N° d'ordre : 08/2012-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 : Mathématiques
Spécialité : Recherche Opérationnelle
Option : Mathématiques de Gestion

Par : **BRAHAM MOUSSA Hassina**

Sujet

Grilles de calcul pour les problèmes d'optimisation multi-objectifs

Soutenu publiquement le *14 Novembre 2012*, devant le jury composé de :

Mr M. MOULAÏ, Professeur, à l'USTHB	Président
Mr A. BERRACHEDI, Professeur, à l'USTHB	Directeur de Mémoire
Mr M.E.A. CHERGUI, Maître de Conférences/A, à l'USTHB	Examinateur
Mme A. El-MAOUHAB, Chargé de Recherche, au CERIST	Invitée

A ma mère,

A la mémoire de mon père,

A mes beaux parents,

A mon mari et à mes enfants,

A ma sœur,

A tous les membres de ma famille,

A toutes mes amies.

Remerciements

Je tiens d'abord à remercier Dieu de m'avoir guidée et menée jusqu'ici.

J'adresse mes remerciements à Monsieur Berrachedi Abdelhafid, Professeur à l'USTHB, pour m'avoir fait l'honneur de diriger ce mémoire, pour son soutien exemplaire et pour la confiance et la liberté qu'il m'a accordées.

J'exprime toute ma gratitude à Monsieur Chergui Mohamed El Amine, Maître de Conférences à l'USTHB, pour son aide précieuse, les conseils et les connaissances dont il a su me faire profiter.

Je tiens à remercier également Madame El-Maouhab Aouaouche, Chargée de Recherche au CERIST pour avoir éclairé mon chemin, pour ses précieux conseils et infinie patience, de m'avoir soutenu durant cette longue épreuve.

Je remercie Monsieur MOULAÏ Mustapha, Professeur à l'USTHB, qui m'a fait l'honneur d'accepter de présider le jury de ce mémoire.

Que Monsieur Chergui Mohamed El Amine, Maître de Conférences à l'USTHB, trouve ici l'expression de ma reconnaissance pour avoir accepté de juger ce travail en tant qu'examinateur.

Je ne pourrais oublier de remercier mes amies, Khedimi Amina, Bentaleb Ouafa, Aït Mehdi Meriem, Ouail Fatma Zohra, qui m'ont accompagnées durant cette période.

Finalement, je voudrais remercier ma famille, en particulier, ma mère et mon mari, pour leur soutien et appui inconditionnel.

Merci à tous.

Table des matières

Liste des tableaux

Table des figures

Introduction

La résolution des applications d'Optimisation Combinatoire a toujours été limitée par les ressources d'une seule machine : soit par la puissance de calcul, soit par la capacité mémoire, le plus souvent les deux à la fois.

Nous nous plaçons dans ce mémoire, dans le contexte de l'optimisation multiobjectif discrète où il s'agit de déterminer l'ensemble des solutions efficaces d'un problème de programmation linéaire multiobjectif en nombres entiers.
Sur le plan des méthodes de résolution, elles sont principalement scindées en deux grandes familles complémentaires : les méthodes exactes et les méthodes approchées. Les méthodes exactes permettent de trouver le ou les meilleure(s) solution(s) avec preuve d'optimalité. Certaines de ces méthodes sont fondées sur le paradigme de " séparation et évaluation " (Branch-and-Bound en anglais) qui consiste à effectuer un parcours implicite et intelligent d'une arborescence de sous-ensembles de solutions afin de découvrir les solutions optimales.

Pour accélérer les calculs, les grilles de calcul représentent donc une solution vitale

pour le traitement de ces applications à travers la parallélisation de ces méthodes de résolution. Néanmoins leur déploiement efficace sur des grilles de calcul requiert une reconstitution et une reformulation afin de les adapter à une architecture de grilles. L'objectif de ce travail est de revoir quelques méthodes de résolution en programmation linéaire multiobjectif en nombres entiers basées sur le Branch-and-Bound dans le but d'implémenter une d'elles sur les grilles de calcul.

Ce mémoire comporte quatre chapitres :

- Dans le premier chapitre, nous présentons le premier volet du cadre de notre travail qui concerne l'Optimisation Multiobjectif Discrète. Nous commençons par un bref rappel de quelques résultats importants en programmation linéaire, nous formulons ensuite le problème de programmation linéaire à variables discrètes en exhibant sa difficulté, pour finir, nous décrivons la méthode duale fractionnaire [Gom58] permettant de résoudre ce type de problèmes. Nous définissons par la suite le problème de programmation linéaire multiobjectif en nombres entiers et présentons les concepts fondamentaux relatifs tels que la dominance, l'efficacité, les solutions supportées et non supportées. Nous donnons en dernier, une méthode graphique pour la détection des solutions efficaces.]
- Le chapitre 2 est entièrement consacré à la description de quelques méthodes existantes d'optimisation multiobjectif discrète ([CAM12], [KH82], [GM92], [Mou02b], [SC04]) en illustrant certaines par des exemples.
- Le chapitre 3 aborde le second volet de notre travail, dans lequel nous commençons par décrire les grilles informatiques en général en donnant quelques définitions, concepts et terminologies de termes utilisés dans ce domaine. Nous décrivons par la suite les grilles de calcul étant le type de grilles qui répond aux besoins de notre travail, de quoi sont elles composées ?, comment les utiliser ?

et dans quel but ?.

– Le chapitre 4 présente notre contribution pour le domaine multiobjectif. Nous proposons en effet une implémentation parallèle de la méthode de Chergui et Al [CAM12] sur la grille de calcul sous Matlab en permettant de réduire le temps d'exécution. Nous exposons d'abord les stratégies de parallélisation, l'environnement Matlab installé dans la grille de calcul qui est différent de celui qui s'installe sur une seule machine, puis nous proposons une adaptation de la méthode dont il est question ainsi que les principales étapes de son execution sur la grille. Pour finir, nous présentons les principaux résultats obtenu des expérimentations efféctuées sur la dite méthode.

Enfin, nous terminons par une conclusion reprenant les différentes contributions apportées dans ce mémoire et donnant quelques perspectives de recherche.

CHAPITRE 1

Introduction à l'optimisation multiobjectif discrète

Dans de nombreuses situations pratiques, il est nécessaire d'utiliser des variables discrètes dans la modélisation du problème, par exemple, dans un problème de planification manufacturière, le nombre d'unités produites doit être un nombre entier. On parle alors de programmation linéaire discrète ou en nombres entiers. L'objet de ce chapitre introductif est de présenter quelques notions fondamentales concernant les problèmes de programmation linéaire en nombres entiers dans le cas où l'objectif à optimiser est unique et celui où plusieurs critères sont à considérer.

1.1 Programmation linéaire unicritère discrète

De nombreux problèmes d'optimisation relèvent de l'optimisation discrète. Dans ces problèmes, les variables de décision sont astreintes à prendre des valeurs entières

et cette restriction les rend particulièrement difficiles (résolution, temps d'exécution, ...).

1.1.1 Rappel de quelques définitions et résultats fondamentaux de la programmation linéaire

Définition 1.1. *Un programme linéaire est un problème dans lequel les variables sont des réels qui doivent satisfaire un ensemble d'équations et/ou d'inéquations linéaires (dites contraintes) et la valeur d'une fonction linéaire de ces variables (appelée fonction objectif), doit être rendue maximum (ou minimum).*

Sans perte de généralité, nous supposerons par la suite que nous considérons des problèmes de maximisation.

Définition 1.2. *Soient une $m \times n$-matrice A, un m-vecteur colonne b et un n-vecteur ligne c. Les programmes linéaires :*

$$(PC) \begin{cases} Max\ z = cx \\ \\ Ax \leq b \qquad\qquad x \geq 0 \end{cases}$$

$$(PS) \begin{cases} Max\ z = cx \\ \\ Ax = b \qquad\qquad x \geq 0 \end{cases}$$

sont écrits sous forme canonique et sous forme standard respectivement.

Théorème 1.1. *Tout programme linéaire peut être écrit sous forme canonique ou sous forme standard.*

Définition 1.3. *Le système linéaire (où A est une $m \times n$-matrice) :*

$$Ax = b \tag{1.1}$$

est dit de plein rang si le rang de A, c'est-à-dire le nombre de colonnes de A linéairement indépendantes, est égal à m. On considère le programme linéaire

$$(P) \begin{cases} Max\ z = cx \\ \\ Ax = b \qquad\qquad x \geq 0 \end{cases}$$

tel que le système $Ax = b$ soit de plein rang. Une base de ce programme linéaire est un ensemble $B \subset \{1, 2, \ldots, n\}$ d'indices de colonnes tel que A^B soit carrée[1] *non singulière.*

Définition 1.4. *À une base B du programme linéaire (P), on associe la solution de base $\overline{x}$ correspondant à B, solution de (1.1) définie par*

$$\overline{x}_j = 0 \text{ pour } j \notin B$$

Une base B est dite réalisable si la solution de base associée est réalisable pour (P), c'est-à-dire $\overline{x}_B = (A^B)^{-1}b \geq 0$.

Définition 1.5. *Etant donnée une base réalisable B du programme linéaire (P), le programme linéaire, équivalent à (P) :*

$$(PC) \begin{cases} \widehat{c}^N x_N = z(Max) - \pi b \\ \\ x_B + (A^B)^{-1}A^N x_N = (A^B)^{-1}b \qquad\qquad x \geq 0 \end{cases}$$

1. A^B désigne la $m \times |B|$-matrice obtenue à partir de A en retenant les colonnes A^j pour $j \in B$.

où :

- $N = \{1, 2, \ldots, n\} \setminus B$
- $\pi = c^B (A^B)^{-1}$ *est dit vecteur multiplicateur relatif à la base* B
- $\widehat{c} = c - \pi A$ *est dit vecteur coût réduit relatif à la base* B

est dit forme canonique de (P) *par rapport à la base* B*.*

Théorème 1.2. *Si le vecteur coût réduit* $\widehat{c}$ *relatif à une base réalisable* B *est négatif ou nul, la solution de base correspondante est solution optimale de* (P)*. La base* B *est alors dite base optimale.*

Théorème 1.3 (théorème fondamental de la programmation linéaire)**.** *Etant donné un programme linéaire :*

- *S'il admet une solution réalisable, il admet une solution réalisable de base.*
- *S'il admet une solution optimale, il admet une solution optimale de base.*
- *S'il admet une solution réalisable et si la valeur de la fonction objectif est bornée supérieurement, il admet une solution optimale de base.*

Définition 1.6. *Le domaine des solutions réalisables d'un programme linéaire est un convexe particulier, intersection d'un nombre fini de demi-espaces fermés, appelé polyèdre convexe.*

Théorème 1.4. *Une solution de base réalisable du programme linéaire* (P) *correspond à un sommet du polyèdre convexe* D *des solutions réalisables. Une itération de l'algorithme du simplexe*[2] *correspond à un mouvement d'un sommet à un sommet adjacent de* D*.*

2. Algorithme dédié à la résolution des programmes linéaires en variables continues, dû à G.B.Dantzig 1947 [Dan63].

1.1.2 Formulation et complexité du problème

Si dans le programme linéaire (P), on contraint le vecteur x à être entier, on obtient le programme (Q), qui est un *programme linéaire en nombres entiers* (PLNE) :

$$(Q)\begin{cases} Max\ z = cx \\ Ax = b \qquad\qquad x \geq 0 \\ x \in \mathbb{Z}^n \end{cases}$$

Ces contraintes supplémentaires (x_j entier pour $j = \overline{1,n}$) sont dites contraintes d'intégrité. En général, un tel problème n'a de solutions que si A et b sont aussi entiers.

Les principales motivations derrière la programmation linéaire en nombres entiers sont :

- le besoin de variables entières (un nombre de camions à acheter par exemple) ;
- la modélisation de contraintes et de conditions ingérables par la programmation linéaire en variables continues (c'est le cas par exemple du choix entre deux alternatives incompatibles ; on peut introduire la contrainte $x+y \leq 1$ avec x et y deux variables binaires valant 1 si l'alternative correspondante est choisie).

La difficulté de résolution de tels problèmes réside dans la construction de l'enveloppe convexe du domaine des solutions entières, c'est-à-dire le plus petit ensemble convexe contenant ce dernier. Aussi, ces problèmes sont connus pour être NP-Complets [NW88].

Les deux principales familles de méthodes actuellement connues pour résoudre ces programmes sont les méthodes arborescentes et les méthodes de troncatures (coupes). Nous présentons ci-dessous une méthode de coupe, proposée en 1958 par

R.Gomory [Gom58].

1.1.3 Méthode duale fractionnaire

Considérons le PLNE sous forme standard :

$$\begin{cases} Max\ z = cx \\ Ax = b \qquad\qquad x \geq 0 \\ x \in \mathbb{Z}^n \end{cases} \tag{1.2}$$

La relaxation continue de (1.2) est :

$$\begin{cases} Max\ z = cx \\ Ax = b \qquad\qquad x \geq 0 \end{cases}$$

Ce problème est obtenu à partir de (1.2) en relachant (c'est-à-dire en oubliant) les contraintes d'intégrité.

L'idée principale des méthodes de coupes est d'ajouter des contraintes qui n'excluent aucun point entier réalisable. La méthode consistera à ajouter de telles contraintes linéaires une par une, jusqu'à ce que la solution optimale de la relaxation soit entière. Les contraintes ajoutées sont appelées troncatures ou coupes.

Description de la méthode

Commençons par résoudre la relaxation continue par l'algorithme du simplexe [Dan63] et considérons le tableau obtenu à l'optimum ($\overline{A}x = \overline{b}$) : parmi les variables de base, choisissons une variable x_r, $r \in B$, fractionnaire (s'il n'y en a pas, on est à

l'optimum). La contrainte correspondant à x_r (i$^{\text{ème}}$ ligne du tableau optimal) s'écrit :

$$x_r + \sum_{j \in N} \overline{A}_i^j x_j = \overline{b_i} \tag{1.3}$$

Notations : étant donné un nombre réel y, on désigne par :

- $\lfloor y \rfloor$ le plus grand entier inférieur ou égal à y ;
- $\langle y \rangle = y - \lfloor y \rfloor$.

$\langle y \rangle$ est appelée la partie fractionnaire de y et $\lfloor y \rfloor$ sa partie entière.

Puisque toutes les variables sont positives ou nulles dans (1.3), on a :

$$\sum_{j \in N} \left\lfloor \overline{A}_i^j \right\rfloor x_j \leq \sum_{j \in N} \overline{A}_i^j x_j$$

Donc on a :

$$x_r + \sum_{j \in N} \left\lfloor \overline{A}_i^j \right\rfloor x_j \leq \overline{b_i}$$

La partie gauche de cette inéquation est entière. Le second membre (partie droite) peut donc être remplacé par $\lfloor \overline{b_i} \rfloor$:

$$x_r + \sum_{j \in N} \left\lfloor \overline{A}_i^j \right\rfloor x_j \leq \lfloor \overline{b_i} \rfloor \tag{1.4}$$

(1.3)−(1.4) donne :

$$\sum_{j \in N} (\overline{A}_i^j - \left\lfloor \overline{A}_i^j \right\rfloor) x_j \geq \overline{b_i} - \lfloor \overline{b_i} \rfloor$$

Finalement, nous obtenons la contrainte :

$$\sum_{j \in N} \left\langle \overline{A}_i^j \right\rangle x_j \geq \langle \overline{b_i} \rangle$$

C'est la coupe de Gomory.

Nous voulons ajouter cette coupe au problème initial. Pour garder un problème écrit sous forme canonique par rapport à une base, nous multiplions cette dernière inéquation par (-1) et ajoutons une variable d'écart x_s. On obtient :

$$-\sum_{j\in N}\left\langle \overline{A}_i^j \right\rangle x_j + x_s = -\left\langle \overline{b_i} \right\rangle$$

La nouvelle base formée de la base optimale précédente à laquelle on ajoute s vérifie les conditions d'optimalité (coûts réduits négatifs ou nuls) et n'est pas réalisable.

On peut appliquer l'algorithme dual du simplexe [Lem54] pour résoudre le nouveau programme linéaire formé.

Par ce procédé, on ajoute ainsi, une par une des coupes jusqu'à obtention d'une solution de base entière qui est alors l'optimum de (1.2).

1.2 Programmation linéaire multiobjectif discrète

Comme son nom l'indique, un problème d'optimisation multiobjectif ou multicritère consiste à optimiser simultanément k fonctions objectifs ($k \geq 2$) souvent contradictoires (deux objectifs sont contradictoires lorsque la diminution de l'un entraîne une augmentation de l'autre). De ce fait, la notion d'optimalité perd son sens, on essaie de proposer des solutions de compromis.

1.2.1 Formulation du problème

Un programme linéaire multiobjectif en nombres entiers (MOILP : **M**ultiple **O**bjective **I**nteger **L**inear **P**rogram) peut être formulé comme suit :

$$\begin{cases} Max & z_1 = c^1 x \\ & \vdots \\ Max & z_k = c^k x \\ & x \in S = \{x \in \mathbb{Z}^n | Ax = b,\, x \geq 0\} \end{cases}$$

Où $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ et $c^i \in \mathbb{R}^{1 \times n}$, $i = \overline{1,k}$.

Il peut être aussi écrit de la manière suivante :

$$\text{“}Max\text{”}\ \{z = Cx | x \in S\}$$

où C est une $k \times n$-matrice composée des vecteurs lignes c^i.

On met Max entre guillemets pour dire qu'en général, on ne peut pas trouver une solution réalisable qui maximise les k objectifs simultanément.

1.2.2 Concepts de base

Soit (P) un programme linéaire multiobjectif en nombres entiers. S l'ensemble des solutions réalisables de (P) dans l'espace des décisions (l'espace $\mathbb{Z}^n$ où se situe S).

Solutions réalisables dans l'espace des critères

L'ensemble Z donné par :

$$Z = \left\{ z \in \mathbb{R}^k | z = Cx,\, x \in S \right\}$$

représente l'ensemble des points réalisables dans l'espace des critères (l'espace $\mathbb{R}^k$ où se situe Z). En d'autres termes, Z est l'ensemble des images de tous les points de S. On impose une relation d'ordre partiel (une solution peut être meilleure qu'une autre sur certains objectifs et moins bonne sur les autres) sur cet ensemble de points, appelée *relation de dominance*.

Dominance

Soient deux vecteurs critères $z^1, z^2 \in Z$. On dit que z^1 *domine* z^2 si et seulement si $z_i^1 \geq z_i^2$ pour tout i avec au moins une inégalité stricte.

Efficacité

Une solution $\overline{x} \in S$ est *efficace* (ou Pareto optimale, ou encore non inférieure) si et seulement s'il n'existe pas de solution $x \in S$ telle que $c^i x \geq c^i \overline{x}$ pour tout i avec au moins une inégalité stricte.

Efficacité faible

Une solution $\overline{x} \in S$ est *faiblement efficace* si et seulement s'il n'existe pas de solution $x \in S$ telle que $Cx > C\overline{x}$.

Le vecteur critère associé à une solution (faiblement) efficace $\overline{x}$ donné par $C\overline{x}$ est dit solution (faiblement) non dominée.

Solutions supportées et non supportées

Considérons le problème

$$(P')\begin{cases} Max & z_1 = c^1 x \\ & \vdots \\ Max & z_k = c^k x \\ & x \in D = \{x \in \mathbb{R}^n | Ax = b,\ x \geq 0\} \end{cases}$$

relaxation continue du problème (P).

Théorème 1.5 (Geoffrion [Geo68]). *Étant donné le problème* $(P_\lambda) : Max\,\{\lambda^t Cx | x \in D\}$ *avec* $\lambda \in \Lambda = \left\{\lambda \in \mathbb{R}^k \mid \sum_{i=1}^{k} \lambda_i = 1,\ \lambda_i \geq 0,\ i = \overline{1,k}\right\}$ *; alors* $\overline{x}$ *est une solution efficace pour* (P') *si et seulement si* $\overline{x}$ *est une solution optimale du problème paramétrique* (P_λ).

D'après ce théorème, l'ensemble des solutions efficaces du problème (P') est bien caractérisé par la solution de (P_λ). Différemment du cas continu, la difficulté principale rencontrée lorsqu'on traite les problèmes multiobjectifs à variables discrètes est l'existence de solutions efficaces qui ne sont pas optimales pour (P_λ) (en remplaçant D par S) et ce en raison de la non convexité du domaine des solutions réalisables S, ces solutions sont dites solutions efficaces *non supportées*. Celles qui sont optimales pour (P_λ), sont appelées solutions efficaces *supportées*.

L'exemple numérique suivant (introduit par Bowman [Bow76]) illustre ces deux notions :

$$(P^{Bow})\begin{cases} Max\ z_1 = 6x_1 + 3x_2 + x_3 \\ Max\ z_2 = x_1 + 3x_2 + 6x_3 \\ x_1 + x_2 + x_3 \leq 1 \\ x_1, x_2, x_3 \in \{0, 1\} \end{cases}$$

Les solutions optimales du problème

$$(P_\lambda^{Bow}) \begin{cases} Max\ \lambda(6x_1 + 3x_2 + x_3) + (1-\lambda)(x_1 + 3x_2 + 6x_3) \\ \qquad x_1 + x_2 + x_3 \leq 1 \\ \qquad x_1, x_2, x_3 \in \{0,1\}\ avec\ 0 < \lambda < 1 \end{cases}$$

sont des solutions efficaces. Elles sont données par $x^1 = (1,0,0)$ et $x^2 = (0,0,1)$ de vecteurs critères respectifs (6,1) et (1,6). Or il est facile de constater que la solution $x^3 = (0,1,0)$ dont le vecteur critère est (3,3) est également efficace mais n'est pas optimale pour le problème paramétrique (P_λ^{Bow}).

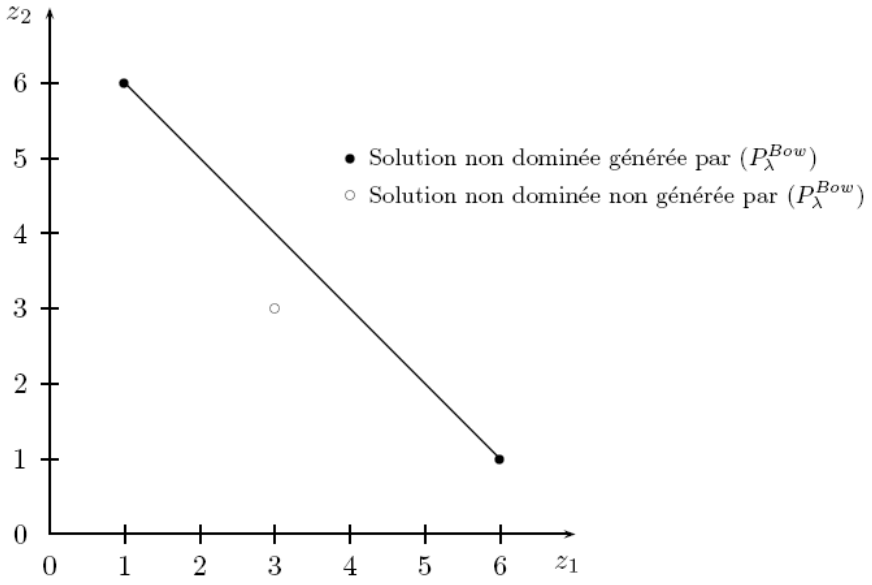

Figure 1.1 – Exemple proposé par Bowman

1.2.3 Détection graphique des solutions efficaces

Pour définir graphiquement la notion d'efficacité, nous avons besoin d'introduire les concepts suivants :

Définition 1.7 (Cône)**.** *Soit $v \in V \subset \mathbb{R}^n, V \neq \emptyset$. Alors, V est un cône si et seulement si $\alpha v \in V$ pour tout scalaire $\alpha \geq 0$. L'origine $0 \in \mathbb{R}^n$ est contenu dans chaque cône.*

Définition 1.8 (Vecteurs générateurs d'un cône)**.** *Soit un ensemble de vecteurs $\{v^1, \ldots, v^k\}$ de $\mathbb{R}^n$ et l'ensemble V tel que :*

$$V = \{v \in \mathbb{R}^n | v = \sum_{i=1}^{k} \alpha_i v^i,\ \alpha_i \geq 0\ \forall i\}.$$

V est l'ensemble de toutes les combinaisons linéaires à coefficients non négatifs des v^i, $i = \overline{1,k}$ et est le cône convexe généré par l'ensemble $\{v^1, \ldots, v^k\}$.

Un générateur $v^r \in \{v^1, \ldots, v^k\}$ est non essentiel si $\{v^1, \ldots, v^k\} \backslash \{v^r\}$ peut générer V. En d'autres termes, c'est celui qui peut s'écrire comme combinaison linéaire à coefficients non négatifs des autres générateurs. Il est essentiel sinon.

Définition 1.9 (Cône polaire semi-positif)**.** *Soit $V \subset \mathbb{R}^n$ un cône convexe généré par $\{v^1, v^2, \ldots, v^k\}$. Le cône convexe $V^>$ donné par :*

$$V^> = \{y \in \mathbb{R}^n \,|\, y^t v^i \geq 0\ \forall\, i \text{ et } y^t v^i > 0 \text{ pour au moins un } i \in \{1, \ldots, k\}\} \cup \{0 \in \mathbb{R}^n\}$$

est le cône polaire semi-positif de V.

Définition 1.10 (Ensemble de dominance)**.** *Soit $\overline{x} \in S$ et $C^>$ le cône polaire semi-positif du cône généré par les gradients des k fonctions objectifs.*
L'ensemble de dominance en $\overline{x}$ est donné par :

$$\begin{aligned} D_{\overline{x}} &= \{\overline{x}\} \oplus C^> \\ &= \{x \in \mathbb{R}^n | x = \overline{x} + y,\ y \in C^>\} \end{aligned}$$

L'ensemble de dominance en $\overline{x}$ contient tous les points dont les vecteurs critères dominent le vecteur critère de $\overline{x}$. Le théorème suivant montre l'importance de cet ensemble dans la détection des solutions efficaces :

Théorème 1.6. *Soit $D_{\overline{x}}$ l'ensemble de dominance en $\overline{x} \in S$. Alors, $\overline{x}$ est efficace si et seulement si $D_{\overline{x}} \cap S = \{\overline{x}\}$.*

> Si l'intersection de l'ensemble de dominance en $\overline{x}$ avec l'ensemble des solutions réalisables S contient uniquement $\overline{x}$, alors $\overline{x}$ est efficace.

1.2.4 Exemple illustratif

Soit à déterminer graphiquement l'ensemble des solutions efficaces du programme linéaire (P) à deux objectifs et deux variables bornées entières suivant :

$$(P)\begin{cases} Max\ z_1 = x_1 \\ Max\ z_2 = -x_1 + 3x_2 \\ \qquad (x_1, x_2) \in S = \{(x_1, x_2) \in \mathbb{Z}^2 | x_1 + 2x_2 \leq 7,\ 0 \leq x_1 \leq 5,\ 0 \leq x_2 \leq 3\} \end{cases}$$

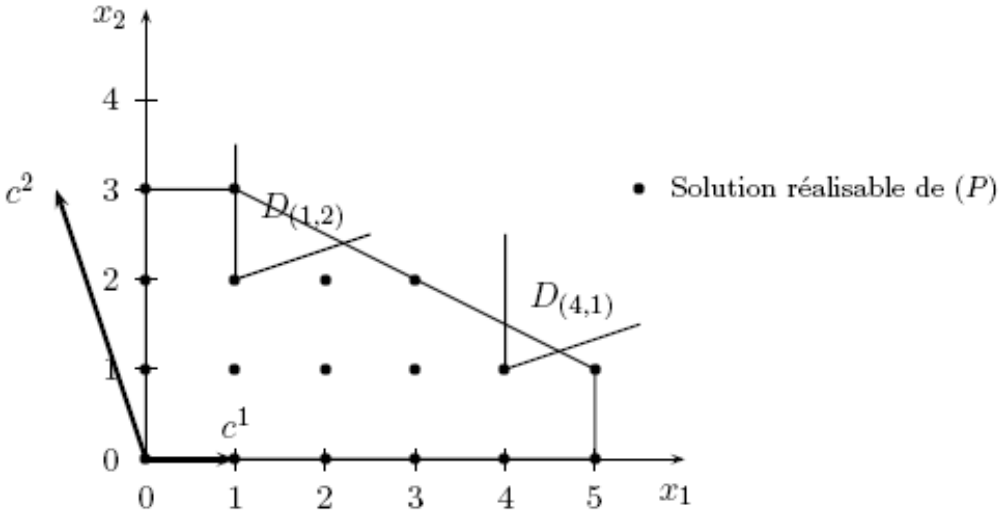

Figure 1.2 – Détection graphique des solutions efficaces de (P)

Dans la figure 1.2, c^1 et c^2 sont les vecteurs gradients des fonctions objectifs z_1 et z_2 respectivement. Pour chaque point (x_1, x_2) de S, $D_{(x_1,x_2)}$ représente l'ensemble de dominance en ce point.

Comme le montre la figure 1.2, la solution réalisable $(1, 2)$ n'est pas efficace pour le problème (P) car $D_{(1,2)} \cap S = \{(1, 2), (1, 3)\} \neq \{(1, 2)\}$. Par contre, la solution $(4, 1)$ est efficace puisque $D_{(4,1)} \cap S = \{(1, 2)\}$. En suivant ce raisonnement, on obtient les solutions efficaces du problème (P) représentées par des carrés blancs et noirs dans la figure 1.3.

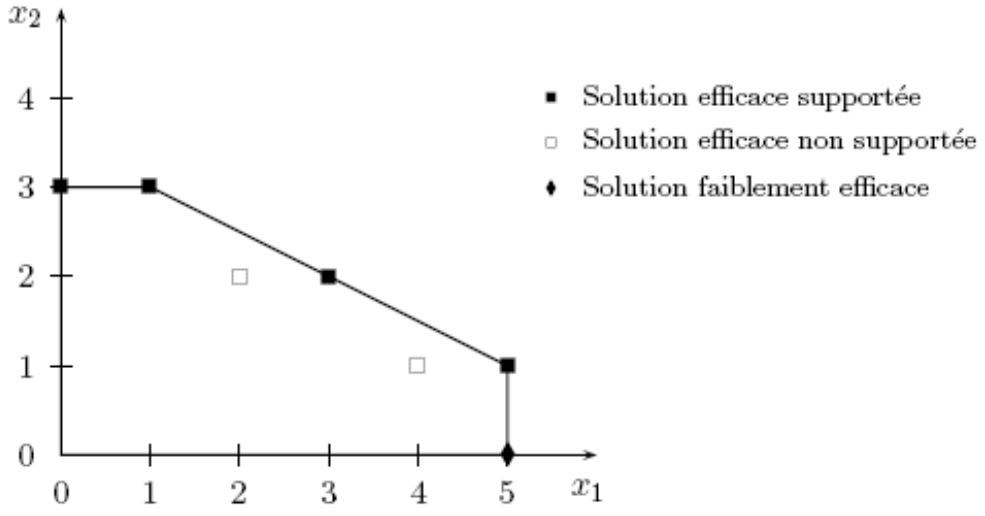

Figure 1.3 – Représentation des différents types de solutions dans l'espace des décisions

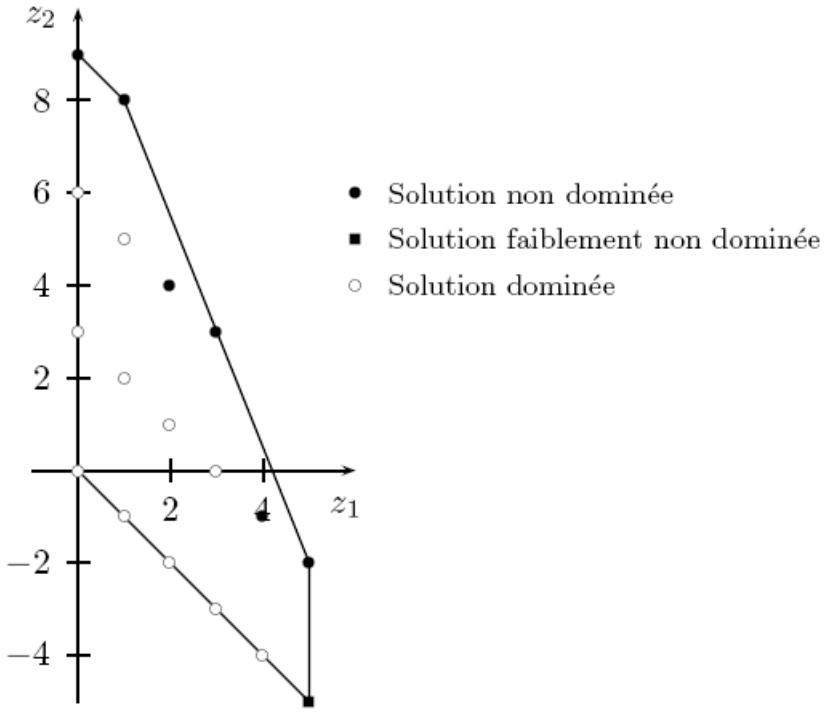

Figure 1.4 – Représentation des différents types de solutions dans l'espace des critères

La figure 1.4 est une illustration de l'ensemble Z, image de l'ensemble S, où on distingue différents types de solutions (dominées, faiblement non dominées et non dominées). Il est clair que la solution $(5, -5)$ est faiblement non dominée puisqu'il n'existe pas de solution $z = (z_1, z_2) \in Z$ telle que $z_1 > 5$.

Ce chapitre est consacré à l'introduction de la programmation linéaire unicritère en nombres entiers ainsi qu'à la programmation linéaire multi-objectif en nombres entiers, en particulier. Les définitions et les concepts de base sont présentés ainsi que les principaux résultats nécessaires à la résolution d'un problème MOILP. Nous avons décrit à la fin l'approche graphique de résolution d'un programme linéaire multi-objectif.

CHAPITRE 2

Quelques méthodes d'optimisation multiobjectif discrète

Résoudre un problème multiobjectif consiste à choisir parmi les solutions efficaces, une solution de meilleur compromis. Dès lors, une étape préalable importante concerne l'optimisation multiobjectif qui recherche les solutions efficaces. Dans ce chapitre, nous donnons une description détaillée de quelques méthodes existantes d'optimisation multiobjectif discrète en illustrant certaines par des exemples.

Considérons le programme linéaire multiobjectif en nombres entiers (P) suivant :

$$(P)\begin{cases} Max & z_1 = c^1 x \\ & \vdots \\ Max & z_k = c^k x \\ & x \in S = \{x \in \mathbb{Z}^n | Ax = b,\, x \geq 0\} \end{cases}$$

Où $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$ et $c^i \in \mathbb{R}^{1\times n}$, $i = \overline{1,k}$.

Dans les paragraphes 2.1, 2.3, 2.4 et 2.5, les paramètres coûts du problème (P) sont supposés entiers, c'est-à-dire $c^i \in \mathbb{Z}^{1\times n}$, pour $i = \overline{1,k}$.

Notons par :

- D l'ensemble des solutions réalisables du problème relaxé de (P).
  $D = \{x \in \mathbb{R}^n | Ax = b,\ x \geq 0\}$;
- $ESE(P)$ (resp. $SND(P)$) l'ensemble de toutes les solutions efficaces (resp. non dominées) de (P).

2.1 Méthode de Klein et Hannan [KH82]

Nous présentons ci-dessous une technique proposée par D. Klein et E. Hannan pour générer l'ensemble $ESE(P)$ ou une partie de ce dernier :

Étape 0 **:**

Choisir arbitrairement un critère l parmi les k critères du problème général (P) et résoudre le problème unicritère (P_0) suivant :

$$(P_0) \quad Max \left\{ c^l x | x \in S \right\}.$$

Trouver toutes les solutions optimales de (P_0) et retenir celles dont les vecteurs critères sont non dominés formant l'ensemble $\Theta(P_0)$.

Étape j $(j \geq 1)$ **:**

Soient $(x^p : p = \overline{1,r})$ les solutions efficaces trouvées aux étapes précédentes. On

résout le problème unicritère (P_j) définit comme suit :

$$(P_j)\begin{cases} Max & c^l x \\ & x \in S \\ & \bigcap_{p=1}^{r}\left(\bigcup_{i=1,i\neq l}^{k} c^i x \geq c^i x^p + f_i\right) \end{cases}$$

Où $f_i \geq 1$ et entier pour $i = \overline{1,k}$.

Le problème (P_j) est obtenu à partir de (P_0) en lui ajoutant des contraintes qui assurent que les solutions de (P_j) seront meilleures que toutes les solutions $(x^p : p = \overline{1,r})$ pour au moins un critère $i \neq l$.

Construire l'ensemble $\Theta(P_j)$ des solutions optimales de (P_j) dont les vecteurs critères sont non dominés.

La procédure s'arrête à l'étape (t) où le problème (P_t) est irréalisable.

Les auteurs de la méthode ont démontré que :

- $\Theta(P_j)$ constitue une partie de $ESE(P)$ pour tout j ;
- Si $\forall i,\ f_i = 1$, alors $\bigcup_{j=0}^{t-1} \Theta(P_j) = ESE(P)$.

Remarque

Il est important de noter que le nombre de sous-problèmes sous-jacents considérés à chaque étape j $(j \geq 1)$ est donné par $(k-1)^r$ où k est le nombre de critères du problème et r est le nombre de solutions efficaces trouvées aux étapes précédentes (c'est-à-dire de 0 à $(j-1)$).

Exemple illustratif

Considérons le programme linéaire multiobjectif en nombres entiers (P) suivant :

$$(P)\begin{cases} Max\ z_1 = & x_2 \\ Max\ z_2 = -x_1 + 2x_2 \\ Max\ z_3 = & 2x_1 + x_2 \\ & (x_1, x_2) \in S \end{cases}$$

Où $S = \{(x_1, x_2) \in \mathbb{Z}^2 \mid x_1 + x_2 \leq 3,\ x_2 \leq 2,\ x_1 - 2x_2 \leq 0,\ x_1 \geq 0,\ x_2 \geq 0\}$.

Prenons $l = 1$ et $f_i = 1$ pour tout i.

Étape (0)

La résolution de $(P_0) : Max\ \{z_1 = x_2 | (x_1, x_2) \in S\}$ donne le tableau optimal suivant :

B	x_1	x_4	x_B
x_3	1	-1	1
x_2	0	1	2
x_5	1	2	4
$-z_1$	0	-1	-2

Tableau 2.1

(P_0) admet deux solutions optimales $x^1 = (0, 2)$ et $x^2 = (1, 2)$ dont les vecteurs critères sont respectivement $(2, 4, 2)$ et $(2, 3, 4)$. Ces derniers sont non dominés, donc $\Theta(P_0) = \{(0, 2), (1, 2)\}$.

Étape (1)

On considère le problème (P_1) suivant :

$$(P_1)\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & -x_1 + 2x_2 \geq 5 \quad ou \quad 2x_1 + x_2 \geq 3 \\ & -x_1 + 2x_2 \geq 4 \quad ou \quad 2x_1 + x_2 \geq 5 \end{cases}$$

Pour trouver les solutions optimales de (P_1), il faut résoudre les quatre sous-problèmes suivants :

$$(P_{11})\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & -x_1 + 2x_2 \geq 5 \\ & -x_1 + 2x_2 \geq 4 \end{cases} \qquad (P_{12})\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & -x_1 + 2x_2 \geq 5 \\ & 2x_1 + x_2 \geq 5 \end{cases}$$

$$(P_{13})\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & 2x_1 + x_2 \geq 3 \\ & -x_1 + 2x_2 \geq 4 \end{cases} \qquad (P_{14})\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & 2x_1 + x_2 \geq 3 \\ & 2x_1 + x_2 \geq 5 \end{cases}$$

Les trois premiers sous-problèmes sont irréalisables et le quatrième possède une unique solution optimale $x^3 = (2, 1)$ qui est en fait l'unique solution optimale de (P_1). Le vecteur critère de celle-ci est $(1, 0, 5)$. Donc, $\Theta(P_1) = \{(2, 1)\}$.

Étape (2)

Considérons le problème (P_2) suivant :

$$(P_2)\begin{cases} Max & z_1 = x_2 \\ & (x_1, x_2) \in S \\ & -x_1 + 2x_2 \geq 5 \quad ou \quad 2x_1 + x_2 \geq 3 \\ & -x_1 + 2x_2 \geq 4 \quad ou \quad 2x_1 + x_2 \geq 5 \\ & -x_1 + 2x_2 \geq 1 \quad ou \quad 2x_1 + x_2 \geq 6 \end{cases}$$

Le problème (P_2) est irréalisable (les huit sous-problèmes sous-jacents ne possèdent pas de solutions). Terminer.
Finalement, $ESE(P) = \bigcup_{j=0}^{1} \Theta(P_j) = \{(0,2),(1,2),(2,1)\}$.

2.2 Méthode de Gupta et Malhotra (1ère procédure) [GM92]

Dans ce paragraphe, nous avons besoin d'introduire la définition suivante : Soient le programme linéaire $(Q) : Max\{z = cx | x \in D\}$, B une base de (Q), $\overline{x} = (\overline{x}_i)_{i=\overline{1,n}}$ la solution de base correspondant à B et lig l'application définie par :

$$\begin{array}{rccl} lig: & \{1,\ldots,n\} & \longrightarrow & \{1,\ldots,m\} \\ & i & \longmapsto & lig(i) = \text{ligne correspondant à la variable de base } x_i \end{array}$$

Une arête E_j incidente à la solution $\overline{x}$ est définie comme étant l'ensemble :

$$E_j = \left\{ x = (x_i) \in D \;\middle|\; \begin{array}{l} x_i = \overline{x}_i - \theta_j \widehat{A}^j_{lig(i)},\ i \in B \\ x_j = \theta_j \\ x_l = 0,\ l \in N \backslash \{j\} \end{array} \right\}$$

où $\widehat{A}^j = (A^B)^{-1} A^j$, $N = \{1, 2, \ldots, n\} \backslash B$ et $0 \leq \theta_j \leq \min\limits_{i \in B} \left\{ \dfrac{\overline{x}_i}{\widehat{A}^j_{lig(i)}},\ \widehat{A}^j_{lig(i)} > 0 \right\}$.

Remarque

Pour déterminer les solutions entières sur l'arête E_j incidente à une solution réalisable entière $\overline{x}$, il faut rajouter la condition : θ_j est un entier positif et $\theta_j \widehat{A}^j_{lig(i)}$ sont des entiers pour tout $i \in B$.

Notations

À une solution de base x^p, optimale pour le problème unicritère suivant :

$$Max \left\{ c^1_p x | A_p x = b_p,\ x \geq 0 \right\}$$

on associe :

B_p (resp. N_p) : ensemble des indices des variables de base (resp. hors base) de x^p ;

$\widehat{c}^i_p$: vecteur coût réduit relatif au critère i, $i = \overline{1, k}$, donné par :

$$\widehat{c}^i_p = c^i_p - (c^i_p)^{B_p} (A^{B_p}_p)^{-1} A_p$$

Γ_p : l'ensemble défini comme suit :

$$\Gamma_p = \left\{ j \in N_p \mid (\widehat{c}_p^1)^j < 0 \text{ et } (\widehat{c}_p^i)^j > 0 \text{ pour au moins un } i \in \{2, \ldots, k\} \right\}$$

où $(\widehat{c}_p^i)^j$ est la $j^{ème}$ composante du vecteur $\widehat{c}_p^i$, $i = \overline{1,k}$.

La méthode de Gupta et Malhotra construit progressivement l'ensemble $SND(P)$ en se basant sur la technique des coupes. Elle procède comme suit :

Étape 1 **:**

Résoudre le problème (P_1) : $Max\{z_1 = c^1 x | x \in S\}$ en utilisant la méthode duale fractionnaire de Gomory. Au lieu de (P_1), on peut construire d'une manière similaire l'un des problèmes (P_i), $i = \overline{2,k}$ et le résoudre.

1.1 Si la solution optimale de (P_1) est unique, soit $x^1 = (x_i^1)$, déterminer le vecteur critère $(z_1^1, \ldots, z_k^1)$ correspondant à x^1 et former l'ensemble Z_0 contenant ce k-uplet.

1.2 Si le problème (P_1) admet plusieurs solutions optimales, déterminer toutes ces solutions ainsi que le vecteur critère correspondant à chacune d'elles. Eliminer les vecteurs critères dominés par comparaison deux à deux et garder seulement ceux non dominés formant l'ensemble Z_0. Soient $(z_1^1, \ldots, z_k^1)$ le premier k-uplet de Z_0 dans l'ordre lexicographique (c'est-à-dire celui qui a la plus grande valeur de z_2, si toutes les secondes composantes sont identiques, on choisit celui qui a la plus grande valeur de z_3 et ainsi de suite) et $x^1 = (x_i^1)$ la solution correspondante.

Étape 2 **:**

Choisir $j_1 \in \Gamma_1$ et calculer le nombre $\theta = \min\limits_{i \in B_1} \left\{ \dfrac{x_i^1}{(\widehat{A}_1)_{lig(i)}^{j_1}},\ (\widehat{A}_1)_{lig(i)}^{j_1} > 0 \right\}$ correspondant à la solution x^1.

2.1 Si $\theta < 1$, alors aucune solution entière ne peut être obtenue sur l'arête E_{j_1}.

2.2 Si $\theta \geq 1$, déterminer toutes les solutions entières se trouvant sur l'arête E_{j_1} et former l'ensemble Z_1 des vecteurs critères correspondant à ces solutions.

Eliminer l'arête E_{j_1} par la coupe $\sum_{j \in N_1 \setminus \{j_1\}} x_j \geq 1$ et appliquer la méthode duale du simplexe et les coupes de Gomory si nécessaire pour obtenir une solution réalisable entière x^2 dans la région tronquée. Ajouter le vecteur critère de celle-ci à Z_1 s'il n'est pas dominé par l'un des k-uplets retenus précédemment.

Dans le cas où pour tout $j \in \Gamma_1$, $\theta < 1$, choisir n'importe quel indice $j_1 \in \Gamma_1$ et appliquer la coupe $\sum_{j \in N_1 \setminus \{j_1\}} x_j \geq 1$.

Étape générale p $(p \geq 3)$ **:**

Choisir $j_{p-1} \in \Gamma_{p-1}$. Explorer l'arête $E_{j_{p-1}}$ pour d'éventuelles solutions entières et déterminer les vecteurs critères correspondants (en cas d'existence). Former l'ensemble Z'_{p-1} à partir de Z_{p-2} en lui ajoutant ces nouveaux k-uplets. Eliminer tous les k-uplets dominés de Z'_{p-1} pour obtenir l'ensemble Z_{p-1}. Eliminer ensuite l'arête $E_{j_{p-1}}$ par la coupe $\sum_{j \in N_{p-1} \setminus \{j_{p-1}\}} x_j \geq 1$ et déterminer une solution optimale entière x^p dans la région tronquée.

Étape finale $(n+1)$ **:**

La procédure prend fin à l'étape $(n+1)$ dans laquelle :

(1) $\Gamma_n = \emptyset$ et $(\widehat{c}_n^1)^j < 0,\ \forall j \in N_n$

ou (2) $\Gamma_n \neq \emptyset$ mais pour tout $j \in \Gamma_n$, toutes les solutions entières se trouvant sur l'arête E_j sont inefficaces.

Finalement, l'ensemble $SND(P)$ est la réunion des ensembles Z_0 et Z_{n-1}.

D'après Gupta et Malhotra, cette méthode permet de générer toutes les solutions efficaces d'un programme linéaire multiobjectif en nombres entiers. Malheureusement ce n'est pas toujours le cas, ceci est dû au second critère d'arrêt posé par les auteurs. Moulaï [Mou02a] et Chaabane [Cha05] ont présenté un contre exemple illustrant l'erreur de ce dernier et ont apporté des rectifications et des améliorations à celle-ci.

Nous donnons ci après un autre contre exemple pour la méthode.

Considérons le programme linéaire multiobjectif en nombres entiers (P) suivant :

$$(P)\begin{cases} Max\ z_1 = x_1 + 3x_2 \\ Max\ z_2 = x_1 - x_2 \\ x_1 + 2x_2 \leq 7 \\ x_1 \leq 5 \\ -x_1 + x_2 \leq 2 \\ x_i \geq 0\ entier,\ i = \overline{1,2} \end{cases}$$

Étape (1)

La résolution de (P_1) donne le tableau optimal suivant :

B	x_3	x_5	x_B
x_1	$\frac{1}{3}$	$-\frac{2}{3}$	1
x_4	$-\frac{1}{3}$	$\frac{2}{3}$	4
x_2	$\frac{1}{3}$	$\frac{1}{3}$	3
$-z_1$	$-\frac{4}{3}$	$-\frac{1}{3}$	-10
$-z_2$	0	1	2

Tableau 2.2

La solution de (P_1), $x^1 = (1, 3)$ est unique et le premier couple non dominé est $(z_1^1, z_2^1) = (10, -2)$. $Z_0 = \{(10, -2)\}$.
$N_1 = \{3, 5\}$, $\Gamma_1 = \{5\}$.

Étape (2)

$j_1 = 5$, $\theta = 6 > 1$. Les solutions entières se trouvant sur l'arête E_5 sont $(3, 2)$ et $(5, 1)$. Les vecteurs critères correspondant à ces solutions sont respectivement $(9, 1)$ et $(8, 4)$. $Z_1 = \{(9, 1), (8, 4)\}$.
L'arête E_5 sera donc tronquée par $x_3 \geq 1 \Rightarrow -x_3 + x_6 = -1$. En ajoutant cette contrainte au tableau précédent et en appliquant la méthode duale du simplexe et des coupes de Gomory, on obtient le tableau optimal suivant :

B	x_6	x_7	x_B
x_1	1	-2	2
x_4	-1	2	3
x_2	0	1	2
x_3	-1	0	1
x_5	1	-3	2
$-z_1$	-1	-1	-8
$-z_2$	-1	3	0

Tableau 2.3

$x^2 = (2, 2)$ et $(z_1^2, z_2^2) = (8, 0)$ qui est dominé par rapport aux couples retenus précédemment. $N_2 = \{6, 7\}$, $\Gamma_2 = \{7\}$.

Étape (3)

$j_2 = 7$, l'arête E_7 contient un seul point entier, le point $(4, 1)$. Ceci donne le couple dominé $(7, 3)$. $\Gamma_2 \neq \emptyset$ et toutes les solutions entières se trouvant sur l'arête E_7 sont inefficaces ; d'après Gupta et Malhotra, la procédure s'arrête en donnant l'ensemble des solutions efficaces $ESE(P) = \{(1, 3), (3, 2), (5, 1)\}$ et celui des solutions non dominées $SND(P) = \{(10, -2), (9, 1), (8, 4)\}$. Il se trouve que le point $(5, 0)$ dont le vecteur critère est $(5, 5)$ est efficace pour le problème (P) et pourtant, il n'a pas été généré par la méthode.

Conclusion

Le second critère d'arrêt, à savoir, $\Gamma_n \neq \emptyset$ mais pour tout $j \in \Gamma_n$, toutes les solutions entières se trouvant sur l'arête E_j sont inefficaces, n'est pas correct. Si ce dernier n'est pas pris en considération, la méthode permet effectivement de générer toutes les solutions efficaces ainsi que toutes les solutions non dominées du problème.

2.3 Méthode de Gupta et Malhotra (2ème procédure) [GM92]

Cette méthode est une version améliorée de celle de Klein et Hannan [KH82]. Elle permet de réduire considérablement le nombre de contraintes additionnelles à chaque étape.

Étape 0 **:**

Résoudre le problème (P_0) : $Max\,\{z_1 = c^1 x | x \in S\}$.

Trouver toutes les solutions optimales de (P_0) et retenir celles dont les vecteurs cri-

tères sont non dominés. Notons par $\Theta(P_0)$ l'ensemble de toutes les solutions efficaces trouvées à l'étape (0).

Étape j :

Choisir une solution $x^* \in \Theta(P_{j-1})$, ensemble de toutes les solutions potentiellement efficaces obtenues à l'étape $(j-1)$ telle que $z_1(x^*) = \max_{x \in \Theta(P_{j-1})} \{c^1 x\}$ et résoudre le problème (P_j) suivant :

$$(P_j) \begin{cases} Max & z_1 = c^1 x \\ & x \in S \\ & c^1 x \leq c^1 x^* - 1 \\ & c^i x \geq c^i x^* + 1 \quad \textit{pour au moins un } i \in \{2, \ldots, k\} \end{cases}$$

Former l'ensemble $\Theta(P_j)$ de toutes les solutions potentiellement efficaces obtenues à l'étape (j).

Étape finale n :

La procédure prend fin à l'étape (n) dans laquelle toutes les solutions du problème (P_n) sont inefficaces pour (P) ou le problème (P_n) est irréalisable.

Les méthodes qui vont suivre permettent de générer toutes les solutions non dominées et pas nécessairement toutes les solutions efficaces de (P), c'est-à-dire au moins une solution efficace pour chaque point non dominé.

2.4 Méthode de Abbas et Moumene [Mou02b]

La méthode de Abbas et Moumene se présente comme suit :

Initialisation :

La première étape de cette méthode consiste à éliminer les vecteurs critères non essentiels. Pour ce faire, résoudre pour chaque $l \in \{1, \ldots, k\}$, le programme linéaire (Rd_l) suivant :

$$(Rd_l) \begin{cases} Min & \varphi_l = \sum_{j=1}^{n} v_j \\ & \sum_{i=1, i\neq l}^{k} c_j^i y_i + v_j = c_j^l \quad j = \overline{1,n} \\ & y_i \geq 0 \quad i = \overline{1,k},\ i \neq l \\ & v_j \geq 0 \quad j = \overline{1,n} \end{cases}$$

Si $\varphi_l = 0$, supprimer le $l^{ème}$ critère du problème (P).

Réordonner les k' vecteurs critères restants (essentiels).

Créer une liste F des problèmes non encore résolus et l'initialiser à (P_0), le programme linéaire unicritère à variables entières suivant :

$$(P_0) \quad Max \left\{ \sum_{i=1}^{k'} \lambda_i c^i x | x \in S \right\}.$$

Où $\lambda_i > 0$ pour $i = \overline{1,k'}$.

Étape générale :

Si la liste F est vide, terminer, l'ensemble des solutions non dominées $SND(P)$ est entièrement déterminé. Sinon, résoudre le problème (P_r) de plus fort indice r dans F.

Si (P_r) n'admet pas de solution, faire $F \leftarrow F \setminus \{P_r\}$.

Sinon, soit x^r la solution optimale de (P_r), celle-ci est efficace pour (P). Former k'

problèmes identiques au problème (P_r) et les noter $(P_{r+1}), \ldots, (P_{r+k'})$. Ajouter alors à (P_{r+1}), la contrainte $c^1 x \geq c^1 x^r + 1$ et à (P_{r+i_0}), $i_0 = \overline{2, k'}$, le système

$$\begin{cases} c^{i_0} x \geq c^{i_0} x^r + 1 \\ c^{i_1} x \leq c^{i_1} x^r, \ i_1 = \overline{1, (i_0 - 1)} \end{cases}$$

Actualiser la liste F ($F \leftarrow F \cup \{P_{r+1}, \ldots, P_{r+k'}\} \setminus \{P_r\}$).

Contrairement à ce qu'il a été affirmé par les auteurs de cette méthode, une solution optimale d'un problème (P_r) n'est pas nécessairement efficace pour le problème initial (P) comme nous allons le voir à travers l'exemple ci-dessous.

Soit à résoudre le problème (P) suivant :

$$(P) \begin{cases} Max\ z_1 = x_1 \\ Max\ z_2 = \quad x_2 \\ Max\ z_3 = \quad\quad x_3 \\ \quad\quad (x_1, x_2, x_3) \in S \end{cases}$$

Où $S = \{(x_1, x_2, x_3) \in \mathbb{Z}^3 \mid x_1 + 2x_2 \leq 6,\ x_2 + 2x_3 \leq 6,\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0\}$.
Dans cet exemple, les solutions réalisables et les vecteurs critères correspondants sont confondus.
Prenons $\lambda_1 = 1$, $\lambda_2 = 3$ et $\lambda_3 = 2$.

Soit (P_0) le problème : $Max\,\{z = x_1 + 3x_2 + 2x_3 \mid (x_1, x_2, x_3) \in S\}$ et $F \leftarrow \{P_0\}$.

Étape (1)

On sélectionne le problème (P_0). La résolution de ce dernier donne le point optimal $(2, 2, 2)$.
On génère trois sous-problèmes (P_1), (P_2) et (P_3) en ajoutant respectivement au

problème (P_0), les systèmes de contraintes suivants :

$$\left\{ x_1 \geq 3 \right. \quad , \quad \left\{ \begin{array}{l} x_2 \geq 3 \\ x_1 \leq 2 \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x_3 \geq 3 \\ x_1 \leq 2 \\ x_2 \leq 2 \end{array} \right.$$

$F \leftarrow \{P_1, P_2, P_3\}$.

Étape (2)

On sélectionne le problème (P_3). La résolution de ce dernier donne le point optimal $(2, 0, 3)$.

On génère trois sous-problèmes (P_4), (P_5) et (P_6) en ajoutant respectivement au problème (P_3), les systèmes de contraintes suivants :

$$\left\{ x_1 \geq 3 \right. \quad , \quad \left\{ \begin{array}{l} x_2 \geq 1 \\ x_1 \leq 2 \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x_3 \geq 4 \\ x_1 \leq 2 \\ x_2 \leq 0 \end{array} \right.$$

$F \leftarrow \{P_1, P_2, P_4, P_5, P_6\}$.

Étape (3)

On sélectionne le problème (P_6). Ce dernier est irréalisable. $F \leftarrow \{P_1, P_2, P_4, P_5\}$.

Étape (4)

On sélectionne le problème (P_5). Ce dernier est irréalisable. $F \leftarrow \{P_1, P_2, P_4\}$.

Étape (5)

On sélectionne le problème (P_4). Ce dernier est irréalisable. $F \leftarrow \{P_1, P_2\}$.

Étape (6)

On sélectionne le problème (P_2). La résolution de ce dernier donne le point optimal $(0, 3, 1)$.

On génère trois sous-problèmes (P_3), (P_4) et (P_5) en ajoutant respectivement au problème (P_2), les systèmes de contraintes suivants :

$$\left\{ x_1 \geq 1 \right. \quad , \quad \left\{ \begin{array}{l} x_2 \geq 4 \\ x_1 \leq 0 \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x_3 \geq 2 \\ x_1 \leq 0 \\ x_2 \leq 3 \end{array} \right.$$

$F \leftarrow \{P_1, P_3, P_4, P_5\}$.

Étape (7)

On sélectionne le problème (P_5). Ce dernier est irréalisable. $F \leftarrow \{P_1, P_3, P_4\}$.

Étape (8)

On sélectionne le problème (P_4). Ce dernier est irréalisable. $F \leftarrow \{P_1, P_3\}$.

Étape (9)

On sélectionne le problème (P_3). Ce dernier est irréalisable. $F \leftarrow \{P_1\}$.

Étape (10)

On sélectionne le problème (P_1). La résolution de ce dernier donne le point optimal $(6, 0, 3)$.

On génère trois sous-problèmes (P_2), (P_3) et (P_4) en ajoutant respectivement au problème (P_1), les systèmes de contraintes suivants :

$$\left\{ x_1 \geq 7 \right. \quad , \quad \left\{ \begin{array}{l} x_2 \geq 1 \\ x_1 \leq 6 \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x_3 \geq 4 \\ x_1 \leq 6 \\ x_2 \leq 0 \end{array} \right.$$

$F \leftarrow \{P_2, P_3, P_4\}$.

Étape (11)

On sélectionne le problème (P_4). Ce dernier est irréalisable. $F \leftarrow \{P_2, P_3\}$.

Étape (12)

On sélectionne le problème (P_3). La résolution de ce dernier donne le point optimal $(4, 1, 2)$.

On génère trois sous-problèmes (P_4), (P_5) et (P_6) en ajoutant respectivement au problème (P_3), les systèmes de contraintes suivants :

$$\left\{ x_1 \geq 5 \right. \quad , \quad \left\{ \begin{array}{l} x_2 \geq 2 \\ x_1 \leq 4 \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x_3 \geq 3 \\ x_1 \leq 4 \\ x_2 \leq 1 \end{array} \right.$$

$F \leftarrow \{P_2, P_4, P_5, P_6\}$.

Étape (13)

On sélectionne le problème (P_6). Ce dernier est irréalisable. $F \leftarrow \{P_2, P_4, P_5\}$.

Étape (14)

On sélectionne le problème (P_5). Ce dernier est irréalisable. $F \leftarrow \{P_2, P_4\}$.

Étape (15)

On sélectionne le problème (P_4). Ce dernier est irréalisable. $F \leftarrow \{P_2\}$.

Étape (16)

On sélectionne le problème (P_2). Ce dernier est irréalisable. $F \leftarrow \emptyset$. Terminer.

Finalement, les solutions trouvées par la méthode sont : $(2,2,2)$, $(2,0,3)$, $(0,3,1)$, $(6,0,3)$ et $(4,1,2)$. Il est clair que la solution $(2,0,3) \notin SND(P)$ puisqu'elle est dominée par la solution $(6,0,3)$. Il sera donc plus judicieux de comparer entre toutes les solutions générées afin de trouver l'ensemble de toutes les solutions non dominées du problème.

2.5 Méthode de Sylva et Crema [SC04]

Cette méthode est une variante de celle de Klein et Hannan [KH82], maximisant à chaque étape non pas un des critères du problème mais une combinaison positive de toutes les fonctions objectifs. Ainsi, la solution optimale de chaque programme linéaire en nombres entiers résolu est efficace pour le problème multiobjectif.
Les résultats suivants permettent de justifier la méthode :

Proposition 2.1. *Soient $x^1, \ldots, x^r$ des solutions efficaces pour le problème (P) et $\Delta_l = \left\{x \in \mathbb{Z}^n | Cx \leq Cx^l\right\}$ où C est une $k \times n$-matrice composée des vecteurs lignes c^i, $i = \overline{1,k}$. Soit x^* une solution efficace pour le problème (P_r) suivant :*

$$(P_r) : \text{“}Max\text{”} \left\{ Cx | x \in S - \bigcup_{l=1}^{r} \Delta_l \right\}.$$

Alors x^* est une solution efficace pour (P).
De plus, si le problème (P_r) est irréalisable, alors $\left\{Cx^l\right\}_{l=1}^{r}$ est l'ensemble de tous les vecteurs critères non dominés de (P).

Corollaire 2.1. *Soient $x^1, \ldots, x^r$ des solutions efficaces pour le problème (P) et*

$\Delta_l = \left\{x \in \mathbb{Z}^n | Cx \leq Cx^l\right\}$. *Si* x^* *est une solution optimale pour le problème*

$$(P_r^\lambda) : \text{“}Max\text{”} \left\{\lambda^t Cx | x \in S - \bigcup_{l=1}^{r} \Delta_l\right\}$$

pour certaines valeurs de $\lambda \in \mathbb{R}^k$, $\lambda > 0$, *alors elle est efficace pour* (P).

Description de la méthode

Étape 0 **:**

Après avoir choisi le paramètre λ, on résout le problème $(P_0) : Max\{\lambda^t Cx | x \in S\}$. Si ce problème est irréalisable, il en est de même pour (P). Sinon, une solution optimale x^1 est trouvée ; celle ci est efficace pour le problème (P).

Étape r $(r \geq 1)$ **:**

Résoudre le problème :

$$(P_r) \begin{cases} Max & \lambda^t Cx \\ & x \in S \\ & (Cx)_i \geq ((Cx^l)_i + 1)y_i^l - M_i(1 - y_i^l) \quad \forall\, l = \overline{1,r}\ ;\ i = \overline{1,k} \\ & \sum_{i=1}^{k} y_i^l \geq 1 \quad \forall\, l = \overline{1,r} \\ & y_i^l \in \{0,1\} \quad \forall\, l = \overline{1,r}\ ;\ i = \overline{1,k} \end{cases}$$

où $-M_i$ est un minorant pour toute valeur réalisable de la $i^{ème}$ fonction objectif. Dans le cas où les éléments de C sont tous positifs, M_i peut être fixé à 0 pour $i = \overline{1,k}$. Si (P_r) est irréalisable, terminer. $\left\{Cx^l\right\}_{l=1}^{r}$ est l'ensemble de tous les vecteurs critères non dominés de (P).

Sinon, une nouvelle solution efficace pour (P), notée x^{r+1}, est trouvée.

Pour des problèmes de grande taille, l'énumération de toutes les solutions non dominées n'est plus envisageable. Néanmoins, un sous-ensemble de ces dernières peut être obtenu en remplaçant le problème (P_r) par :

$$\begin{cases} Max & \lambda^t Cx \\ & x \in S \\ & (Cx)_i \geq ((Cx^l)_i + f_i)y_i^l - M_i(1 - y_i^l) \quad \forall\, l = \overline{1,r}\ ;\ i = \overline{1,k} \\ & \sum_{i=1}^{k} y_i^l \geq 1 \quad \forall\, l = \overline{1,r} \\ & y_i^l \in \{0,1\} \quad \forall\, l = \overline{1,r}\ ;\ i = \overline{1,k} \end{cases}$$

Où f_i est un entier supérieur à 1, représentant l'amélioration minimale dans le $i^{ème}$ critère pour $i = \overline{1,k}$.

2.6 Méthode de Chergui et Al [CAM12] : Génération des solutions entières non dominées du problème multi-objectif linéaire

Cette méthode a été présentée par M.E.A Chergui dans la section (4) de sa thèse de doctorat [Che10], elle génère l'ensemble de toutes les solutions entières non dominées d'un programme linéaire multiobjectif en nombres entiers (MOILP). C'est une méthode branch and bound qui fait appel à des coupes efficaces dans l'espace des critères ; c'est-à-dire des contraintes faisant intervenir les critères. On fait appel à la méthode du simplexe pour résoudre le programme linéaire (P_l) à l'étape l de la méthode. Les vecteurs critères sont alors adjoints au tableau du simplexe et évoluent de façon dynamique dans ce dernier de la même façon que le critère z de (P_l). Si la

solution optimale obtenue n'est pas entière, ce qui correspond à un noeud de type 1 de l'arbre, seul un processus de branchement est effectué pour détecter une solution entière.

Quand une solution entière est obtenue, le noeud est de type 2, le vecteur critère correspondant est comparé à ceux déjà trouvés et l'ensemble des solutions potentiellement non dominées est mis à jour.

En vue de rechercher une autre solution entière, les directions d'amélioration des critères sont utilisés pour construire des coupes efficaces permettant d'éviter l'exploration de domaines ne contenant que des solutions entières dominées.

Notations

Soit (P) le programme linéaire multi-objectif à variables entières (MOILP) :

$$(P)\begin{cases} Max & Cx \\ & x \in X \\ & x\ vecteur\, entier \end{cases}$$

L'ensemble des vecteurs critères réalisables Y du problème MOILP est défini comme suit :

$$Y = \left\{ z \in \mathbb{R}^k | z^i = c^i x, x \in X \right\} = f(X) \tag{2.1}$$

Soit $x_l \in X$ une solution entière du problème (P), on définie les ensembles suivants en x_l et relativement à chaque critère i ; $i = 1, \ldots, k$, :

$$H_l^i = \left\{ j \in N_l | \hat{c}_j^i > 0 \right\} \tag{2.2}$$

où N_j est l'ensemble des indices des variables hors-base et $\hat{c}_j^i$ est la $j$ème composante

du coût réduit du vecteur critère i.

$$K_l = \left\{ i \in \{1, \ldots, k\} \,|H_l^i \neq \emptyset \right\} \tag{2.3}$$

indique l'ensemble des critères pouvant être améliorés à partir de x_l.

Une contrainte est une coupe efficace pour le problème (P) dans l'espace des critères, si son adjonction n'élimine pas de solutions entières non dominées de l'ensemble Y. A partir du tableau du simplexe au point x_l ; nous avons la relation suivante :

$$c^i x = c^i x^l + \sum_{j \in N_l} \hat{c}_j^i x_j, \forall i \in \{1, \ldots, k\} \tag{2.4}$$

et sous l'hypothèse que la matrice C est à coefficients entiers, on en déduit la contrainte suivante pour chaque critère $i \in K_l$:

$$c^i x \geq c^i x^l + \sum_{j \in N_l \setminus H_l^i} \left\lfloor \hat{c}_j^i \right\rfloor x_j + \max \left\{ 1, \left\lfloor \hat{c}_{j0}^i \right\rfloor \right\} \tag{2.5}$$

où $\hat{c}_{j0}^i = \min_{j \in H_i} \left\{ \hat{c}_j^i \right\}$ pour $i \in K_l$. cette contrainte définie une coupe efficace dans l'espace des critères pour le problème (P).

La méthode de Chergui et Al peut être décrite comme suit :

Étape 0 **(initialisation) :**

l'ensemble des solutions non dominées (SND) de (P)est vide.

Résoudre le programme linéaire (P_0).

Poser x la solution optimale trouvée.

Étape r $(r \geq 1)$ **:**

Résoudre le programme linéaire (P_l), $0 \leq l < r$,

Considérer une solution optimale x^l de (P_l),

Si x^l n'est pas entière, Choisir une coordonée j de x dont la valeur n'est pas entière et séparer le noeud l sur cette coordonnées en deux nouveaux noeuds et revenir à l'étape r.

Si x^l est entière alors :

- Si le vecteur critère correspondant Cx^l n'est dominé par aucun autre vecteur Z appartenant à l'ensemble des solutions non dominées (SND) alors mettre à jour ce dernier en lui rajoutant le vecteur Cx^l.
- S'il existe une solution $Z \in SND$ qui est dominée par Cx^l alors remplacer Z par Cx^l dans l'ensemble des solutions non dominées.

Déterminer l'ensemble K_l des critères pouvant être améliorés à partir de x_l, pour tout $i \in K_l$ rajouter la contrainte 2.5 pour obtenir $|K_l|$ nouveaux programmes linéaires $(P_k), k > l$ et revenir à l'étape r.

Critère d'arrêt :

La procédure s'arrête dès que $K_l = \emptyset$ c'est-à-dire aucun critère ne peut être amélioré ou bien (P_l) n'admet pas de solutions réalisables pour toute étape l telle que $0 \leq l \leq r$.

Exemple illustratif

On considère le problème $MOILP$ suivant :

$$(P)\begin{cases} Max\ z_1 = x_1 + 3x_2 \\ Max\ z_2 = \quad -x_2 \\ \qquad\qquad (x_1, x_2) \in S \end{cases}$$

Où $S = \{(x_1, x_2) \in \mathbb{Z}^2 \mid 2x_1 + 3x_2 \leq 5,\ 2x_1 + x_2 \leq 4,\ x_1 \geq 0,\ x_2 \geq 0 =\}$.

L'ensemble des solutions réalisables du problème relaxé est :

$X = \{(x_1, x_2) \in \mathbb{R}^2 \mid 2x_1 + 3x_2 \leq 5,\ 2x_1 + x_2 \leq 4,\ x_1 \geq 0,\ x_2 \geq 0\}$.

Le programme linéaire (P_0) est :

$$(P_0)\begin{cases} Max\ z(x) = z_1(x) + z_2(x) = x_1 + 2x_2 \\ \qquad\qquad (x_1, x_2) \in X \end{cases}$$

La résolution de (P_0) donne :

	b	x_1	x_2	x_3	x_4
x_2	$\frac{5}{3}$	$\frac{2}{3}$	1	$\frac{1}{3}$	0
x_4	$-\frac{7}{3}$	$\frac{4}{3}$	0	$-\frac{1}{3}$	0
$-z$	$-\frac{10}{3}$	$-\frac{1}{3}$	0	$-\frac{2}{3}$	0
z_1	-5	-1	0	-1	0
z_2	$\frac{5}{3}$	$\frac{2}{3}$	0	$\frac{1}{3}$	0

Tableau 2.4

La solution optimale est non entière $(0, \frac{5}{3})$. La séparation se fait donc par rapport à la variable x_2 et on obtient les deux sous problèmes :

$$(P_1)\begin{cases} (P_0) \\ x_2 \leq 1 \end{cases} \quad (P_2)\begin{cases} (P_0) \\ x_2 \geq 2 \end{cases}$$

Le problème (P_2) étant non réalisable, le noeud correspondant est sondé.

La résolution de (P_1) donne :

	b	x_1	x_2	x_3	x_4	x_5
x_2	1	0	1	0	0	1
x_4	1	0	0	-1	1	2
x_1	1	1	0	$\frac{1}{2}$	0	$-\frac{3}{2}$
$-z$	-3	0	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$
$-z_1$	-4	0	0	$-\frac{1}{2}$	0	$-\frac{3}{2}$
$-z_2$	1	0	0	0	0	1

Tableau 2.5

La solution optimale entière est $x^1 = (1, 1)$ dont le vecteur critère est :

$(f_1(x^1), f_2(x^1)) = (4, -1)$.

Ainsi $SND := (4, -1), N_1 = 3, 5, H_1^1 = \emptyset, H_1^2 = 5, K_1 = 2, N_1 \backslash H_1^2 = 3 et \min_{j \in H_1^2} \{\hat{c}_j^2\} = \hat{c}_5^2 = 1$.

Comme $|K| = 1$, alors un seul problème est créé, (P_2), en rajoutant la coupe efficace : $f_2(x \geq f_2(x^1) + \lfloor \hat{c}_3^2 \rfloor x_3 + \max\{1, \lfloor \hat{c}_5^2 \rfloor\}$, dont l'expression est donnée par : $-x_2 \geq -1 + \lfloor 0 \rfloor x_3 + 1$.

La résolution de (P_3) donne :

	b	x_1	x_2	x_3	x_4	x_5	x_6
x_2	0	0	1	0	0	0	1
x_3	1	0	0	1	1	0	-2
x_1	2	1	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$
x_5	1	0	0	0	0	1	-1
$-z$	-2	0	0	0	$-\frac{1}{2}$	0	$-\frac{3}{2}$
$-z_1$	-2	0	0	0	$-\frac{1}{2}$	0	$-\frac{5}{2}$
$-z_2$	0	0	0	0	0	0	1

Tableau 2.6

La solution optimale entière est $(2,0)$ et son vecteur critère est aussi $(2,0)$.

$SND := (4,-1),(2,0)$ puisque $(2,0)$ n'est pas dominé par $(4,-1)$, $N_3 = 4,6, H_3^1 = \emptyset, H_3^2 = 6, K_3 = 2, N_3 \setminus H_3^2 = 4 et \min_{j \in H_3^2} \{\hat{c}_j^2\} = \hat{c}_6^2 = 1$.

Un nouveau problème (P_4) est créé à partir de (P_3) après rajout de la coupe efficace : $-x_2 \geq 0 + 0.x_4 + 1$

.

	b	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_2	-1	0	1	0	0	0	0	1
x_3	3	0	0	1	1	0	0	-2
x_1	$\frac{5}{2}$	1	0	0	$\frac{1}{2}$	0	0	$-\frac{1}{2}$
x_5	2	0	0	0	0	1	0	-1
x_6	1	0	0	0	0	0	1	-1
$-z$	$-\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	0	$-\frac{3}{2}$
$-z_1$	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	0	$-\frac{5}{2}$
$-z_2$	-1	0	0	0	0	0	0	1

Tableau 2.7

Le problème (P_4) est non réalisable et par conséquent, le noeud correspondant est sondé.

Tous les noeuds de l'arborescence étant sondé, l'algorithme se termine et l'ensemble des solutions non dominées du problème (P) est $SND := (4,-1),(2,0)$.

2.6. MÉTHODE DE CHERGUI ET AL [CAM12] : GÉNÉRATION DES SOLUTIONS ENTIÈRES NON DOMINÉES DU PROBLÈME MULTI-OBJECTIF LINÉAIRE

Dans ce chapitre, une liste non exhaustive de méthodes exactes de résolution de MOILP est donnée. Ces méthodes sont décrites avec quelques remarques mentionnées sur chacune. Il est bien connu que pour des problèmes de grande taille, l'énumération de toutes les solutions efficaces et/ou non dominées n'est plus envisageable, c'est dans cette perspective que nous avons été motivé au cours de ce travail pour étudier la technologie des grilles de calul afin de connaître ce qu'elle offre comme possibilités d'amélioration en termes de temps d'exécution de telles méthodes, cette étude fait l'objet du prochain chapitre.

CHAPITRE 3

Les grilles informatiques

Les grilles de calcul sont de plus en plus utilisées, aussi bien dans le monde de la recherche que dans celui de l'industrie. Il s'agit en effet d'un instrument puissant, permettant de traiter des problèmes difficiles faisant appel à de nombreuses données de taille importante. Au delà d'une solution pour les calculs intensifs, l'approche grille permet la constitution d'organisations virtuelles (VO) regroupant des communautés d'utilisateurs partageant des ressources globalisées de traitement, de stockage, des instruments et des services largement distribuées, quelles qu'en soient les organisations propriétaires.

3.1 Historique et définitions

3.1.1 Les origines des grilles informatiques :

Les origines des grilles informatiques sont assez flous. Aux alentours des années 70, le docteur Richard Crandall (NeXT) serait le premier à avoir expérimenté les grilles informatiques, grâce à son programme de calcul parallèle distribué baptisé Zilla, le programme utilisait des machines châinées entre elles pour des traitements mathématiques complexes [Bou08].

D'autres disent que l'idée serait venue de trois personnes, du Docteur en mathématiques et en informatique Ian Foster (Directeur du Laboratoire National Argonne à Chicago aux Etats-Unis), de Monsieur Carl Kesselman chercheur en Informatique à « The University of Southern California » et de Steve Tuecke ingénieur en informatique au Laboratoire National Argonne. Ces trois sont surnommés *« fathers of the Grid »* (Les Fondateurs du Grid), et sont à l'origine de l'une des plus importantes organisations pour le Grid « The Globus Alliance [7]».

Le mot « grille » vient de l'anglais grid qui a été choisi par analogie avec le système de distribution d'électricité américain (electric power grid), ce terme a été répandu en 1998 par l'ouvrage de Ian Foster et Carl Kesselman [FK98]. En effet, une grille peut être vue comme un instrument qui fournit de la puissance de calcul et/ou de la capacité de stockage de la même manière que le réseau électrique fournit de la puissance électrique (chaque utilisateur a toutes les ressources dont il a besoin au moyen d'une interface simplifiée, une prise de courant. Toute la complexité du réseau sous-jacent(de la centrale électrique au particulier) est complètement cachée).

La vision des inventeurs de ce terme est qu'il sera possible, à terme, de se brancher sur une grille informatique pour obtenir de la puissance de calcul et/ou de stockage de données sans savoir ni où, ni comment cette puissance est fournie.

3.1.2 Exemple sur l'analogie des réseaux de distribution électrique et des grilles informatiques [3] :

Nous présentons un réseau de distribution électrique, avec plusieurs sources de production d'énergie et un consommateur d'énergie ; ce consommateur, physiquement relié à ce réseau va pouvoir utiliser l'énergie (les ressources) mise à disposition sans se soucier de quelle source provient exactement cette énergie.

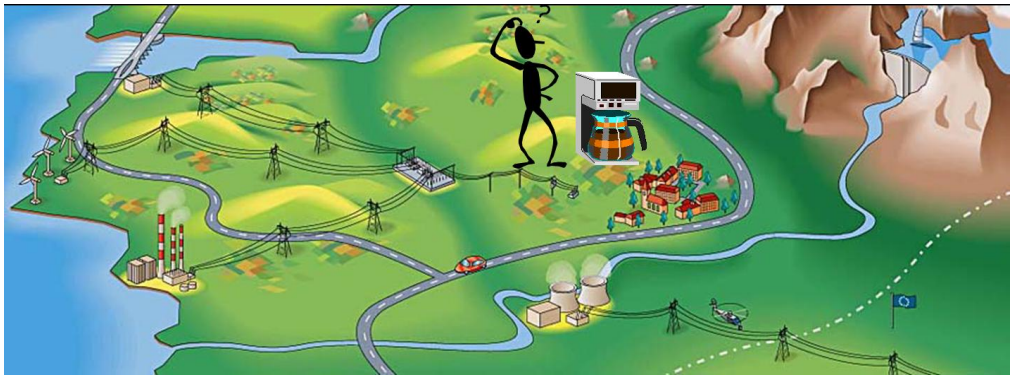

Figure 3.1 – Approche pour la distribution de la puissance électrique, les réseaux électrique et la haute-tension [3]

Cette idée est utilisée dans les grilles de calcul. Un utilisateur possédant un ordinateur et nécessitant une énorme puissance de calcul peut, avec le système de grille de calcul, soumettre ses tâches à plusieurs "supercalculateurs" enregistrés sur la grille.

L'analogie avec le système de distribution d'électricité permet de cerner la vision d'une grille d'un point de vue utilisateur, nous avons donc besoin de définir com-

Figure 3.2 – Approche pour la distribution de la puissance informatique [3]

ment cette puissance est fournie.

Nous commençons par définir ce qu'est un site d'une grille, pour cela nous reprenons la définition de [LAC05].

Définition 3.1. ***Site :*** *Un site est un ensemble de ressources informatiques localisées géographiquement dans une même organisation (campus universitaire, centre de calcul, entreprise ou chez un individu) et qui forment un domaine d'administration autonome, uniforme et coordonné.*
Les ressources informatiques sont aussi bien des liens réseau, des machines (simples PC ou calculateurs parallèles) ou des éléments logiciels. Nous pouvons maintenant définir une grille informatique.

Définition 3.2. ***Grille Informatique :*** *Une grille informatique mutualise un ensemble de ressources informatiques géographiquement distribuées dans différents sites. Néaumoins, il n'existe pas de définitions trés précises des grilles de calcul.*

- ***Buyya*** *définit la grille comme une infrastructure qui implique l'utilisation intégrée et collaborative d'ordinateurs, réseaux, bases de données et d'instruments scientifiques appartenant et gérés par des organisations multiples [BV05].*

- *Le **CERN** (Centre Européen de recherche Nucléaire), un des plus gros consommateurs de puissance de calcul à travers la technologie du grid computing, il la définit dans son site web GridCafé [4] comme un service pour le partage de puissance informatique et de capacité de stockage à travers l'internet .*

- *Mais la définition qui nous paraît la plus complète est celle donnée par **Ian Foster** dans son article "What is the Grid ? A Three Point Checklist" [Fos02], où il fait une synthèse de plusieurs définitions et sort avec une liste de trois points selon laquelle une grille est un système qui :*

  1. *Coordonne des ressources informatiques dont l'administration n'est pas centralisée,*
  2. *utilise des méthodes et des normes standardisées, ouvertes, pour des fins générales (une grille est construite à partir des protocoles multi-usages et des interfaces qui permettent l'authentification, découverte de ressources, et l'accès aux ressources,...)*
  3. *offre une qualité non négligeable de service (relativement aux temps de réponse par exemple, le débit, la disponibilité, la sécurité, et / ou co-affectation de plusieurs types de ressources pour répondre aux besoins complexes des utilisateurs.*

Remarque 3.1. *Il est important de noter que les grilles informatiques sont, par nature, dynamiques : 1) les sites peuvent quitter ou rejoindre la grille à tout moment ; 2) de même, au sein de chaque site, de nouvelles ressources peuvent être ajoutées, d'autres peuvent tomber en panne ou être déconnectées.*

Remarque 3.2. *Une grille de calcul est une architecture permettant à plusieurs*

communautés de partager des ressources informatiques. L'idée centrale introduite par la grille est la notion d'organisation virtuelle (VO Virtual Organization) qui signifie un ensemble de personnes, d'institutions ou d'organismes qui ont un but commun dans leur utilisation de la grille. On peut citer la plus connue des organisations virtuelles : LHC (Large Hadron Collider), le collisionneur de particules du CERN (Centre Européen pour la Recherche Nucléaire) [8] produit des quantités extraordinaires de données par an qui seront visualisés et analysés par 6000 physiciens dans le monde via des grilles de calcul répartis sur quatre niveaux. L'intérêt d'une VO est de proposer un accès simplifié aux données partagées par l'organisation ce qui évite aux utilisateurs de rapatrier des quantités de données croissantes dans leurs propres PC.

Il existe différents types de grille informatique. On distingue notamment les grilles de données et les grilles de calcul [Bou08].

Les grilles de données (data grid) : sont principalement utilisées pour le stockage de grandes masses de données. Elles sont généralement composées de ressources offrant une grande capacité de stockage, en mémoire et sur disque.

Les grilles de calcul (computational grid) : sont dédiées aux calculs intensifs. Une grande importance est alors donnée à la puissance des processeurs des noeuds qui la composent. Nous parlerons dans la suite de ce mémoire que de ce type de grilles car nous nous intéressons qu'à des applications scientifiques effectuant des calculs sur des grilles de calcul.

3.2 Quelques projets de grilles informatiques

- **EGEE**(Enabling Grids for E-sciencE) [2] : Ce projet a permis de construire la plus grande infrastructure de grille pluridisciplinaire du monde, avec plus de 120 organisations qui collaborent pour fournir des ressources de calcul scientifique à la communauté mondiale et européenne de la recherche. EGEE comprend 250 sites dans 48 pays et plus de 68 000 unités centrales mises à la disposition de 8000 utilisateurs, 24 heures sur 24 et 7 jours sur 7.

  Cette grille s'appuie sur le réseau à grand débit GEANT de l'Union Européenne, le projet se concentre sur trois axes :

  1) construire une grille cohérente, robuste et sécurisée ;

  2) améliorer continûment la qualité du logiciel pour fournir un service fiable aux utilisateurs ;

  3) attirer de nouveaux utilisateurs scientifiques ou industriels en leur faisant découvrir le nouveau potentiel offert par cette grille et s'assurer qu'ils reçoivent une formation et un support de qualité.

- **EUMEDGRID** [1] : L'initiative d'EUMEDGRID vise à développer dans le bassin Méditerranéen une infrastructure de grid pour la Recherche, qui peut devenir une partie d'EGEE et être intégrée avec des initiatives analogues dans les Balkans, l'Europe du nord, l'Amérique Latine et le Loin-Est Asiatique. Ce projet dont l'Algérie est partenaire à travers le CERIST(CEntre de Recherche sur l'Information Scientifique et Technique) accompagne l'ensemble des pays partenaires dans la réalisation de leurs propre grille de calcul et de l'intégrer à la grille mondiale.

  Un autre but d'EUMEDGRID est d'accroître les compétences relatives aux

grilles des chercheurs qui travaillent dans la Région Méditerranéenne, pour les rendre capables de bénéficier de ce nouvel et puissant instrument, pour stimuler la collaboration avec des projets Européens et dans le monde entier et pour soutenir le développement scientifique et industriel dans la Région.

- **EUChinaGrid** [9] : L'objectif du projet a été dans un premeir temps de faciliter le transfert de données scientifiques et leur traitement à travers un premier échantillon d'applications pilotes où une collaboration forte entre l'Europe et la Chine a déjà existé. Ces applications ont immédiatement bénéficié de la nouvelle infrastructure, ce qui a permis de tester et de déployer une infrastructure de grille efficace entre l'Europe et la Chine.

- **OSG**(Open Science Grid) [6] : C'est une infrastructure américaine de calcul en grille pour les applications scientifiques, basée sur une collaboration ouverte entre chercheurs, développeurs logiciels et fournisseurs de ressources de calcul, de stockage de données et d'accès réseau.

3.3 Topologie d'une grille de calcul

La grille est physiquement constituée de noeuds, qui sont des processeurs avec leurs disques, l'ensemble étant inter connecté via un réseau, ces noeuds sont des machines plus ou moins puissantes, voire des PC, ou des groupes d'ordinateurs (appelés grappes ou clusters [1]).

1. **Cluster :** Regroupement d'unités informatiques qui coopèrent pour un but final commun et forment une seule unité informatique virtuelle sur les plans fonctionnel et administratif [5]

Un logiciel de pilotage est installé sur chaque noeud. Il assure le bon déroulement de l'activité locale du noeud (selon le rôle de ce dernier) et permet de faire collaborer tous les éléments composant la grille (les personnes et les ressources).

L'ensemble des logiciels assurant la gestion de la grille est dénommé l'intergiciel (middleware) de la grille. Celui-ci représente en quelque sorte le "moteur" de grille, il gère toutes les ressources de la grille, et est informé en permanence de leur état : unités de calcul et de stockage constitutives des noeuds, ne doit pas seulement trouver les données dont les scientifiques ont besoin, mais également utiliser les programmes et les ressources calculs nécessaires pour les exécuter. Les tâches doivent être distribuées n'importe où dans le monde dès qu'il y a une ressource suffisante disponible, puis retourner le résultat aux scientifiques.

Le middleware assure aux utilisateurs et aux applications les services de :

- réservation et allocation des ressources nécessaires
- ordonnancement et lancement des travaux
- suivi de l'activité
- administration du système

Le middleware utilisé par un grand nombre de projets de grille est GT (Globus Toolkit). La grille scientifique américaine OGS (Open Grid Science) et la grille européenne EGEE (Enabling Grids for E-sciencE) l'utilisent depuis leur création même si EGEE utilise maintenant une évolution du middleware appelé gLite. Ce dernier est également le middleware utilisé par la grille Euro-Méditerranéenne EUMEDGRID.

la grille Algérienne (le DZ-GRID) qui comprend jusque là trois clusters installés sur trois sites : DZ-01 (au niveau du CERIST-Alger), DZ-02 (au niveau de l'université

de Batna), DZ-03 (au niveau de l'université de Sétif) utilise le middleware gLite, nous allons par conséquent définir dans ce qui suit les composants d'une grille de calcul basée sur gLite.

3.3.1 gLite : Lightweight Middleware for Grid Computing

C'est un middleware développé par le projet EGEE (Enabling Grids for E-sciencE). L'architecture de la grille de calcul suivant gLite se compose d'un ensemble de services, nous présentons les plus importants [BCL+09](voir Figure 3.3) :

- **La sécurité :** Pour pouvoir accéder aux ressources de la grille l'utilisateur doit être capable de s'authentifier auprès de ces ressources, il doit ainsi posséder un "certificat numérique" X.509 (passeport électronique) qui est une clé publique cryptée associée à une clé privée, le certificat est délivré par une autorité de certification "CA" (Certification Authority).

  L'utilisateur doit ensuite être inscrit dans une "VO" (Virtual Organization) [2]. Muni de son certificat, il peut s'inscrire dans une ou plusieurs VOs. Sa demande sera ensuite validée par le «VO Manager».

  Le certificat d'utilisateur, dont la clé privée est protégée par un mot de passe, est utilisé pour générer et signer un certificat temporaire, appelé un "certificat proxy" (ou simplement un proxy) qui permet l'identification de l'utilisateur ainsi que ses jobs lors de leur exécution.

- **L'interface utilisateur (UI "User Interface") :** Le point d'accès aux ressources de la grille est l'interface utilisateur, c'est une machine sur laquelle

2. VO ou Virtual Organization signifie un ensemble de personnes, d'institutions ou d'organismes qui ont un but commun dans leur utilisation de la grille

l'utilisateur de la grille doit avoir un compte et installer son certificat.

Une UI permet grâce à une CLI (Command Line Interface) (voir le glossaire) fournie par gLite d'effectuer les opérations de base de la grille :

- lister toutes les ressources adéquates pour exécuter une tâche donnée ;
- soumettre des jobs [3] ;
- annuler des travaux ;
- interroger le statut d'un job et récupérer ses résultats ;
- copier, reproduire et supprimer des fichiers de la grille ;
- récupérer l'état des différentes ressources du système d'information.

- **L'élément de calcul (CE "Computing Element") :** le CE est un service qui représente une ressource informatique, il est le point d'entrée sur les fermes de calcul.

  Les applications et logiciels dont ont besoin les utilisateurs y sont installées par un utilisateur ayant un rôle spécial dans la VO (software manager).

  Le CE est composé d'un :

  - Système d'information
  - Système de Batchs [4] local (Exemple : Torque / MAUI)
  - Ensemble de Worker Nodes (machines homogènes en général)

  Le CE soumet les jobs aux worker nodes via le système de batch local.

- **Les Noeuds de calcul ("WN" Worker Node) :** Ce sont les noeuds qui fournissent les services de calcul et d'exécution des jobs envoyés par le CE, chaque site doit en posséder plusieurs.

3. Le job est la tâche (travail, application scientifique) qu'exécute un utilisateur sur la grille de calcul

4. Définition du mot BATCH, Désigne le fonctionnement par file d'attente d'un système : les jobs ne sont pas effectués lors de l'arrivée, mais d'abord toutes regroupées dans la file puis exécutées en une séquence.

- **Le WMS (Workload Management System) :** c'est le chef d'orchestre de la grille et est installé sur une machine, il permet d'accepter le job soumis par l'utilisateur, l'assigne au CE approprié, enregistre son statut et récupère sa sortie.

  Les jobs soumis sont décrits à l'aide du JDL (Job description Language) sous forme de fichier qui spécifie quel exécutable et quels paramètres doivent être lancés sur la grille. A l'aide d'un processus appelé « match-maker » le job est soumis au meilleur CE en fonction des informations fournies dans le fichier JDL.

  Le but du WMS est d'ordonnancer et de manager les ressources de la grille en optimisant leur utilisation. Il permet aux utilisateurs de soumettre leurs jobs et de les exécuter le plus vite possible.

- **L'élément de stockage ("SE" Storage Element) :** Ces noeuds permettent d'accéder aux disques de stockages massifs. la gestion de l'espace disque est assurée par une solution légère à l'aide du DPM (Disk Pool Manager).

- **Le service dinformation (SI) :** fournit des informations sur les ressources de la grille et leur statut ainsi que sur les jobs. Les informations publiées sur la grille par le SI servent à la supervision et l'étude des performances des ressources.

  Le SI utilisé par gLite est le BDII (Berkley Database Information Index), il existe deux types de machines BDII :

  – le Site-BDII : installé au niveau de chaque site, il rapporte le statut à l'échelle du site lui même

– le Top-BDII : il représente le noeud supérieur de la grille, recueille les informations de tous les sites appartenant à la grille à travers les sites-BDII.

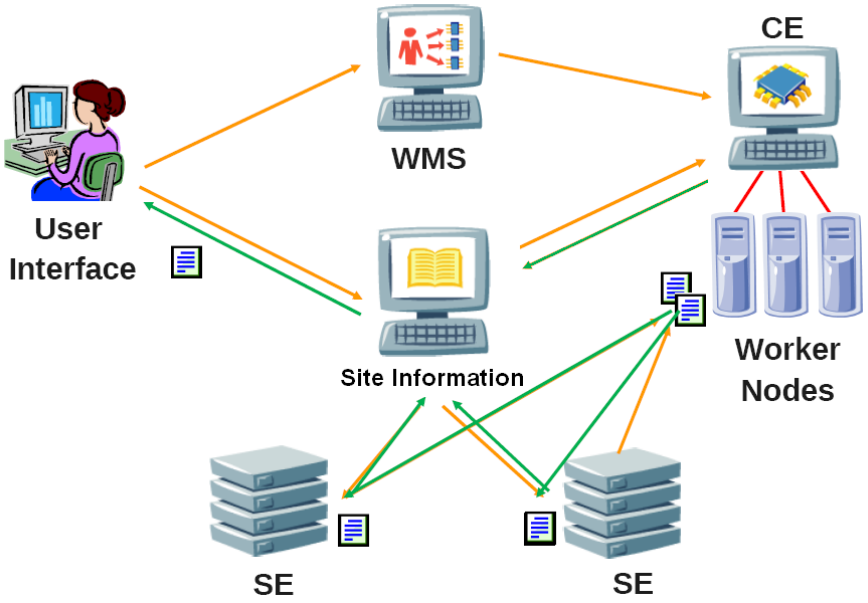

Figure 3.3 – Schéma d'un site de grille utilisant le middleware gLite [Gal10].

3.4 Étapes d'un job avec gLite [BCL+09]

La soumission d'un job sur une grille commence par l'écriture d'un script (petit fichier texte) dans le langage JDL (Job Description Language) dont nous parleront ultérieurement. Il est ensuite soumis par l'utilisateur via le User Interface (UI), et transféré au Workload Management Sytem (WMS), il est considéré comme "**submitted**".

Une fois accepté par le WMS, le job reste en attente de traitement par le Workload Manager : Le job est dans l'état "**Waiting**".

Lorsque le job est assigné à un Computing Element (CE) approprié mais n'y est pas encore transféré, il est considéré comme "**Ready**".

Une fois le job transféré au CE il se met sur la fille d'attente des jobs et est dans l'état "**Scheduled**".

Ensuite le job est envoyé sur un des Workers Node est son exécution commence, il prend alors l'état "**Running**". Dès que son execution se termine avec succès il devient "**Done(OK)**", à ce moment là l'utilisateur peut récupérer ses résultats, une fois que ces derniers sont transférés au UI le job est "**Cleared**".
Si l'execution du job échoue alors l'état du job retourne "**Done(Failed)**".

A n'importe quel moment l'utilisateur peut arrêter son job sur la grille, et le job est considéré comme "**Cancelled**", comme le job peut rester trop longtemps sur la grille jusqu'à ce que le proxy expire et dans ce cas le job est automatiquement arrêté par le WMS et il se met à l'état "**Aborted**".
La figure ainsi que le tableau qui suivent illustrent les étapes d'un job avec gLite.

3.5 Préparation d'un job

Pour pouvoir soumettre un job, il faut :

- Donner une description du job : quel programme (exécutable) ? quelles données ? quels softs, OS, besoins spécifiques ?
- Que le job soit le plus portable possible (exécutable sur une plateforme inconnue) et sans liens absolus, path spécifiques, etc.

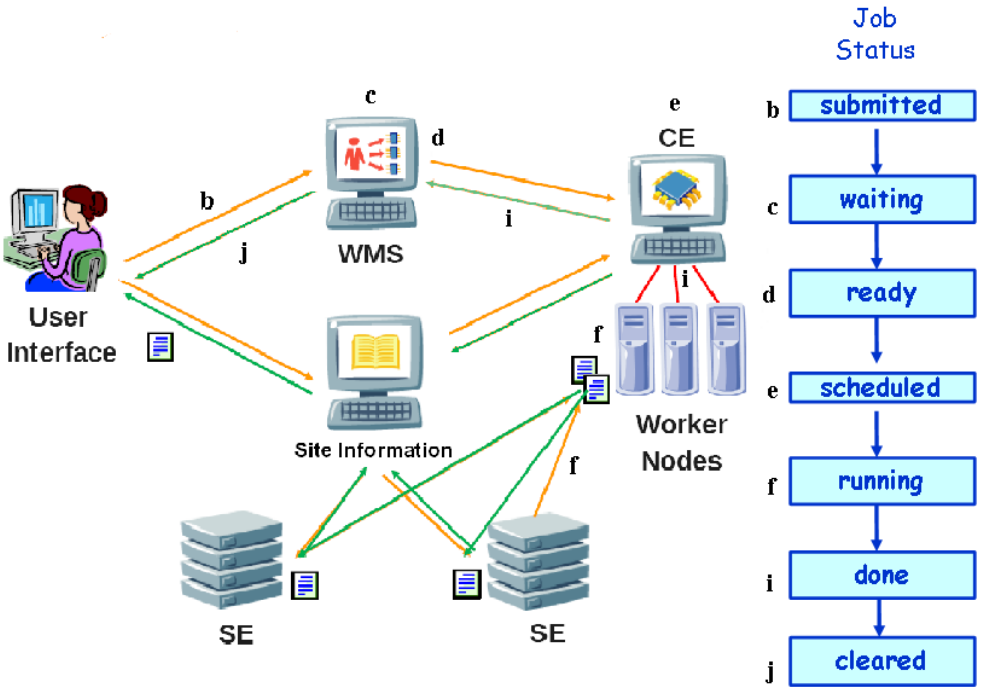

Figure 3.4 – Les étapes d'un job avec gLite.[BCL+09]

États	Définition
SUBMITTED	Le job a été soumis par le user et en cours de transfert vers le WMS
WAITING	Le job a été accépté par le WMS mais encore traité par le Workload Manager
READY	Le job a été assigné à un computing element mais n'y est pas encore transféré
SCHEDULED	Le job est en attente dans la queue du coputing element
RUNNING	Le job est en cours d'exécution sur un worker node
DONE	L'exécution s'est terminée
CLEARED	Les résultats ont été transférés à la user interface
ABORTED	Le job a été arrêté par le WMS (parcequ'il est resté trop longtemps sur la grille, ou le proxy a expiré, etc)
CANCELED	Le job a été arrêté par le user

Tableau 3.1 – États d'un job selon gLite.[Rea10]

- Préciser (si nécessaire) les données entrantes et/ou sortantes.

3.6 Le langage JDL

Un job (fichier de description d'un job) est construit avec une séquence d'attributs dans un langage appelé JDL (Job Description Language).
Il y a deux grandes catégories d'attributs :

- Ceux concernant le job : définissent le job lui-même
- Ceux concernant les ressources

– Ils sont utilisés par le WMS pour déterminer les ressources nécessaires au job.
– Ils permettent de préciser les caractéristiques de calcul requises (librairies disponibles, etc.).
– Ils permettent de définir les caractéristiques liées aux données : données entrantes, SE utilisé pour les données, les protocoles daccès aux données, ...

CHAPITRE 4

Contribution et implémentation

Nous nous intéressons dans ce chapitre à l'implémentation d'une des méthodes d'optimisation multiobjectif discrète décrites précédemment sous un environnement de grilles de calcul. Il s'agit de la méthode de Chergui et Al [CAM12]. Nous présentons tout d'abord une revue rapide de cette dernière, puis nous exposons en détail le modèle parallèle proposé pour cette méthode et son implémentation sur la grille de calcul et pour finir, des résultats expérimentaux sont donnés et commentés.

4.1 Introduction à la Contribution

La méthode de Chergui et Al [CAM12] génère toutes les solutions entières non dominées sans passer en revue toutes les solutions possibles de Y. C'est une méthode bassée sur une technique de branchement.

Deux types de nuds sont considérés dans l'arbre de recherche. Le premier type carac-

térise les solutions non entières qui sont trouvés et transformés en solutions entières en appliquant une procédure de branchement. Le deuxième type de nuds contient une solution entière et dans ce cas des coupes efficaces sont utilisées afin d'éliminer les domaines réalisables ne contenant que des solutions entières dominées.

Concernant les expérimentations faites sur la méthode, M.E.A Chergui explique dans sa thèse de doctorat [Che10] que la version actuelle du programme de traitement des données est focalisée sur la résolution du problème MOILP ; mais les procédures utilisées ne sont pas optimisées, c'est pour cette raison que seuls les problèmes de petite et moyenne taille ont été résolus.
La méthode en question a été mise en oeuvre dans un environnement Matlab 7.0. Toutes les procédures ont été programmées, y compris les méthodes du simplexe et du dual du simplexe.
En se basant sur les conclusions faites par M.E.A Chergui, l'arboresc ence de l'algorithme proposé et qui se base sur le branch and bound pourrait être parallélisée afin de permettre la résolution des problèmes de grande taille.

Afin de proposer un modèle parallèle pour la méthode en question nous avons étudié les différentes possibilités de paralléliser les algorithmes de Branch and Bound.

4.2 Sources de Parallélisme

L'utilisation du parallélisme a été développée dans le but d'améliorer les performances de la recherche durant le Branch and Bound. Trois approches de base, désormais classiques, ont été classifiées par les chercheurs du domaine (tels que Crianic, Le Cun,Gendron et Roucairol [CCR09] et [GC94]) pour la conception d'algorithmes

branch and bound parallèles :

1. **Parallélisme de type 1 (basé sur les noeuds)**, qui introduit le parallélisme sur une opération particulière, principalement au niveau d'un noeud ou un sous problème généré : Calcul en parallèle des bornes inférieures ou supérieures, inversion d'une matrice, l'évaluation en parallèle des fils, génératio n de coupes, et ainsi de suite.

   Cette classe de stratégies ne vise pas à modifier la trajéctoire de l'exploration dans l'arbre, ni sa dimension. L'accélérer est le seul objectif.

2. **Parallélisme de type 2 (basé sur l'arbre)**, qui vise à construire et à explorer l'arborescence branch and bound en parallèle en effectuant des opérations de manière simultanée sur plusieurs sous-problèmes, qui veut dire offrir à l'ensemble des processeurs la possibilité de sélectionner en parallèle des sommets à explorer. Chaque processeur exécute le cycle de décomposition de la même façon qu'en séquentiel.

3. **Parallélisme de type 3**, qui consiste à définir plusieurs branch and bound pour explorer un même domaine parallèlement avec ou sans communication entre les processeurs (approche "Multisearch"). Ce type de parallélisme peut consister aussi à décomposer le domaine des solutions réalisables et utiliser le Branch and Bound pour résoudre le problème sur chaque partition.

   Contrairement au Parallélisme de type 1, le parallélisme de type 2 et 3 peut engendrer la modification de la structure initiale de l'arbre.

Ces stratégies ne sont pas mutuellement incompatibles. En effet, lorsque les instances de problèmes sont particulièrement grandes et difficiles, plusieurs stratégies peuvent être combinées dans une conception algorithmique globale. Par exemple, Une stratégie basée sur les noeuds pourrait lancer la recherche et générer rapide-

ment des sous-problèmes intéressants, suivie d'une exploration parallèle de l'arbre. Dans d'autres cas, plusieurs Branch and Bound peuvent être consididérés simultanément pour explorer un même domaine, chaque arbre pouvant être construit et exploré en parallèle.

Les stratégies de parallélisation basées sur l'arbre donnent des algorithmes irréguliers et les difficultés correspondantes ont été bien identifiés dans le projet STRATAGEMME [Rou87] :

- Les tâches sont créées dynamiquement au cours de l'algorithme.
- La structure de l'arbre à explorer n'est pas connue à l'avance.
- Le graphe de dépendance entre les tâches est imprévisible.
- L'attribution des tâches aux processeurs doit être faite dynamiquement.

En outre, le parallélisme peut créer des tâches qui sont redondantes ou inutiles pour le travail à effectuer, ou qui réduissent les performances de la recherche dans le branch and bound. Dans les années quatre-vingt, plusieurs chercheurs ont montré que, sous certaines conditions, des implémentations parallèles peuvent parfois ne pas donner les accélérations espérées.

Il est donc important de prendre en considération les aspects algorithmiques au moment de l'exécution, comme le partage, l'équilibrage de la charge de travail ou la transmition de l'information entre les processeurs.

Après avoir étudié les différentes manières de paralléliser le branch and bound et avant de proposer un modèle parallèle pour la méthode de Chergui et Al, nous expliquons dans la prochaine section ce qu'offre l'environnement Matlab en terme de programmation parallèle et d'intégration aux technologies des grilles de calcul. Pour pouvoir par la suite donner une conception complète de la version parallèle de la méthode en question.

4.3 Matlab et le Calcul Parallèle

En informatique, le calcul parallèle consiste en l'exécution simultanée d'une même tâche, partitionnée et adaptée afin de pouvoir être répartie entre plusieurs processeurs en vue de traiter plus rapidement des problèmes plus grands.
Matlab est un langage technique de haut niveau et un environnement interactif pour le développement d'algorithmes, il est largement utilisé pour résoudre des problèmes dans plusieurs domaines d'application.
Les ingénieurs et les scientifiques ont tendance à résoudre des problèmes de plus en plus complexes, avec des temps d'execution et des ensembles de données qui dépassent de loin les capacités des systèmes monoprocesseurs traditionnels. En même temps, les progrès réalisées concernant la puissance de traitement informatique ont permis un accès facile aux processeurs multicoeurs. Ces deux facteurs ont stimulé la demande à rendre Matlab facilement exploitable sur des architectures multiprocesseurs.
L'entreprise MathWorks a répondu à cette demande en créant des boîtes à outils (Matlab toolboxes) qui permettent de supporter le calcul parallèle dans Matlab. Deux produits ont été sortis en 2004 : "Parallel Computing Toolbox" qui est la boîte à outils qui permet d'intégrer le parallélisme dans un code Matlab et le "Matlab Distributed Computing Server" qui permet de distribuer le calcul sur plusieurs processeurs [Mat08b, Mat08a].
Les grilles de calcul étant utilisées par une large communauté scientifique, l'intégration de cette technologie avec Matlab a été une demande naturelle. Ainsi, pour exécuter des programmes parallèles, les utilisateurs de Matlab peuvent résoudre de plus grands problèmes à l'aide des Grilles de calcul tout en restant dans l'environnement Matlab et sans changement significatif du code existant.

4.3.1 Comment fonctionne le mode parallèle sous MATLAB ?

L'exécution parallèle d'une application Matlab sur un système de grille, requiert que l'utilsateur possède sur son poste de travail une licence Matlab muni de la Parallel Computing Toolbox. Ce processus implique typiquement les étapes suivantes :

i L'utilisateur ouvre une session matlab sur son poste de travail, doit d'abord faire quelques transformations sur son code Matlab initial, choisit une Configuration Distribuée "locale", met en marche les agents de calcul Matlab (matlab Workers) locaux et teste le programme parallèle localement avec ces Matlab Workers ;

ii Une fois que le programme tourne avec succès, l'utilisateur choisit une Configuration pour le système de grille et exécute le programme de nouveau. Cette fois, sans intervention de l'utilisateur, Matlab transférera les données en entrée (inputs) et le code à la grille, exécutera le code parallèle sur les agents Matlab (Matlab Workers) propres à la grille et récupérera des données en sortie (outputs).

Le mode parallèle sous Matlab se compose de deux parties principales comme l'illustre la figure 4.1 :

La partie utilisateur : Sur son poste de travail l'utilisateur doît posséder Matlab muni de la boîte à outils Parallel Computing Toolbox(PCT). Ce dernier a la possibilité de paralléliser son code matlab grâce à des fonctions et des structures prédéfinies qu'offre la PCT, le code est divisé en sous tâches qui peuvent s'exécuter en parallèle et c'est à l'utilsateur de décider comment les diviser.

Il pourra ensuite tester et déboguer localement son programme avant de le lancer sur la grille.

La partie grilles : Le programme une fois arrivé à la grille, est géré par un compo-

sant nommé "MATLAB Distributed Computing Server (MDCS)". Il se compose d'un ordonnanceur qui reçoit les jobs(tâches, calculs) envoyés par l'utilisateur et qui est à la tête d'un ensemble de noeuds de calcul appelés "Matlab Workers" qui vont effectuer les calculs.

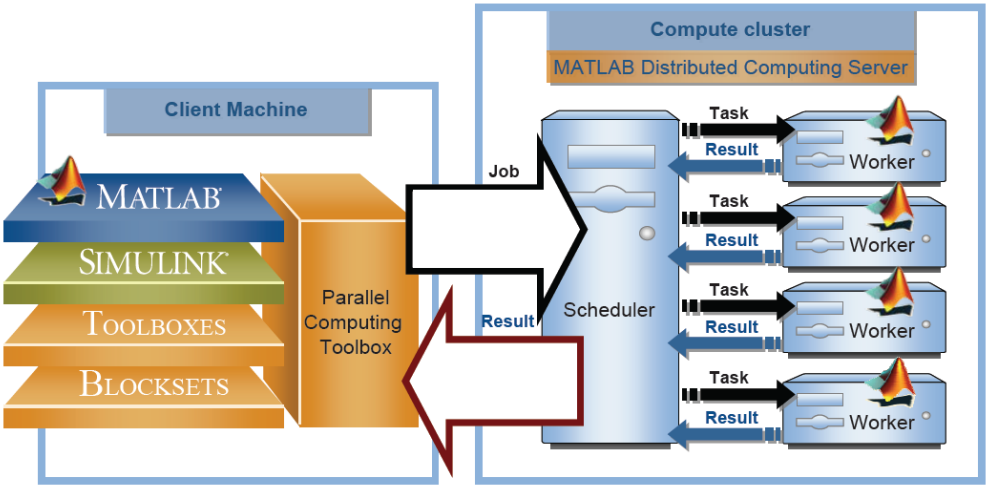

Figure 4.1 – MDCS-Matlab Distributed Computing Server

4.3.2 L'intégration de Matlab avec gLite

L'intégration de Matlab et les grilles de calcul basées sur le middleware gLite est venue de manière tout affet naturelle. Il a suffit de superposer les même composantes que celles illustrées dans la figure précédente (4.1) sur les éléments qui composent une grille basées sur gLite (ces éléments ont fait l'objet d'une étude au chapitre 3) comme l'illustre la figure 4.2 :

La partie utilisateur : Sur son poste de travail, l'utilisateur choisit la configuration de la grille gLite et pourra accéder à la grille à travers l'interface utilisateur de gLite (User Interface), lorsqu'il lance l'éxécution de son programme, c'est Matlab qui génère automatiquement le fichier JDL (Job Description Language)

spécifique à gLite, transfère le code et les fichiers d'entrée sur l'élément de stokage de la grille (SE) et se charge de récupérer les fichiers de sortie une fois l'exécution terminée.

La partie grilles : A ce niveau, Le Matlab Distributed Computing Server est installé sur le Computing Element de gLite et ses Worker Nodes ; l'ordonnanceur est installé sur l'élément de calcul (Computing Element) de gLite, et les Matlab Workers seront lancés sur les noeudes de calculs (Worker Nodes) de la grille. lorsque l'utilsateur lance l'éxécution de son programme, le MDCS, lance les calculs de manière parallèle sur les Matlab Workes de la grille. Une fois l'exécution terminée les fichiers de sortie sont copiés sur le "SE" et Matlab se chargera de les transférer sur le poste de travail de l'utilisateur.

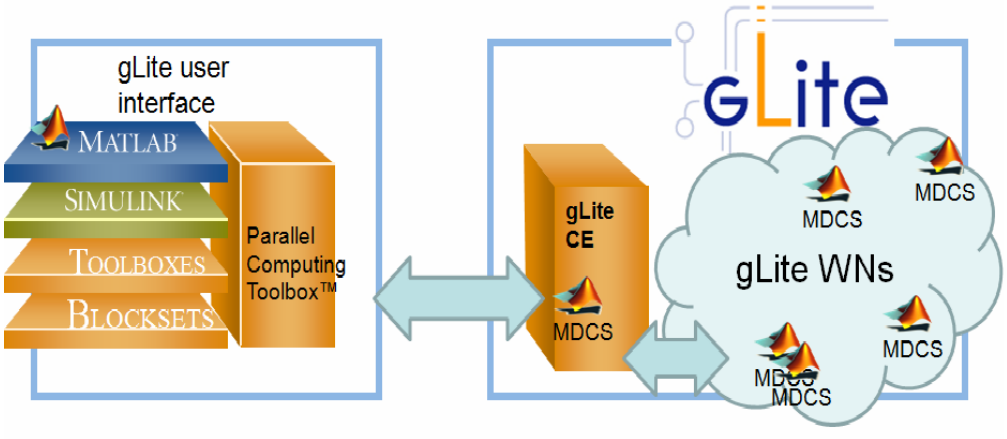

Figure 4.2 – Matlab Distributed Computing Server sur gLite

4.3.3 Le langage parallèle

Il existe sous matlab un ensemble de constructions, de fonctions et de structures de données dédiées à la programmation parallèle, ces constructions se divisent en deux types [CGFL+08] :

- Programmation parallèle de haut niveau : comme les boucles for parallèles et les tableaux distribuées ;
- Programmation parallèle de bas niveau ou avancée : qui utilise le protocole de la programmation par passage de messages, plus connu sous le nom de "Message Passing Interface"(MPI) [1].

La boucle for parallèle, parfor : Cette boucle peut remplacer une boucle for dans un programme, sous condition que les itérations de la boucle for soient complètement indépendantes entre elles. Ainsi, pendant l'exécution, les itérations de la boucle parfor seront distribuées à travers les Matlab workers disponibles.
A titre d'exemple, Supposons qu'un code comprend une boucle pour créer une onde sinusoïdale pour tracer la forme d'onde :

```
for i=1:1024
A(i) = sin(i*2*pi/1024);
end
plot(A)
```

Chaque itération étant complètement indépendante de l'autre, il est possible de remplacer la boucle for par parfor, comme suit :

```
parfor i=1:1024
A(i) = sin(i*2*pi/1024);
end
plot(A)
```

1. **Message Passing Interface(MPI) :** Ce protocole a été conçu en 1993-94, il représente une norme définissant une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran. Elle permet d'exploiter par passage de messages des noeuds exécutant des programmes parallèles sur des systèmes à mémoire distribuée tels que les grilles de calcul.

Au moment de l'exécution chaque itération sera calculée par un Matlab worker comme l'illustre la figure 4.3 [Mat08b] :

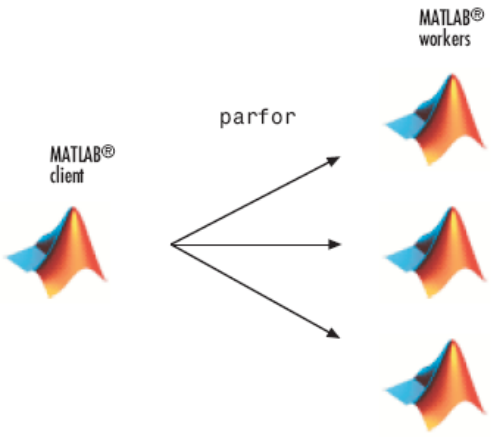

Figure 4.3 – Exemple d'exécution parallèle avec la boucle parfor

Les tableaux distribués de Matlab : Il est possible de distribuer une matrice à travers les Matlab workers et travailler avec de grands ensembles de données. Les opérations de Matlab comme la multiplication, décomposition des matrices peuvent utiliser les tableaux distribuées de Matlab. Chaque worker traite une partie de la matrice et tous les workers communiquent entre eux et sont informés de la partie que chaque worker traite. La figure 4.4 illustre ce principe [CC08].

Programmation par passage de messages (The Message Passing Interface) : Les experts du calcul parallèle ont l'option d'utiliser les fonctions de la programmation parallèle avancée pour exercer un meilleur contrôle sur leurs applications Matlab. Lorsque le programme requiert la communication (envoi et reception de données, synchronisation par exemple) entre les noueds(Matlab workers) le protocole "Message Passing Interface" serait la solution.

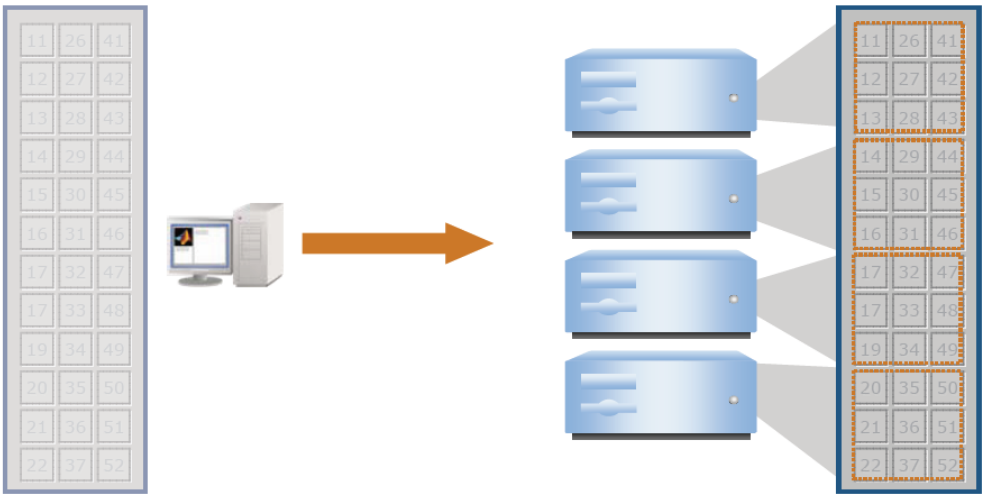

Figure 4.4 – Les tableaux distribuées de Matlab [CC08]

4.4 Le Modèle Parallèle de l'Algorithme de Chergui et Al [CAM12]

Après avoir étudié les diférentes façons de paralléliser un algorithme Branch and Bound et vu les différentes fonctionnalités qu'offre Matlab en termes de programmation parallèle, nous avons pu dégager un modèle parallèle pour la méthode de Chergui et Abbas. Ce modèle repose sur une combinaison de deux stratégies de parallélisme qui sont le type 1 et 2.

Dans la méthode de Chergui et Al [CAM12] : A l'étape r $(r \geq 1)$; après Résolution du programme linéaire (P_l), $0 \leq l < r$ et détermination d'une solution optimale x^l de (P_l).

Si la solution optimale x^l trouvée est entière, ce qui correspond à un noeud de type 2 de la méthode et en vue de trouver une nouvelle solution entière, l'ensemble K_l des critères pouvant être améliorés à partir de x_l est determiné.

Pour tout $i \in K_l$, une contrainte est rajoutée à (P_l) pour obtenir $|K_l|$ nouveaux programmes linéaires. Ces $|K_l|$ programmes linéaires seront à leur tour résolus et

ainsi de suite.

Puisque chaque programme linéaire sera résolu indépendemment de l'autre, alors nous avons proposé de résoudre ces $|K_l|$ programmes linéaires de manière parallèle en utilisant la boucle parfor de Matlab, c'est à dire que chaque programme sera résolu sur un worker à part de la grille.

Pour ce modèle nous avons donc choisi d'appliquer le parallélisme au niveau du noeud de type 2 de l'algorithme (parallélisme basé sur les noeuds ou de type1) et de traiter l'arbre qui est généré par ce noeud de manière parallèle (parallélisme basé sur l'arbre ou de type2).

La version parallèle de la méthode de Chergui et Abbas peut être décrite comme suit :

Étape 0 **(initialisation) :**

l'ensemble des solutions non dominées (SND) de (P)est vide.

Résoudre le programme linéaire (P_0).

Poser x la solution optimale trouvée.

Étape r $(r \geq 1)$ **:**

Résoudre le programme linéaire (P_l), $0 \leq l < r$,

Considérer une solution optimale x^l de (P_l),

Si x^l n'est pas entière, Choisir une coordonée j de x dont la valeur n'est pas entière et séparer le noeud l sur cette coordonnées en deux nouveaux noeuds et revenir à l'étape r.

Si x^l est entière alors :

- Si le vecteur critère correspondant Cx^l n'est dominé par aucun autre vecteur Z appartenant à l'ensemble des solutions non dominées (SND) alors mettre à jour ce dernier en lui rajoutant le vecteur Cx^l.

- S'il existe une solution $Z \in SND$ qui est dominée par Cx^l alors remplacer Z par Cx^l dans l'ensemble des solutions non dominées.

Déterminer l'ensemble K_l des critères pouvant être améliorés à partir de x_l,
parfor $i \in K_l$ rajouter la contrainte 2.5 pour obtenir $|K_l|$ nouveaux programmes linéaires $(P_k), k > l$ et revenir à l'étape r.

Critère d'arrêt :

La procédure s'arrête dès que $H_l = \emptyset$ c'est-à-dire aucun critère ne peut être amélioré ou bien (P_l) n'admet pas de solutions réalisables pour toute étape l telle que $0 \leq l \leq r$.

4.5 Implémentation et Résultats

Nous avons programmé l'algorithme parallèle présenté dans ce chapitre en utilisant le logiciel Matlab version 7 muni de la Parallel Computing Toolbox et effectué les expérimentations sur la grille DZ-01 (au niveau du CERIST-Alger).
Ces expérimentations ont été effectuées sur des instances du problème MOILP avec deux, trois et quatre objectifs.
Ces instances ont été construites grâce au générateur aléatoire décrit ci-dessous :

Algorithme 1 Génération aléatoire d'instances pour un problème multiobjectif

n : nombre de variables
k : nombre de critères
m : nombre de contraintes
les matrices A, C et b formant le problème multiobjectif
$A \leftarrow$ **randint**$(m, n, [-10, 99])$
$C \leftarrow$ **randint**$(k, n, [-10, 99])$
Pour $i = 1$ to $\frac{m}{n}$ **faire**
  $b_i \leftarrow round(\sum_{j=1} A_i^j / 5)$
Fin pour

randint$(m, n, [p_1, p_2])$ est une fonction prédéfinie de Matlab qui renvoie une matrice à m lignes et n colonnes dont les coefficients sont des entiers indépendamment et uniformément distribués dans lintervalle entre p_1 et p_2.

round(X) est elle aussi une fonction prédéfinie de Matlab qui arrondit les éléments de la matrice X aux entiers les plus proches.

Les poids relatifs aux critères sont tous pris égaux à 1.

Notons que la machine sur laquelle est installé Matlab et la PCT est un serveur quadri-core avec 4 GB de mémoire vive configuré avec Scientific Linux 5.4 comme système d'exploitation, la machine a pour nom ui01.
La grille DZ-01 sur laquelle est installé le MDCS est un cluster muni de 5 workers, chaque worker est un serveur doté de 2 processeurs Intel Pentium 4 cadencé à 3.6 GHz avec 4 GB de mémoire vive et 160 GB de disque, configuré également avec Scientific Linux 5.4 comme système d'exploitation.
Toutes les procédures ont été programmées, y compris les méthodes du simplexe et du dual du simplexe.
Pour chaque instance nous avons effectué l'exécution en local sur la machine ui01

d'abord et ensuite sur la grille. Et ce, dans Le but de comparer entre une exécution sur une seule machine et une exécution parallèle sur 5 machines.

Le tableau suivant donne pour chaque instance et pour les deux types d'exécution, la moyenne sur cinq exécutions indépendantes, du temps de calcul en secondes et du nombre de solutions non dominées obtenues :

Instances			Temps de calcul(s)		Nombre de solutions non dominées
n	m	k	Exécution sur une seule machine	Exécution sur sur la grille DZ-01	
15	5	2	17.32	18.9	12.40
20	5	2	92.49	93.53	12.80
25	5	2	400.6	414.3	13.60
15	5	3	20.2	21.45	18.50
20	5	3	99.87	101.85	25.50
25	5	3	356.67	340.3	21.20
20	10	3	64.34	50.45	24
25	10	3	200.23	170.65	20.60
15	5	4	18.12	21.45	19.10
20	5	4	128.1	119.36	130
25	5	4	378.7	350.99	128.90
30	10	4	632.82	500.34	89
50	10	4	x	10800	75.30
60	10	4	x	18034	64
70	10	4	x	x	x

Tableau 4.1 – Résultats des exécutions parallèles sur machine unique et sur la grille avec temps CPU en secondes.

D'après le tableau, il apparaît que pour des instances de petite taille aucune amélioration n'est obtenue lors de l'exécution parallèle à travers les Matlab Workers. Il peut même y avoir un léger ralentissement par rapport à l'éxecution sur une seule machine comme dans le cas des instances (15,5,2), (20,5,3) et (15,5,4). Ce ralentissement est dû au fait que le temps d'exécution des instances est négligeable devant le temps de transfert de données et de communication entre l'ordonnanceur et ses Matlab workers et ainsi que les workers entre eux. Le temps d'exécution en parallèle est alors ralenti.

Nous commençons à percevoir une légère amélioration dans le temps d'exécution parallèle des instances de moyenne taille, comme pour les instances (20,10,3), (25,10,3) et (30,10,4). Ici le temps d'exécution des instances devient considérable devant le temps de transfert de données et de communication et c'est pour cette qu'on arrive à obtenir des améliorations.

Lors d'instances de grande taille telles que (50,10,4) et (50,10,4) il est clair que l'exécution sur une seule machine ne donne aucun résultat après 24h de calcul car le temps d'exécution et l'espace mémoire requis pour le calcul deviennent très grands, cependant même si avec un temps d'exécution qui est très grand par rapport aux instances précédantes l'exécution parallèle donne l'ensemble de toutes les solutions non dominées et parvient à résoudre les instances en question.

Au delà de l'instance (70,10,4) aucun résultat n'est obtenu et ce aprés un temps d'attente qui dépasse les 24 heures.

Au regard de ces résultats, il apparaît que distribuer le calcul à travers plusieurs workers est avantageux pour des instances avec des temps d'exécution plus grands que les temps de communication à travers le réseau.

Nous pouvons aussi déduire que grâce à la parallélisation de la méthode, nous

sommes parvenus à résoudre jusqu'à une certaine limite des instances de grande taille ce qui n'est pas négligeable.

Dans ce chapitre, nous avons expliqué notre approche pour la parallélisation de la méthode de Chegui et Al [CAM12]. Nous avons conclu que les techniques de calcul parallèle peuvent aider à réduire le temps nécessaire pour arriver à une solution. Pour tirer profit de la parallélisation, il est important de choisir une approche qui convient pour le problème d'optimisation. Après avoir testé la méthode de parallélisation par la boucle parfor et dans le but de résoudre de plus grandes instances, il serait intéressant dans un avenir proche de tester d'autres méthodes de parallélisation comme la programmation par passage de messages "MPI" qui permet de mieux gérer la communication entre les Matlab workers ou encore utiliser les tableaux distribués de Matlab et peut être même utiliser une combinaison de plusieurs méthodes parallèles.

Conclusion et Perspectives

De nombreux problèmes rencontrés dans la pratique, dans divers secteurs liés à l'industrie, l'économie, etc. nécessitent la prise en charge de plusieurs objectifs qui, très souvent, sont conflictuels. Pour ce type de problèmes, il ne sagit plus de rechercher une solution optimale, mais un ensemble de solutions Pareto optimales qui se distinguent par les différents compromis réalisés entre les différentes fonctions objectifs considérées.

Dans les trente dernières années, la plupart des travaux réalisés dans ce domaine ont porté sur la programmation linéaire multiobjectif en variables continues. Les raisons principales de cet intérêt sont d'une part le développement de la programmation linéaire monoobjectif en recherche opérationnelle et la facilité relative de traiter de tels problèmes, et d'autre part l'abondance de cas pratiques pouvant être formulés sous cette forme. Dans cette étude, nous nous sommes particulièrement intéressés aux problèmes de programmation linéaire multiobjectif en nombres entiers pour lesquels relativement peu de travaux ont été réalisés et pour beaucoup d'entre eux

seules des instances de petites tailles ont été résolues avec un temps raisonnable sur une seule machine.

Pour accélérer les calculs, les grilles de calcul représentent donc une solution vitale pour le traitement de ces problèmes à travers la parallélisation de ces méthodes de résolution.

Nous avons commencé ce travail par introduire les notions de base concernant l'optimisation multiobjectif, ce qui a fait l'objet du premier chapitre de ce mémoire.

Après une étude détaillée de quelques méthodes existantes d'optimisation multiobjectif discrète dans le deuxième chapitre, en l'occurrence, la méthode décrite par Klein et Hannan [KH82], Gupta et Malhotra [GM92], Sylva et Crema [SC04], Abbas et Moumene [Mou02b] et Chergui et Al [CAM12], nous avons exploré le domaine des grilles de calcul, leur topologie, caractéristiques et mode de fonctionnement. Cette étude a fait l'objet du troisième chapitre.

Nous avons par la suite proposé à titre illustratif de gridifier la méthode Chergui et Al [CAM12] et ce dans le but de parvenir à résoudre des problèmes de grande taille par le biais de la programmation parallèle.

Pour ce faire, Après avoir exposé quelques démarches de la programmation parallèle et étudié la manière d'implémenter la méthode en question sur les grilles, nous avons pu dégagé un modèle parallèle pour la méthode de Chergui et Al [CAM12].

Dans le but de faire une étude comparative par rapport à une exécution sur une seule machine, nous avons programmé le modèle en utilisant le logiciel Matlab version 7 et la parallel Computing Toolbox, nous avons utilisé pour la programmation parallèle la boucle for parallèle de Matlab. Nous avons effectué par la suite des expérimentations, exposé les résultats et dégagé quelques interprétations.

Nous avons constaté au cours de ce travail que les techniques de calcul parallèle appliquées sur les technologies des grilles de calcul peuvent aider à réduire le temps

nécessaire pour arriver à une solution. Pour tirer profit de la parallélisation, il est important de choisir une approche qui convient pour le problème d'optimisation. Après avoir testé la méthode de parallélisation par la boucle parfor de Matlab et dans le but de résoudre de plus grandes instances, il serait intéressant dans un avenir proche de tester d'autres méthodes de parallélisation comme la programmation par passage de messages "MPI" qui permet de mieux gérer la communication entre les Matlab workers ou encore utiliser les tableaux distribués de Matlab et peut être même utiliser une combinaison de plusieurs méthodes parallèles.

Ce travail nous a permis de découvrir des perspectives prometteuses en terme d'amélioration des méthodes en optimisation multiobjectif discrète afin de pouvoir bénéficier au maximum des technologie des grilles de calcul et du domaine de la programmation parallèle.

Bibliographie

[BCL+09] S. Burke, S. Campana, E. Lanciotti, P. Méndez Lorenzo, V. Miccio, C. Nater, R. Santinelli, and A. Sciabà. *GLITE 3.1 USER GUIDE*, 18 Décembre 2009.

[Bou08] A. Bouali. *Recherche et Filtrage d'information basés sur le Text Mining sous la technologie GRID.* Mémoire de magister, Université Badji Mokhtar de Annaba, Algérie, 2008.

[Bow76] V.J. Bowman. *On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives.* In H. Thiriez & S. Zionts (eds), MCDM, Springer-Verlag, Berlin, pages 76-85, 1976.

[BV05] R. Buyya and S. Venugopal. A gentle introduction to grid computing and technologies. *Computer Society of India*, 2005.

[CAM12] M.E.A Chergui, M. Abbas, and M. Aït Mehdi. *Efficient cuts for generating the non-dominated vectors for multiple objective integer linear*

programming. International Journal of Mathematics in Operational Research (IJMOR),Vol. 4, No.3 pp. 302 - 316, 2012.

[CC08] A.J. Chakravarti and E. Chan. *Hands-On Session for Parallel Computing with MATLAB and gLite*. The MathWorks, Inc, 2008.

[CCR09] T.G. Crainic, B. Le Cun, and C. Roucairol. *Parallel Branch-and-Bound Algorithms, Chapitre 1*. John Wiley Sons, Inc, 2009.

[CGFL+08] A.J. Chakravarti, S. Grad-Freilich, E. Laure, M. Jouvin, G. Philippon, C. Loomis, and E. Floros. *Enhancing e-Infrastructures with Advanced Technical Computing :Parallel MATLAB on the Grid*. The MathWorks, Inc, 2008.

[Cha05] D. Chaabane. *Optimisation multicritère en nombres entiers*. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2005.

[Che10] M.E.A. Chergui. *Contribution à la programmation non linéaire multicritère*. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2010.

[Dan63] G.B. Dantzig. *Linear programming and extensions*. Princeton University Press, Princeton, 1963.

[FK98] I. FOSTER and C. KESSELMAN. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1ère édition, Juillet 1998.

[Fos02] I. Foster. *What is Grid ? A Three Point Checklist*. PhD thesis, Laboratoire National d'Argonne et l'Université de Chicago, Etats Unis, Juillet 2002.

[Gal10] F. Galeazzi. *Architecture of the gLite Data Management System.* Presentation dans l'EUMEDGRID School for Application porting, Algiers, July 2010.

[GC94] B. Gendron and T.G. Crainic. *Parallel Branch-and-Bound Algorithms :Survey and Synthesis.* Operations Research, Vol. 42, No 6, pages 1042-1066, 1994.

[Geo68] A.M. Geoffrion. *Proper efficiency and the theory of vector maximization.* Journal of Mathematical Analysis and Applications 22, pages 618-630, 1968.

[GM92] R. Gupta and R. Malhotra. *Multi-criteria integer linear programming problem.* Cahiers de CERO 34, pages 51-68, 1992.

[Gom58] R.E. Gomory. *Outline of an algorithm for integer solutions to linear programs.* Bulletin of the American Mathematical Society 64, pages 275-278, 1958.

[KH82] D. Klein and E. Hannan. *An algorithm for the multiple objective integer linear programming problem.* European Journal of Operational Research 9, pages 378-385, 1982.

[LAC05] S. LACOUR. *Contribution à l'automatisation du déploiement d'applications sur les grilles de calcul.* PhD thesis, Université de Rennes 1, IRISA, Rennes, France, Décembre 2005.

[Lem54] C.E. Lemke. *The dual method for solving the linear programming problem.* Naval Research Logistic Quarterly 1, pages 36-47, 1954.

[Mat08a] The MathWorks. *Matlab Distributed Computing Server System Administrator's Guide.* The MathWorks, Natick, MA, March 2008.

[Mat08b] The MathWorks. *Parallel Computing Toolbox User's Guide*. The MathWorks, Natick, MA, March 2008.

[Mou02a] M. Moulaï. *Optimisation multicritère fractionnaire linéaire en nombres entiers*. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2002.

[Mou02b] A. Moumene. *Contribution algorithmique pour la résolution d'un programme linéaire multi-objectifs en nombres entiers*. Mémoire de magister, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2002.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley, New York, 1988.

[Rea10] M. Reale. *gLite Job Management*. Presentation dans l'EUMEDGRID School for Application porting, Algiers, July 2010.

[Rou87] C. Roucairol. *A Parallel Branch and Bound Algorithm for the Quadratic Assignment Problem*. Discrete Applied Mathematics, No 18, pages 211-225, 1987.

[SC04] J. Sylva and A. Crema. *A method for finding the set of non-dominated vectors for multiple objective integer linear programs*. European Journal of Operational Research 158, pages 46-55, 2004.

Webographie

[1] EUMEDGRID web site

http://www.eumedgrid.eu/

[2] EGEE Portal

http://www.eu-egee.org/

[3] M. COSNARD and T. PRIOL. Introduction au tutoriel Globalisation des ressources informatiques et des données, Présentation de l'*ACI GRID*, 2002.

www-sop.inria.fr/aci/grid/public/Library/Hammamet-9-4-02.ppt

[4] Qu'est-ce que la Grille ? de GridCafé

http://www.gridcafe.org/gridcafe-f/whatisgrid/whatis.html

[5] Grille informatique, de wikipedia

http://fr.wikipedia.org/wiki/Grille_informatique

[6] Open Science Grid

http://www.opensciencegrid.org/

[7] The Globus Alliance

http://www.globus.org/alliance/

[8] CERN - Centre Européen pour la Recherche Nucléaire

http://public.web.cern.ch/public/

[9] the EUChinaGRID initiative

http://www.euchinagrid.org/

Grilles de calcul pour les problèmes d'optimisation multiobjectifs

Résumé

La résolution des applications d'Optimisation multiobjectif discrète a toujours été limitée par les ressources d'une seule machine : soit par la puissance de calcul, soit par la capacité mémoire, le plus souvent les deux à la fois. Pour accélérer les calculs, les grilles de calcul représentent donc une solution vitale pour le traitement de ces applications à travers la parallélisation de ces méthodes de résolution. Dans ce mémoire, nous nous sommes intéressés à l'étude de quelques méthodes de résolution en programmation linéaire multiobjectif en nombres entiers basées sur le Branch-and-Bound ainsi qu'à l'étude de la technologie des grilles de calcul. dans le but d'implémenter sur grilles de calcul. Cette étude nous a permis de proposer une implémentation de la méthode de Chergui et Abbas sur la grille de calcul en réduisant le temps d'exécution. Afin de mettre en valeur notre apport, les principaux résultats obtenus sont présentés.

Mots clés : optimisation multiobjectif, programmation linéaire en nombres entiers, grilles de calcul.

Grid Computing for multi-objective optimization problems

Abstract

Solving multiobjective discrete optimization applications has always been limited by the resources of one machine : by computing power or by memory, most often both. To speed up the calculations, the grid computing represents a primary solution for the treatment of these applications through the parallelization of these resolution methods. In this work, we are interested in the study of some methods for solving linear programming multiobjective integer based on Branch-and-Bound and the study technology of grid computing. in order to implement on grid computing. This study allowed us to propose an implementation of the method of Abbas and Chergui on the grid by reducing the execution time. To enhance our contribution, the main results are presented.

Keywords : multiobjective optimization , integer linear programming, grid computing.