chaque itération.
Si λ_k est fixé, on parle de la **méthode de gradient à pas prédéterminé**. L'inconvénient de cette procedure est que la convergence est très dépendante du choix du pas de déplacement. La convergence peut être très lente si le pas est mal choisi. L'intérêt principal de cette méthode est de pouvoir se généraliser au cas de fonctions non partout différentiables.

Actuellement la méthode la plus utilisée de cette famille est la **méthode de la plus grande pente**. Elle permet de se libérer du choix de λ_k mais elle introduit un critère d'arret. Le but de cette méthode est de minimiser la fonction de λ_k.

$$g(\lambda_k) = f(x_k - \lambda \nabla f(x_k));$$

L'algorithme de la plus forte pente est à la base de l'algorithme de **Hill-Climbing** appelé également algorithme de la descente de gradient.[9]

Algorithme 1 Algorithme de la plus grande pente

a)- Choisir un point x_0 et faire $k = 0$.
b)- A l'itération k, $d_k = -\nabla f(x_k)$.
Rechercher $\lambda_k \geq 0$ tel que $f(x_k + \lambda_k.d_k) = \min\{f(x_k + \lambda.d_k)\}$
Pour $x_{k+1} = x_k + \lambda_k.d_k$.
c)- Si le test d'arrêt est verifié **alors** fin
sinon $k = k + 1$ et aller en b).

Méthodes multistart

La méthode multistart effectue une recherche locale à partir de plusieurs points répartis dans l'espace de recherche. Avant de lancer le processus de recherche, il faut réaliser un maillage de l'espace de recherche. L'efficacité de cette méthode dépend de la bonne adéquation entre le maillage et la fonction objectif. Si le maillage est trop grand, alors la probabilité de trouver l'optimum global est faible car certaines vallées[1] ne seront pas traitées. Si le maillage est trop petit, la recherche globale sera inefficace car plusieurs points vont être présents dans la même vallée et convergeront vers le même optimum.

Pour éviter ce dernier inconvénient, la méthode a été améliorée en introduisant la notion de cluster[2]. Le regroupement des points voisins permet de diminuer le nombre de calculs en évitant les recherches locales redondantes. Les points à partir desquels la recherche locale est relancés étant choisis minutieusement dans chaque cluster, l'efficacité de l'algorithme est ainsi augmenté [9].

La méthode de Nelder Mead ou la méthode du simplexe

L'intérêt principal de la méthode du simplexe par rapport aux deux autres méthodes précédentes est qu'elle ne nécessite pas le calcul de gradient. Elle est uniquement basée sur l'évaluation de la fonction objectif, celà qui la rend utilisable pour des fonctions bruitées. Cette méthode de recherche locale effectue une recherche multidimensionelle dans l'espace d'état [Nelder and Mead 1965]. Soit une fonction f à minimiser. On appelle simplexe de $\mathbb{R}^n$ tout ensemble $x_0, x_1, \ldots, x_n$ de points de $\mathbb{R}$ tels que $f(x_0) \geq f(x_i), \forall i \in [0.n]$. x_0 est donc le meilleur élément.

Après la construction d'un simplexe initial, l'algorithme va modifier celui-ci en appliquant des calculs de réflexion, expansion et contraction jusqu'à la validation d'un critère d'arrêt.[9]

[1] Le terme vallée est utilisé car nous considérons que nous sommes dans un cas du minimisation si non, nous utilisons le terme pic pour le cas de maximisation

[2] Dans certaines problèmes multiobjectif l'ensemble pareto-optimal est trop grand, on utilise alors une technique de clustering pour supprimer un certain nombre de solutions

Algorithme par séparation et évaluation Branch and Bound

Le principe de cette méthode est de découper (branch) l'espace initial de recherche en domaines de plus en plus restreints afin d'isoler l'optimum global (principe de séparation). L'algorithme forme ainsi un arbre dont chaque nœud représente une partie de l'espace. Ensuite, chaque nœud est évalué pour déterminer sa borne (bound) inférieure [3] en fonction d'un critère d'évaluation. Si cette borne n'est pas meilleure que la solution courante, alors la recherche sur ce nœud est stoppée, sinon la séparation continue.

L'efficacité de cette méthode dépend essentiellement du choix des critères de séparation. Un bon choix permettra à l'algorithme de réduire l'arbre de recherche en évitant la construction de certaines branches. Dans le pire des cas, un mauvais choix peut mener l'algorithme à explorer tout l'espace d'état.[9]

La méthode du tunneling

Cette méthode recherche l'optimum global d'une fonction en utilisant des recherches successives d'optima locaux, de sorte que la valeur de la fonction s'améliore. L'algorithme se décompose en deux phases : une phase de recherche d'un optimum local et une phase dite de "tunneling".
Si on se place dans le cas d'une fonction f à minimiser, la première phase peut utiliser une méthode de gradient pour trouver l'optimum local $f^* = f(x^*)$.

La deuxième phase consiste à trouver un point x dans une autre vallée à l'aide d'une fonction T de "tunneling" de type $T(x) = f(x) - f^* \leq 0$.
Le principal inconvénient de cette méthode est l'utilisation de la fonction "tunneling".

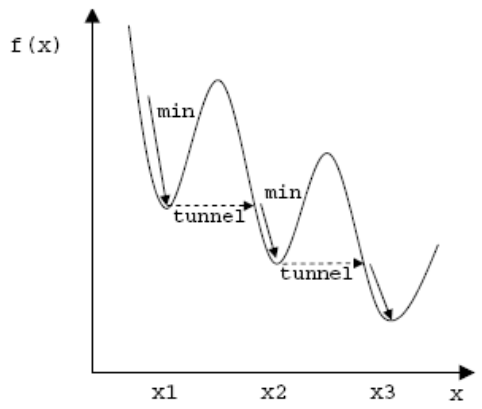


FIG. 1.2 – *Méthodes du tunneling[9]*

Remarque 2 : les méthodes vues ici sont très efficaces sur des problèmes particuliers de petite taille. Mais sur des problèmes de grande taille, la probabilité de trouver un optimum global dépend essentiellement de la bonne connaissance du problème de la part du décideur.

[3] Inférieure dans le cas de minimisation, supérieur dans le cas de maximisation

Ainsi, ce dernier pourra choisir avec précision la bonne méthode et les bons paramètres.

Si les conditions exposées ci-dessus ne sont pas réunies, le décideur devra plutôt s'orienter vers des méthodes stochastiques.

1.4.2 Méthodes stochastiques

Les méthodes stochastiques sont caractérisées par[9] :

- un processus de création des points aléatoires ou pseudo-aléatoires dans l'espace d'état,
- une heuristique qui permet de guider le processus de convergence de l'algorithme.

Ces méthodes sont utilisées dans les problèmes où on ne connaît pas d'algorithme de résolution en temps polynomial, et pour lesquels on espère trouver une solution approchée. Dans la pratique, les méthodes les plus populaires sont le recuit simulé, la recherche tabou, les algorithmes évolutionnaires, la méthode de Monté-Carlo, les colonies de formis, l'optimisation par particules d'essaims, les réseaux neurologiques et le branch and bound stochastique. Elles présentent les avantages suivants :

- facilité d'implementation,
- flexibilité par rapport aux contraintes du problème,
- qualité élevée des solutions.

D'un point de vue théorique, il existe des théorèmes de convergence pour les algorithmes génétiques [Goldberg 1989, Raphaë Cerf 1994], et pour le recuit simulé [Hajek 1988] justifient l'usage de ces méthodes. En général, il est établi qu'on a une probabilité très élevée de trouver une solution optimale si un temps de calcul très important est alloué.[9]

La plupart de ces méthodes sont généralisée aux cas des problèmes multiobjectif.

Branch and bound stochastique

Cette méthode utilise le même principe que le branch and bound déterministe, mais la borne inférieure (ou supérieure) est stochastique. Elle apparaît plus simple à paramètrer, et elle peut être utilisée dans des espaces de recherche plus grands[9].

1.5 Optimisation Multiobjectif

Problèmes d'optimisation multiobjectif

La plupart des problèmes d'optimisation réels sont décrits à l'aide de plusieurs objectifs ou critères souvent contradictoires, devant être optimisés simultanément. Alors que, pour les problèmes n'incluant qu'un seul objectif, l'optimum cherché est clairement défini, celui ci reste à formaliser pour les problèmes d'optimisation multiobjectif. En effet, pour un

problème à deux objectifs contradictoires, la solution optimale cherchée est un ensemble de points correspondant aux meilleurs compromis possibles pour résoudre notre problème.

Prenons le cas d'une personne souhaitant acheter une voiture d'occasion. La voiture idéale est celle qui est peu chère avec peu de kilomètres, mais cette voiture idyllique n'existe pas. Notre acheteur va donc devoir identifier les meilleurs compromis possibles correspondant à son budget (voir figure 1.3).

Les problèmes d'optimisation multiobjectif sont une généralisation à n fonctions ob-

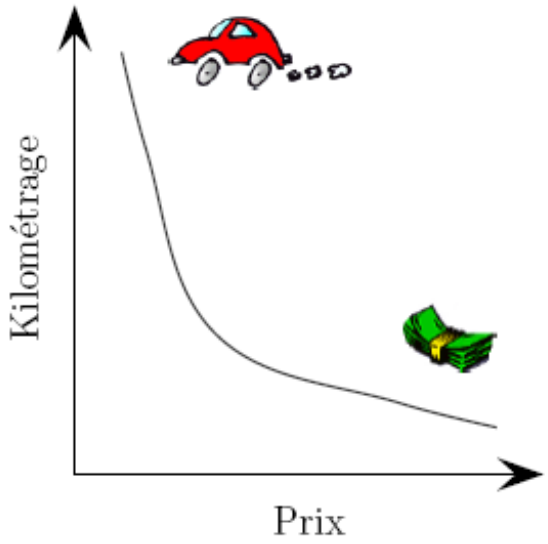


FIG. 1.3 – *Relation entre kilométrage et prix[4].*

jectif des problèmes d'optimisation classique (cf. Section 1.1). Ils sont définis formellement comme suit :
Considérons un problème de minimisation :

Soit $f : \mathbb{R}^n \to \mathbb{R}$ et soit χ un ensemble fermé de $\mathbb{R}^n$;
alors l'ensemble des minimiseurs est :
$\widehat{\chi} = \{x \in \chi | \forall x' \in \chi, (\exists i \in [1, \ldots, m], f_i(x) < f_i(x'))\} \bigvee (\forall i \in [1, \ldots, m], f_i(x) = f(x'))$;
et le minimum du problème est : $f(\widehat{\chi})$.

D'après cette définition, il est clair que l'optimum n'est plus une simple valeur comme pour les problèmes à un seul objectif, mais un ensemble de points, appelé ensemble des meilleurs compromis ou front Pareto. Dans la suite de ce travail, nous adopterons la formulation suivante, équivalente à la précédente, mais plus souvent utilisée dans les travaux actuels :

$$\begin{array}{ll} \textit{Minimiser} \quad f(x), & (m \text{ fonctions à optimiser}) ; \\ \textit{sachant que} \quad g(x) \leq 0, & (q \text{ contraintes à satisfaire}) ; \\ \textit{avec} \quad x \in \mathbb{R}^n, f(x) \in \mathbb{R}^m, g(x) \in \mathbb{R}^q & . \end{array}$$

1.6 Relations d'ordre et de Dominance

Comme la solution rechercher est une multitude de points de $\mathbb{R}^m$, il est vital pour identifier ces meilleurs compromis de définir une relation d'ordre entre ces éléments. Dans le cas des problèmes d'optimisation multiobjectif, ces relations d'ordre sont appelées relations de dominance. Plusieurs relations de dominance ont déjà été présentées : la *a-dominance* [Othmani, 1998], la dominance au sens de *Geoffrion* [Ehrgott, 2000], la *cône-dominance* [Collette and Siarry, 2002], ...

Mais la plus célèbre et la plus utilisée, c' est la dominance au sens de Pareto.≪ Au XIXème siècle, Vilfredo Pareto, un mathématicien italien, formule le concept : «*dans un problème multiobjectif, il existe un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères*»≫. C'est cette relation de dominance que nous allons définir et utiliser dans ce travail. De manière à définir clairement et formellement cette notion, les relations $=, \leq$ et $<$ usuelles sont étendues aux vecteurs.

1.6.1 Vocabulaire et définitions

Définition 1.12 [4]*Soient u et v, deux vecteurs de même dimension :*

$$\left\| \begin{array}{lll} u = v, & ssi & \forall i \in \{1, 2, \ldots, m\}, u_i = v_i \, ; \\ u \leq v, & ssi & \forall i \in \{1, 2, \ldots, m\}, u_i \leq v_i \, ; \\ u < v, & ssi & u \leq v \bigwedge u \neq v. \end{array} \right.$$

Les relations $\geq$ et $>$, sont définies de manière analogue.

Les relations définies précédemment ne couvrent pas tous les cas possibles. En effet, il est impossible de classer les points $a = (1; 2)$ et $b = (2; 1)$ à l'aide d'une de ces relations. Contrairement aux problèmes à un seul objectif, où les relations usuelles $<, \leq, \ldots$ suffisent pour comparer les points, elles sont insuffisantes pour comparer des points issus des problèmes multiobjectif. Nous définissons donc maintenant la relation de dominance au sens de Pareto, permettant de prendre en compte tous les cas de figures rencontrés lors de la comparaison de deux points (ici des vecteurs).

Définition 1.13 [4]**La dominance au sens de Pareto :***Considérons un problème de minimisation. Soient u et v deux vecteurs de décision :*

$$\left\| \begin{array}{lll} u \prec v (u \textit{ domine } v), & ssi & f(u) < f(v) \, ; \\ u \preceq v (u \textit{ domine } faiblement \; v), & ssi & f(u) \leq f(v)), \; f(u_i) < f(v_i) \, ; \\ u \sim v (u \textit{ est incomparable (ou non-dominé) avec } v), & ssi & f(u) \nleq f(v) \textit{ et } f(v) \nleq f(u). \end{array} \right.$$

Pour un problème de maximisation, ces relations sont définies de manière symétrique.

Définition 1.14 [4]*Une solution $x \in \chi_f$ est dite non dominée par rapport à un ensemble $\chi_a \subseteq \chi_f$ si et seulement si :*

$$\nexists X_a \subseteq \chi_a, \quad X_a \prec X;$$

Illustrons maintenant cette relation par un exemple en dimension 2 (cf. figure 1.4). Sur cette figure, $\mathcal{F}$ (l'espace réalisable dans l'espace des objectifs) est l'image de χ. Ainsi, chaque point y^i est l'image de x^i par $f : y^i = f(x^i)$. Prenons le point y^1 comme point de référence. Nous pouvons distinguer trois zones :

- La zone de préférence : est la zone contenant les points dominés par y^1,
- La zone de dominance : est la zone contenant les points dominant y^1,
- La zone d'incompatibilité : contient les points incomparables avec y^1.

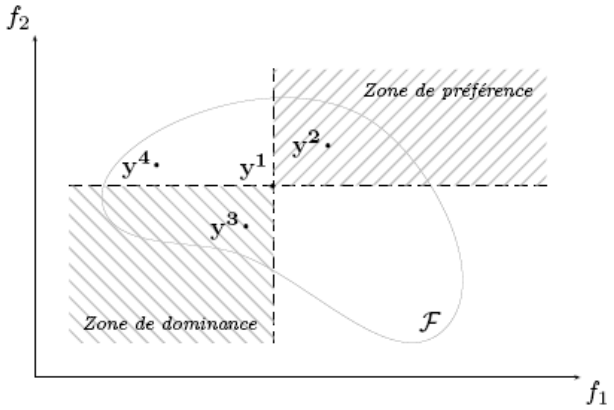


FIG. 1.4 – *Exemple de dominance [16].*

Ainsi, il est clair que y^2 est dominé par y^1 (y^1 est préféré à y^2), que y^3 domine y^1 (y^3 est préféré à y^1), et que y^4 est non dominé (incomparable) avec y^1.

Forts de ce nouvel outil, nous pouvons maintenant définir l'optimalité dans le cas de problèmes multiobjectif. Nous pouvons définir les concepts d'optimalité globale et locale au sens de Pareto (c.à.d utilisant la dominance au sens de Pareto) :

Définition 1.15 [4] *Un vecteur de décision $x \in \chi$ est dit Pareto globalement optimal si et seulement si : $\nexists y \in \chi, y \prec x$.*

Définition 1.16 [4] *Un vecteur de décision $x \in \chi$ est dit Pareto optimal localement si et seulement si, pour un $\delta > 0$ fixé : $\nexists y \in \chi; f(y) \in B(f(x); \delta)$ et $y \prec x$, où $B(f(x); \delta)$ représente une boule de centre $f(x)$ et de rayon δ.*

La figure 1.5 donne un exemple en dimension 2 d'optimalité locale. Le point $f(x)$ est localement optimal, car il n'y a pas de point compris dans la boule B le dominant.

1.6.2 Front Pareto et surface de compromis

Nous rappelons que la solution que nous cherchons pour un problème d'optimisation multiobjectif n'est pas un point unique, mais un ensemble de points, que nous avons appelé à la Section 1.5 ensemble des meilleurs compromis.

Nous avons défini jusqu'ici les notions de dominance (au sens de Pareto). Il nous reste

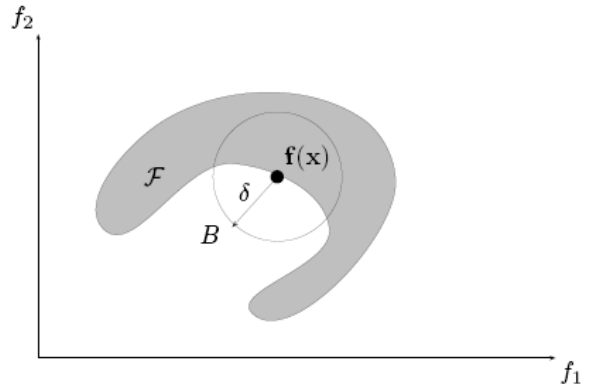


FIG. 1.5 – *Exemple d'optimalité locale[16].*

maintenant à définir la solution d'un problème multiobjectif utilisant ces concepts. Cet ensemble des meilleurs compromis, appelé aussi surface de compromis ou front Pareto, est composé des points qui ne sont dominés par aucun autre. Nous définissons d'abord formellement *l'ensemble des solutions non dominées* :

Définition 1.17 [4][4]*Soit $\mathcal{F}$ l'image dans l'espace des objectifs de l'ensemble réalisable χ. L'ensemble des solutions non dominées de χ, est défini par l'ensemble $ND(\chi)$:*

$$ND(\chi)) = \{x \in \chi | x \ \ est \ \ non \ \ dominé \ \ par \ \ rapport \ \ à \ \ \chi\};$$

Nous pouvons définir le front Pareto de $\mathcal{F}$ de manière analogue :

Définition 1.18 [4]*Soit $\mathcal{F}$ l'image dans l'espace des objectifs de l'ensemble réalisable χ. Le front Pareto $ND(\mathcal{F})$ de $\mathcal{F}$ est défini comme suit :*

$$ND(\mathcal{F})) = \{y \in \mathcal{F} | \ \nexists z \in \mathcal{F}, z < y\}$$

Le front Pareto est aussi appelé ***l'ensemble des solutions efficaces*** *ou* ***la surface de compromis***

1.6.3 Caractérisation du front Pareto

Solutions supportées / non supportées

Le front Pareto comporte plusieurs solutions de meilleur compromis appelées solutions pareto optimales. Nous allons caractériser les différents types de solutions composant ce front. Deux types de solutions le composent.

a- Les solutions supportées : leurs images dans Y se trouvent sur l'enveloppe convexe de Y. Chacune de ces solutions peut être trouvée en optimisant une agrégation linéaire des objectifs (théorème de Geoffrion).

b- Les solutions non-supportées : elles sont pareto optimales mais leurs images dans Y ne sont pas situées sur l'enveloppe convexe de Y. Aucune de ces solutions ne peut être trouvée en optimisant une agrégation linéaire des objectifs (corollaire du théorème de Geoffrion).

Ensemble Pareto minimal complet / Ensemble Pareto maximal complet

Un ensemble Pareto est dit complet, si toutes les valeurs Pareto qui peuvent être obtenues sont représentées parmi les solutions de l'ensemble. Certaines méthodes ne permettent pas de trouver l'ensemble Pareto complet (ex : la méthode par agrégation).

Il est important de remarquer que le nombre de solutions Pareto dépend de l'espace considéré (espace décisionnel/ espace objectif). En effet, deux solutions différentes de l'espace décisionnel peuvent avoir le même vecteur coût et donc représentent le même point de l'espace des objectifs. L'ensemble Pareto de l'espace décisionnel est appelé *ensemble Pareto maximal* et celui des objectifs est appelé *ensemble Pareto minimal*.

Il est donc nécessaire de définir l'ensemble recherché car cela fera varier le nombre de solutions Pareto [40].

Ces notions sont illustrées dans la figure 1.6.

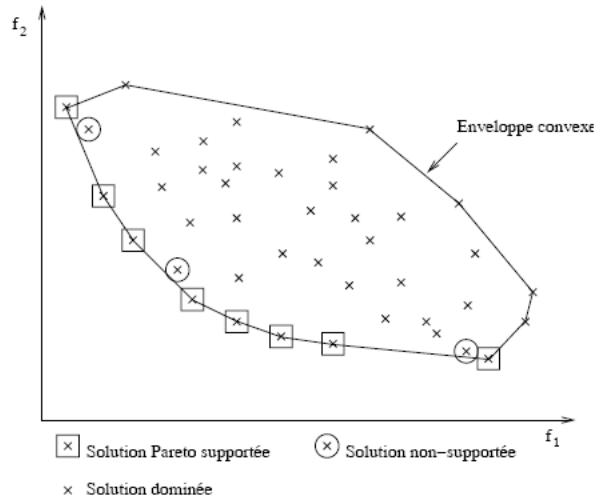


FIG. 1.6 – *Solutions supportés et non supportés[18]*

l'ensemble complet minimal ne contient qu'une seule solution pour chaque point non dominé dans l'espace des objectifs ;
l'ensemble complet maximal contient toutes les solutions équivalentes pour chaque point non dominé dans l'espace des objectifs.

Un exemple de surface de compromis (front Pareto) en dimension 2 est montré à la figure 1.7. Dans cet exemple, le problème considéré est un problème de minimisation avec deux critères. Deux points particuliers apparaissent clairement : le point idéal et le point Nadir. Ces deux points sont calculés à partir du front Pareto. Le point idéal (resp. le point

Nadir) domine (resp. est dominé par) tous les autres points de la surface de compromis. Bien que ces points ne soient pas forcément compris dans la zone réalisable, ils servent souvent de pôle d'attraction (resp. de répulsion) lors de la résolution du problème. Les coordonnées de ces deux points sont maintenant définies formellement.

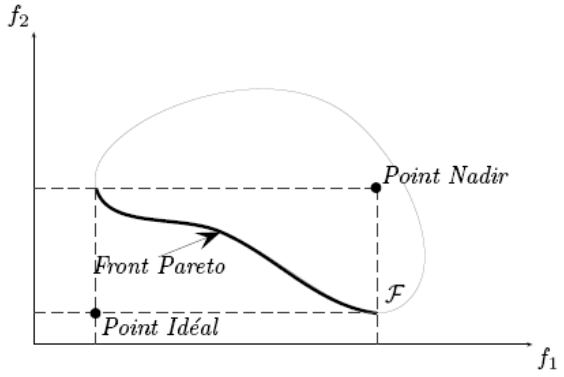


FIG. 1.7 – *Le front Pareto[16].*

Définition 1.19 *Les coordonnées du point idéal* (pi) *correspondent aux meilleures valeurs de chaque objectif des points du front Pareto. Les coordonnées de ce point correspondent aussi aux valeurs obtenues en optimisant chaque fonction objectif séparément.*

$$pi_i = \min\{y_i | y \in ND(\mathcal{F})\} \textit{avec } y \textit{ est l'éspace des objectif;}$$

Définition 1.20 *Les coordonnées du point Nadir* (pn) *correspondent aux pires valeurs de chaque objectif des points du front Pareto.*

$$pn_i = \max\{y_i | y \in ND(\mathcal{F})\};$$

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 [16].

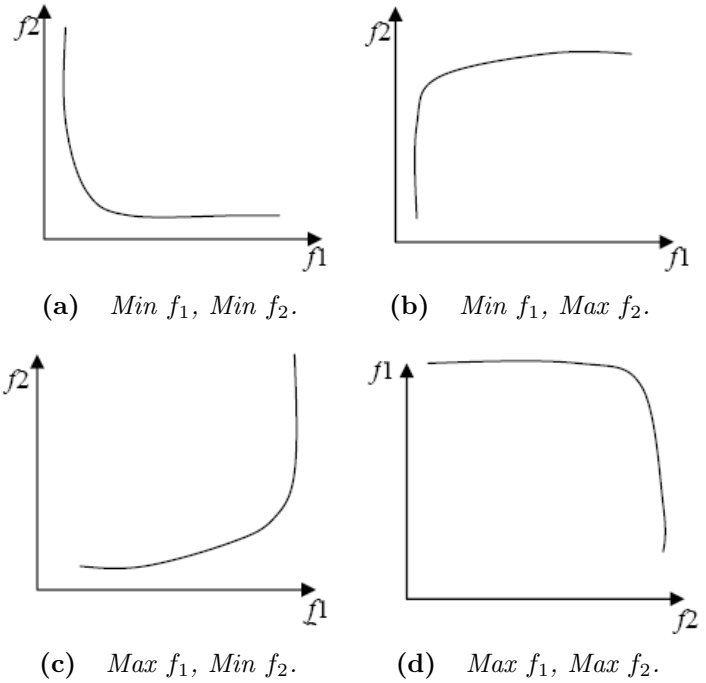
La figure 1.7 nous indique aussi que le front Pareto peut avoir des propriétés particulières quant à sa forme. La principale caractéristique utilisée pour comparer les formes de ces courbes est la convexité. Nous en rappelons donc la définition.

Définition 1.21 *Un ensemble A est convexe, si et seulement si l'équivalence suivante est vérifiée :*

$$\forall x \in A \quad et \quad y \in A \Leftrightarrow Segment(x; y) \subset A;$$

La convexité est le premier indicateur de la difficulté du problème. En effet, certaines méthodes sont dans l'incapacité de résoudre des problèmes non convexes de manière optimale. Mais il existe d'autres indicateurs tout aussi importants, notamment la continuité, la multimodalité, la nature des variables de décision (entières ou réelles),...

Une propriété est remarquable : en fonction du type du problème que l'on cherche à résoudre, on obtient une forme de surface de compromis. Les formes les plus courantes de surfaces de compromis sont réunies à la figure 1.8. Ces formes de surface de compromis sont typiques pour un problème d'optimisation multiobjectif sur un ensemble de solutions convexes. C'est ce type d'ensemble que l'on rencontre la plupart de temps.



(a) *Min* f_1, *Min* f_2. **(b)** *Min* f_1, *Max* f_2.

(c) *Max* f_1, *Min* f_2. **(d)** *Max* f_1, *Max* f_2.

FIG. 1.8 – *Formes les plus courantes de la surface de compromis dans le cas à deux objectifs[4].*

1.6.4 Complexité des algorithmes

La complexité (temporelle) d'un algorithme est le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuées par cet algorithme. Ce nombre s'exprime en fonction de la taille n des données. On s'intéresse au coût exact quand c'est possible, mais également au coût moyen (que se passe-t-il si en moyenne sur toutes les exécutions du programme sur des données de taille n ?), au cas le plus favorable, ou bien au pire cas. On dit que la complexité de l'algorithme est $O(f(n))$ où f est d'habitude une combinaison de polynômes, logarithmes ou exponentielles. Ceci reprend la notation mathématique classique, et signifie que le nombre d'opérations effectuées est borné par $cf(n)$, où c est une constante, lorsque n tend vers l'infini.

Les algorithmes usuels peuvent être classés en un certain nombre de grandes classes de complexité.

✓ Les algorithmes sous-linéaires, dont la complexité est en général en $o(\log n)$. C'est le cas de la recherche d'un élément dans un ensemble ordonné fini de cardinal n ;

✓ Les algorithmes linéaires en complexité $o(n)$ ou en $o(n \log n)$ sont considérés comme rapides, comme l'évaluation de la valeur d'une expression composée de n symboles ou les algorithmes optimaux de tri ;

✓ Plus lents sont les algorithmes de complexité située entre $o(n^2)$ et $o(n^3)$, c'est le cas de la multiplication des matrices et du parcours dans les graphes ;

✓ Au delà, les algorithmes polynômiaux en $o(n^k)$ pour $k > 3$ sont considérés comme lents, sans parler des algorithmes exponentiels (dont la complexité, est supérieure à tout polynôme en n), que l'on s'accorde à dire inutilisables dès que la taille des données est supérieure à quelques dizaines d'unités.

1.7 Relation de forme dérivée par la dominance

La relation de dominance n'offre pas trop de degrès de liberté dans la définition. Par exemple, il n'est pas possible d'inclure dans la définition de la relation de dominance une préférence pour une fonction objectif relativement à une autre. Pour surmonter ce manque de flexibilité, des relations dérivées de la relation de dominance ont été développées. Toutes les solutions que nous pouvons trouver avec ces relations dérivées sont toujours optimales dans le sens de Pareto [16].

1.7.1 Optimalité lexicographique

Cette définition de l'optimalité permet d'inclure une préférence entre objectifs

Définition 1.22 *la solution $x^* \in \chi$ est optimale au sens lexicographique si $x^* \leq_{lex} x. \forall x \in \chi - \{x^*\}$*

Exemple 1 Considerons les deux points A et B de $\mathbb{R}^6$

$$A = (1, 2, 3, 4, 5, 6)$$

$$B = (1, 2, 3, 9, 4, 9)$$

Pour ces deux points, on a $A \leq_{lex} B$ car, jusqu'à la troisième position, on a $A_i = B_i, i = 1, 2, 3$ et, pour la quatrième position, on a $4 < 9$. On conclut donc que la solution A domine lexicographiquement la solution B.

1.7.2 Optimalité extrême

Définition 1.23 *Une solution $x^* \in \chi$ est extrême-optimale si, étant donné un vecteur de poids $\lambda \in \mathbb{R}^m$ tel que $\sum_i \lambda_i = 1$, x^* est une solution optimale du problème de minimisation monocritère ayant pour fonction objectif*

$$\left\| \begin{array}{l} \sum_{i=1}^{k} \lambda_i.x_i; \\ \sum_{i=1}^{k} \lambda_i.x_i^* \leq \sum_{i=1}^{k} \lambda_i.x_i, \forall x \in \chi - \{x^*\}. \end{array} \right. \tag{1.1}$$

1.7.3 Optimalité maximale

Définition 1.24 *Une solution $x^* \in \chi$ est max-optimale si la valeur du pire objectif est aussi petite que possible :*

$$\max_{k\in\{1,\dots,m\}} x_k^* \leq \left| \begin{array}{ll} \max & x_k \\ k \in \{1,\dots,m\}; & \\ x \in \chi - \{x^*\}. & \end{array} \right. \tag{1.2}$$

Exemple 2 :Illustrons cette relation en prenant l'exemple précédent. On peut dire que la solution A max-domine la solution B car :

$$\max A = 6 < \max B = 9.$$

1.7.4 Cône optimalité

Une autre relation de classement existe. Il s'agit de la relation de cône-dominance. Cette relation possède l'avantage, par rapport à la relation de Pareto, d'être "réglable".

Définition 1.25 [16]*Un cône de pente λ est défini de la manière suivante*
Si $0 < \lambda < 1$

$$C_\lambda(x_1, x_2) = \{x \in \mathbb{R}^2 | x_1 \geq 0, x_2 \geq 0, \lambda.x_1 \leq x_2 \leq \frac{1}{\lambda}.x_1\}.$$

si $\lambda < 0$:

$$C_\lambda(x_1, x_2) = \{x \in \mathbb{R}^2 | \lambda.x_1 \leq x_2 \quad et \quad \lambda.x_2 \leq x_1\};$$

si $\lambda = 0$:

$$C_\lambda(x_1, x_2)\{x \in \mathbb{R}^2 | x_1 \geq x_0 \quad et \quad x_2 \geq 0\}.$$

Cette relation peut être étendue au cas multidimensionnel. On trouvera dans [16] une relation différente permettant aussi de régler la forme du cône de domination.

Définition 1.26 *Soit un cône C_λ. Un vecteur $x \in \mathbb{R}^n$ cône-domine un vecteur $y \in \mathbb{R}^n$ (cette relation est notée $x \leq_{C_\lambda} y$) si $Y - x \in C_\lambda$ et $x \neq y$ ((ou encore $Y - x \in C_\lambda - \{0\}$).*

1.7.5 La a-dominance

Cette définition de dominance a été introduite dans [16].

Définition 1.27 *Une solution $x^* \in \chi$ a-domine une solution $x \in \chi$ s'il existe un ensemble de combinaisons de $k+1-a$ critères (on note $I_{(k+1-a)}$ l'ensemble des indice correspondant à l'ensemble des combinaisons de ces critères) tel que :*

1. $x_j^* \leq x_j$ *pour tout* $j \in I_{k+1-a}$, *et*
2. $x_j^* < x_j$ *pour au moins un* $j \in I_{k+1-a}$.

1.7.6 La dominance au sens de Geoffrion

Une dernière forme de dominance importante dans le monde de l'optimisation multiobjectif est la dominance au sens de Geoffrion. Les solutions optimales obtenues par ce type de dominance sont appelées les solutions Pareto optimales propres.

Définition 1.28 *Une solution $x^* \in \chi$ est appelée solution Pareto optimale propre si :*

- *elle est Pareto optimale,*
- *il existe un nombre $M > 0$ tel que $\forall i = 1, \dots m$ et $\forall x \in \chi$ vérifiant $f_i(x) < f_i(x^*)$, il existe un indice j tel que $f_j(x^*) < f_j(x)$ et :*

$$\frac{f_i(x^*) - f_i(x)}{f_j(x) < f_j(x^*)} \leq M;$$

avec m est la dimension de x

Un théorème relatif à la méthode de pondération des fonctions objectif utilisant ce résultat est le suivant :

Théorème 1.1 Soit la méthode d'agrégation des fonctions objectif suivantes :

$$f_{eq}(x) = \sum_{i=1}^{m} w_i.f_i(x);$$

Supposons que $\forall i = 1, \dots, N, w_i > 0 \;\; avec \;\; \sum_{i=1}^{N} w_i = 1$
Si x est une solution optimale obtenue en utilisant la méthode d'agrégation ci-dessus, alors cette solution est aussi Pareto optimale propre.

1.8 Compromis entre exploration et exploitation

La capacité d'exploitation est l'aptitude d'une méthode à utiliser des résultats obtenus pour faire converger l'algorithme.

La capacité d'exploration est l'aptitude d'une méthode à explorer l'espace d'état. Cette aptitude a tendance à ralentir la convergence. Lors de la conception d'une méthode d'optimisation, le chercheur doit choisir entre maintien de la diversité et convergence alors que le décideur doit choisir entre *efficacité et efficience* d'une méthode.

Une méthode efficace fournira un résultat optimal ou proche de l'optimum. Dans ce cas, le temps de calcul n'a aucune importance. Pour assurer un résultat optimale, ces méthodes doivent d'effectuer une recherche exploratoire très importante de façon à ne laisser aucune partie de l'espace des états non visitée.

Une méthode efficiente est une méthode capable de donner un résultat satisfaisant

en un temps de calcul relativement court. Dans ce genre de méthode, le décideur privilégie le temps de calcul à la précision du résultat. La conception de ces méthodes est basée sur une capacité d'exploration importante des résultats précédents. Ainsi, le processus de convergence est accéléré au risque de voir la méthode trompée, et dirigée vers une zone de l'espace non optimale. Cette réutilisation systématique des caractères des génération passées entraîne une perte rapide de la diversité. L'adaptation des individus n'a pas le temps de devenir optimale.

Conclusion

La dominance est donc un outil, permettant de reproduire une démarche de recherche d'optimum. Elle n'est pas la seule, bien entendu, mais elle constitue un élément important pour la résolution d'un problème.

Dans ce chapitre nous avons donné un court aperçu sur les problèmes d'optimisation à savoir monobjectif et multiobjectif et les méthodes de résolution pour les cas monobjectif ainsi que quelques définitions mathématiques concernant le domain. Dans le chapitre suivant nous allons focaliser notre étude sur les méthodes de résolutions des problèmes multiobjectif.

2

Méthodes de résolution

Introduction

Qu'ils comportent un ou plusieurs objectifs, les problèmes d'optimisation sont en général difficiles à résoudre. De plus, le temps de calcul nécessaire à leurs résolutions peut devenir si important que l'algorithme développé devient inutilisable en pratique. Il n'existe pas d'algorithme générique capable de résoudre efficacement toutes les instances de tous les problèmes. De nombreuses 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 précis et les méthodes plus génériques pouvant s'appliquer à une catégorie de problèmes.

Contents

Dans ce chapitre, nous essayons de faire un état d'art sur les différentes méthodes d'optimisation multiobjectif, on distingue deux grandes classes à savoir les méthodes exactes et les méthodes approchées.

Les méthodes exactes examinent, souvent de manière implicite, la totalité de l'espace de recherche. Ainsi, elles ont l'avantage de produire une solution optimale lorsqu'aucune contrainte de temps n'est donnée. Néanmoins, le temps de calcul nécessaire pour atteindre une solution optimale peut devenir vite prohibitif, et ce malgré les diverses techniques et heuristiques qui ont été développées pour accélérer l'énumération des solutions.

Dans de telles situations (et dans beaucoup d'autres), les méthodes approchées constituent une alternative indispensable et complémentaire. Le but d'une telle méthode n'est plus de fournir une solution optimale au problème donné. Elle cherche avant tout à produire une solution sous-optimale de meilleure qualité possible avec un temps de calcul raisonnable. En général, une méthode approchée examine seulement une partie de l'espace de recherche.

A partir de deux méthodes de résolution différentes, il est possible de concevoir des méthodes hybrides dans le but de fournir des résultats supérieurs à ceux des deux méthodes qui les composent.

2.1 Schéma de méthodes

Il existe un nombre important de méthodes, que certains auteurs les classifient en cinq groupes :[16]

- Méthodes scalaires ;
- Méthodes interactives ;
- Méthodes floues ;
- Méthodes exploitant une métaheuristique ;
- Méthodes d'aide à la décision.

Les méthodes de ces cinq groupes peuvent aussi être rangées en trois familles de méthodes d'optimisation multiobjectif [62] :

- **Méthodes à préférence a priori**
  Dans ces méthodes, l'utilisateur définit le compromis qu'il désire réaliser (il fait part de ses préférences) avant de lancer la méthode d'optimisation. On retrouve dans cette famille la plupart des méthodes par agrégation (où les fonctions objectif sont fusionnées en une seule).
- **Méthodes à préférence progressive**
  Dans ces méthodes, l'utilisateur affine son choix de compromis au fur et à mesure du

déroulement de l'optimisation. On retrouve dans cette famille les méthodes interactives.

- **Méthodes à préférence a posteriori**
  Dans ces méthodes, l'utilisateur choisit une solution de compromis en examinant toutes les solutions extraites par la méthode d'optimisation. Les méthodes de cette famille fournissent, à la fin de l'optimisation, une surface de compromis.

Il existe des méthodes d'optimisation multiobjectif qui n'entrent pas exclusivement dans une de ces familles. Par exemple, on peut utiliser une méthode à préférence a priori en lui fournissant des préférences choisies au hasard. Le résultat sera alors un grand nombre de solutions qui seront présentées à l'utilisateur pour qu'il décide de la solution de compromis. Cette combinaison forme alors une méthode à préférence a posteriori. [16] D'autres types de classement classifient les méthodes d'optimisation multiobjectif selon les principes mathématiques, (utilisation les dérivées des fonctions objectif ou non).

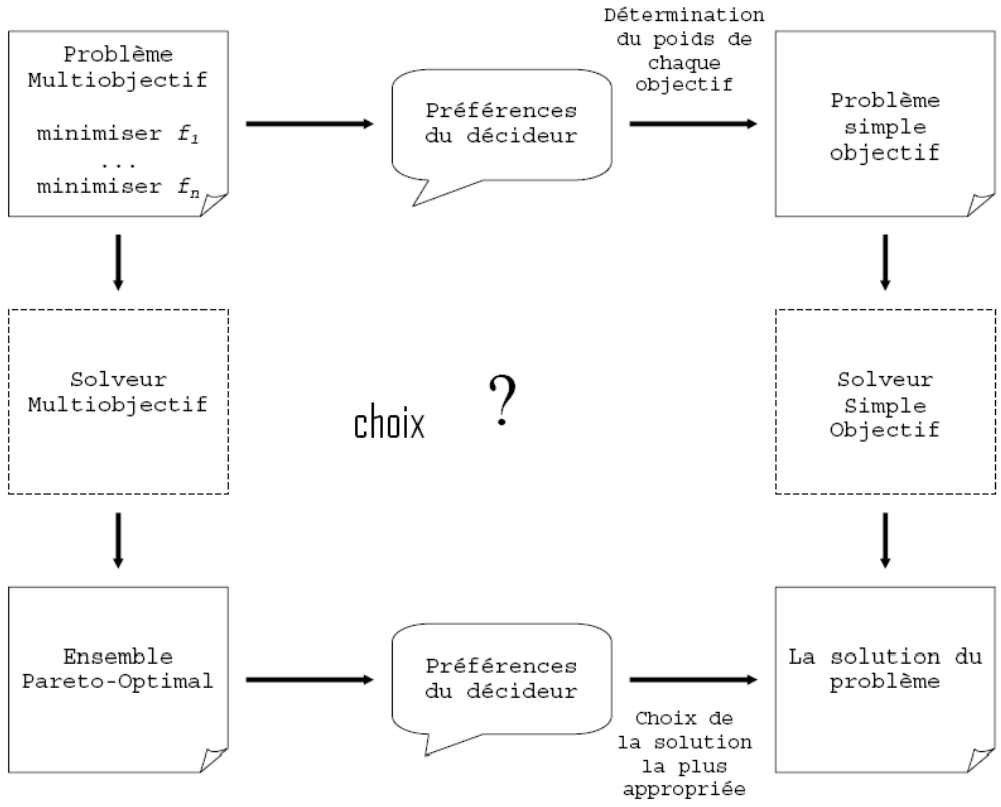


FIG. 2.1 – *Schéma de résolution*

2.2 Méthodes scalaires

2.2.1 La méthode de pondération de fonctions objectif

Cette approche de la résolution d'un problème d'optimisation multiobjectif est la plus évidente. D'ailleurs, on appelle aussi cette méthode "approche naïve" de l'optimisation multiobjectif [Coello [34]]. Le but, ici, est de revenir à un problème d'optimisation monobjectif, dont il existe de nombreuses méthodes de résolution. La manière la plus simple de procéder consiste à prendre chacune des fonctions objectif, à leur appliquer un coefficient de pondération et à faire la somme des fonctions objectif pondérées. On obtient alors une nouvelle fonction objectif [16].

$$(p) \left\| \begin{array}{ll} \text{Minimiser} & f_{eq}(x) = \sum_{i=1}^{m} w_i.f_i(x); \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & h(x) = 0. \end{array} \right.$$

Souvent, les coefficients de pondération respectent la relation suivante : $w_i \geq 0$ pour tous $i \in \{1, \ldots, k\}$ et $\sum_{i=1}^{k} w_i = 1$.

2.2.2 La méthode de Keeney-Raiffa

Cette méthode utilise le produit des fonctions objectif pour se ramener à un problème d'optimisation monobjectif. L'approche utilisée est semblable à celle utilisée dans la méthode de pondération des fonctions objectif. La fonction objectif ainsi obtenue s'appelle la fonction d'utilité de Keeney-Raiffa[16].

$$\left\| \begin{array}{ll} \text{Minimiser} & f_{eq}(x) = \prod_{i=1}^{k} w_i.f_i(x); \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & h(x) = 0. \end{array} \right.$$

On retrouve cette fonction dans la théorie de la fonction d'utilité multi-attribut exposée dans [Keeney et al. [37]] (M.A.U.T, Multi Attribute Utility Theory). Cette théorie, issue des sciences d'aide à la décision, traite des propriétés et de la manière de créer une "fonction d'utilité".[16]

2.2.3 La méthode de la distance à un objectif de référence

Cette méthode permet de transformer un problème d'optimisation multiobjectif en un problème d'optimisation monobjectif.

La somme que l'on va utiliser ici prendra la forme d'une distance ($\sqrt[k]{()^k}$) où la racine $k^{\text{ème}}$ de la somme d'éléments élevée à une certaine puissance k. De plus, dans certains cas on normalisera les éléments par rapport à une valeur avant d'en faire la somme [Azarm [3], Miettinen [47]].

On part encore du problème P. Dans les definitions suivantes, le vecteur F (de coordonnées $F_i, i \in \{1, \ldots, k\}$) est un vecteur dont les coordonnées correspondent à celles d'un objectif idéal (ou de reference) au sens défini par l'utilisateur (cela peut être le point ideal, ou un autre point qui représente mieux les préférences de l'utilisateur). Le vecteur F est choisi par l'utilisateur. On peut, par exemple, utiliser la distance absolue. Voici la definition de cette distance :

$$L_r\big(f(x)\big) = \big[\sum_{i=1}^{k} |F_i - f_i(x)|^r\big]^{\frac{1}{r}};$$

avec $0 \leq r \leq \infty$.

2.2.4 Méthode de compromis (approche par ϵ-contrainte)

Une autre façon de transformer un problème d'optimisation multiobjectif en un problème monobjectif est de convertir $m-1$ des m objectifs du problème en contraintes et d'optimiser séparément l'objectif restant [16].

La démarche est la suivante :

- On choisit un objectif à optimiser prioritairement ;
- On choisit un vecteur de contraintes initial ;
- On transforme le problème conservant l'objectif prioritaire et on transforme les autres objectifs en contraintes d'inégalité.

On appelle aussi cette méthode la méthode de la ϵ-contrainte [Miettinen [47]]. Le problème peut être reformulé de la manière suivante [4] :

$$\left\|\begin{array}{ll} \text{Minimiser} & f_i(x); \\ \text{tel que,} & f_1(x) \leq \epsilon_1; \\ & \vdots; \\ & f_{i-1}(x) \leq \epsilon_{i-1}; \\ & f_{i+1}(x) \leq \epsilon_{i+1}; \\ & \vdots \\ & f_m(x) \leq \epsilon_m; \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & x \in \mathbb{R}^n, f(x) \in \mathbb{R}^m, g(x) \in \mathbb{R}^q. \end{array}\right.$$

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 par rapport aux autres de ne pas être trompée par les problèmes non convexes. Ainsi la figure 2.2 illustre, en dimension 2, le cas où un point $(\epsilon; f_{1_{min}})$, de la partie non convexe, est trouvé. La figure 2.2 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.

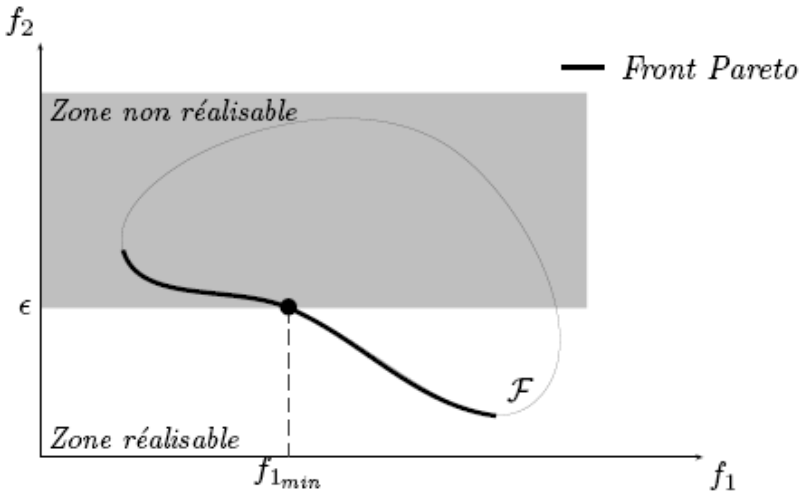


FIG. 2.2 – *Interprétation graphique de l'approche par ϵ-contrainte[4].*

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. Il est clair qu'une bonne connaissance du problème a priori est requise.

2.2.5 Méthode du but à atteindre (Goal attainment method)

Cette approche, comme celle de min-max, utilise un point de référence pour guider la recherche. Mais elle introduit aussi une direction de recherche, si bien que le processus de résolution devra suivre cette direction. À la différence de l'approche min-max, qui utilise des normes pour formaliser la distance au point de référence, l'approche du *but à atteindre* utilise des contraintes, à l'instar de l'approche ϵ-contrainte, pour déterminer la position du point de référence (aussi appelé le but). L'écart par rapport à ce but est contrôlé grâce à la variable λ introduite à cet effet :

$$\left\|\begin{array}{ll} \text{Minimiser} & \lambda; \\ \text{tel que,} & f_1(x) - w_1.\lambda \leq B_1; \\ & \vdots; \\ & f_m(x) - w_m.\lambda \leq B_m; \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & x \in \mathbb{R}^n, f(x) \in \mathbb{R}^m, g(x) \in \mathbb{R}^q. \end{array}\right.$$

Ainsi en minimisant λ et en vérifiant toutes les contraintes, la recherche va s'orienter vers le but B et s'arrêter sur le point A faisant partie de la surface de compromis (voir l'illustration en 2 dimensions à la figure 2.3. Cette approche permet, comme l'approche par ϵ-contrainte et l'approche min-max, de trouver les parties non convexes des fronts Pareto.

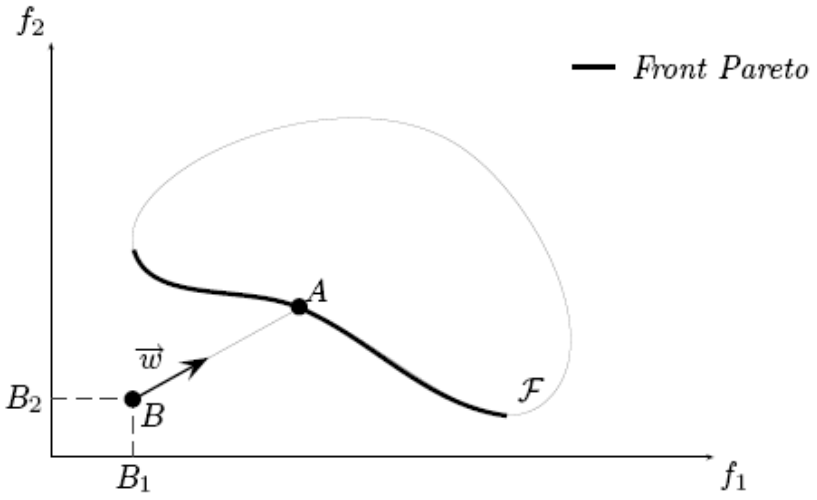


FIG. 2.3 – *Interprétation graphique de l'approche par "but à atteindre"[4].*

Cependant, cette approche, comme les précédentes, doit être itérée plusieurs fois dans le but d'obtenir un ensemble de points Pareto optimaux. Les paramètres w et B doivent être bien choisis par l'utilisateur. Bien que ces paramètres permettent une grande flexibilité de la recherche (orientation et but), s'ils sont mal choisis, ils peuvent, dans certains cas extrêmes, donner des résultats non cohérents.

Il y a lieu de noter que cette approche est très similaire à celle de la "goal programming" [Charnes and Cooper, 1961 ; Romero, 1991 ; Coello, 1998], où les contraintes deviennent des égalités. Des variables d'écart sont alors introduites [4].

2.2.6 Programmation par but (Goal programming method)

Cette méthode est proche de la méthode de but à atteinte. La différence principale est que, après avoir transformé le problème d'optimisation, nous avons des contraintes d'égalité au lieu des contraintes d'inégalité. Cette approche est la suivante [16] :

- Choisi un vecteur initial des fonctions objectif F,
- Associons à chaque objectif deux nouvelles variables "déviations" liées au vecteur initial des fonctions objectif choisies.
- minimiser après une des deux variables ; le choix de la variable se fait en fonction du type de dépassement que l'on veut (au-dessus ou au-dessous de l'objectif que l'on s'est fixé).

Présentation de la méthode On part du problème P. On choisit un vecteur de fonctions objectif initial $F \in \mathbb{R}^m$. On associe aussi un ensemble de variables d_i^+ et d_i^- à chaque fonction objectif $f_i(x), i \in \{1, \ldots, k\}$.

On obtient alors le problème suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & (d_1^+ \ \ ou \ \ d_1^-, \ldots, d_k^+ \ \ ou \ \ d_k^-); \\ \text{tel que,} & f_i(x) = F_1 + d_1^+ - d_1^- \quad i = 1 \ldots k-1; \\ & \vdots; \\ & f_k(x) = F_k + d_1^+ - d_k^-; \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & h(x) = 0. \end{array} \right.$$

Les variables de déviation que l'on cherche à minimiser doivent respecter certaines contraintes :
$d_1^+ \geq 0$ et $d_k^- \geq 0$ et $d_1^+ \cdot d_k^- = 0$ avec $i \in \{1, \ldots, k\}$

En fonction de la façon dont on désire atteindre le but F, on cherchera à minimiser différentes combinaisons des coefficients d_i^+ et d_i^-.

2.2.7 Méthode hybride

Comme nous allons le voir, il est possible d'exploiter plusieurs méthodes pour en former une nouvelle. Dans ce cas, nous obtenons une "méthode hybride".

Présentation de la méthode La méthode hybride la plus connue est la méthode de Corley. Cette méthode utilise la méthode de pondération des fonctions objectif et la méthode du compromis. On part du problème P. On le transforme de la manière suivante :

$$\left\| \begin{array}{ll} \text{Minimiser} & \sum_{i=1}^{m} w_i . f_i(x); \\ \text{tel que,} & f_j(x) \leq \varepsilon_j, j = 1, \ldots, m; \\ \text{et que} & g(x) \leq 0; \\ \text{avec,} & h(x) = 0. \end{array} \right.$$

2.2.8 Méthode lexicographique

Cette méthode est très intuitive. En effet, elle consiste à considérer les fonctions objectif les unes après les autres et à minimiser un problème d'optimisation monobjectif, en complétant au fur et à mesure l'ensemble des contraintes [15].

Présentation de la méthode On part du problème P. On procède ensuite en m étapes (autant d'étapes qu'il y a de fonctions objectif). Commençons avec la première fonction objectif. On résout :

$$\left\| \begin{array}{ll} \text{Minimiser} & f_1(x); \\ \text{tel que,} & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

On note f_1^* la solution de ce problème.

Ensuite, on transforme la première fonction objectif en contrainte d'égalité puis on

prend la seconde fonction objectif et on résout le problème :
On répète cette démarche jusqu'à la fonction objectif k. Et, pour finir, on aura :

$$\left\| \begin{array}{ll} \text{Minimiser} & f_k(x); \\ \text{tel que} & f_1(x) = f_1^*, \ldots, f_{k-1}^*; \\ & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

La dernière valeur de x est celle qui minimise tous les objectifs.

Remarque 3 : D'autres méthodes existent appartenant à cette classe dans la littérature nous pouvons citer :

- La méthode des contraintes d'égalité propres ou Proper-equality-constraints-method ;
- La méthode des contraintes d'inégalité propres ou Proper-inequality-constraints-method ;
- L'algorithme de Lin-Tabak ;
- L'algorithme de Lin-Giesy ;
- etc.

2.3 Méthodes interactives

Les méthodes interactives permettent de chercher une et une seule solution. Elles forment la famille des méthodes progressives et permettent à l'utilisateur de déterminer ses préférences vis-à-vis d'un compromis entre objectifs au cours de l'optimisation. Ces méthodes sont à comparer aux méthodes :

▷ À préférence a priori, où l'utilisateur choisit le compromis à réaliser entre objectifs avant de lancer l'optimisation ;

▷ À préférence a posteriori, où l'utilisateur n'a pas à choisir de compromis avant de lancer l'optimisation. La méthode d'optimisation calcule toutes les solutions efficaces et l'utilisateur peut ainsi les comparer et en choisir une[16].

Nous allons maintenant présenter quelques méthodes interactives (ou plutôt quelques principes d'interaction).

2.3.1 Méthode de compromis par substitution

Cette méthode (Surrogate Worth Tradeoff (SWT) ou "méthode du compromis par substitution") a été très utilisée pour l'optimisation de ressources en eau [Haimes et al. [34]]. Elle est basée sur la méthode du compromis, à laquelle on a ajouté un processus interactif, pour que la méthode converge vers la solution la plus susceptible de satisfaire l'utilisateur [16].

Présentation de la méthode Cette méthode se décompose en sept étapes : (voir algorithme 2) :
Le plus difficile dans cette méthode est de bien comprendre la signification du coefficient W_{1j}.

2.3.2 Méthode de Fandel

Le but de cette méthode est d'aider l'utilisateur dans le choix des coefficients de pondération [25].

Présentation de la méthode On part du problème P. On modifie le problème de la manière suivante :

$$P = \left\| \begin{array}{ll} \text{Minimiser} & \sum_{i=1}^{m} w_i.f_i(x); \\ \text{tel que} & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

On a aussi $w_i \geq 0$ et $\sum_{i=1}^{k} w_i = 1$. On retrouve cette condition dans la méthode de pondération des fonctions objectif.

Comme pour la méthode de pondération des fonctions objectif, la variation des coefficients w_i permet de déterminer les points de la surface de compromis. En revanche, dans la méthode de Fandel, on suppose que ces coefficients ne sont pas connus. On suppose aussi que l'utilisateur cherche une solution qui soit proche de la solution idéale.

2.3.3 Méthode STEP

Cette méthode est similaire à la méthode de Fandel. Ici aussi, les informations sur la préférence de l'utilisateur permettent de restreindre l'espace de recherche étape par étape [25].

Présentation de la méthode On part du problème P et on le transforme de la manière suivante :

$$\left\| \begin{array}{ll} \text{Minimiser} & \beta; \\ \text{tel que} & \left[f_i(x) - \overline{Y}\right]^t \cdot w_i < \beta; \\ & g(x) \leq 0; \\ \text{avec} & h(x) = 0; \\ & f(x) \leq \overline{Y}. \end{array} \right.$$

ou le vecteur $\overline{Y}$ correspond a un vecteur de bornes supérieures servant à restreindre l'espace de recherche. Le vecteur w_j est défini dans la suite de cette section.

De la même manière que l'approche précédente, on forme la matrice B.
Les coefficients de pondération w_j des différentes fonctions objectif f_j sont nécessaires pour la résolution du problème. On va déterminer ces coefficients. On va donner un poids plus important aux fonctions f_j dont l'excursion est importante (f_j prend ses valeurs dans un domaine étendu). Ces coefficients ont la forme suivante :

$$w_j = \frac{v_j}{\sum_{i=1}^{k} v_j};$$

Algorithme 2 Surrogate Worth Tradeoff (SWT)-algorithm

Étape 1 Trouver le minimum de la fonction f_j en résolvant

$$\left\| \begin{array}{ll} \text{Minimiser} & f_j(x); \\ \text{sachant que} & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

La solution de ce problème devient la $j^{\text{ème}}$ composante du vecteur f_{min}, qui regroupe les $k-1$ valeurs minimales de f_j, $j = 1, \ldots, k$. Si possible, on peut chercher à cette étape les composantes du vecteur f_{max}, qui regroupe les $k-1$ valeurs maximales de f_j, $j = 2, \ldots, k$.

Étape 2 Choisir la valeur de départ de $\epsilon \geq f_{j_{min}}$, $j = 2, \ldots, k$.

Étape 3 Résoudre le problème suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & f_1(x); \\ & f_2(x) \geq \epsilon_2, \ldots, f_k(x) \geq \epsilon_k; \\ \text{sachant que} & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

Si certaines des contraintes de ce problème ne peuvent pas être respectées, on pose $\epsilon_j = f_j(x)$ pour j correspondant aux indice des contraintes non respectées (on relâche les contraintes). Il faut ensuite recommencer cette étape. Si les contraintes sont respectées, on appelle x^* le vecteur solution et on passe à l'étape suivante.

Étape 4 Si les contraintes sont maintenant suffisamment relâchées, on passe à l'étape 5 sinon, on choisit une nouvelle valeur de $\epsilon_j \geq f_{j_{min}}$, pour tout $j = 2, \ldots, k$ et on retourne à l'étape 3.
Une méthode pour sélectionner une nouvelle valeur de ϵ consiste à commencer par une grande valeur de ϵ puis à diminuer chaque ϵ_j, pour tout $j = 2, \ldots, k$ d'un certain $\Delta_j > 0$, chaque fois que les contraintes sont respectées.

Si les contraintes ne sont pas respectées, on pose $\epsilon_j = f_j(x)$ avec j indice des contraintes non respectées (ici, x désigne la valeur courante de la solution).

Étape 5 On demande maintenant à l'utilisateur de choisir les coefficients W_{1j}, pour tout $j = 2, \ldots, k$. Ces coefficients représentent l'avis de l'utilisateur vis-à-vis d'un accroissement de la fonction f_j de ω_j unités (le mode de calcul de ω_{1j} est précisé plus loin). Ce coût est mesuré sur une échelle de -10 à +10, où -10 représente un avis défavorable de l'utilisateur vis-à-vis de cet accroissement et +10 un avis favorable. 0 représente l'indifférence.

Cette opération est répétée pour j variant de 2 à k.

Étape 6 On répète l'étape 5 jusqu'à ce que l'on trouve pour les coefficients W_{1j}, $j = 2, \ldots, k$ des valeurs égales à zéro. On note f_j^*, pour tout $j = 2, \ldots, k$ les valeurs des fonctions objectif correspondant à ces coefficients.

Étape 7 Le vecteur solution préféré x^*. est déterminé en résolvant le problème suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & f_1(x); \\ \text{sachant que} & f_2(x) = f_2^*, \ldots, f_k(x) = f_k^*; \\ \text{et} & g(x) \leq 0; \\ \text{avec} & h(x) = 0. \end{array} \right.$$

avec

$$v_j = \frac{\max_i\{f_j^{*i}\} - f_j^{*j}}{\min_i\{f_j^{*i}\}} \cdot \frac{1}{f_j^{*j}};$$

avec $i = 1, \ldots, k$; Par conséquent les poids respectent les conditions $w > 0$ et $\sum_{j=1}^{k} w_j = 1$.

2.3.4 Méthode de Jahn

Cette méthode est complètement différente des deux autres. On commence par choisir un point de départ, puis le problème est résolu pas à pas à travers l'espace de recherche en direction de la solution optimale au sens de Pareto [25].

Cette méthode est fondée sur une méthode de minimisation cyclique (aussi appelée "méthode d'optimisation scalaire dans les directions réalisables" de Zoutendjik).

présentation de la méthode On transforme le problème P de la manière suivante :

$$\left\| \begin{array}{ll} \text{Minimiser} & \beta; \\ \text{tel que} & w_j.\left[\nabla_x f_j^u\right]^t \cdot \delta_f \leq \beta; \\ & v_i.\left[\nabla_x g_i^u\right]^t \cdot \delta_g \leq \beta; \\ & t_l.\left[\nabla_x h_l^u\right]^t \cdot \delta_h = \beta. \end{array} \right.$$

On a $\delta_f \in \mathbb{R}^m$, $\delta_g \in \mathbb{R}^m$, $\delta_h \in \mathbb{R}^p$. Où :

β : fonction scalaire de préférence ;

∇_x : opérateur nabla de la variable x, $\nabla_x A = \sum_i \frac{\partial A}{\partial_{x_i} x_i}$ avec A est la matrice augmenté des f, g, h ;

δ : direction du pas de minimisation.

w_j : coefficient de pondération de la j^e fonction objectif.

v_i : coefficient de pondération de la $i^{\text{ème}}$ contrainte d'inégalité ;

t_l : coefficient de pondération de la $l^{\text{ème}}$ contrainte d'égalité.

La solution de notre nouveau problème commence par le choix d'un vecteur de depart réalisable x^0 et par le calcul du vecteur des fonctions objectif correspondant $f(x^0)$.
Le choix du point de départ influence les alternatives futures possibles, en supposant que l'espace de recherche $\widehat{Y}^0$ soit restreint à :

$$\widehat{Y}^0 = \left\{ f(x) | f(x) \leq f(x^0) \right\};$$

En fonction du vecteur objectif $f(x)$, l'utilisateur choisit une fonction objectif f_i, qui doit être minimisée prioritairement. A cet effet, un pas de direction δ et de longueur appropriée l doit être déterminé, et il doit satisfaire la relation suivante :

$$x^{k+1} = x^k + \lambda^k \cdot \delta^k;$$

Il faut que x^{k+1} et x^k satisfassent les contraintes du problème (h et g). De plus, il faut que ces points vérifient :

$$f_i^{k+1} < f_i^k \quad \forall i \in I = \{1, \ldots, m\};$$
$$f_j^{k+1} \leq f_j^k \quad \forall j \in J = \{I | j \neq i\};$$

On obtient la direction du pas en résolvant notre problème modifié (le problème de départ transformé). En agissant ainsi, les coefficients de pondération w_j, v_i et t_l choisis par l'utilisateur influencent la direction du pas. L'utilisateur peut maintenant choisir la longueur du pas λ^k dans un intervalle $[0, \lambda_{\max}]$. On trouve $\lambda_{\max}$ en résolvant le problème suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & \lambda_{\max}\,; \\ \text{sachant que} & f_i(x^k + \lambda_{\max} \cdot \delta^k) < f_i(x^k), \forall i \in I = \{1, \ldots, k\}\,; \\ & f_j(x^k + \lambda_{\max} \cdot \delta^k) \leq f_j(x^k) \quad \forall j \in J = \{I | j \neq i\}\,; \\ & g(x^k + \lambda_{\max} \cdot \delta^k) \leq 0\,; \\ & h(x^k + \lambda_{\max} \cdot \delta^k) = 0. \end{array} \right.$$

Le nouveau vecteur fonction objectif f^{k+1}, pour le pas de direction et de longueur données, est calculé $f^{k+1} = f(x^k + \lambda^k \cdot \delta^k)$. Si ce vecteur objectif n'est pas accepté, on détermine à la prochaine itération une nouvelle direction, en résolvant notre problème modifié.

2.3.5 Méthode de Geoffrion

Cette approche est similaire à la méthode de Jahn. Elle repose sur l'algorithme de Franck-Wolfe [25]

Présentation de la méthode On transforme le problème (P) de la manière suivante :

$$\left\| \begin{array}{ll} \text{Minimiser} & \left(\nabla_x p \cdot [fx]\right)^t \cdot \delta\,; \\ \text{sachant que} & h(x) = 0\,; \\ & g(x) \leq 0. \end{array} \right.$$

avec $\delta \in \mathbb{R}^m$, et p est une probabilité de choix ;

La fonction de preference $p \cdot [f(x)]$ n'est pas forcément connue par l'utilisateur. Pendant le déroulement de l'algorithme, l'utilisateur devra fournir des informations sur les compromis qu'il désire réaliser. Il doit aussi fournir la longueur du pas.

Deux méthodes sont possibles pour déterminer le point de depart x^0. Soit l'utilisateur sélectionne un vecteur en utilisant la méthode de Jahn, soit le vecteur est calcule en résolvant le problème de substitution suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & \sum_{i=1}^m w_i \cdot f_i(x)\,; \\ \text{sachant que} & h(x) = 0\,; \\ \text{et} & g(x) \leq 0\,; \end{array} \right.$$

On reconnaît là la méthode de pondération des fonctions objectif décrite dans les sections précédentes. En résolvant ce problème, on obtient le point $x^0 = x^*$.

A ce moment, on peut résoudre le problème p'. Cette résolution nous donne la direction de recherche δ. Des informations à propos de la fonction de préférence sont maintenant requises. Il

va falloir transformer le problème P' en utilisant la relation suivante :

$$\nabla_x p \cdot \big[f(x)\big] = \sum_{i=1}^{k} \big[\frac{\partial p}{\partial f_i}\big]_x \cdot \nabla_x \cdot f_i(x);$$

On obtient le problème suivant :

$$\left\| \begin{array}{ll} \text{Minimiser} & \sum_{i=1}^{k} w_i \cdot (\nabla_x f_i(x^k))^t \cdot \delta\,; \\ \text{sachant que} & h(x) = 0\,; \\ \text{et} & g(x) \leq 0. \end{array} \right.$$

On obtient

$$w_i = \frac{\big[\frac{\partial p}{\partial f_i}\big]_x}{\big[\frac{\partial p}{\partial f_1}\big]_x};$$

avec $i \in \{1, \ldots, k\}$, et la direction de l'étape est donnée par $\delta = x^{k+1} - x^k$.

Les coefficients de pondération reflètent le compromis qui est fait par l'utilisateur entre la fonction objectif f_i et la fonction objectif de référence f_1.

Après résolution du problème de substitution ci-dessus, la longueur du pas λ doit être déterminée par l'utilisateur lui-même, qui doit considérer le but :

$$\left\| \begin{array}{ll} \text{Minimiser} & p\big[f(x_k + \lambda^k \cdot \delta^k)\big]\,; \\ \text{sachant que} & h(x_k + \lambda^k \cdot \delta^k) = 0\,; \\ \text{et} & g(x_k + \lambda^k \cdot \delta^k) \leq 0. \end{array} \right.$$

en utilisant l'hypothèse suivante :

$$0 \leq \lambda^k \leq 1;$$

Une fois que la longueur du pas est déterminée, le nouveau vecteur objectif peut être calculé :

$$f^{k+1} = f(x_k + \lambda^k \cdot \delta^k);$$

L'utilisateur évalue alors le résultat, et décide si le processus peut continuer, ou si la solution peut être acceptée comme solution de compromis. S'il décide de continuer, il peut alors parcourir la surface de compromis.

2.3.6 Méthode de Simplexe

Cette méthode n'a rien à voir avec la méthode du simplexe utilisée en programmation linéaire. Il s'agit là d'une méthode de recherche séquentielle [Nelder et al. [49]] de l'optimum d'un problème d'optimisation. Le logiciel [Multisimplex] réalise cette optimisation pour un problème d'optimisation multiobjectif de manière interactive.

Cette méthode utilise $k + 1$ essais (où k représente la dimension de la variable de décision x) pour définir une direction d'amélioration des fonctions objectif. L'amélioration des fonctions objectif est obtenue en utilisant une méthode d'agrégation floue.

On commence par choisir au hasard $k+1$ valeurs pour la variable de décision x. L'algorithme évalue alors les $k+1$ points et supprime le point le moins "efficace". Il crée alors un nouveau point à partir du point supprimé (voir figure 2.4) et recommence l'évaluation.
L'algorithme comporte quelques règles, qui permettent de choisir des points qui évitent de tourner autour d'une mauvaise solution. Les deux principales règles sont les suivantes :

Règle 1 : rejeter les pires solutions. Une nouvelle position de la variable de decision x est calculée par réflexion de la position rejetée (voir la figure 2.4). Après cette transformation, on recherche le nouveau pire point. La méthode est alors répétée en eliminant ce point, etc. A chaque étape, on se rapproche de la zone où se trouve l'optimum recherché.

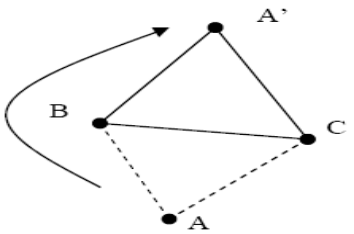


FIG. 2.4 – *Choix d'un nouveau point par symétrie[16].*

Règle 2 : ne jamais revenir sur un point qui vient juste d'être rejeté. Sans cette règle, l'algorithme pourrait osciller entre deux "mauvais" points. [16]
W : point le moins favorable ou point venant d'être rejeté.
B : point le plus favorable.
R : second point le plus favorable.

2.4 Méthodes floues ou brouillées

Dans la vie courante, tout ne peut pas être décrit de manière binaire. Par exemple, la transition entre le jour et la nuit se fait progressivement, l'action sur l'embrayage d'un véhicule est, elle aussi, progressive.
Longtemps, le seul outil de description en logique était binaire. Tout en logique a été décrit en termes de VRAI ou FAUX. Le problème de cette description simpliste en logique est qu'elle ne permet pas de traiter l'incertitude et l'imprécision des connaissances humaines.

L'automaticien L. A. Zadeh a élaboré une nouvelle logique basée sur les ensembles flous. Elle permet de traiter l'imprécision et l'incertitude dans la connaissance humaine ainsi que les transitions progressives entre états. La différence principale entre une logique classique et cette logique floue est l'existence d'une transition progressive entre le VRAI et le FAUX [66].

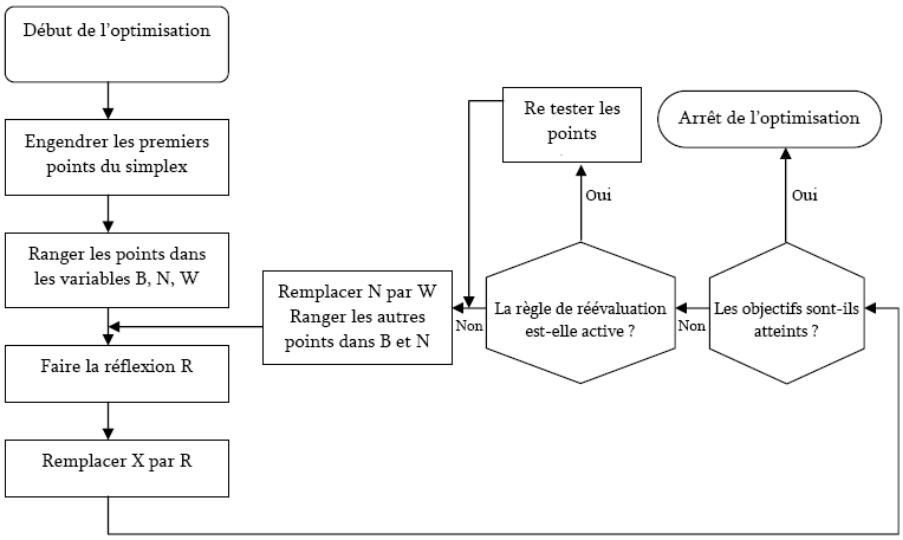


FIG. 2.5 – *L'algorithme de la méthode de Simplex[16].*

2.4.1 Méthode de Sakawa

Cette méthode fait intervenir la logique floue à tous les niveaux (sur les paramètres du problème ainsi que sur les paramètres des contraintes). L'ensemble de solutions que l'on va trouver, lui aussi, fera intervenir la logique floue. Ces solutions auront un niveau d'appartenance. Ce seront des solutions qui auront une corrélation variable avec l'objectif initial [Sakawa et al.[53]] (le niveau de corrélation avec l'objectif initial est fixé par l'utilisateur) [16].

On part du problème suivant :

$$\left\| \begin{array}{l} \text{" min "} f_1(x), \ldots, f_k(x); \\ g(x) \leq 0. \end{array} \right.$$

2.4.2 Méthode de reardon

On présente une méthode simplifiée de traitement multiobjectif utilisant la logique floue. Des exemples utilisant cette méthode peuvent être trouvés dans [Reardon [51]].

Présentation de la méthode On part du problème P. Pour chaque fonction objectif, on définit une fonction d'appartenance, qui aura la forme représentée à la figure 2.6.

Cette fonction d'appartenance permet "d'informer" l'algorithme que la fonction objectif a sa valeur située dans l'intervalle $[[O_i - E_i, O_i + E_i]$ (zone où la fonction d'appartenance vaut 0). Si la valeur de la fonction objectif est située en dehors de cette zone, on pénalise de plus en plus la fonction objectif. Dans ce cas, la pénalité varie entre 0 et S_{min} et S_{max}.
La signification des différents paramètres de la fonction d'appartenance est la suivante :

S_{min} **et** S_{max} **:** facteurs d'échelle flous.

f_{min} **:** valeur minimale de la fonction objectif numéro i.

Algorithme 3 Algorithme de la méthode de Sakawa

Étape 1 : sous la contrainte $g(x) \leq 0$, on minimise, puis on maximise, chaque fonction objectif séparément, de manière a obtenir la plage de variation de la fonction d'appartenance des fonctions objectif.

Étape 2 : On définit les fonctions d'appartenance de la manière suivante :

$$\mu_i(x) = \frac{f_{i1} - f_i(x)}{f_{i1} - f_{i0}}; \tag{2.1}$$

où

- f_{i0} est la valeur de la fonction d'appartenance la moins intéressante,
- f_{i1} est la valeur de la fonction d'appartenance la plus intéressante.

Étape 3 : On définit les fonctions de prise de décision DM_i (Decision Making) comme suit :

$$DM_i(x) = \begin{cases} 0, & \text{si } \mu_i(x) \leq 0\,; \\ \mu_i(x), & \text{si } 0 \leq \mu_i(x) \leq 1\,; \\ 1, & \text{si } \mu_i(x) \geq 1. \end{cases} \tag{2.2}$$

où

- $DM_i(x) = 0$, quand l'objectif n'est pas atteint,
- $DM_i(x) = 1$, quand l'objectif est atteint.

Étape 4 : On maximise les fonctions DM_i.

Étape 5 : S'il n'y a pas de solutions, ou si la solution ne satisfait pas l'utilisateur, alors il faut modifier les fonctions d'appartenance, et retourner à l'étape 3.

Étape 6 : Arrêt et affichage des résultats.

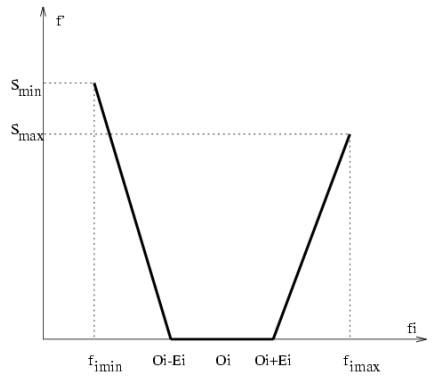


FIG. 2.6 – *Fonction d'adhésion de Reardon[16].*

f_{max} **:** valeur maximale de la fonction objectif numéro i.

O_i **:** valeur expérimentale de la fonction objectif numéro i : on voudrait que $f_i = O_i$.

E_i **:** marge d'erreur acceptable (l'objectif O_i est défini à $2 \cdot E_i$ près)

- si $f_i \leq (O_i - E_i)$ alors $f'(f_i) = [\frac{S_{max}}{f_{min} - (O_i - E_i)}] \cdot (f_i - (O_i - E_i))$;
- si $(O_i - E_i) \leq f_i \leq (O_i + E_i)$ alors $f'(f_i) = 0$;
- si $f_i \geq (O_i + E_i)$ alors $f'(f_i) = [\frac{S_{min}}{(O_i + E_i) - f_{max}}] \cdot (f_i - (O_i + E_i))$.

2.5 Métaheuristiques multiobjectif

2.5.1 Qu'est-ce qu'une métaheuristique ?

Les métaheuristiques sont des méthodes générales de recherche dédiées aux problèmes "d'optimisation difficile" [Sait et al. [52]]. Ces méthodes sont, en général, présentées sous la forme de concept. Comme nous le verrons plus tard, elles reprennent des idées que l'on retrouve parfois dans la vie courante. Les principales métaheuristiques sont le recuit simulé, la recherche tabou et les algorithmes génétiques. Ces méthodes ont des inspirations de l'éthologie comme les colonies de fourmis, de la physique comme le recuit simulé, et de la biologie comme les algorithmes évolutionnaires. Dans la figure suivante 2.7, on voit un classement de ces méthodes selon le principe d'inspiration utilisé, est ce qu'il est basé. Plusieurs définition ont été données pour clarifier le concept de l'heuristique, parmi les quelles on trouve les définitions suivantes :

Définition 2.1 [17] *Une heuristique est une technique trouvant de bonnes solutions pour un coût de calcul raisonnable, sans pouvoir garantir l'admissibilité ou l'optimalité, ou même dans de nombreux cas, préciser la distance à l'optimum d'une solution particulière.[50]*

Typiquement, ce genre de méthodes est particulièrement utile pour les problèmes nécessitant une solution en temps réel (ou très court) ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille. Elles peuvent aussi être utilisées afin d'initialiser une méthode exacte (Branch and Bound par exemple).

A côté des méthodes heuristiques, sont apparues des méthodes qualifiées de métaheuristiques.

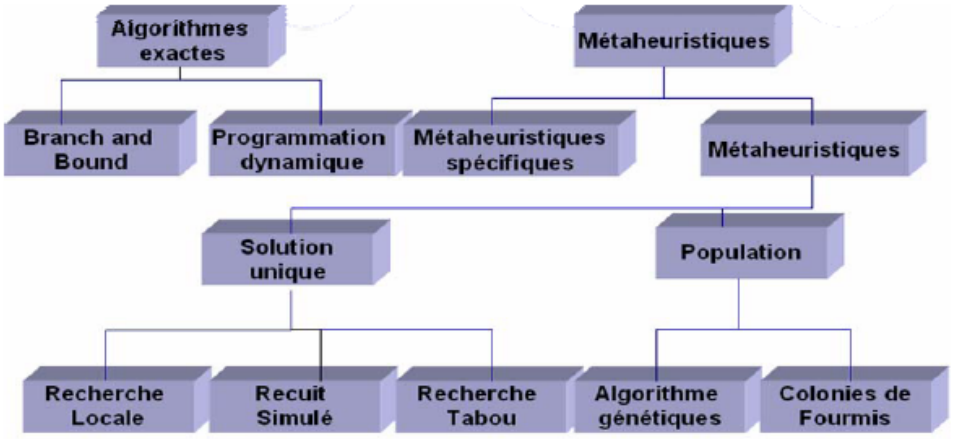


FIG. 2.7 – *Schéma des méthodes basées sur les métaheuristiques[4].*

Plusieurs définitions ont été proposées pour décrire leurs particularités par rapport aux heuristiques.

Définition 2.2 [17]**(Métaheuristique selon Osman et Laporte)** *Une métaheuristique est un processus itératif de génération guidant une heuristique subordonnée en combinant intelligemment différents concepts ; pour explorer et exploiter l'espace de recherche en utilisant des stratégies pour structurer l'information de manière à trouver efficacement des solutions proches de l'optimum [27].*

Cependant, cette définition ne fait pas clairement apparaître l'indépendance des métaheuristiques actuelles vis-à-vis des problèmes à traiter.

Définition 2.3 [17]**(Métaheuristique selon Pirlot)** *Les métaheuristiques ne sont pas à proprement parler des heuristiques, mais des schémas généraux, des "moules" à heuristiques ; le schéma général doit être adapté à chaque type particulier de problème. [50]*

Malgré la grande diversité de métaheuristiques (algorithmes génétiques, méthode de bruitage, méthode GRASP, recherche à voisinage des variable, recherche dispersée, recherche tabou, recuit simulé, systèmes de fourmis, ...), leurs principes sont souvent assez proches. Taillard [58] distingue trois éléments principaux constitutifs de toutes les métaheuristiques :

- la structure de voisinage permettant de modifier une solution,
- la mémoire des solutions déjà obtenues,
- la construction de solutions entièrement nouvelles.

Ainsi, chaque métaheuristique utilise au moins l'un de ces trois éléments. De plus, l'hybridation de différentes métaheuristiques est une pratique de plus en plus courante permettant d'obtenir des heuristiques performantes. Taillard [57] propose d'ailleurs une unification de la plupart des métaheuristiques telles qu'elles sont implantées actuellement sous la dénomination de programmation à mémoire adaptative.[17]

Une métaheuristique est donc une méthode très générale, qui nécessite quelques transformations (mineures en générale) avant de pouvoir être appliquée à la résolution d'un problème particulier. Si l'on considère les différentes métaheuristiques, on constate que celles-ci se répartissent principalement en trois familles :

2.5.2 Méthodes déterministes de recherche d'optimum local

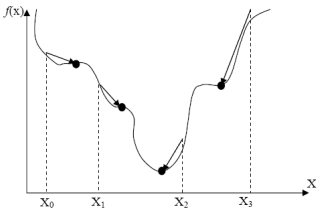
Ces méthodes convergent rapidement mais, la plupart du temps, elles ne trouvent pas l'optimum global. Elles se contentent de trouver un optimum local et ne reposent pas sur un processus stochastique pour la recherche de l'optimum (voir 2.8(a)).

2.5.3 Méthodes déterministes de recherche d'optimum global

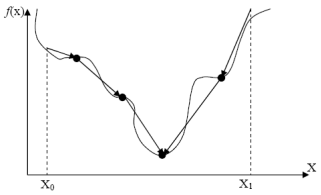
Ces méthodes permettent de trouver un optimum global rapidement et ne reposent pas sur un processus stochastique pour la recherche de l'optimum (voir 2.8(b)).

2.5.4 Méthodes stochastiques de recherche d'optimum global

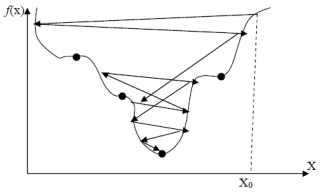
Ces méthodes reposent sur un processus stochastique chargé d'effectuer la recherche de l'optimum. Elles sont moins performantes (du point de vue rapidité) que les méthodes déterministes, mais elles peuvent trouver, un optimum global difficile à atteindre(voir 2.8(c)).



(a) *Méthode déterministe de recherche de l'optimum local.*



(b) *Méthode déterministe de recherche d'optimum global.*



(c) *Méthode stochastique de recherche d'optimum global.*

FIG. 2.8 – *Différentes familles de métaheuristiques.*

2.5.5 Méthode de recherche locale

La première classe des métaheuristiques présentées regroupe les méthodes utilisant les principes de la recherche locale. Ces méthodes résolvent le problème d'optimisation de manière

itérative. Elles font évoluer la configuration courante en la remplaçant par une autre issue de son voisinage, ce changement de configuration est couramment appelé mouvement. Parmi les méthodes de recherche locale la plus simple et la plus utilisée dans la literature est La descente[4].

2.5.6 Descente

La descente est une méthode d'amélioration itérative simple permettant d'atteindre le premier optimum local. La descente pour un problème de minimisation peut être définie très simplement :

Algorithme 4 Pseudo-code de la méthode de descente.

Entrées: Paramètres d'entrée : N ; f ; x_0

$x_{suiv} \leftarrow x_0$;

répéter

$x \leftarrow x_{suiv}$;

$x_{suiv} \in \{x' | x' \in N(x) \bigwedge f(x') = \min(\{f(y) | y \in N(x)\})\}$;

jusqu'à $f(x_{suiv}) > f(x)$;

Renvoyer x.

où N est la fonction de voisinage, f la fonction d'évaluation, et x_0 la configuration initiale servant de point de départ à l'algorithme.

2.5.7 Recuit simulé

Cette méthode de recherche a été proposée par des chercheurs d'IBM qui étudiaient les verres de spin. Ici, on utilise un processus métallurgique (le recuit) pour trouver un minimum. En effet, pour qu'un métal retrouve une structure proche du cristal parfait (l'état cristallin correspond au minimum d'énergie de la structure atomique du métal), on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement (voir [Bonomi], [Siarry] et [Siarry][12, 54, 54]).

Ce processus métallurgique a été transposé à l'optimisation et a donné une méthode simple et efficace. Le pseudo-code du recuit simulé est représenté dans l'algorithme qui suit. Le fonctionnement de cet algorithme est le suivant :

- On commence par choisir un point de départ au hasard (x) ;
- On calcule un voisin de ce point ($\gamma = \mathcal{V}(x)$) ;
- On évalue ce point voisin et on calcule l'écart par rapport au point d'origine ($\Delta C = C(\gamma) - C(x)$) ;
- Si cet écart est négatif, on prend le point γ comme nouveau point de départ, s'il est positif, on peut quand même accepter le point y comme nouveau point de départ, mais avec une probabilité $e^{-\frac{\Delta C}{T}}$ (qui varie en sens inverse de la température T) ;
- Au fur et à mesure du déroulement de l'algorithme, on diminue la température $T(T = \alpha(T))$, souvent par paliers ;
- On répète toutes ces étapes tant que le système n'est pas figé (par exemple, tant que la température n'a pas atteint un seuil minimal).

Au début de la simulation, les points ont une grande capacité d'exploration de l'espace d'état car l'algorithme accepte des déplacements très importants. Au fur à mesure que T diminue P augmente, la capacité de déplacement d'un point diminue et les points améliorant leur valeur sont de plus en plus nombreux, la chance d'une solution négative d'être acceptée diminue. Quand $T \rightarrow 0$ seuls les points ameliorant leurs valeurs sont acceptés. A la fin il y a une chance de zéro que n'importe quel changement négatif de température peut être alloué. A ce point, le « refroidissement » est dit d'avoir lieu, et le système doit avoir convergé à la solution optimale.

Algorithme 5 Recuit simulé.

Init T (température initial), Init x (point de départ), Init ΔT (temperature) ;
tantque not end **faire**
  $y = Voisin(x)$, $\Delta C = C(y) - C(x)$;
  si $\Delta C < 0$ **alors**
    $y = x$;
  sinon
    si $alea(0,1) < e^{-\frac{\Delta C}{T}}$ **alors**
      $y = x$ $T = \alpha(T)$;
    finsi
  finsi
  si $T < \Delta T$ **alors**

  finsi
fin tantque
REPEAT(while) ;

Dans la littérature des méthodes d'optimisation multiobjectif, on trouve deux importantes extensions de la méthode du Recuit Simulé basées sur les frontières de Pareto. La méthode P.A.S.A (Pareto Archived Simulated Annealing et La méthode M.O.S.A (Multiple objectif Simulated Annealing).

2.5.8 Recherche tabou

Cette méthode, mise au point par F. Glover, est conçue en vue de surmonter les minima locaux de la fonction objectif. C'est une technique d'optimisation combinatoire que certains présentent comme une alternative au recuit simulé.

A partir d'une configuration initiale quelconque, Tabou engendre une succession de configurations qui doit aboutir à la configuration optimale. A chaque itération, le mécanisme de passage d'une configuration, soit x_n, à la suivante, soit x_{n+1}, est le suivant :

- on construit l'ensemble des "voisins" de x_n, c'est-à-dire l'ensemble des configurations accessibles en un seul "mouvement" élémentaire à partir de x_n (si cet ensemble est trop vaste, on en extrait aléatoirement un sous-ensemble de taille fixée) : soit Voisinage(x_n) l'ensemble (ou le sous-ensemble) envisagé ;
- on évalue la fonction objectif f du problème pour chacune des configurations appartenant à Voisinage(x_n). La configuration x_{n+1}, qui succède à la configuration x_n dans la chaîne de

Markov construite par Tabou, est la configuration de Voisinage(x_n) en laquelle f prend sa valeur minimale.

Notons que la configuration x_{n+1} est adoptée même si $f(x_{n+1}) > f(x_n)$: c'est grâce à cette particularité que Tabou permet d'éviter les minima locaux de f .

Cependant, telle quelle, la procedure ne fonctionne généralement pas, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui provoque l'apparition d'un cycle. Pour éviter ce phénomène, on tient à jour, à chaque iteration, une "liste Tabou" de mouvements interdits ; cette liste - qui a donné son nom à la méthode - contient les mouvements inverses ($x_{n+1} \rightarrow x_n$) des m derniers mouvements ($x_n \rightarrow x_{n+1}$) effectués (typiquement m=7). La recherche du successeur de la configuration courante x_n est alors restreinte aux voisins de x_n qui peuvent être atteints sans utiliser un mouvement de la liste tabou. La procédure peut être stoppée des que l'on a effectué un nombre donné d'itérations, sans améliorer la meilleure solution atteinte jusqu'ici.
Le pseudo-code de cette méthode est représenté dans l'algorithme 6.

Algorithme 6 Algorithme Tabou standard TS[4].

1: Soit x une configuration réalisable, et L le nombre d'itérations
2: $x^* = x$ (x^* est la meilleure solution obtenu auparavant) ;
3: **pour** $i = 0$ jusqu'à L **faire**
4: Choisir le meilleur mouvement m autorisé selon $opt(x)$ qui est une fonction d'evaluation définie par le décideur ;
5: Mettre à jour la liste Tabou en fonction de m
6: **si** $f(x) < f(x^*)$ **alors**
7: $x^* \leftarrow x$;
8: **finsi**
9: **fin pour**
10: Renvoyer (x).

2.5.9 Méthode GRASP

La métaheuristique *GRASP* a été proposée initialement par Féo et Resende [26]. C'est une méthode multi-départ en deux phases pour la résolution approchée de problèmes difficiles de l'optimisation combinatoire. La première phase correspond à la construction d'une solution initiale à l'aide d'une procédure gloutonne aléatoire. L'incorporation d'une composante aléatoire permet d'obtenir des solutions dans des zones diverses de l'espace des solutions. La seconde phase correspond à une recherche locale améliorant ces solutions. Ce processus est répété un grand nombre de fois. De plus, plusieurs nouveaux composants sont venus compléter le schéma de base de *GRASP* comme par exemple :
Une implementation de *GRASP* pour un problème particulier va être caractérisée par six éléments principaux :

– l'algorithme glouton utilisé,

- l'importance de la composante aléatoire (fixée par un paramètre $\alpha \in [0;1]$, une valeur $\alpha = 1$ correspondant à un glouton pur et une valeur $\alpha = 0$ correspondant à un algorithme aléatoire),
- le type de recherche locale et le voisinage considéré,
- la phase d'intensification éventuelle,
- le critère d'arrêt,
- la phase post-optimisation éventuelle.

L'adaptation de *GRASP* pour résoudre un problème particulier est assez facile lorsqu'il existe des algorithmes de construction et de recherche locale pour ce problème. *GRASP* a ainsi été appliqué avec succès à un grand nombre de problèmes d'optimisation comme des problèmes d'ordonnancement, de routage ou encore des problèmes théoriques dans les graphes.

2.5.10 Méthode des Colonies de Fourmis

La recherche guidée par Marco Dorigo produit un nouveau membre de cette classe d'algorithmes : l'algorithme de «système de fourmis».[28]

Algorithme de base Le point de départ de cet algorithme se base sur l'observation des fourmis qui construisent des chemins entre une source de nourriture et leur nid. Les fourmis sont capables de déposer sur le sol une certaine quantité d'une substance chimique volatile (le phéromone) qu'elles peuvent détecter ensuite. Les fourmis se déplacent au hasard, dans ce cas, on peut dire qu'elles sont aveugles, mais sont attirées par les chemins de phéromone déposées par d'autres fourmis. Ainsi, plus les fourmis empruntent un chemin, plus il y aura de fourmis attirées par cet itinéraire. Mais dans le cas de cet algorithme de base, quelques modifications sont apportées aux capacités des fourmis telles que :

1. elles possèdent une mémoire ;
2. elles ne sont pas totalement aveugles ;
3. le temps est discret.

- J_i^k : la liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i :
- $\eta_{ij} = \frac{1}{d_{ij}}$: l'inverse de la distance entre les villes, appelée visibilité. Cette information " statique" est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines ;
- τ_{ij} la quantité de pheromone déposée sur l'arête reliant les deux villes, appelée l'intensité de la piste. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système.

L'algorithme 7 donne la structure générale de système de fourmis pour le TSP.

Autres variantes

Il existent d'autres variantes de la méthode telles que : AS rank qui est une version élitiste de AS, le Max-Min Ant System qui introduit l'utilisation des valeurs τ_{max} et τ_{min} pour l'intensité

Algorithme 7 Algorithme de colonies de fourmis de base "Ant System"

Initialisation : $\tau ij \longrightarrow \tau_0 \quad \forall(i,j)$
aléatoirement chaque fourmi sur une ville.
pour t = 1, ...,t_max **faire**
  pour chaque fourmi k = 1,...,m **faire**
    Choisir une ville au hasard ;
    pour chaque ville non visitée i **faire**
      Choisir une ville j, dans la liste J_i^k des villes restantes ;
    fin pour
    Déposer une piste $\Delta\tau_{ij}^k(t)$ sur le trajet $T^k(t)$;
  fin pour
  Évaporer les pistes
fin pour

de la trace de phéromone. Ceci permet d'éviter que certains chemins soient trop favorisés, Ant Colony System (ACS) a été introduit pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles. Elle est fondée sur des modifications du AS.

2.5.11 Méthode Monté Carlo

Les méthodes Monté Carlo consistent en des simulations expérimentales ou informatiques des problèmes mathématiques ou physiques, basées sur le tirage des nombres aléatoires. Généralement, on utilise des séries de nombres pseudo-aléatoires générées par des algorithmes spécialisés. Les propriétés de ces séries sont très proches de celles d'une véritable suite aléatoire.

Le grand avantage de cette méthode est sa simplicité. Elle permet entre autres de visualiser l'effet de différents paramètres et de donner ainsi des orientations, d'étudier des structures intéressantes qui auraient été a priori écartées et de trouver facilement des structures que l'on n'aurait pas aussi bien optimisées «à la main».

Algorithme 8 Algorithme de Monté Carlo :

étape 1 On génère un point initial x dans l'espace d'état, considéré comme solution courante.

étape 2 On génère aléatoirement un point x'.

étape 3 Si x' est meilleur que x alors x' devient la solution courante.

étape 4 Si le critère d'arrêt est satisfait alors fin sinon retour en à la deuxième étape

2.5.12 Optimisation par essaims de particules

Ł'optimisation par essaims de particules (OEP) est une méthode née en 1995 aux États-Unis sous le nom de Particle Swarm Optimization (PSO). Initialement, ses deux concepteurs, Russel Eberhart et James Kennedy [14], cherchaient à modéliser des interactions sociales entre des

«agents» devant atteindre un objectif donné dans un espace de recherche commun, chaque agent ayant une certaine capacité de mémorisation et de traitement de l'information. La règle de base était qu'il ne devait y avoir aucun chef d'orchestre, ni même aucune connaissance par les agents de l'ensemble des informations, seulement des connaissances locales. Un modèle simple fut alors élaboré.

Dès les premières simulations, le comportement collectif de ces agents évoquait celui d'un essaim d'êtres vivants convergeant parfois en plusieurs sous-essaims vers des sites intéressants. Ce comportement se retrouve dans bien d'autres modèles, explicitement inspirés des systèmes naturels Ici, la métaphore la plus pertinente est probablement celle de l'essaim d'abeilles, particulièrement du fait qu'une abeille ayant trouvé un site prometteur sait en informer certaines de ses consoeurs et que celles-ci vont tenir compte de cette information pour leur prochain déplacement.

2.5.13 Algorithmes évolutionnaires

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. Comme exemple des algorithmes évolutionnaires, nous présentons maintenant les algorithmes génétiques [4].

2.5.14 Algorithmes génétiques

Les algorithmes évolutionnaires sont parmi les métaheuristiques à base de population. Ils sont inspirés de la biologie et introduisent le théorème de Darwin dans l'évolution des espèces. Les algorithmes génétiques (AGs) ont été introduits par Holland [Holland, 1975 [35]] comme un modèle de méthode adaptative. Ils s'appuient sur un codage de l'information sous forme de chaînes 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 [4].

Récemment, beaucoup de recherches ont été menées sur l'application des algorithmes évolutionnaires aux problèmes d'optimisation multiobjectif. Celles-ci ont permis de mettre en avant l'intérêt d'utiliser des méthodes d'optimisation basées sur le concept de population. Nous présentons maintenant deux algorithmes évolutionnaires représentatifs, résolvant des problèmes d'optimisation multiobjectif [4].

Non dominated sorting genetic algorithm II"

$NSGA-II$ [4] est un algorithme élitiste n'utilisant pas d'archive externe pour stocker l'élite. Pour gérer l'élitisme, NSGA-II assure qu'à chaque nouvelle génération, les meilleurs individus rencontrés soient conservés.

Algorithme 9 Pseudo-code de l'algorithme $NSGA-II$.

tantque critère_d'arrêt_non_rencontré **faire**
  Créer $R_t = P_t \bigcup Q_t$
  Calculer différents fronts F_i de la population R_t par un algorithme de "ranking" ;
  Mettre $P_{t+1} = \phi$; et $i = 0$,
  tantque $|P_{t+1}| + |F_i| < N$ **faire**
    $P_{t+1} = P_t \bigcup F_i$
    $i = i + 1$
  fin tantque
  Inclure dans P_{t+1} les $(N - |Pt + 1|)$ individus de F_i les mieux répartis
  au sens de la distance de "crowding"
  Sélectionner dans P_{t+1} et créer de Q_{t+1} par application des opérateurs de croisement et mutation ;
fin tantque

Strength Pareto evolutionary algorithm

La méthode SPEA [4] implementée par Zitzler et Thiele [Zitzler and Thiele, 1998b] est l'illustration même d'un algorithme évolutionnaire élitiste. Pour réaliser cet élitisme, $SPEA$ maintient une archive externe contenant le meilleur front de compromis rencontré durant la recherche.

La première étape consiste à créer une population initiale (constituée par exemple d'individus générés aléatoirement). L'archive externe, au départ initialisée à l'ensemble vide, est mise à jour régulièrement en fonction des individus non dominés de la population.

À chaque itération de l'algorithme, on retrouve les étapes classiques *sélection* + *croisement* + *mutation*. Puis les nouveaux individus non dominés découverts viennent s'ajouter à l'archive, et les individus de l'archive dominés par le nouvel arrivant sont supprimés. Si l'archive vient à excéder une certaine taille (fixée au départ), alors une phase de clustering est appliquée dans le but de garder les meilleurs représentants. La figure 2.9 (extraite de la thèse de Zitzler [Zitzler, 1999]) illustre le schéma général de fonctionnement de l'algorithme.

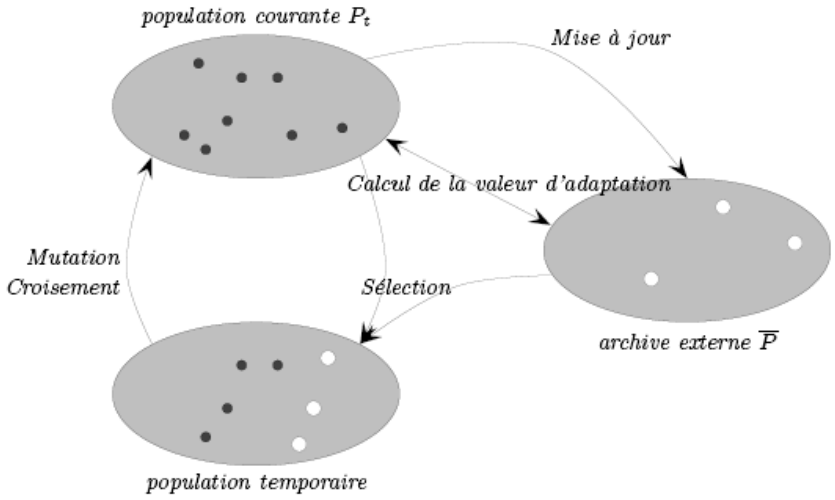


FIG. 2.9 – *Fonctionnement général de l'algorithme SPEA[4].*

2.6 Méthodes d'aide à la décision

Les méthodes que nous avons présentées jusqu'ici sont basées sur la relation de dominance. Cette relation (que l'on peut définir de plusieurs manières : la dominance de Pareto, la dominance lexicographique par exemple) permet de filtrer les éléments d'un ensemble, et de ne retenir que les éléments incomparables entre eux. Cependant, il existe une autre approche pour obtenir un ensemble de solutions, qui repose sur l'établissement d'une relation d'ordre entre les différents éléments. Ainsi, on peut, en fonction de la relation d'ordre définie, obtenir un ensemble de solutions (relation d'ordre partiel) ou une et une seule solution (ordre complet). L'autre différence majeure, par rapport aux méthodes d'optimisation multiobjectif classiques, vient du fait que les méthodes d'aide à la décision ne travaillent que sur des ensembles discrets de points (les méthodes d'optimisation multiobjectif "classiques" peuvent travailler sur des ensembles continus)[4].

De plus, les méthodes d'aide à la décision permettent de répondre à plusieurs problématiques, réunies dans le tableau 2.1.

L'aide à la décision a été développée à partir de la constatation explicitée maintenant.
Dans certains cas, lorsque l'on est amené à comparer trois actions, on peut rencontrer un bouclage entre ces actions. Ce phénomène est appelé l'intransitivité de la préférence, et de l'indifférence, ou paradoxe de Condorcet. Il se résume de la manière suivante : soient trois actions A, B et C,
on peut avoir $A \geq B$, $B \geq C$ et $C \geq A$ (ici, le symbole $\geq$ désigne la preference).

L'aide à la décision nous propose donc une prise en compte de ces propriétés d'intransitivité des relations d'ordre, et elle nous propose aussi des familles de méthodes destinées à résoudre des problématiques différentes (Sélection, Tri, Rangement) de celles traitées par l'optimisation multiobjectif classique.

Problématique	Résultat	Procédure
Choix d'un sous-ensemble des actions les "meilleures" ou, a défaut, les plus "satisfaisantes".	Choix	Sélection
Tri par affectation des actions à des catégories prédéfinies.	Tri	Affectation
Rangement de classes d'équivalence composées d'actions, ces classes étant ordonnées de façon complète ou partielle.	Rangement	Classement

TAB. 2.1 – *Les divers problèmes répondus par des méthodes d'aide de décision*

Lorsque l'on est confronté au domaine de l'aide à la décision, on remarque un certain nombre de termes redondants : Action désigne un objet, une décision, un candidat ou autre chose encore. C'est sur cette entité que va s'opérer la sélection (ou le tri, ou le classement).

Une méthode d'aide à la décision va donc permettre de choisir la meilleure action, ou de classer des actions en fonction d'un ou plusieurs critères.

En fonction du type de classement que l'on désirera effectuer, on pourra utiliser différents types de règles de classement ou de choix. Ces règles vont définir un ordre complet (si toutes les actions sont classées) ou partiel (après le classement, il subsiste des actions incomparables, donc non classées).
Il existe plusieurs méthodes d'aide à la décision, on cite par example la famille ELECTRE (I, IS, II, III, IV et TRI), et la famille PROMETHEE (I et II)[4].

2.7 Difficultés de l'optimisation multiobjectif

Dans cette section, nous présentons toutes les difficultés rencontrées par le processus d'optimisation multiobjectif. Un processus d'optimisation multiobjectif doit résoudre les deux tâches suivantes[9] :

- guider le processus de recherche vers la frontière de Pareto,
- maintenir une diversité des solutions pour assurer une bonne répartition sur la frontière Pareto.

L'accomplissement de ces tâches est très délicat car les difficultés rencontrées dans un problème multiobjectif sont identiques à celles d'un problème monobjectif. Elles sont amplifiées par la présence d'objectif dépendant les uns des autres.

2.7.1 Guider le processus de recherche vers la frontière Pareto

Le processus de recherche est souvent ralenti ou totalement dérouté par des fonctions possédant une des caractéristiques suivantes : **multimodalité** (lorsque une fonction possède plusieurs optimums globaux, dès lors, chaque optimum exerce une attraction sur le processus de résolution ce qui peut piéger la convergence de la méthode), **isolation d'un optimum** (il existe des problème dans lesquels un optimum peut être entouré de grandes zones pratiquement plates. Cet optimum

se trouve alors isolé car l'espace de recherche qui l'entoure ne peu pas guider vers lui les individus de la population) et **tromperie** (un problème est trompé lorsqu'il guide la convergence vers une zone non optimale de la fonction).

2.7.2 Maintenir la diversité sur le front Pareto

La difficulté à maintenir une bonne répartition des solutions sur la frontière de Pareto résulte principalement des caractéristiques suivantes : **convexité ou non convexité** de la frontière de Pareto (certains problèmes ont une frontière de Pareto non convexe), **discontinuité** de cette frontière (si une frontière de Pareto est discontinue, nous retrouvons le même principe que pour une fonction multimodale, les différentes parties de cette frontière vont exercer proportionnellement à leurs tailles, une attraction plus ou moins importante sur les individus d'une population, certaines d'entre elles pourront donc ne pas être découvertes) **non uniformité** de la distribution (les solutions sur la frontière de Pareto peuvent ne pas être réparties uniformément, la raison principale vient du choix des fonctions objectifs, si une des fonction objectif est multimodale, elle va influencer de manière très différente la répartition des solutions sur la frontière de Pareto).

Conclusion

Dans ce chapitre, nous avons essayé de rassembler un état d'art sur les problèmes d'optimisation multiobjectif, tout en montrant la manière de définir un problème pareil, ses contraintes, ses objectifs ainsi que les méthodes de résolution utilisées, tout en respectant le concept de Compromis et les frontières de Pareto. Nous avons parlé des méthodes utilisées dans le domaine de l'optimisation multiobjectif à savoir exactes et métaheuristiques. Dans le chapitre qui suit, on va essayer de présenter une classe de problèmes d'optimisation combinatoire qui sera le cadre d'étude de ce mémoire.

3

Les problèmes de type Sac-à-Dos

Introduction

Dans ce chapitre, nous nous intéressons à la résolution des problèmes combinatoires multiobjectif, c'est-à-dire les problèmes dont les variables sont discrètes. Il existe beaucoup de problèmes combinatoires réputés. Parmi eux le problème du sac-à-dos (ou knapsack problem) qui est un problème classique d'optimisation appartenant à la classe des problèmes NP-difficiles. On le retrouve sous de nombreuses formes, comme par exemple, loger des morceaux de musique dans une cassette de sorte à avoir la plus longue durée de musique.

Contents

Le problème est de remplir un sac-à-dos supportant un poids maximum C avec des objets ayant un poids w_j pour $j = 1, \ldots, n$, et un indice de satisfaction de telle sorte que la satisfaction totale (fonction objectif) $Z(x)$ soit maximale. La question qui se pose est de savoir quels objets doit-on mettre dans le sac ?

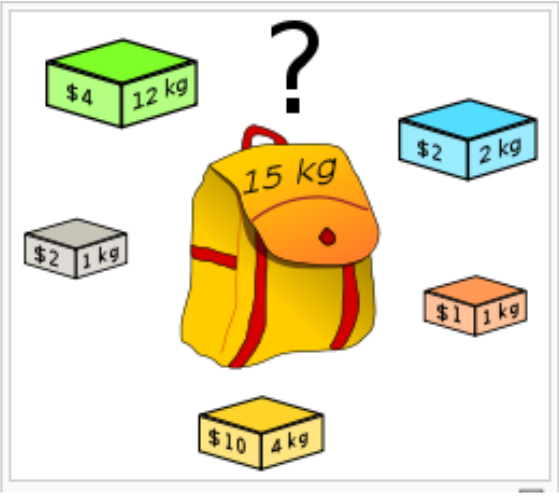


FIG. 3.1 – *Problème du sac-à-dos*

Ce problème intervient comme un sous problème dans plusieurs problèmes, par exemple, les problèmes de découpe (voir Glomre et Gomory [4]) lors de la génération d'un pivot du simplexe.

Sous le terme "sac-à-dos" sont regroupées différentes variantes qui, bien qu'elles semblent très proches, ne font pas du tout appel aux mêmes méthodes de résolution exactes ou approchées pour une variante donnée. Dans ce chapitre, nous rappelons les diverses formes de problèmes, a savoir Sac-à-dos monobjectif unidimensionnel, borné et non borné multidimensionnel, sac-à-dos multiobjectif et sac-à-dos multiobjectif multidimensionnel, ainsi les Différentes méthodes de résolutions pour ces derniers.

3.1 Problème de sac-à-dos (knapsack problem)

Pour une bonne illustration, nous adoptons l'exemple suivant :
Un randonneur doit décider de l'équipement qu'il va emporter pour sa prochaine expédition. Pour des raisons évidentes, il veut limiter le poids total des objets emportés et s'est fixé une borne supérieure de C kg à ne pas dépasser. Chacun des n objets qu'il peut emporter possède un poids w_i, (entier) $i = 1, \ldots, n$ ainsi qu'une utilité $c_i, i = 1, \ldots, n$, (réelle à priori), cette dernière étant d'autant plus grande que le randonneur juge l'objet intéressant.
Définissons pour chaque objet une variable de décision binaire

$$\begin{cases} x_i = 1, & \text{si l'objet } i \text{ est retenu ;} \\ x_i = 0, & \text{sinon.} \end{cases}$$

Il s'agit ici de la formulation en $0-1$ du problème de sac-à-dos. Il désire remplir le sac de façon à maximiser la somme des utilités des objets emportés, en respectant la contrainte de capacité.

Autrement dit, on désire résoudre le problème de programmation linéaire entière suivant :

$$x \in \arg\max_{j\in\{1,\dots,n\}} \Big\{ \sum_{1\le j\le n} c_j x_j, \text{sous la containte} \sum_{1\le j\le n} w_j x_j \le C \Big\}.$$

Le vecteur $x = (x_j, 0 \le j \le n) \in \{0,1\}^n$ est une solution du problème avec $x_j = 1$, si l'objet j est emporté dans le sac et $x_j = 0$ sinon.

Le problème consiste donc à choisir un sous-ensemble d'objets parmi la collection d'objets initialement qui maximise la fonction objectif : $Z(x) = \sum_{j=1}^{n} c_j x_j$, avec l'hypothèse suivante $\forall i \in \{1,\dots,n\},\ w_j \le C$ et $\sum_{j=1}^{n} w_j > C$.

Le vecteur binaire $(x_1,\dots,x_n)$ est appelé solution du problème du sac-à-dos, KP. On dit qu'une solution est réalisable, notée $\overline{x}$, si elle vérifie la contrainte de capacité, c'est-à-dire $\sum_{j=1}^{n} w_j \overline{x_j} \le C$. La solution est dite optimale, notée x^*, si elle est à la fois réalisable et si elle maximise la somme des valeurs profits des objets mis dans le sac. En d'autres termes, pour toute solution réalisable $\overline{x}$, on a :

$$\sum_{j=1}^{n} c_j \overline{x_j} \le \sum_{j=1}^{n} c_j x_j^*.$$

La formulation du problème du sac-à-dos unidimensionnel, KP, en variables binaires est la suivante :

$$\left|\begin{array}{lll} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} c_j x_j & \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \le C & \\ & x_j \in \{0,1\}, & \text{pour } j = 1,\dots,n. \end{array}\right.$$

Diverses approches aussi bien exactes qu'approchées ont été développées pour résoudre ce problème.

3.2 Sac-à-dos à variables bornées

Le problème défini dans la section précédente peut être généralisé en assignant pour chaque objet j une borne b_j à ne pas dépasser ($b_j \le C/w_j$), nous obtiendrons alors le problème de Sac-à-dos à variables bornées dont la formulation mathématique est la suivante :

$$\left|\begin{array}{lll} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} c_j x_j; & \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \le C; & \\ & 0 \le x_j \le b_j, & j = 1,\dots,n; \\ & x_j \ \text{ entier}, & j = 1,\dots,n. \end{array}\right.$$

Dans le cas où $b_j = +\infty$ le problème s'appelle Sac-à-dos à variable non bornées (unbounded knapsack problem). Et pour $b_j = c_j$, ce problème est intitulé "sub-set-sum problem" ou bien la somme des sous ensembles.

Un problème de Sac-à-dos à variables non bornées très particulier est consideré lors que, $c_j = 1,\quad j = 1,\dots,n$, et en imposant l'égalité pour la contrainte capacité $\sum_{j=1}^{n} w_j x_j = C$, ce

problème s'appelle "Change-making problem", cependant ce problème peut avoir une extension à variables non bornées (Unbounded Change-making problem).

3.3 Problème de la distribution équitable

Dans cette partie, nous présentons la modélisation du problème de la distribution équitable (Knapsack sharing Problem : KSP). Il s'agit d'un problème en max − min. Ce problème possède un bon nombre d'applications dans les domaines du commerce, du transport, des télécommunications, d'allocation des ressources, etc. Pour ce problème, la distribution équitable ne consiste pas à partager en parts égales les ressources mais plutôt à maximiser la plus petite part. Autrement dit privilégier le moins favorisé.

Prenons l'exemple de l'État qui a le souci de partager équitablement le budget C alloué aux différents ministères qui sont d'un nombre fixé à m. Chaque ministère possède ses différentes priorités pour la réalisation d'un certain nombre de projets. Allouée équitablement le budget aux différents ministères, ne revient pas à diviser le budget C en parts égales $\frac{C}{m}$, entre les différents départements ministériels. Mais comme chacun des projets d'un ministère $i, i = 1, \ldots, m$, dispose d'un coût de réalisation et d'un profit estimé, alors le problème de partage équitable revient à maximiser la plus petite valeur du profit global par ministère sans dépasser le budget prévu pour la réalisation de ces projets.

Plus précisément, nous considérons dans cette partie le problème KSP à variables binaires. Pour ce problème, à chaque objet j sont associés un profit v_j et un poids w_j. On dispose d'un ensemble $\mathcal{N}$ de n objets où $\mathcal{N}$ est composé de m classes disjointes d'objets. Si J_i désigne la i-ème classe d'objets $i, i = 1, \ldots, m$, alors $\forall p = 1, \ldots, m$, $\forall q = 1, \ldots, m$, pour $p \neq q$, $J_p \cap J_q = \phi$ et $\bigcup_{i=1}^{m} J_i = \mathcal{N}$.

L'objectif de ce problème est de déterminer le sous-ensemble d'objet que nous allons retenir dont la capacité a pour valeur C. Ce sous-ensemble est choisi de manière à maximiser la valeur de la fonction objectif qui réalise le minimum par rapport à l'ensemble de toutes les classes.
La formulation de ce problème est la suivante :

$$(KSP) \left| \begin{array}{ll} \text{Maximiser} & \min_{1 \leq i \leq m} \left\{ \sum_{j \in J_i} v_j x_j \right\}; \\ \text{s.c} & \sum_{j \in \mathcal{N}} w_j x_j \leq C; \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1, \ldots, n. \end{array} \right.$$

3.4 Problème de sac-à-dos multidimensionnel à choix multiple

Le problème de sac-à-dos généralisé à choix multiple ou problème de sac-à-dos multidimensionnel à choix multiple ou encore problème de sac-à-dos multicontraintes à choix multiple, est une version bien particulière du problème du KP. Il peut être considéré comme une variante qui généralise encore plus de deux problèmes généralisant à leur tour le problème de sac-à-dos : le problème $MDKP$ et $MCKP$. Ces deux généralisations ont bien été étudiées par le passé (voir Martello et Toth [43]) et autres, pour lesquels diverses approches exactes et heuristiques efficaces on été proposées pour les résoudre.

Nous rappelons que le problème du $MMKP$ est caractérisé par la donnée d'un vecteur capacité ou resources $C = (C^1, \ldots, C^m)$, et d'un ensemble $\mathcal{J} = \{J_1, \ldots, J_n\}$ d'objets répartis sur n classes disjointes telles que pour chaque couple (p, q), $p \neq q$, $p \leq n$ et $q \leq n$, nous avons $J_p \cap J_q = \phi$ et $\cup_{i=1}^{n} J_i = \mathcal{J}$, Chaque classe $i, i = 1, \ldots, n$, est de cardinalité r_i (nombre d'objets appartenant à la dite classe i).

Á chaque objet j de la classe i sont associés un profit v_{ij} et un vecteur poids $W_{ij} = (w_{ij}^1, \ldots, w_{ij}^m)$. Le but est d'attribuer au sac, exactement un et un seul objet par classe dans le but de maximiser la valeur totale du choix, sans que l'une ou l'autre des contraintes capacités ne soient à un moment ou un autre violée. Le $MMKP$ peut donc être formulé de la façon suivante :

$$(MMKP) \left| \begin{array}{ll} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} \sum_{j=1}^{r_i} v_{ij} x_{ij} \, ; \\ \text{s.c} & \sum_{i=1}^{n} \sum_{j=1}^{r_i} w_{ij}^k x_{ij} \leq C^k, \qquad k = 1, \ldots, m \, ; \\ & \sum_{j=1}^{r_i} x_{ij} = 1, \qquad (3.4) \, ; \\ & x_{ij} \in \{0, 1\}, \text{ pour } i = 1, \ldots, n, \quad j = 1, \ldots, r_i. \end{array} \right.$$

Sans perte de généralité et pour éviter les solutions triviales, nous formulons l'hypothèse suivante :

$$(H) : \sum_{i=1}^{n} \min_{1 \leq j \leq r_i} \left\{ w_{ij}^k \right\} \leq C^k \leq \sum_{i=1}^{n} \max_{1 \leq j \leq r_i} \left\{ w_{ij}^k \right\}, pour \quad k = 1, \ldots, m.$$

Nous rappelons que le problème du $MMKP$ est une généralisation des problèmes $MDKP$ et $MCKP$. En effet, si $m = 1$, alors le $MMKP$ devient $MCKP$ et si $n = 1$ et on supprime la contrainte du choix (3.4), alors le $MMKP$ devient un $MDKP$.

3.5 Sac-à-dos multiple

Le problème de Sac-à-dos multiple (MKP) consiste à répartir un ensemble d'objets dans plusieurs sacs à dos de capacités différentes. La valeur d'un objet dépend maintenant du sac dans lequel il est placé. Par exemple : on peut considérer qu'un euro a plus de valeur sur un compte d'épargne que sur un compte courant.

3.6 Sac-à-dos multidimensionnel multiobjectif

Le problème du Sac-à-dos multidimensionnel multiobjectif (MOKP) est l'un des problèmes d'optimisation combinatoire $\mathbb{NP}$-complets fondamentaux les plus étudiés dans la communauté multiobjectif. Nous avons donc choisi ce problème pour nos travaux et nous allons maintenant le présenter.

Soit un ensemble d'articles (ou objets), à chacun d'entre eux sont associés des valeurs de profit ainsi que des poids. Le problème du Sac-à-dos multidimensionnel multiobjectif en $0-1$ ($MOKP/01$) consiste à sélectionner un sous-ensemble d'objets maximisant une fonction multiobjectif, exprimée en fonction des valeurs de profit, tout en respectant un ensemble de contraintes de type Sac-à-dos. Le problème du $MOKP/01$ peut être défini plus formellement de la manière suivante :

$$MOKP01 \left| \begin{array}{lll} \max & Z^j(x) = \sum_{i=1}^{n} c_i^j x_i, & j = 1, \ldots, m\,; \\ t.q. & \sum_{i=1}^{n} w_i^l x_i \leq b_l, & l = 1, \ldots, q\,; \\ & x_i \in \{0,1\}, & i = 1, \ldots, n. \end{array} \right.$$

où n est le nombre d'objets, x_i une variable de décision, m le nombre d'objectifs, z^j la $j^{\text{ème}}$ composante de la fonction multiobjectif Z et q le nombre de contraintes Sac-à-dos du problème.

Le **MOKP** peut être utilisé pour modéliser de nombreux problèmes réels comme la répartition de budgets et l'allocation de ressources. Plusieurs algorithmes heuristiques ont été développés pour résoudre le **MOKP**.

Le problème de sac-à-dos a été étudié pour la première fois à la fin des années 50. Dès cette date un grand nombre d'algorithmes et de techniques a été proposé. Peu d'algorithmes exacts pour le problème sac-à-dos monobjectif existe dans la littérature[4]. Le plus célèbre algorithme a été proposé par Martello et Toth [43] et c'est une procédure **Branch and Bound**.

Le problème sac-à-dos monobjectif peut être aussi résolu d'une manière heuristique pour s'approcher de la solution optimale. Une heuristique typique pour ce problème est l'algorithme "Glouton" qui peut résoudre soit des problèmes simples ou complexes en un temps polynômial.

Des méthodes de voisinage comme le recuit simulé [Serafini, 1992 ; Ulungu et al., 1999] et la recherche Tabou [Gandibleux et al., 1997], des méthodes évolutionnaires : "vector evaluated genetic algorithm" (VEGA) [Schafer, 1984], "nondominated sorting genetic algorithm" (NSGA) [Srinivas and Deb, 1994], "strength Pareto evolutionary algorithm" (SPEA) [Zitzler and Thiele, 1998a], et "memetic Pareto archived evolution strategy algorithm" (M-PAES) [Corne and Knowles, 2000]. Très récemment, des algorithmes hybrides combinant la recherche génétique et la recherche locale (limitée à des méthodes de descente à notre connaissance) tels que "multiple objectif genetic local search" (MOGLS) [Jaszkiewicz, 2002], ainsi que des approches hybrides parallèles, [Jozefowiez et al., 2002] ont été proposées [4].

Une approche hybride GTS^{MOKP} est présentée dans la thèse du V. Barichard [4], c'est une combinaison d'un algorithme génétique et d'une méthode de recherche Tabou pour le $MOKP$. Elle est inspirée d'un algorithme hybride pour le problème de coloriage de graphes [Galinier and Hao, 1999], et de l'algorithme MOGLS [Jaszkiewicz, 2000b]. Une procédure du branch and bound pour le cas bi-critère a été proposée pour ce problème dans l'ouvrage de M. Ehrgott [22].

3.7 Problème de sac-à-dos et branch and bound

Nous présentons ici un algorithme branch and bound multiobjectif qui a été donné dans l'ouvrage de M. Ehrgott [22], il s'agit d'un algorithme conçu pour le problème sac-à-dos bi-critère. Le problème sac-à-dos bi-critère est un programme en nombres entiers binaires qui se formule comme suit :

$$\left|\begin{array}{ll} \max f_1(x) = \sum_{i=1}^{n} c_i^1 x_i; & \\ \max f_2(x) = \sum_{i=1}^{n} c_i^2 x_i; & \\ t.q. \qquad \sum_{i=1}^{n} w_i \cdot x_i \leq W; & \\ \qquad x_i \in \{0,1\}, & i = 1,\ldots,n. \end{array}\right. \tag{3.1}$$

Le problème est évidemment qu'il appartient à la classe des problèmes NP-dur, en tant qu'une généralisation d'un problème monobjectif NP-dur.
Nous présenterons ci-après, le fonctionnement de l'algorithme de Branch and bound. Pour éviter les solutions insignifiantes, et avoir un problème significatif, nous faisons quelques hypothèses de base sur les paramètres du problème de sac-à-dos. Nous supposons que toutes les valeurs de c_i^k, tous les poids w_i aussi bien que la capacité W, sont non négatifs. En outre, aucun poids ne dépasse la capacité, c-à-d. $w_i \leq W$, pour tout $i = 1,\ldots,n$, et la somme des poids de tous les articles est supérieur à W, $\sum_{i=1}^{n} w_i > W$.

Les valeurs des rapport c_i^k/w_i, sont d'une importance essentielle pour la resolution des problèmes de sac-à-dos. Dans le cas du problème sac-à-dos linéaire monobjectif (où $x_i \in \{0,1\}$ est remplacé par $0 \leq x_i \leq 1$), ces rapports sont utilisés pour trouver facilement une solution optimale.

$$\left|\begin{array}{lll} \max & \sum_{i=1}^{n} c_i.x_i\,; & \\ s.c & \sum_{i=1}^{n} w_i.x_i \leq W\,; & \\ & x_i \geq 0, \quad i = 1,\ldots,n & \\ x_i \leq 1, & i = 1,\ldots,n. & \end{array}\right. \tag{3.2}$$

Supposons que les articles $1,\ldots,n$ sont ordonnés de sorte que

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \ldots \geq \frac{c_n}{w_n}. \tag{3.3}$$

Soit $i^* = \{i, \sum_{j=1}^{i} w_j > W\}$ le plus petit indice tel que la somme des objets de 1 à i dépasse la capacité totale. L'indice i s'appelle indice critique. La résolution du problème continu de sac-à-dos est simplement donnée en prenant tous les points de 1 à $i^* - 1$ et une fraction de l'indice critique, $x_i = 1$ pour $i = 1,\ldots,i^* - 1$ et

$$x_{i^*} = \frac{W - \sum_{i=1}^{i^*-1} w_i}{w_{j^*}}.$$

De bons algorithmes pour le problème monobjectif utilisent ce fait et se concentrent sur l'optimisation des indices autour de i^*, voir par exemple Martello et Toth (1990) ; Pisinger (1997) ; Kellerer et autres (2004). Des idées de tels algorithmes ont été adaptées au cas bi-critère par Ulungu et Teghem (1997).

Les deux critères induisent deux ordres de valeur différents des rapports de poids. Soitent $\mathcal{O}_k$ l'ordre (3.3) selon $c_i^k/w_i, k = 1, 2$, r_i^k le rang ou la position du point i dans l'ordre $\mathcal{O}_k$ et $\mathcal{O}$ l'ordre selon les valeurs croissantes de $(r_i^1 + r_i^2)/2$, le rang moyen d'un indice.

La méthode "branch and bound" crée les solutions partielles en assignant des zéros et des uns aux sous-ensembles de variables notées B_0 et B_1, respectivement. Ces solutions partielles constituent des nœuds de l'arboresence de recherche. Les variables non assignées à "Zéro" ou "un" s'appellent les variables libres pour une solution partielle et définissent l'ensemble $\mathcal{F} \subseteq \{1, \ldots, n\}$ tel que $\{1, \ldots, n\} = B_1 \cup B_0 \cup \mathcal{F}$.

Une solution est constituée en assignant aux variables libres une valeur qui s'appelle l'accomplissement d'une solution partielle. Les variables d'une solution partielle seront assignées à une valeur selon l'ordre $\mathcal{O}$. Il est commode de numéroter les indices dans cet ordre de sorte que nous ayons pour un certain l.

$$B_1 \cup B_0 = \{1, \ldots, l-1\}; \mathcal{F} = \{l, \ldots, n\}.$$

En outre nous appelons r^k l'indice de la première variable dans $\mathcal{F}$ selon l'ordre $\mathcal{O}_k$, pour $k = 1, 2$.

Des bornes évaluées par vecteur seront utilisées pour élaguer une branche de l'arborescence, quand aucun accomplissement de la solution partielle (B_1, B_0) ne peut contenir une solution efficace. Une borne inférieure $(\underline{z_1}, \underline{z_2})$ à une solution partielle est simplement donnée par la somme des valeurs des variables à qui a déjà été assignée la valeur 1.

$$(\underline{z_1}, \underline{z_2}) = (\sum_{i \in B_1} C_i^1, \sum_{i \in B_1} C_i^2). \tag{3.4}$$

Pour le calcul des bornes supérieures nous définissons :

$$\overline{W} = W - \sum_{i \in B_1} w_i \geq 0.$$

qui sera la capacité libre restante du sac-à-dos après fixation des variables dans B_1. En outre, on note :

$$S_k = \max \{ l_k \in \mathcal{F}, \sum_{j_k = r_k}^{l_k} w_{jk} < \overline{W} \}.$$

Le dernier indice qui peut être choisi pour être ajouté à une solution partielle selon $\mathcal{O}_k$. Ainsi, $S_k + 1$ est en fait l'indice critique dans l'ordre $\mathcal{O}_k$, en tenant compte des variables déjà fixes dans B_0 et B_1.
La borne supérieure pour chaque valeur objectif à une solution partielle est calculée selon la règle de Martello et de Toth (1990).

$$\overline{z_k} = \underline{z_k} + \sum_{j_k = r_k}^{s_k} c_{j_k}^k + \max \{ [\overline{W}_k \frac{c_{s_k+2}^k}{w_{s_k+2}}], [c_{s_k+1}^k - (w_{s_k+1} - \overline{W}_k) \frac{c_{s_k}^k}{w_{s_k}}] \}. \tag{3.5}$$

où $\overline{W}_k = \overline{W} - \sum_{j_k = r_k}^{s_k} w_{j_k}$. Cette borne $\overline{z_k}$ peut être justifiée par le fait que x_{s_k+1} ne peut pas

admettre des valeurs fractionaires. Par conséquent, elle doit être mise à zéro ou à un. Elle est calculée à partir de toutes les valeur des variables déjà assignées dans $(\underline{z}_k)$, plus les valeur des indices qui peuvent être ajoutés au sac selon l'ordre $\mathcal{O}_k$, plus le maximum terme 3.5. Le premier terme dans le maximum (3.5) se justifie, par fait de mettre $x_{s_k+1} = 0$ et de remplir la capacité restante par x_{s_k+2}, tandis que le second terme signifie la mise de $x_{s_k+1} = 1$ et enlever une partie de x_{s_k} pour satisfaire la capacité totale du sac-à-dos.

Une solution potentiellement efficace est obtenu à partir d'une solution partielle et en assignant des zéros et des uns aux variables libres. Ce problème lié à la solution partielle (B_1, B_0) est encore un problème Sac-à-dos bi-critère :

$$\left|\begin{array}{ll} \max \sum_{i\in\mathcal{F}} c_i^1 \cdot x_i + \sum_{i\in B_1} c_i^1; & \\ \max \sum_{i\in\mathcal{F}} c_i^2 \cdot x_i + \sum_{i\in B_1} c_i^2; & \\ s.c \sum_{i\in\mathcal{F}} w_i \cdot x_i \leq \overline{W}; & \\ \quad x_i \in \{0.1\}, & i = 1, \ldots, n. \end{array}\right. \tag{3.6}$$

Nous avons maintenant toutes les données nécessaires pour formuler l'algorithme. L'algorithme poursuit initialement une stratégie de profondeur. C'est-à-dire, le mouvement en bas d'une branche de l'arborescence de recherche, et donc faire fixer plus de variables dans B_1 ou B_0, et en préfère d'étudier les solutions partielles avec peu de variables fixes. Des successeurs d'un nœud dans une arborescence (embranchement) sont distingués par différentes tailles de B_1 et de B_0. En fait, l'arborescence peut être dessiné de sorte que les ensembles B_1 de successeurs d'un nœud devient de plus en plus petits, pendant que des variables sont déplacées de B_1 à B_0, (voir le schéma 3.2), l'idée est de fixer d'abord beaucoup de variables selon la valeur des rapports de poids. Une bonne solution faisable est obtenue rapidement, de sorte que beaucoup de branches de l'arbre puissent être élaguées tôt.

Pendant la resolution, on garde une liste $\mathcal{L}$ de points potentiellement non dominés identifiés jusqu'ici (notons qu'en raison de la maximisation y^1 domine y^2 signifie $y^1 > y^2$), et une liste des Nœuds $\mathcal{N}$ toujours à traiter.

La liste $\mathcal{N}$ est maintenue en dernière priorité dans la première file d'attente. Des nœuds sont coupés si les bornes montrent qu'elles peuvent seulement mener aux solutions dominées, si elles ont été complètement étudiées (elles représentent une solution complète), ou si aucun autre Nœud successeur ne peut être construit.

Quand un Nœud est coupé, l'algorithme fait marche arrière et crée un nouveau nœud en déplaçant le dernier article de B_1 à B_0, en enlevant tous les articles après ce nouvel article de B_0. Si, cependant, le dernier article dans B_1 est n alors l'algorithme choisit le plus petit v tel que tous les articles $\{v, \ldots, n\}$ étaient dans B_1, les enlève tous, et définit B_0 pour tous les éléments précédents de B_0 jusqu'à $v - 1$ et pour inclure le v.

Quand un nœud n'est pas coupé, l'algorithme crée un nouveau nœud successeur. Ceci peut

encore être fait de deux manières. Si B_1 permet l'addition du premier article l de $\mathcal{F}$, autant d'articles aussi possibles sont inclus dans B_1, selon l'ordre $\mathcal{O}$, c-à-d. qu'ils apparaissent dans $\mathcal{F}$. Si, d'une autre part, la capacité restante $\overline{W}$ ne permet pas à l'article l d'être ajouté à B_1, le premier article possible r de $\mathcal{F}$, qui peut être ajouté à B_1 est cherché et l'article r est ajouté à B_1. Naturellement tous les points $i, \ldots, r-1$ doivent être ajoutés à B_0. Puisque le Nœud courant n'est pas coupé, alors un tel r doit exister.

Nous illustrons l'algorithme par un exemple qui a été utilisé dans (Ulungu et Teghem (1997)). Les itérations de l'arborescence illustrent clairement le fonctionnement de l'algorithme.

Exemple 3 : Nous considérons le problème :

$$\left|\begin{array}{lll} \max & & 11x_1 + 5x_2 + 7x_3 + 13x_4 + 3x_5; \\ \max & & 9x_1 + 2x_2 + 16x_3 + 5x_4 + 4x_5; \\ s.c & & 4x_1 + 2x_2 + 8x_3 + 7x_4 + 5x_5 \leq 16; \\ & x_i \in \{0.1\}, & i = 1, \ldots, n. \end{array}\right. \tag{3.7}$$

les ordres sont :

$$\mathcal{O}_1 = \{x_1, x_2, x_4, x_3, x_5\}.$$
$$\mathcal{O}_2 = \{x_1, x_3, x_2, x_5, x_4\}.$$
$$\mathcal{O} = \{x_1, x_2, x_3, x_4, x_5\}.$$

Le premier Nœud N_0 est créé avec $B_1 = \phi$, $B_0 = \phi$, $\mathcal{F} = \{1, 2, 3, 4, 5\}$ et les bornes sont initialisées à $\underline{z} = \binom{0}{0}$, $\overline{z} = \binom{\infty}{\infty}$ et $\mathcal{L} = \phi$, $\mathcal{N} = \{N_0\}$

-1- Le Nœud N_0 est sélectionné et un nouveau Nœud N_1 est créé avec $B_1 = \{1, 2, 3\}$, $B_0 = \phi$, $\mathcal{F} = \{4, 5\}$ alors $\mathcal{N} = \{N_0, N_1\}$.

-2- Le Nœud N_1 est sélectionné, on calcule $\overline{W} = 2$, $\underline{z} = \overline{z} = \binom{23}{27}$, et on l'ajoute à $\mathcal{L}$; $\mathcal{L} = \{\binom{23}{27}\}$.
Tant que $\{j \in \mathcal{F}, w_j < \overline{W}\} = \phi$ le Nœud N_1 est coupé et on crée le Nœud N_2 tant que $t = 3$ on aura $B_1 = \{1, 2\}$, $B_0 = \{3\}$, $\mathcal{F} = \{4, 5\}$, $\mathcal{N} = \{N_0, N_2\}$.

-3- Le Nœud N_2 est sélectionné, on calcule $\overline{W} = 10$, $\underline{z} = \binom{16}{11}$
On choisit $\{j \in \mathcal{F}, w_j < \overline{W}\} = \{4, 5\} \neq \phi$
La borne supérieure est $\overline{z} = \binom{29}{18}$.
Le Nœud N_3 est créé avec $s = 4$ et $B_1 = \{1, 2, 4\}$, $B_0 = \{3\}$, $\mathcal{F} = \{5\}$, $\mathcal{N} = \{N_0, N_2, N_3\}$

-4- Le Nœud N_3 est sélectionné, on calcule $\overline{W} = 3$, $\underline{z} = \overline{z} = \binom{29}{16}$, et on l'ajoute à $\mathcal{L}$; $\mathcal{L} = \{\binom{23}{27}, \binom{29}{16}\}$.
Tant que $\{j \in \mathcal{F}, w_j < \overline{W}\} = \phi$ le Nœud N_1 est coupé.
Le Nœud N_4 est créé avec $t = 4$, $B_1 = \{1, 2\}$, $B_0 = \{3, 4\}$, $\mathcal{F} = \{5\}$, $\mathcal{N} = \{N_0, N_2 N_4\}$.

Algorithme 10 Branch and bound pour le knapsack problem

Entrées: Valeurs du c_i^1 et c_i^2, poids de w_i pour $i = 1, \ldots, n$ et capacité W.

Initialisation : Créer le Nœud racine N_0 comme suit.
Placer $B_1 := \phi, B_0 := \phi, \mathcal{F} := \{1, \ldots, n\}, \underline{z} := \binom{0}{0}, \overline{z} := \binom{\infty}{\infty}$, $\mathcal{L} := \phi, \mathcal{N} := \{N_0\}$.
tantque $\mathcal{N} \neq \phi$ **faire**
  Choisir le Nœud suivant $N \in \mathcal{N}$, Calculer $\overline{W}$ et $\underline{z}$.
  Ajouter $\underline{z}$ à $\mathcal{L}$ s'il n'est pas dominé, Calculer $\overline{z}$.
  si $\{i \in \mathcal{F} : w_i \leq \overline{W}\} = \phi$ **ou** $\overline{z}$ est dominé par un certain $y \in \mathcal{L}$ **alors**
    couper le Nœud N. $\mathcal{N} := \mathcal{N}/\{N\}$.
    Créer un nouveau Nœud N' comme suit.
    Soit $t := \max\{i : i \in B_1\}$.
    si $t < n$ **alors**
      $B_1 := B_1/\{t\}, B_0 := (B_0 \cap \{1, \ldots, t-1\}) \cup \{t\}, \mathcal{F} := \{t+1, \ldots, n\}$
    finsi
    si $t = n$ **alors**
      soit $u = \min\{j : \{j, j+1, \ldots, t-1, t\} \subset B_1\}$, soit $v = \max\{j : j \in B_1/\{u, \ldots, t\}\}$.
      $B_1 := B_1/\{v, u, u+1, \ldots, t-1, t\}, B_0 := (B_0 \cap \{1, \ldots, v-1\}) \cup \{v\}$. $\mathcal{F} := \{v+1, \ldots, n\}$
    finsi.
    $\mathcal{N} := \mathcal{N} \cup \{N\}$
    si l'ensemble B_1 de N' est plus petit que B_1 des Nœuds prédécesseur de N, qui ne sont pas des prédécesseurs de N' **alors**
      coupe ces Nœuds.
    finsi
    si aucun nouveau Nœud ne peut être créé ($B_1 = \phi$) **alors**
      ARRÊTER
    finsi
  sinon
    Créer un nouveau Nœud N' comme suit.
    Soit $s := \max\{t \in \mathcal{F} : \sum_{j=l}^{t} w_j < \overline{W}\}$ selon l'ordre $\mathcal{O}$.
    si $w_l > \overline{W}$ **alors**
      soit $s := l - 1$.
    finsi
    si $s \geq l$ **alors**
      $B_1 := B_1 \cup \{l, \ldots, s\}, B_0 := B_0, \mathcal{F} := \mathcal{F}/\{l, \ldots, s\}$.
    finsi
    si $s = i - 1$ **alors**
      Soit $r := \min\{j : j \in \mathcal{F}, w_j < \overline{W}\}$ selon l'ordre $\mathcal{O}$.
      $B_1 := B_1 \cup \{r\}, B_0 := B_0 \cup \{l, \ldots, r-1\}, \mathcal{F} := \mathcal{F}/\{l, \ldots, r\}$
    finsi
    $\mathcal{N} := \mathcal{N} \cup N$
  finsi
fin tantque.

Sorties: Toutes les solutions efficaces.

-5- On sélectionne le Nœud N_4 et on calcule $\overline{W} = 10$, $\underline{z} = \binom{16}{11}$ et $\overline{z} = \binom{16+3}{11+4} = \binom{19}{15}$.
$\overline{z}$ est dominante, alors N_4 est coupé.
Le Nœud N_5 est créé avec $t = 2$ et $B_1 = \{1\}$, $B_0 = \{2\}$, $\mathcal{F} = \{3, 4, 5\}$
B_1 dans N_5 est plus petit que B_1 dans N_2 alors N_2 est coupé $\mathcal{N} = \{N_0, N_5\}$

-6- Le Nœud N_5 est sélectionné, en ce Nœud $\overline{W} = 12$, $\underline{z} = \binom{11}{9}$
tant que $\{j \in \mathcal{F}, w_j < \overline{W}\} \neq \phi$ $\overline{z} = \binom{27}{27}$ est non dominante.
Le Nœud N_6 est créé avec $s = 3$ et $B_1 = \{1, 3\}$, $B_0 = \{2\}$, $\mathcal{F} = \{4, 5\}$ $\mathcal{N} = \{N_0, N_5, N_6\}$

-7- On sélectionne N_6, $\overline{W} = 4$, $\underline{z} = \overline{z} = \binom{18}{25}$ puisque $\{j \in \mathcal{F}, w_j < \overline{W}\} = \phi$, alors N_6 est coupé.
On crée N_7 avec $t = 3$, $B_1 = \{1\}$, $B_0 = \{2, 3\}$, $\mathcal{F} = \{4, 5\}$, $\mathcal{N} = \{N_0, N_5, N_7\}$.

-8- On sélectionne N_7, $\overline{W} = 12$, $\underline{z} = \binom{11}{9}$, $\{j \in \mathcal{F}, w_j < \overline{W}\} \neq \phi$ borne supérieure $\overline{z} = \binom{27}{18}$ est non dominante.
On crée N_8 $s = 5$, $B_1 = \{1, 4, 5\}$, $B_0 = \{2, 3\}$, $\mathcal{F} = \phi$ $\mathcal{N} = \{N_0, N_5 N_7, N_8\}$.

-9- Le Nœud N_8 est sélectionné, $\overline{W} = 0$, $\underline{z} = \overline{z} = \binom{27}{18}$ est ajouté à $\mathcal{L}$ pour obtenir $\mathcal{L} = \{\binom{23}{27}, \binom{29}{16}, \binom{27}{18}\}$ tant que $\{j \in \mathcal{F}, w_j < \overline{W}\} = \phi$, N_8 est coupé.
Le Nœud N_9 est créé avec $t = 5$, $u = 4$, $v = 1$ et $B_1 = \phi$, $B_0 = \{1\}$, $\mathcal{F} = \{2, 3, 4, 5\}$
B_1 dans N_9 est plus petit que B_1 dans N_7 et N_5, N_7 sont coupés.
$\mathcal{N} = \{N_0, N_9\}$

-10- On sélectionne N_9, $\overline{W} = 16$, pour calculer $\overline{z}$ on utilise $s_1 = 4$, $s_2 = 5$ alors

$$\overline{z} = \begin{pmatrix} 0 + 5 + 13 + \max\{[7\frac{3}{5}], [7 - (8 - 7)\frac{13}{7}]\} \\ 0 + 16 + 2 + 4 + \max\{[0], [5 - (7 - 1)\frac{4}{5}]\} \end{pmatrix} = \begin{pmatrix} 23 \\ 22 \end{pmatrix}$$

Le processus de résolution est illustré par la figure 3.2 où chaque Nœud contient son numéro et les ensembles B_1 et B_0, $\mathcal{L}$ correspondant.

Il y a trois solutions efficaces x^1 avec $x_1 = x_2 = x_3 = 1$, x^2 avec $x_1 = x_2 = x_4 = 1$, x^3 avec $x_1 = x_4 = x_5 = 1$.

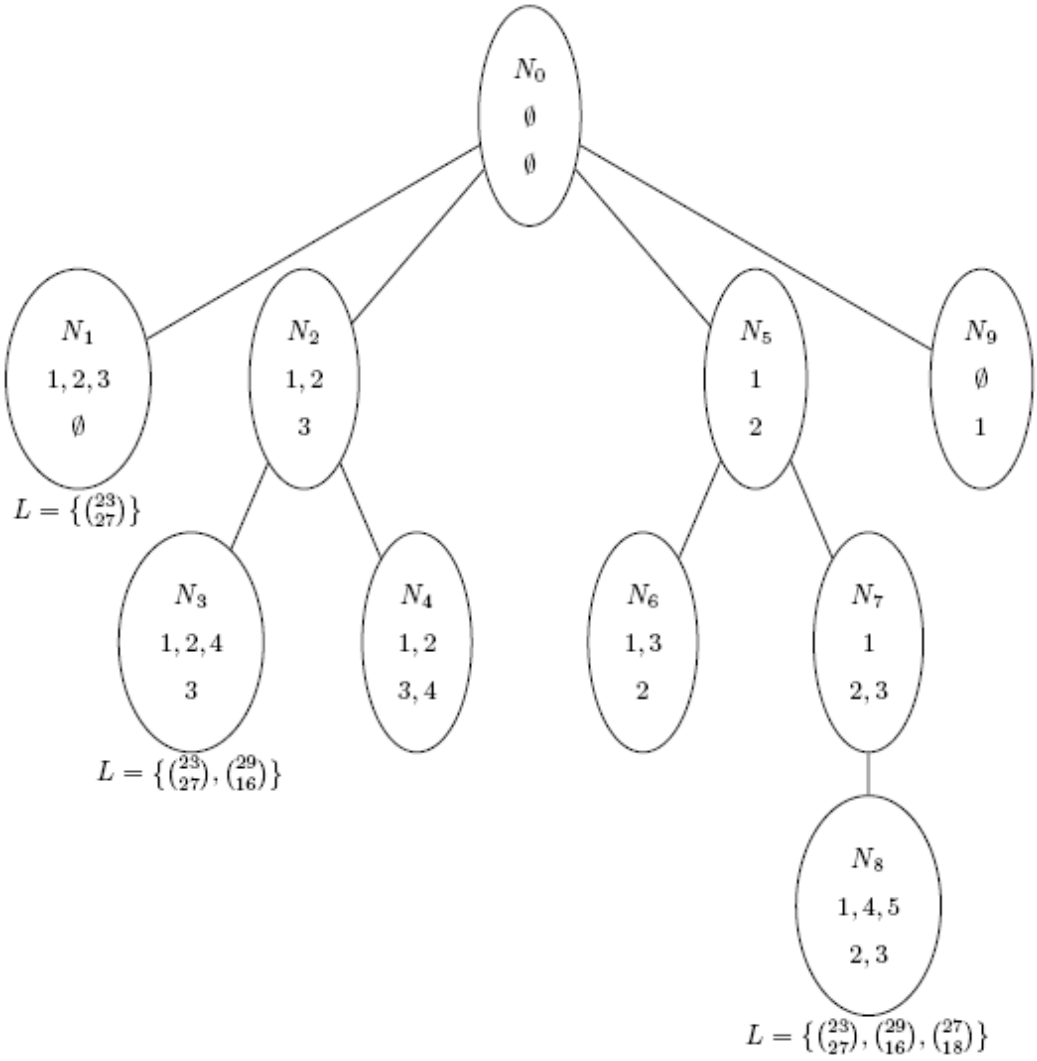


FIG. 3.2 – *Arbre de branch and bound pour l'exemple 3[22].*

3.8 Branch and bound pour le sac-à-dos multiobjectif

Nous essayons ici de modifier l'algorithme branch and bound bi-objectif qui a été donné dans l'ouvrage Ehrgott [22] au cas multiobjectif c'est-à-dire, plus de deux objectifs. Le problème Sac-à-dos multiobjectif est le programme en nombres entiers binaires suivant :

$$MOKP01 \left| \begin{array}{lll} \max & Z^j(x) = \sum_{i=1}^{n} c_i^j x_i, & j = 1, \ldots, m\,; \\ t.q. & \sum_{i=1}^{n} w_i x_i \leq b, & ; \\ & x_i \in \{0,1\}, & i = 1, \ldots, n. \end{array} \right.$$

La démarche est la même que celle utilisée par l'algorithme précédent, avec quelques adap-

tations au niveau des objectifs. Nous supposons que toutes les valeurs de c_i^k, tous les poids w_i ainsi que la capacité W sont non négatifs. En outre, aucun poids ne dépasse la capacité, c-à-d. $w_i \leq W$ pour tous $i = 1, \ldots, n$, et la somme des poids de tous les articles est plus grande que W, $\sum_{i=1}^{n} w_i > W$.

Nous introduisons un nombre d'ordres de rapports de poids égal au nombre de critères ; $\mathcal{O}_k$ selon $c_i^k/w_i, k = 1, \ldots m$. Soit r_i^k le rang ou la position du point i dans l'ordre $\mathcal{O}_k$ et soit $\mathcal{O}$ l'ordre selon les valeurs croissantes de $\frac{1}{m}\sum_{k=1}^{m} r_i^k$, le rang moyen d'un indice.

Enfin pour l'évaluation, nous utilisons des vecteurs de dimension m au lieu de dimension 2. Pour les bornes nous modifions les précédentes comme suit :

$$(\underline{z}_1, \ldots, \underline{z}_m) = (\sum_{i \in B_1} C_i^1, \ldots, \sum_{i \in B_1} C_i^m).$$

Pour le calcul des bornes supérieures nous définissons

$$\overline{W} = W - \sum_{i \in B_1} w_i \geq 0.$$

Conclusion

Dans ce chapitre, nous avons présenté une étude sur le problème de Sac-à-dos, sa formulation et ses différentes variantes telles que sac-à-doc monoobjectif, le problème de la distribution équitable, le problème de sac-à-dos généralisé aux choix multiples et le problème de sac-à-dos multidimensionnel multiobjectif en 0-1 (MOKP). Nous avons présenté une procedure branch and bound pour la résolution du problème dans le cas biobjectif, et nous avons tenté d'adapter ce dernier au cas de plus deux objectifs. Dans le chapitre qui suit nous allons hybrider ce dernier avec une méthode de recherche locale (Tabou Search).

4

Un algorithme hybride pour le Knapsack

Introduction

Afin d'exploiter les avantages des différentes approches et de surpasser leurs inconvénients, il est utile de combiner différents types de méthodes pour répondre au mieux aux attentes des décideurs qui se résument dans un large choix de solutions efficaces. Dans ce qui suit, nous présentons une approche hybride de résolution pour le Sac-à-dos multiobjectif uni-dimensionnel en variables 0/1. Cette approche exploite les différents avantages de la méthode Branch & Bound et celles de la Recherche Tabou.

Contents

Présenté par : Larbi ASLI *Sous la direction de : Méziane AIDER, Professeur à l'USTHB*

4.1 Hybridation de Branch & Bound et Tabou search

L'hybridation des métaheuristiques avec le branch & bound est une pratique courante. Cette dernière augmente l'efficacité du procédé de résolution, ainsi que la qualité et le nombre de solutions. Dans notre cas, cette hybridation est une composition de deux algorithmes, l'algorithme Branch & Bound multiobjectif dédié aux problèmes de type de sac-à-dos multiobjectif uni-dimensionnel en 0-1, donnée dans l'ouvrage de M. Ehrgott [22], et une méthode de recherche locale, qu'est une métaheuristique bien connue (Tabou Search). Dans la suite, nous allons définir quelques concepts liés à la méthode Tabou puis nous présentons l'algorithme ainsi obtenu.

4.2 Algorithmes Tabou

Pour que la Recherche Tabou présentée auparavant, soit performante, elle doit établir un bon compromis entre exploration et exploitation durant la recherche. Ainsi, l'exploration et l'exploitation sont respectivement assurées par des opérateurs de diversification et d'intensification. Nous mettons en avant, dans les sections suivantes, l'importance de la diversification dans le cadre des problèmes d'optimisation multiobjectif.

4.3 Caractéristiques des algorithmes Tabou

Les algorithmes tabou possèdent des caractéristiques communes que nous allons présenter maintenant.

4.3.1 Espace de recherche

Une configuration est donnée par un vecteur binaire $x = (x_1, x_2, \ldots, x_n)$ vérifiant toutes les contraintes de Sac-à-dos, c.à.d, $\forall l \in \{1, \ldots, q\}$, $\sum_{i=1}^{n} w_i^l \cdot x_i \leq b_l$. L'espace de recherche S est alors composé de tous ces vecteurs binaires, c.à.d, $S = \{x \in \{0,1\}^n / \sum_{i=1}^{n} w_i^l \cdot x_i \leq b_l, \ \ l \in \{1, \ldots, q\}\}$.

4.3.2 Voisinage

Le voisinage $\mathcal{N}$ de notre problème est défini de la manière suivante : soient $x, x^{'} \in S$, x et $x^{'}$ sont voisins ($x^{'} \in \mathcal{N}(x)$) s'ils diffèrent par la valeur d'une seule variable. Plus formellement, $\mathcal{N}(x) = \{x^{'} \in S \ \text{distanceHamming}(x, x^{'}) = 1\}$. Il en résulte qu'à partir d'une configuration x, il est possible d'obtenir une configuration voisine $x^{'}$ en ajoutant ou en enlevant un objet ($x_i : 0 \rightarrow 1 \quad \text{ou} \quad 1 \rightarrow 0$) de x tel que $x^{'}$ reste réalisable. Le mouvement de x vers $x^{'}$ est alors caractérisé par l'entier i représentant l'indice de la variable x_i qui a été changé. C'est cet indice qui sera considéré comme l'attribut du mouvement.

Dans notre cas, ce voisinage est défini d'une manière adapté au problème de sac-à-dos, de telle sorte que cette fonction cherche à transformer les variables égales à zéro, qui ne sont pas encore affectés (les variables libres $\mathcal{F}$) qu'on a déjà définies auparavant, et de temps en temps faire une exception en acceptant des variables déjà fixées à zéro (B_0) avec une probabilité p.

4.3.3 Evaluation du voisinage

L'évaluation du voisinage est basée sur l'approche pondérée de Tchebychef [Miettinen, 1999 ; Ehrgott and Gandibleux, 2000 ; Deb, 2001][1] . Cette approche peut se décrire formellement ainsi : soient $x = (x_1, x_2, \ldots, x_n) \in S$ et $x^{'} = (x_1^{'}, x_2^{'}, \ldots, x_n^{'}) \in \mathcal{N}(x)$, l'évaluation de $x^{'}$ est définie par la fonction suivante :

$$eval(x, x^{'}) = \min_j \lambda_j (z^j(x^{'}) - z^j(x))$$

où

- $z^j(x^{'}) = \sum_{i=1}^{n} c_i^j \times x^{'}$, est la valeur du $j^{\text{ème}}$ objectif de $x^{'}$;
- λ_j, est la $j^{\text{ème}}$ composante du vecteur $\lambda \in [0;1]^m$, où $(\sum_{j=1}^{m} \lambda_j = 1)$. Le vecteur λ correspond à une direction dans l'espace des objectifs, et permet d'orienter la recherche dans une autre direction, donc de la diversifier.

4.3.4 Choix du meilleur voisin

Soit x la configuration courante de l'algorithme. La configuration voisine $x^{'}$ devant remplacer x est déterminée grâce à l'équation suivante : $eval(x, x^{'}) = \max_{y \in \mathcal{N}(x)} eval(x, y)$. Si plusieurs configurations vérifient l'équation, l'algorithme en choisit une aléatoirement.

4.3.5 Gestion de la liste Tabou

Chaque fois qu'un mouvement est appliqué pour aller de x à $x^{'}$, l'indice de la variable changée est enregistré dans la liste Tabou. Ainsi, le mouvement inverse (correspondant au retour à la configuration de départ) est interdit pour les k prochaines itérations de l'algorithme. Pour implementer la liste Tabou, nous utilisons un tableau T de n éléments. Chaque fois qu'un indice est ajouté à la liste, la valeur du nombre courant d'itérations augmenté de la valeur Tabou k est placée dans la case de T correspondant à l'élément ajouté. Á chaque instant, il est facile de vérifier si un mouvement donné est marqué comme Tabou ou non, simplement en comparant le nombre courant d'itérations avec celui enregistré dans le tableau T.

4.3.6 Critère d'aspiration

Le mécanisme précédent est suffisant pour empêcher l'algorithme de rester bloqué dans des cycles courts. Cependant, un tel mécanisme peut interdire certaines configurations qui n'ont pas été encore visitées. Pour remédier à ce problème, un critère d'aspiration standard est introduit. Un mouvement marqué comme Tabou est quand même choisi si celui-ci conduit à l'obtention d'une configuration dont l'évaluation est meilleure que la meilleure configuration rencontrée jusqu'ici par l'algorithme.

[1] 1. Deux autres approches réputées sont basées sur des techniques d'agrégation [Cohon, 2000] ou de ranking [Goldberg, 1989].

4.3.7 Rôle de la diversification

La diversification est un mécanisme très important durant le processus de resolution des problèmes d'optimisation multiobjectif car il permet d'orienter la recherche pour couvrir tout l'espace de recherche donc permet de trouver un ensemble de solutions plus diversifié. Pour mieux comprendre ce dernier, on va illustrer ce mécanisme par deux algorithmes tabou présentés dans la thèse de V. Barichard [4], le premier utilise un principe de diversification basé sur la marche aléatoire et le second sur la distance de Hamming.

4.4 Algorithme tabou avec marche aléatoire

Le principe de la marche aléatoire (Random Walk (RW)) est bien connu dans la littérature et couramment utilisé dans des algorithmes réputés comme WSAT [Selman et al., 1994]. La marche aléatoire consiste à réaliser de temps en temps un mouvement qui n'est plus guidé par la fonction d'évaluation et constitue donc un schéma de diversification. Intégrer la marche aléatoire dans un algorithme Tabou standard permet d'obtenir une recherche plus diversifiée.

À chaque itération de l'algorithme **TS+RW**, une valeur réelle $rw \in [0,1]$ est choisie aléatoirement. Soit $r_{wseuil} \in [0,1]$ la valeur seuil, alors, si $rw > r_{wseuil}$ l'algorithme exécutera un mouvement classique, dans le cas contraire, l'algorithme effectuera un mouvement aléatoire réalisable. L'algorithme **TS+RW** peut être décrit ainsi :

Algorithme 11 Algorithme Tabou avec marche aléatoire **TS+RW**. [4]

Soit x une configuration réalisable, et L le nombre d'itérations
rw_{seuil} le seuil de perturbation
$ND \leftarrow \{x\}$ (ND est l'ensemble des solutions non dominées)
pour $i = 0$ jusqu'à L **faire**
    Tirer aléatoirement un nombre $rw \in [0,1]$
    si $rw \leq rw_{seuil}$ **alors**
        Choisir un mouvement m au hasard
    sinon
        Choisir le meilleur mouvement m autorisé
    finsi
    Mettre à jour la liste Tabou en fonction de m
    Effectuer le mouvement m dans x
    Mettre à jour l'ensemble des solutions non dominées ND en fonction de x
fin pour
Renvoyer ND

4.5 Algorithme Tabou et distance de Hamming

Le second algorithme Tabou est basé sur une technique de diversification originale et plus rationnelle (par rapport à **TS+RW**). L'idée principale consiste à superviser l'évolution de la

diversité des configurations récemment visitées. Si le niveau de diversité tombe sous un certain seuil, une phase de diversification est exécutée.

La diversité des configurations est calculée en utilisant la distance de Hamming. Pour appliquer cette idée, il est nécessaire de pouvoir calculer exactement et de manière incrémentale la distance de Hamming sur l'ensemble des configurations données.

À chaque itération de l'algorithme **TS+HW**, la diversité des l dernières configurations (l est fixé de manière empirique) est calculée. Si la valeur de la diversité est inférieure à un certain seuil d, l'algorithme change de comportement et entame une phase de diversification. Il existe plusieurs manières de réaliser cette phase de diversification. Dans certains articles, les auteurs augmentent la valeur Tabou des mouvements les plus fréquents, puis redémarré la recherche depuis une configuration détériorée (par exemple, la configuration dans laquelle toutes les variables sont mises à 0). L'algorithme TS+HW est décrit dans l'Algorithme 12.

Algorithme 12 Algorithme Tabou avec la distance de Hamming **TS+HW** [4].

Soit x une configuration réalisable, et L le nombre d'itérations ;
d : le seuil de diversité ;
$ND \leftarrow \{x\}$ (ND est l'ensemble des solutions non dominées) ;
pour $i = 0$ jusqu'à L **faire**
    Tirer aléatoirement un nombre $rw \in [0, 1]$;
    si $Niveau - de - diversité < d$ **alors**
        Augmenter la valeur Tabou des mouvements les plus fréquemment effectués durant la dernière phase d'intensification ;
        $x \leftarrow$configuration-zéro ;
    finsi
    Choisir le meilleur mouvement m autorisé ;
    Mettre à jour la liste Tabou en fonction de m ;
    Effectuer le mouvement m dans x ;
    Mettre à jour l'ensemble des solutions non dominées ND en fonction de x ;
fin pour
Renvoyer ND.

4.6 Le $BBTS^{MOKP}$

Au cour de ce chapitre, nous avons vu que la diversification de la recherche joue un rôle important dans la résolution des problèmes d'optimisation multiobjectif. Ainsi, l'intensification, bien qu'étant une action opposée à la diversification, influe grandement sur la qualité des résultats. Cette phase est assurée dans notre cas par la méthode Tabou Search.

La recherche Tabou est une méthode bien connue pour son aptitude à intensifier la recherche, cependant avec la nouvelle variante **TS+RW**, cette dernière devient puissante en terme de diversification. Pour observer le rôle de la diversification, nous avons choisi de partir d'un algorithme exact (Branch & Bound) performant et de lui implanter une méthode de diversification puissante (Tabou Search with Random Walk **TS+RW**). Dans ce but, nous développons un nouvel

algorithme pour le $MOKP$. Cet algorithme, appelé $BBTS^{MOKP}$, combine une approche exacte (Branch & Bound) et une méthode Tabou. Dans ce schéma, la diversification est assurée par les actions combinées de la marche aléatoire et un mécanisme de déplacement des indices de B_1 à B_0. Ce dernier mécanisme sert à créer une nouvelle branche après avoir stérilisé un nœud.

Algorithme 13 Hybrid Branch & bound with Tabou Search for knapsack problem $BBTS^{MOKP}$

Entrées: Valeurs de c_i^1 et c_i^2, poids de w_i pour $i = 1, \ldots, n$ et capacité W.

Initialisation : Créer le Nœud racine N_0 comme suit.
Placer $B_1 := \phi$, $B_0 := \phi$, $\mathcal{F} := \{1, \ldots, n\}$, $\underline{z} := (0, \ldots, 0)^t$, $\overline{z} := (\infty, \ldots, \infty)^t$, $\mathcal{L} := \phi$, $\mathcal{N} := \{N_0\}$, $K := n$; $T := \phi$.
tantque $\mathcal{N} \neq \phi$ **faire**
  Choisir le dernier nœud $N \in \mathcal{N}$, Calculer $\overline{W}$ et $\underline{z}$.
  Ajouter $\underline{z}$ à $\mathcal{L}$ s'il n'est pas dominé, Calculer $\overline{z}$.
  si $\{i \in \mathcal{F} : w_i \leq \overline{W}\} = \phi$ **ou** $\overline{z}$ est dominé par un certain $y \in \mathcal{L}$ **alors**
    Stériliser le nœud N. $\mathcal{N} := \mathcal{N}/\{N\}$.
    Créer un nouveau nœud N' comme suit.
    Enlever le dernier élément de B_1 et le mettre le dans B_0
    Mettre à jour $B_1, B_0, \mathcal{F}$
    $\mathcal{N} := \mathcal{N} \cup \{N\}$
    si l'ensemble B_1 de N' est plus petit que B_1 des nœuds prédécesseurs de N, qui ne sont pas des prédécesseurs de N' **alors**
      Stériliser ces nœuds.
    finsi
    si aucun nouveau nœud ne peut être créé ($B_1 = \phi$) **alors**
      ARRÊTER
    finsi
  sinon
    Créer un nouveau nœud N' comme suit.
    Tabou-search(x, K, rw_{seuil}) (c'est une procédure tabou adaptée expliquée dans les sections précédentes).
    Mettre à jour $B_1, B_0, \mathcal{F}$
    $\mathcal{N} := \mathcal{N} \cup N$
  finsi
fin tantque.

Sorties: Toutes les solutions efficaces.

4.7 Implémentation et résultats

L'algorithme ainsi développé est implementé sur machine en se servant des avantages du langage de programmation mathématique le Matlab 7 (matrix laboratory 7), puis on a programmé d'autres algorithmes exposés auparavant dans ce mémoire tels que branch and bound et le branch

and bound modifié [2], ainsi que deux variantes de la méthode tabou qui sont Tabou search standard et Tabou search avec marche aléatoire **TS+RW**.

Tous ces algorithmes sont implementés sur une machine dont les caractéristiques :(processor P4, FSB 800 Mb, DDR 1 512 Mo) en utilisant le même langage de programmation (matlab 7) et testés sur un ensemble d'instances de sac-à-dos biobjectifs. Ces algorithmes sont exécutés une seule fois sauf les variantes de la méthode tabou ainsi le $BBTS^{MOKP}$ en raison de la présence du facteur aléatoire dans leurs structures. Les résultats sont présentés dans ce qui suit :

Exemple 4 : Cet exemple est tirer de l'ouvrage de M.Ehrgott [22], il est constitué de deux objectifs et cinq objets.

$$\left|\begin{array}{ll} \text{Minimiser} & Z_1(x) = 11x_1 + 5x_2 + 7x_3 + 13x_4 + 3x_5 \\ \text{Minimiser} & Z_2(x) = 9x_1 + 2x_2 + 16x_3 + 5x_4 + 4x_5 \\ \text{s.c} & 4x_1 + 2x_2 + 8x_3 + 7x_4 + 5x_5 \leq 16 \end{array}\right.$$

Algorithmes/Paramètres	TS+RW	$BBTS^{MOKP}$
N^{bre} d'itérations	130	3
Seuil de la marche aléatoire	0.15	0.15
Taille de la liste Tabou	10	7

TAB. 4.1 – *Table de paramètrage pour l'exemple 4*

Algorithmes/Paramètres	B&B	B &B modifié	TS	TS+RW	$BBTS^{MOKP}$
Temps (s)	0.328000	0.031000	0.156000	0.953000	**0.015000**
N^{bre} de solutions x	3	3	1	3	3
N^{bre} des évaluations f	3	3	1	3	3

TAB. 4.2 – *Table des résultats pour l'exemple 4*

Exemple 5 : Cet exemple comporte 8 objets.

$$\left|\begin{array}{ll} \text{Minimiser} & Z_1(x) = 11x_1 + 4x_2 + 5x_3 + x_4 + 7x_5 + 13x_6 + 4x_7 + 3x_8 \\ \text{Minimiser} & Z_2(x) = 9x_1 + 8x_2 + 2x_3 + 2x_4 + 16x_5 + 5x_6 + 9x_7 + 4x_8 \\ \text{s.c} & 4x_1 + 3x_2 + 2x_3 + x_4 + 8x_5 + 7x_6 + 6x_7 + 5x_8 \leq 20 \end{array}\right.$$

[2] A la première phase, on enlève deux éléments au lieu d'un seul comme dans le branch and bound défini auparavant

Algorithmes/Paramètres	TS+RW	$BBTS^{MOKP}$
N^{bre} d'itération	400	5
Seuil de la marche aléatoire	0.15	0.15
Taille de la liste Tabou	10	7

TAB. 4.3 – *Table de paramétrage pour l'exemple 5*

Algorithmes/Paramètres	B&B	B &B modifié	TS	TS+RW	$BBTS^{MOKP}$
Temps (s)	0.312000	**0.094000**	1.171000	1.516000	0.246000
N^{bre} de solutions x	4	4	1	4	4
N^{bre} des évaluations f	4	4	1	4	4

TAB. 4.4 – *Table des résultats pour l'exemple 5*

Dans cette section nous avons présenté les résultats expérimentaux obtenus pour le $BBTS^{MOKP}$ et Branch & bound modifié ainsi que le Branch & bound multiobjectif et les algorithmes tabou. Sur la table de l'exemple 4, nous remarquons que le $BBTS^{MOKP}$ obtient le résultat en moins de temps que les autres algorithmes. Ainsi Branch & bound modifié présente une amélioration par rapport au Branch & bound standard et à la méthodes tabou. Pour l'exemple 5, l'algorithme Branch and Bound modifie présente un temps meilleur que les autres.

4.8 Résultats des instances de grande taille

Ces instances sont publié dans le site "http ://www.terry.uga.edu/mcdm/" de Zitzler and Thiele 1999. Le temps d'exécution est évalue en second.

Instances	Branch & Bound			Tabou Search				B&B modifie			$BBTS^{MOKP}$		
	T	SE	LE	T	SE	LE	N	T	SE	LE	T	SE	LE
2KP10	0.62500	2	2	2.00000	4	4	300	0.32800	2	2	**0.14100**	2	2
2KP20	27.36000	2	2	**4.56200**	9	9	1500	13.26600	5	5	14.04700	4	4
2KP30	115.45300	3	3	16.36000	11	11	3000	**2.70300**	4	4	7.31200	3	3
2KP40	31.39100	9	9	38.73400	13	13	5000	**14.56200**	13	13	31.87500	8	8
2KP50-11	123.4300	13	13	97.87500	14	14	8000	105.45300	7	7	**15.73400**	9	9
2KP50-50	89.53400	2	2	156.60900	3	3	8000	76.98000	2	2	**10.11000**	2	2
2KP50-92	73.79700	4	4	108.34400	2	2	8000	**7.10900**	4	4	64.4600	4	4

TAB. 4.5 – *Table des résultats pour les instances de benchmark*

T : Temps d'exécution ;

SE : Nombre de solutions efficaces ;

LE : Nombre d'evaluations ;

N : Nombre d'itérations.

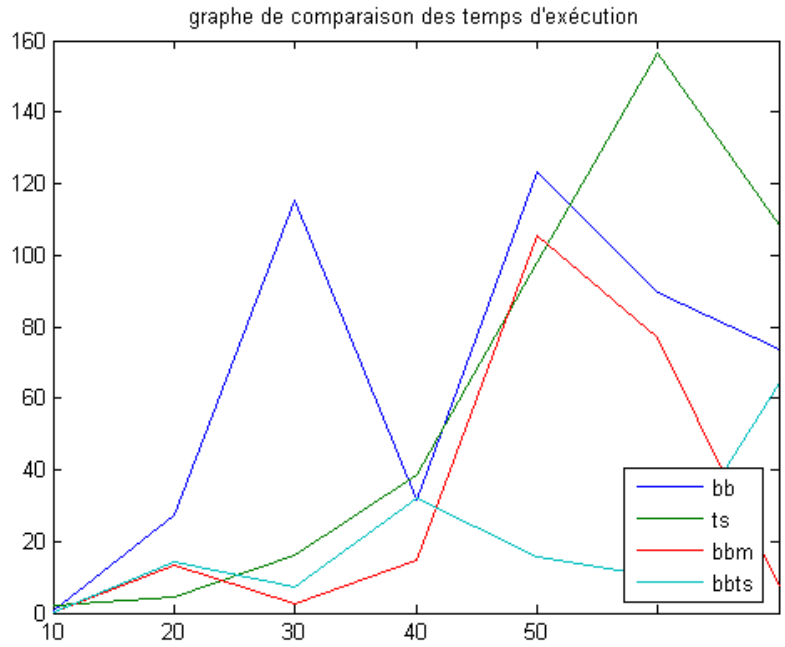


FIG. 4.1 – *graphe de comparaison des temps d'exécution*

4.9 Discussion

Le branch and bound est une méthode exacte. Elle exploite systématiquement la totalité de l'espace de recherche. Dans ce type de méthodes, l'ensemble des solutions trouvées "front Pareto" est efficace. Cependant elle est, de grande complexité, ce qui rend le temps d'exécution très importants. Cela est illustré par l'exécution des différentes instances du sac-à-dos qui donnent des temps d'exécution importants par rapport à ceux trouvés par les autres méthodes. Pour la variante de Branch and bound modifié, on remarque que le temps d'exécution diminue mais l'exploitation de l'espace de recherche n'est pas complète, ceci est dû à la modification du branchement qui se base sur l'enlèvement de deux objets au lieu d'un seul, ce qui conduit à l'ignorance d'un certain nombre de branches qui peuvent être prometteurs ou on peut trouver des solutions efficaces. Donc ce mécanisme accélère la résolution mais diminue l'efficacité de la méthode.

La méthode Tabou avec marche aléatoire est l'une des variantes les plus performantes de la méthode tabou. En particulier, Elle exploite partiellement l'espace de recherche. Pour le cas du sac-à-dos multi objectif aucun facteur ne garantit que l'espace de recherche est complètement parcouru, ce qui influe sur l'ensemble des solutions qui ne sera pas toujours efficace - un ensemble de solutions qui ne se domine pas entre elles - il est appelé Front Pareto potentiellement efficace. De plus, le nombre d'itérations et la valeur de la marche aléatoire influe beaucoup sur les résultats, cela est bien visible par l'exécution des différentes instances du problème qui montre une large différence en terme de taille du front Pareto sauf pour la dernière instance. Les temps d'exécution de cette méthode dépend du nombre d'itérations et de la taille de front trouvé ceci est bien visible

dans la figure 4.1 où les temps augmentent avec l'augmentation du nombre d'objets.

La combinaison des deux méthodes exacte et heuristique donne naissance à une méthode hybride qu'on a appelée algorithme $BBTS^{MOKP}$ qui exploite les avantages des deux approches. On remarque, à partir des résultats, que les temps d'exécution des différentes instances du sac-à-dos obtenus par cette méthode sont en moyenne meilleurs que ceux des deux méthodes qui les constituent. De plus la qualité du front Pareto et sa taille sont bien supérieures à celles trouvées par les autres algorithmes. Toute fois cet algorithme peut être amélioré en développant de nouvelles bornes et de nouveaux mécanismes de recherche.

On remarque aussi, pour toutes les méthodes présentées ici que le résultat est obtenu en peu temps et que, lorsque le nombre de solutions augmente dans l'ensemble des solutions non dominées, la complexité des algorithmes augmente, ce qui justifie l'importance des temps d'exécution pour les instances de grande taille. Cela est dû au grand nombre de verifications qui effectuées au niveau de l'ensemble des solutions non dominées où que chaque solutions trouvée sera comparée à toutes les solutions de cet ensemble.

Conclusion

Dans ce chapitre, nous avons présenté $BBTS^{MOKP}$, un algorithme hybride, combinant l'algorithme branch and bound avec la recherche Tabou. Cette hybridation s'avère très performante pour résoudre le problèmes de sac-à-dos multiobjectif unidimensionnel en 0-1 (MOKP01). Notre algorithme $BBTS^{MOKP}$ combine les concepts généraux de la méthode branch and bound avec les propriétés locales de la recherche Tabou, conduisant à un meilleur compromis entre exploitation et exploration de l'espace de recherche.

Les performances de $BBTS^{MOKP}$ ont été validées sur un ensemble d'instances publiées et comparées avec l'algorithme de branch and bound et le $TS + RW$ ainsi que avec le branch and bound modifié. Les résultats expérimentaux ont montré que $BBTS^{MOKP}$, obtient en moyenne de meilleurs résultats que les autres algorithmes en compétition sur les instances testées.

Conclusion générale

il n'y a pas des problèmes qu'on se pose,
il y a des problèmes qui se posent, il n'y a pas de problèmes résolus,
il y a des problèmes plus ou mois résolus
*** Henri Poincaré***

Dans ce travail, nous avons abordé et étudié la programmation multiobjectif. Nous nous sommes intéressé à l'hybridation des méthodes pour la résolution du problème de Sac-à-dos (knapsack problem).

D'abord, nous avons présenté le problème de sac-à-dos et ses différentes variantes, en nous focalisant sur le cas uni-dimensionnel multiobjectif. Ensuite nous avons développé un algorithme hybride, combinant une méthode de recherche Tabou avec marche aléatoire, qui permet de mettre en évidence la complémentarité des deux phases d'intensification et de diversification, et un algorithme (Branch&Bound). L'algorithme ainsi obtenu est appelé $BBTS^{MOKP}$ (hybrid Branch&Bound and tabou search for the multiobjectif knapsack problem).

Afin de tester l'efficacité de notre méthode, nous avons étudié et programmé d'autres algorithmes pour le même problème. Le premier est un algorithme exact. Il s'agit de l'algorithme de Branch & Bound bi-objectif, développé par Matthias Ehrgott [22]. Deux variantes de la méthode Tabou ont également été étudié "Tabou search standard" **TS** et "Tabou search avec marche aléatoire" **TS+RW** ; ensuite nous avons proposé une amélioration pour Branch & Bound.

Ces méthodes sont implémentées sur machine sous le logiciel matlab 7, puis une étude comparative à été menée sur un ensemble de problèmes tests (benchmark), dont les résultats numériques montrent l'efficacité de notre approche, en termes de temps d'exécution et de qualité de l'ensemble des solutions. Ceci prouve l'intérêt de l'hybridation des méthodes exactes avec des metaheuristiques.

Perspectives de recherche

Nous nous sommes concentrés dans ce mémoire sur l'optimisation combinatoire multiobjectif, dont l'objectif est de fournir l'ensemble le plus complet possible des solutions Pareto à un problème donné. En guise de perspectives, on peut citer :

- Adapter le $BBTS^{MOKP}$ à d'autre problèmes d'optimisation multiobjectif, tels que le problème de bin packing bi-objectif...
- Généraliser le (Branch & Bound et Branch & Bound modifié) au cas multiobjectif.
- Développer d'autres mécanismes de diversification et d'intensification plus performants que le $TS + RW$, et les incorporer dans le (Branch & Bound),
- Améliorer la vitesse de l'algorithme, en développant de nouvelles bornes pour le branch and bound.
- Essayer d'hybrider le Branch and bound avec d'autres metaheuristiques, telles que les algorithmes génétiques, colonies de fourmis ... etc.

Bibliographie

[1] ABDELLI. A. *Optimisation multicritère d'une chaîne éolienne passive.* PhD thesis, Institut Natinal Polytechnique de Toulouse Spécialité : Génie Electrique, 2007.

[2] AVILA. S. L. *Optimisation multiobjectif et analyse de sensibilité appliquées à la conception de dispositifs Application : Synthèse d'antennes à réflecteur embarquées dans un satellite.* PhD thesis, L'ecole centrale de lyon, Année 2006.

[3] AZARM. S. multiobjective optimum desig. *www.glue.umd.edu/azarm/optimum_notes/multi/multi.html* (1996).

[4] BARICHARD. V. *Approches hybrides pour les problémes multiobjectifs.* PhD thesis, Ecole Doctorale d'Angers, 24 Novembre 2003.

[5] BARICHARD. V ; H. JIN KAO. Picpa : un nouvel algorithme évolutionnaire pour les problèmes continus multi-objectifs sous contraintes. *LERIA - Université d'Angers, 2, Bd Lavoisier, 49045 Angers cedex 01* ((vincent.barichard, jin-kao.hao)@info.univ-angers.fr).

[6] BARICHARD. V ; M. EHRGOTT ; X. GANDIBLEUX ; V. T'KINDT. *Multiobjective Programming and Goal Programming ; Theoretical Results and Practical Applications*, spriner ed. No. 618. 2009 Springer-Verlag Berlin Heidelberg, 2009.

[7] BAZGAN. C ; H. HUGOT ; D. VANDERPOOTEN. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research 36 (2009) 260 – 279* (Lamsade, Université Paris-Dauphine, Place du Maréchal de Lattre de Tassigny, 75 775 Paris Cedex 16, France 22 September 2007).

[8] BAZGAN. C ; H. HUGOT ; D. VANDERPOOTEN. Implementing an efficient fptas for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research journal homepage : www.elsevier.com/locate/ejor* (Université Paris-Dauphine, Lamsade, Place du Maréchal de Lattre de Tassigny, 75 775 Paris cedex 16, France (2008)).

[9] BERRO. A. *otimisation multiobjetive et stratégie dévolution dans un enveronnement dynamique.* PhD thesis, Université des sciences sociale Toulouse 1, 2001.

[10] BLUM. C ; C. COTTA ; ANTONIO J. F ; J. E. GALLARDO ; M. MASTROLILLI. Hybridizations of metaheuristics with branch and bound derivates. *Studies in Computational Intelligence (SCI) 114, 85–116 (2008)* (Springer-Verlag Berlin Heidelberg 2008).

[11] BLUM. C ; M. J. BLESA AGUILERA ; A. ROLI ; M. SAMPELS. *Hybrid Metaheuristics An Emerging Approach to Optimization.* 2008 Springer-Verlag Berlin Heidelberg, ISBN 978-3-540-78294-0 e-ISBN 978-3-540-78295-7.

[12] BONOMI. E ; J. L. LUTTON. *Le recuit simulé, pour la science.* number 129, pages 68-77, July 1988.

[13] CHAABANE. D. *contribution a l'optimisation multicritère en variables discrètes*. PhD thesis, Dissertation soumise à la Faculté Polytechnque de Mons en vue l'obtention du titre de Docteur en Sciences Appliquées, 2007.

[14] CLERC. M ; P. SIARRY. Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires. Tech. rep., France Télécom R D ; Université Paris 12, 17/09/2004.

[15] COELLO COELLO. C. A. An updated survuey of g.a based multiobjective optimization techniques. *Technical report Lania-RI-98-08 Laboratorio Nacional Avanzada, Xalapa, Veracruz, Mexico* (Deceùmber 1998).

[16] COLLETTE. Y ; SIARRY P. *Multiobjective Optimization*. Cranfield Bedford MK43 0AL UK, August 2003.

[17] DELORME. X. *Modélisation et résolution de problémes liés à léxploitation dínfrastructures ferroviaires*. PhD thesis, lúniversité de Valenciennes et du Hainaut-Cambrésis, 2003.

[18] DHAENENS-FLIPO. C. *optimisation combinatoire multi-objectif :apport des méthodes coop´eratives et contribution à l'extraction de connaissances*. PhD thesis, Université des Sciences et Technologies de Lille U.F.R. d'I.E.E.A., 2005.

[19] DIETZ. A. R. *Optimisation multicritère pour la conception d'ateliers discontinus multiproduits : aspects économique et environnemental*. PhD thesis, Institut Natinal Polytechnique de Toulouse, 09 décembre 2004.

[20] EGEBLAD. J ; D. PISINGER. Heuristic approaches for the two-dimensional and three-dimensional knapsack packing problem. *Department of Computer Science, University of Copenhagen, Computer Science Universitetsparken 1, 2100 Copenhagen, Denmark Computers. Operations Research 36 (2009) 1026 – 1049* (14 December 2007).

[21] EHRGOTT, M., AND GANDIBLEUX, X. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research* (2007).

[22] EHRGOTT. M. *Multicriteria Optimization*. Springer, March 2005.

[23] EL-DJILLALI. T. *Sélection et réglage de paramètres pour l'optimisation de logiciels d.ordonnancement industriel*. PhD thesis, Thèse réalisée en Convention Industrielle de Formation par la Recherche, Département R D, Finmatica France SA, 150 grande rue de St Clair . Le Sextant, 69731 caluire et cuire cedex / Équipe Production Automatisée Laboratoire Génie de la Production, École Nationale d'Ingénieurs de Tarbes, 47 Avenue d'Azereix, BP 1629, 65016 Tarbes Cedex, Année 2004.

[24] ELGHAZALI. T. *Méthodes dÓptimisation Avancée*. Edition Eyrolles, 2002.

[25] ESCHENAUER. H ; J. KOSKI, A. O. *Multicriteria design optimisation : Procedures and Applications*. Springer, 1990.

[26] FEO T. A ; RESENDE. M. A probabilistic heuristic for a computationally difficult set covering problem. Tech. rep., Opérations Research Letters 8,67-71, 1988.

[27] G, O. I. H. L. Metaheuristics : a bibliography. *Annals of Operations Research* (63 :513-623, 1996).

[28] GAGNE C ; M. GRAVEL. optimisation multi-objectifs a l'aide d'un algorithme de colonie de fourmis. In *Departement d'informatique et de mathimatique universite du Quebec. Canada G7H 2BI Courriel : caroline_gagne@uqac.ca. marc_gravel@uqac.ca* (WILSON L. PRICE Faculte des Sciences de I 'administration. Universite Laval Sfe-Foy, Quebec. Canada GIK 7P4 Courriel : wilson.price@fsa.ulavaica).

[29] GALLARDO. J. E ; C. COTTA ; A. J. FERNANNDEZ. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. *Dept. Lenguajes y Ciencias de la Computaction, ETSI Informatica, University of Malaga, Campus de Teatinos, 29071 - Malaga, Spain* (fpepeg,ccottap,afdezg@lcc.uma.es).

[30] GARY. F. G. L ; A. KOCHENBERGER. *Handbook of Metaheuristics, School of Business University of Colorado at Boulder, College of Business University of Colorado at Denver.* Kluwer Academic Publishers, 2003.

[31] GEIGER. M. J. Bin packing under multiple objectives a heuristic approximation approach. *Department of Industrial Management(510A), University of Hohenheim, 70593 Stuttgart, Germany* (sep 2008).

[32] GILMORE. P. C ; R. E. GOMORY. *A linear programming approach to the cuttingstock problem (part II).* Operations Research 11, 1963.

[33] GRÄBENER. T ; A. BERRO. Optimisation multiobjectif discrète par propagation de contraintes. *IRIT — Université de Toulouse* (Manuscrit auteur, publié dans "JFPC 2008-Quatrièmes Journées Francophones de Programmation par Contraintes, Nantes : France (2008)").

[34] HAIMES. Y. Y ; W. A. HALL ; H. T. FREEDMAN. A multiobjective optimization in water resources systems. *Elsevier Scientific* (1975).

[35] HOLLAND. J.H. *Adaptation in natural and artifcial systems.* Phd thesis, University of Michigan Press, 1975.

[36] JEAN-FRANÇOIS LE GALL. Intégration, probabilités et processus aléatoires. *Ecole normale supérieure de Paris* (Septembre 2006).

[37] KEENEY. R. L ; H. RAAIFFA. Decision with multiple objective : Preference and value tradoff. *Cambridge University Press* (1993).

[38] KHEMAIES. G. M. *Methodoligie de Conception Système à Base de Platformes Reconfigurables et Programables.* PhD thesis, Université Paris xi UFR Scientifique D'Orsay, 01 Mars 2005.

[39] KORTE. B ; J. VYGEN. *Combinatorial Optimization Theory and Algorithms*, fourth edition ed. Springer-Verlag Berlin Heidelberg, 2008.

[40] LEMESRE. J. *Méthodes exactes pour l'optimisation combinatoire multi-objectif : conception et aplication.* PhD thesis, U.F.R. D'I.E.E.A, 2006.

[41] LEMESRE. J. *Méthodes Exactes pour L'optimisation Combinatoire Multi-objectifs : Conception et Application.* PhD thesis, Université des Sciences et Technologies de Lille U.F.R. D'I.E.E.A, Année 2006.

[42] LIU. D. S ; K. C. TAN ; S. Y. HUANG ; C. K. GOH ; W. K. HO. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research 190* ((2008)), 357–382.

[43] MARTELLO. S ; P. TOTH. An upper bound for the zero-one knapsack problem and abranch and bound algorithm. *European Journal of Operational Research Vol.1* (1977), pp. 169–175.

[44] MARTELLO. S ; P.TOH. *Knapsak Problems, Algorithms and Computer Imlementations.* John Wiley, Sons Ltd. Baffins Lane, Chichester West Sussex PO19 IUD, England, 1990.

[45] MATTHIEU. B. *Conception D'algorithmes coopératifs pour l'optimisation Multi-objectif :Application aux Problèmes D'ordonnancement de Type Flow-shop.* PhD thesis, Université des Sciences et Technologies de Lille U.F.R. D'I.E.E.A., 21/06/2005.

[46] MATTHIEU. B ; T. EL-GHAZALI ; A. NEBRO ; E. ALBA. Metaheuristics for multiobjective combinatorial optimization problems : Review and recent issues. Tech. rep., Institut Natinal de Recherche en Informatique et en Automatique, Septembre 2006.

[47] MIETTIENEN. K. M ; M. M. MÄKELÄ. Proper pareto optimality in non convex problems, characterization with tangent and normal cones, technical report. *Jyväskylä University* (November 1998).

[48] NAFI. A. *La Programmation Pluriannelle du Renouvellement des Reseaux D'eau Potable.* PhD thesis, Université Louis Pasteur, Strasbourg I, 08/12/2006.

[49] NELDER. J. A ; R. MEAD. A simplex method for function minimization. *Computer Journal 7* (1965), 308–313.

[50] PIRLOT. M. Métaheuristiques pour lóptimisation combinatoire : un aperçu général.dans j. teghem et m. pirlot, éditeurs, optimisation approchée en recherche opérationnelle - recherches locales, réseaux neuronaux et satisfaction de contraintes, pages 25-55. *Hermés Science Publications* (Paris, 2002).

[51] REARDON. B. J. Fuzzy logic us niched pareto muli-objective genetic algorithm optimization : Part i : Schaffer's problem. Technical report la- ur-97-3675, Los Alamos National Laboratory, 1997.

[52] SAIT. S. M ; H. YOUSSEF. *iterative computer algorithms with applications iningineering : solving combinatorial optimization problems.* IEEE computerssociety, 1999.

[53] SAKAWA. M ; H. YANO. *An interactive fuzzy satisfcing method for multiobjective linear programming problems witth fuzzy parametres*, vol. volume 28. Fuzzy sets and Systems, 1988.

[54] SIARRY. P. La méthode du recuit simulé en électronique. *Habilitation report, Paris-sud University(Orsay)* (1994).

[55] SIARRY. P. Optimisation et classification de données. *Lectures from the dea at Paris University* (1999).

[56] SOURD. F ; O. SPANJAARD. Multi-objective branch-and-bound. application to the bi-objective spanning tree problem. *Laboratoire d'Informatique de Paris 6 (LIP6-UPMC)* (4 Place Jussieu, F-75252 Paris Cedex 05, France).

[57] TAILLARD. E. *La programmation á mémoire adaptative et les algorithmes pseudo-gloutons : nouvelles perspectives pour les métaheuristiques.* PhD thesis, Thése dhabilitation, Université de Versailles Saint-Quentin-en-Yvelines, Versailles, France, 1998.

[58] TAILLARD. E. Principes dímplémentation des métaheuristiques. dans j. teghem et m. pirlot, éditeur, optimisation approchée en recherche opérationnelle - recherches locales, réseaux neuronaux et satisfaction de contraintes, pages 57-79. *Hermés Science Publications* (Paris, 2002).

[59] T'KINDT. V ; JC. BILLAUT. *Multicriteria Scheduling Theory, Models and Algorithms*, second edition ed. Springer-Verlag Berlin Heidelberg, 2002, 2006.

[60] VASQUEZ. M ; H. JIN KAO. une approche hybride pour le sac à dos multidimentionnel en variables 0/1. *RAIRO Operations Research* (RAIRO Oper. Res. 35 (2001) 415-438).

[61] VELDHUIZEN. D. *Multiobjective Evolutionary Algorithms : Classification, Analyses, and New Innovations.* PhD thesis, Air Force Institute of Technology, Air University, 1999.

[62] VELDHUIZEN. D. A. VAN. *Multiobjective Evolutionary Algorithms :Classifications, Analyses and New Innovation.* PhD thesis, Graduate School of Engineering ; Air Force Institute of Technology ; Wright Patterson AFB, Ohio,USA, Janvier ; 1999.

[63] WHITTAKER. G ; R. CONFESOR JR ; S. M. GRIFFITH ; R. FARE ; S. GROSSKOPF ; J. J. STEINER ; G. W. MUELLER-WARRANT ; G. M. BANOWETZ. A hybrid genetic algorithm for multiobjective problems with activity analysis-based local search. *European Journal of Operational Research 193 (2009) 195–203* (2007).

[64] YAMADA. T ; T. TAKEOKA. An exact algorithm for the fixed-charge multiple knapsack problem. *European journal of operational research ISSN* (2009, vol. 192).

[65] YAMADA. T ; T. TAKEOKA. An exact algorithm for the fixed-charge multiple knapsack pr. *Department of Computer Science, The National Defense Academy, Yokosuka, Kanagawa 239-8686, Japan European Journal of Operational Research 192 (2009) 700–705* (22 October 2007).

[66] ZADEH L. A. *Fuzzy sets*, vol. volume 8. Information and Control, 1965.

Résumé

Résumé *: Approche Hybride Pour Les Problèmes multiobjectif : Cas de problème Sac-à-dos.*

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines applicatifs, tel que le traitement d'images, la conception de systèmes, la conception d'emplois du temps,... Ces problèmes sont, en majorité qualifies de difficiles, car pour leur résolution, il n'est pas en général possible de fournir dans tous les cas des solutions en un temps raisonnable. Il en est ainsi du problème de Sac-à-dos. Dans le cas où plusieurs critères sont à optimiser, la problématique se complique davantage car ces critères sont souvent contradictoires. D'autres notions d'optimalité sont utilisées, le plus souvent la Pareto optimalité où il s'agit de déterminer l'ensemble des solutions efficaces. Dans ce mémoire, nous développons pour le problème du Sac-à-dos uni-dimensionnel multiobjectif un algorithme hybride fondé sur les méthodes "Branch and Bound" et "Recherche Tabou". Cet algorithme met en évidence l'importance d'une gestion efficace de la diversité, ainsi que le rôle de l'intensification afin de trouver le front Pareto (solutions efficaces supportées et non-supportées). Cet algorithme est implementé et les résultats expérimentaux obtenus sont discutés.

Mots clés : *problèmes d'optimisation multiobjectif, algorithme hybride, Branch and Bound, Recherche Tabou, Sac-à-dos, front Pareto, solutions efficaces supportées et non-supportées.*

Abstract *: Hybrid approach for multiobjective problems : case of Knapsack problem.*

The combinatorial optimization encompasses a wide class of problems with applications in many application areas, such as image processing, systems design, design schedules,... These problems are mostly skilled difficult, because their resolution, it is generally not possible to provide in all cases solutions in a reasonable time. This is the issue of knapsack problem. Where are several criteria to optimize the problem is further complicated because these criteria are often contradictory. Other notions of optimality are used most often where Pareto optimality is to determine all efficient solutions. In this paper, we develop for the uni-dimensionnel multiobjective knapsack problem an hybrid algorithm based(established) on the methods of Branch and Bound and the Tabou Search. This algorithm brings to light the importance of an effective management of the diversity, so the role of the intensification to find the totality of the Pareto optimal set (efficient supported solutions and non-supported solutions). This algorithm is implemented and experimental results obtained are discussed.

Keywords : *Multiobjective optimization problems, hybrid algorithm, Branch and Bound, Tabu Search, knapsack, front Pareto, efficient supported and unsupported solutions.*