N° d'ordre : 08/2013-M/MT

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ DES SCIENCES ET DE LE TECHNOLOGIE
HOUARI BOUMEDIENE
FACULTÉ DES MATHÉMATIQUES

MÉMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

En : MATHEMATIQUE

Spécialité : Recherche Opérationnelle

Par : BADJARA Mohamed El-Amine

Sujet

APPROCHES EXACTE ET APPROCHÉE POUR LE PROBLÈME DU STABLE MULTI-OBJECTIF

Soutenu publiquement le03/11/2013, devant le jury composé de :

M. M. MOULAÏ, Professeur, à l'USTHB	Président
M. M.E-A. CHERGUI, Maître de Conférences/A, à l'USTHB	Directeur de Mémoire
M. M. AIDER, Professeur, à l'USTHB	Examinateur
Mme. Z. BENMEZIANE, Maître de Conférences/A, à l'USTHB	Examinatrice

Remerciement

Je tiens tout d'abord à remercier **Dieu** *le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce Modeste travail.*

J'adresse de chaleureux remerciements à mon directeur de mémoire, docteur CHERGUI Mohamed El-Amine Maitre de conférences à la faculté de Mathématique de l'USTHB, pour son attention de tout instant sur mes travaux, pour ses conseils avisés et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse. Son énergie et sa confiance ont été des éléments moteurs pour moi. Sa capacité d'analyse et son enthousiasme m'ont montré que le monde de la recherche pouvait être un univers passionnant. En fin, ses nombreuses relectures et corrections de cette thèse ont été très appréciables. J'ai pris un grand plaisir à travailler avec lui.

Je remercie cordialement le professeur MOULAI Mustapha d'avoir accepté présider le jury et pour l'intérêt qu'il a porté à mon travail.

Je souhaite adresser également mes remerciements, au professeur AIDER Meziane et docteur BENMEZIANE Zineb d'avoir fait l'honneur d'examiner mon travail.

J'adresse mes profonds remerciements aussi à l'équipe des enseignants qui ont assurés le suivi des cours et du soutien de la promo du PG, sans oublier bien sûr tous les enseignants du département de Recherche Opérationnelle dont j'ai eu l'honneur de les avoir comme enseignants ou les autres qui font leurs possible pour laisser une bonne image de notre département ainsi que l'université algérienne.

J'exprime ma plus profonde gratitude à ma famille, et notamment à mes parents et, pour m'avoir soutenu durant toutes ces années. Je leur serai toujours redevable de tous les efforts qu'ils ont fournis à mon égard. Sans oublié mon petit frère Mahmoud et mon amie Fatma qui ont sus toujours comment me distraire lors de mes plus difficiles moments.

Enfin, je tiens également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Amine

Table des matières

Préambule

L'être humain est constamment confronté à des problèmes de décisions et/ou d'amélioration de son quotidien. Il fait appel, depuis des siècles, à son expérience et à son vécu pour prendre des décisions dans tous les domaines.

La Recherche Opérationnelle est une discipline carrefour où se rencontrent l'économie, les mathématiques et l'informatique. De ce fait, on peut l'appréhender de manière très différente, suivant que l'on se propose de mettre en relief l'aspect économique, ou bien les méthodes mathématiques, ou encore les particularités de programmation des algorithmes correspondants.

Bien que le concept de Recherche Opérationnelle remonte à l'antiquité et que la plupart de ses méthodes ont été découvertes entre le XVII° siècle et les années 30 du XX° siècle, c'est un fait qu'en dehors du domaine militaire, où l'application systématique date de 1938-1939, la Recherche Opérationnelle civile ne s'est développée qu'à partir du moment où furent vendus dans le public les ordinateurs dits de première génération. Cela s'explique à la fois, par la présence de ces instruments nouveaux de traitement de données, aptes à fournir les résultats de calculs jusqu'alors impossibles, et par croissances des entreprises, d'autant plus difficiles à diriger qu'elles sont de plus grandes tailles, à partir de l'année 1945-1950. Toutefois, la Recherche Opérationnelle ne s'applique qu'aux problèmes devant lesquels le bon sens, ou plus exactement, le sens commun se révèle impuissant.

Il semble qu'on ait tout dit lorsqu'on indique que la Recherche Opérationnelle est le moyen, pour un dirigeant déterminé, d'obtenir le meilleur résultat possible d'une action engagée dans des conditions données. Ce qui revient, lorsqu'on est en train de bâtir un modèle de la réalité, à optimiser une certaine fonction objectif en présence de contraintes multiples.

La Recherche Opérationnelle peut se définir comme étant un ensemble de méthodes scientifiques cherchant à résoudre efficacement les problèmes posés par les activités des organisations humaines. C'est un ensemble de techniques mathématiques qui permettent de formaliser et de résoudre certains problèmes théoriques présentant des analogies avec des problèmes réel de gestion et c'est aussi cet art de représenter des systèmes réels par des modèles mathématiques et l'ensemble des méthodes quantitatives permettant l'exploitation de

ces modèles en vue de leur optimisation et c'est enfin l'application de méthodes scientifiques au contrôle et à la gestion des processus industriels, commerciaux, gouvernementaux et militaires.

En face de la complexité croissante des « opérations », que ce soit dans le domaine du bureau d'étude, de la fabrication, du service commercial ou du marketing, la complexité caractéristique d'une économie hautement développée et souvent puissamment compétitive, le travail du décideur comporte une multitude de tâches, impossible de les considérer indépendamment les unes des autres et pouvant avoir des incidences lointaines. Elles impliquent de nombreuses prises de décision et des problèmes de contrôle.

Avec les résultats surprenants et pratiques qu'à générer la mise en œuvre de la Recherche Opérationnelle, les mathématiciens, informaticiens et chercheurs se sont intéressés de plus près à cette science pour la développer et la rendre de plus en plus performante.

Aujourd'hui, les techniques utilisées en Recherche Opérationnelle se sont considérablement étendues sur un éventail de méthodes et techniques nouvelles. Elle a également largement profité du développement technologique notamment dans le domaine de l'informatique.

Permettant la conception et l'entretien de systèmes logistiques et techniques toujours plus complexes, la Recherche Opérationnelle fait aujourd'hui partie du bagage essentiel à tout ingénieur.

Introduction Générale[1]

L'optimisation combinatoire est une discipline récente des mathématiques discrètes. Plusieurs problèmes sont classés dans ce cadre ont beaucoup d'importance théorique ainsi que pratique, les chercheurs mettent beaucoup d'efforts afin de pouvoir développer et simplifier des méthodes pour les résoudre. L'optimisation combinatoire trouve ses racines dans l'analyse combinatoire, la recherche opérationnelle et l'informatique. Le problème de la recherche d'un stable (un ensemble de sommets indépendants) de poids maximum dans un graphe donné, est un problème d'optimisation combinatoire classique qui va faire une introduction à l'objet d'étude de ce mémoire.

L'optimisation combinatoire multi-objectif est un domaine qui généralise l'optimisation combinatoire classique en posant les mêmes problèmes mais avec plusieurs objectifs, ce cas général met toujours les décideurs face à des situations très conflictuelles et rend les problèmes encore plus délicats à résoudre. C'est pour cela qu'on trouve beaucoup de méthodes qui essayent d'arriver à un compromis qui met terme à cette situation en tentant de satisfaire au maximum les différents objectifs. Le problème du stable multi-objectif dans un graphe donné appartient à cette classe de problèmes et fera l'objet de notre étude.

La théorie de la complexité algorithmique qui traite l'efficacité des algorithmes selon le nombre d'opérations effectuées et la taille du problème traité, prend sa part dans ce travail afin de juger l'efficacité des algorithmes qui seront présentés.

Les algorithmes approximatifs, ou encore nommés heuristiques, méritent leurs places dans ce cadre-là afin de couvrir la faille laissée par les méthodes exactes dans le cas des problèmes difficiles à résoudre puisqu'on a toujours besoin d'une réponse en un temps acceptable, alors les heuristiques donnent des solutions approchées à ces problèmes mais sans une garantie de la qualité de la solution. Les métaheuristiques sont encore plus générales que les heuristiques et s'adaptent à plusieurs problèmes d'optimisation. Dans ce mémoire, le passage par les méthodes approximatives est primordial pour proposer des solutions pratiques pour le sujet étudié.

De ce fait, ce présent mémoire est décomposé en quatre chapitres dont le premier est consacré aux notions fondamentales de la théorie des graphes, la programmation linéaire et la théorie de la complexité. Le deuxième chapitre est une autre gamme de notions fondamentales concernant l'optimisation multi-objectif. On présente aussi la problématique du stable multi-objectif, objet de notre étude ainsi qu'un aperçu sur son état de l'art. Le troisième chapitre présente la méthode exacte développée afin de résoudre ce problème et met en évidence l'ensemble des résultats obtenus lors de l'expérimentation numériques. Enfin, le quatrième chapitre expose l'application d'une des meilleures métaheuristiques connues à nos jours pour résoudre des problèmes d'optimisation multi-objectif, en l'occurrence la méthode connue sous

[1] L'introduction est inspirée du livre « Combinatorial Optimization » pour B.Korte & J.Vygen [19]

le vocable anglophone de ‘’Non-dominated Sorting Genetic Algorithm-II’’ (NSGA-II), et présente les résultats expérimentaux afin de tester son efficacité pour le problème étudié.

Chapitre I :
Concepts de base

I. Introduction

Les gestionnaires des plus hauts niveaux aux petites entreprises sont souvent confrontés à de multiples problèmes, et leur résolution s'avère souvent une tâche difficile. Ces problèmes se présentent sous forme de données, de contraintes dont on doit tenir compte et d'un ou de plusieurs objectifs à atteindre. Pour arriver à résoudre un problème donné, on doit commencer par interpréter tous ses paramètres, et les transformer sous des formes qu'on peut gérer. Donc la première étape dans la résolution d'un problème est sa projection dans un espace qui permet diverses manipulations sur le problème projeté. Ce dernier s'appelle le *modèle* associé au problème.

La modélisation est donc une traduction des paramètres du problème dans un langage accessible par la méthode de résolution utilisée, ou bien c'est une façon de décrire le problème sous une forme qui introduit sa réalisation. Enfin, la modélisation d'un problème doit pouvoir donner une interprétation aux solutions concrètes répondant aux besoins du problème réellement posé.

Dans ce chapitre, on va voir quelques concepts de base sur la modélisation et la résolution des problèmes d'optimisation linéaires.

II. Théorie des graphes

La théorie des graphes est un des domaines de la recherche opérationnelle apparue la première fois dans un magazine de mathématiques présentant le fameux problème des « ponts de Königsberg ». Appliquée après dans l'électricité, la chimie, l'économie et beaucoup d'autres domaines avant même de connaitre ses fondements, d'où elle est devenue l'une des branches les plus reconnues de l'algèbre moderne qui le point sur beaucoup de problèmes mathématiques d'optimisation ou même les problèmes d'algèbre les plus classiques tel que le problème de Cayley sur la possibilité de coloration des pays sur les cartes géographiques en uniquement 4 couleurs de telle sorte que deux pays voisins n'ont pas la même couleur [2] et [3]. On va donner dans cette partie plusieurs généralités, définitions et théorèmes connus dans le domaine qu'on peut les trouver sur plusieurs ouvrages tels que [1], [14], [18] et [19].

II.1. Notions de base

On va définir initialement un graphe, on a deux définitions connues :

Définition d'un graphe selon Berge : Un graphe dénoté **G** est un schéma constitué par un ensemble (supposé fini) de points $v_1, v_2, \dots, v_n$, et par un ensemble de flèches reliant chacune deux de ceux-ci, et dénotées $e_1, e_2, \dots, e_m$. Les points sont appelés les **sommets** du graphe dénotés généralement **V** et les flèches les **arcs** du graphe dénotés généralement **E**, en plus, si les arcs ne sont pas orientés on les appellera les **arêtes**.

Définition algébrique d'un graphe : Un graphe non orienté est le triplet (V, E, Ψ) où V et E sont des ensembles finis et une application $\Psi: E \longrightarrow \{X \subseteq V : |X| = 2\}$. Un graphe orienté est le triplet (V, E, Ψ) où V et E sont des ensembles finis et une application

$\Psi: E \longrightarrow \{(v,w) \in V \times V : v \neq w\}$. Les éléments de V sont appelés *sommets* et les éléments de E sont appelés *arêtes*. On le note généralement $G = (V, E)$.

Quelques définitions de base

- Un graphe est dit ***simple*** s'il n'existe pas deux arêtes e et e' telles que : $\Psi(e) = \Psi(e')$ sinon le graphe est dit ***multi-graphe***.
- Si $e = \{v, w\}$ est une arête de G, les sommets v et w sont appelés les extrémités de l'arête. Si $v = w$, on dit que l'arête e est une ***boucle***.
- Deux sommets v et w sont dits ***adjacents*** s'il existe e dans E $telle\ que\ e = \{v, w\}$.
- Deux arêtes $e = \{v, w\}$ et $e' = \{v', w'\}$ sont dites ***adjacentes*** si elles ont un sommet en commun.
- Une arête $e \in E$ est dite **incidente** à un sommet $x \in V$ si x est une extrémité de e.
- Le ***degré*** d d'un sommet v est le nombre d'arêtes incidentes à v, une boucle étant comptée deux fois.
- On note δ le degré minimum et par Δ le degré maximum des sommets d'un graphe.

Quelques graphes particuliers

Soit $G = (V, E)$ un graphe, tel que $|V| = n$ et $|E| = m$.

- *Graphe complet d'ordre n (clique) :* $\forall\ v, w \in V, v \neq w\ alors\ \{v, w\} \in E$.
  On le note K_n.
- *Sous-graphe :* Un sous-graphe G_W de G engendré par $W \subset V$ est un graphe dont l'ensemble des sommets est W et dont les arêtes sont les arêtes de G qui ont leurs deux extrémités dans W.
- *Graphe partiel :* Un graphe partiel de G engendré par $U \subset E$ est un graphe dont l'ensemble des sommets est V et dont les arêtes sont celles de U.
- *Chaîne :* Une chaîne de longueur q est une séquence d'arêtes $[e_1, e_2, \dots, e_q]$ de G telle que chaque arête de la séquence a une extrémité en commun avec l'arête précédente, et l'autre extrémité en commun avec l'arête suivante. Le nombre d'arêtes de la séquence est la **longueur** de la chaîne. Une chaîne qui ne rencontre pas deux fois le même sommet est dite **élémentaire**. Une chaîne qui n'utilise pas deux fois la même arête est dite **simple**.
- *Cycle :* Un cycle est une chaîne telle que les deux sommets aux extrémités de la chaîne sont confondus $(ou\ e_1 = e_p)$.
- *Graphe connexe :* Un graphe connexe est un graphe tel que pour toute paire de v, w de deux sommets distincts, il existe une chaîne qui les relie.

II.2. Représentation d'un graphe

On peut représenter un graphe par une matrice selon plusieurs façons dont les plus utilisées sont :

- *Matrice d'incidence sommets-arêtes :*
  Soit $G = (V, E)$ un graphe non orienté et $|V| = n$ et $|E| = m$. On peut représenter G par la matrice $M'(G)$ dont les lignes sont les sommets et les colonnes sont les arêtes.

$$M'(G) = (m'_{ij}); 1 \leq i \leq n; 1 \leq j \leq m$$
$$m'_{ij} = \begin{cases} 1; si\ e_j incidente\ \text{à}\ v_i \\ 0; sinon \end{cases}$$

- *Matrice d'adjacence :*
  Soit $G = (V, E)$ un graphe tel que : $|V| = n$et $|E| = m$. On peut représenter G par la matrice $A(G)$ dont les lignes et les colonnes sont les sommets.
  $$A(G) = (a_{ij}); 1 \leq i \leq n; 1 \leq j \leq n$$
  où : a_{ij} est le nombre d'arcs ayant v_i comme extrémité initiale et v_j comme extrémité terminale dans le cas d'un graphe orienté. Pour le graphe non orienté, on peut le considérer comme le nombre d'arêtes entre les sommets $v_i$et v_j.
- *Table de hachage d'adjacence (Liste des adjacents):*

  Soit $G = (V, E)$ un graphe tel que : $|V| = n$ et $|E| = m$. On peut représenter G par la table de hachage $H_G = (V, D_v)$ dont on a un ensemble de clés qui représentent les sommets V où chaque sommet de l'ensemble des clés $v \in V$ pointe sur un ensemble de sommets qui sont adjacents à lui (D_v). En terme informatique, cette architecture consomme moins d'espace mémoire que les présentations précédentes.

III. Programmation linéaire

On va présenter dans ce paragraphe quelques généralités, définitions et théorèmes concernant la programmation linéaire ([13] et [26]).

Définition d'un programme linéaire : Soit (P) définie tel que :

$$(P): \begin{cases} cx = Z(\max) \ldots (1) \\ Ax = b \ldots (2) \\ x \geq 0 \ldots (3) \end{cases} \quad ; \text{avec :}$$

$A: m \times n$ - matrice des contraintes.
$b: m$ - vecteur colonne (second membre).
$c: n$ - vecteur ligne (vecteur des coûts).
$x: n$ - vecteur colonne.
(P) est appelé un programme linéaire. On note $D(P)$ le domaine formé par (2) et (3).
Donc ; un programme linéaire a pour but de résoudre un problème d'optimisation dans lequel :

- Les contraintes (2) e (3) délimitent dans un espace de n dimensions (le nombre de variables), s'ils sont compatibles, un hyper volume convexe dont à l'intérieur on peut trouver le (ou les) point(s) qui satisfai(en)t la fonction économique (1) (appelée aussi la fonction objectif).

Définition : Soit le programme linéaire (P') tel que:

$$(P'): \begin{cases} cx = Z(\max) \\ Ax \leq b \\ x \geq 0 \end{cases} \quad ; \text{ou bien : } (P'): \begin{cases} cx = Z(\min) \\ Ax \geq b \\ x \geq 0 \end{cases}$$

On dit que (P') est écrit sous forme **canonique**.

Définition : Soit le programme linéaire (P') tel que:

$$(P'):\begin{cases} cx = Z(\max) \\ Ax = b \\ x \geq 0 \end{cases} \text{ ; ou bien : } (P'):\begin{cases} cx = Z(\min) \\ Ax = b \\ x \geq 0 \end{cases}$$

On dit que (P') est écrit sous forme **standard**.

Définition : On appelle **base** de (P) un ensemble $J \subseteq \{1, \dots, n\}$ d'indices de colonnes tel que A^J (matrice de base associée à J) soit *carrée régulière*.

Remarque : Les indices j sont de base si $j \in J$ et hors base si $j \in \bar{J}$ d'où on a :

$$Ax = b \Leftrightarrow A^J x_J + A^{\bar{J}} x_{\bar{J}} = b$$

Définition : La solution $x_J = (A^J)^{-1} b$; $x_{\bar{J}} = 0$ est appelé **solution de base** associée à J. Cette solution est dite **dégénérée** si x_J contient des composantes nulles.

Définition : Le m-vecteur ligne $\pi = c^J (A^J)^{-1}$ est appelé **vecteur multiplicatif** associé à la base J. Le n-vecteur ligne $\hat{c}^J = \left(0, c^{\bar{J}} - \pi A^{\bar{J}}\right)$ est appelé **vecteur coût** relatif à la base J ou bien le **coût réduit**.

Définition : Une base J de (P) est dite **réalisable** si la solution associée vérifie $x_J \geq 0$. Une solution de base associée à une base réalisable est dite **solution de base réalisable**.

Théorème : *Soit J une base réalisable de (P). Si le vecteur coût réduit relatif à J est négatif ou nul alors la solution de base associée à J est **une solution optimale** de (P) et la base J est dite **une base optimale**.*

Théorème : *Théorème fondamental de la programmation linéaire*
Etant donné un programme linéaire (P) :

i. *Si $D(P) \neq \emptyset$ alors (P) admet une solution réalisable de base.*
ii. *Si (P) admet une solution optimale alors il admet une solution optimale de base.*
iii. *Si (P) admet une solution réalisable et que la fonction objectif est bornée $sup(P_{max})$ $(Inf(P_{min}))$ alors (P) admet une solution optimale.*

III.1. Dualité

Le concept de dualité est un concept fondamental en programmation linéaire. En fait, il faut considérer que deux programmes linéaires duaux ne constituent pas deux problèmes distincts mais deux aspects du même problème sachant que quand on résout un programme linéaire, on résout simultanément son dual.

Définition formelle du programme linéaire : Soit (P) le programme linéaire (dit **primal**) écrit sous forme canonique :

$$(P):\begin{cases} cx = Z(\max) \\ Ax \leq b \\ x \geq 0 \end{cases}$$

On appelle **dual** de ce programme linéaire, le programme linéaire (D) tel que :

$$(D): \begin{cases} yb = W(\min) \\ yA \geq c \\ y \geq 0 \end{cases}$$

$y: m$ - vecteur ligne.

Plus en détail, le tableau suivant va récapituler les différentes transformations du primal au dual :

Primal (Dual)	**Dual (Primal)**
Fonction objectif à maximiser	**Fonction objectif à minimiser**
$i^{ème}$ contrainte $\geq$	$i^{ème}$ variable ≤ 0
$i^{ème}$ contrainte $\leq$	$i^{ème}$ variable ≥ 0
$i^{ème}$ contrainte $=$	$i^{ème}$ variable $\gtrless 0$
$j^{ème}$ variable ≥ 0	$j^{ème}$ contrainte $\geq$
$j^{ème}$ variable ≤ 0	$j^{ème}$ contrainte $\leq$
$j^{ème}$ variable $\gtrless 0$	$j^{ème}$ contrainte $=$

Tableau 1 : Transformation du primal au dual

Théorème : *Le vecteur multiplicateur relatif à une base optimale de (P) est une solution optimale du dual (D) de (P) et de plus $W_{min} = Z_{max}$.*

Définition : On dit qu'un programme linéaire écrit sous forme standard est **primal réalisable** si $b \geq 0$.

Définition : On dit qu'un programme linéaire écrit sous forme standard est **dual réalisable** si $c \leq 0$.

Théorème : *Une base J est optimale si et seulement si (P) est primal et dual réalisable.*

III.2. Résolution d'un programme linéaire

Il existe plusieurs méthodes pour la résolution d'un programme linéaire (P) selon le type de contrainte qu'on a. On peut citer à titre d'exemples :

- La méthode du simplexe et celle du simplexe révisé pour résoudre un programme linéaire sous la forme standard qui est primal réalisable et on essaye d'obtenir la duale réalisabilité.
- La méthode dual du simplexe qui résout un programme linéaire sous la forme standard qui est dual réalisable et on essaye d'obtenir la primale réalisabilité.
- La méthode du simplexe aux variables bornées qui résout un programme linéaire sous forme standard avec des variables bornées qui est primal réalisable et on essaye d'obtenir la duale réalisabilité.

III.3. Programmation linéaire en nombres entiers

Soit le programme (P) suivant :

$$(P): \begin{cases} cx = Z(\max) \\ Ax \leq b \\ x \in \mathbb{N} \end{cases}$$

(P) est appelé un programme linéaire en nombres entiers.

Les méthodes de résolution d'un programme linéaire en nombres entiers sont essentiellement de type séparation et évaluation progressive, énumération implicite, programmation dynamique et méthodes des coupes (Gomory et autres) [26].

Les applications de ce type de problèmes sont très connues et utilisées, d'ailleurs, les différents problèmes cités au-dessus dans la section dédiée à la théorie des graphes peuvent être formulés au tant que programmes linéaires en nombres entiers. Par exemple :

- Les problèmes du pavage comme le « *problème du pavage de cardinalité maximum* », « *problème du couplage de cardinalité maximum* » ou celui du « *problème du stable de cardinalité maximum* » peuvent être formulé presque de la même façon, si on prend celui du stable de cardinalité maximum, on aura :

$$x_i = \begin{cases} 1; si\ le\ sommet\ i\ est\ pris\ dans\ le\ stable \\ 0; sinon \end{cases}$$

$$(P): \begin{cases} cx = Z(\max) \\ A^t x \leq e \\ x \in \{0,1\}^n \end{cases}$$

A est la matrice d'incidence du graphe étudié.
$c^t = e = \{1, \dots, 1\}^t$

IV. Complexité algorithmique

Dans cette section, on va voir une classification des méthodes de résolution selon leur efficacité en étudiant formellement la quantité de ressource (en temps d'exécution et en espace mémoire) nécessaire pour la résolution d'un problème d'optimisation. On va présenter dans ce paragraphe quelques généralités, définitions et théorème concernant la complexité algorithmique ([24] et [27]).

Définition : Etant données deux fonctions :$f, g : \mathbb{N} \longrightarrow \mathbb{N}$, on dit que f est de l'ordre de g ($O(g)$) s'il existe une constantes c et un rang N telle que :

$$(1) |f(n)| \leq c|g(n)| \ ; \forall\ n > N$$

Une fonction f est dite **polynômiale** si elle est $O(g)$ et si g est un polynôme en n. Ou encore s'il existe deux constantes c, k et un rang N telles que :

$$(2)\ \ f(n) \leq cn^k ; \forall n > N$$

Une fonction polynômiale f est dite « en n^p » si p est égal au plus petit scalaire k pour lequel (2) est satisfait.

Définition : Un algorithme est dit **polynômial** si le nombre d'opérations élémentaires nécessaire pour résoudre un exemple de taille n est borné par un polynôme en n. Un algorithme est considéré comme **efficace** si et seulement si il est polynômial.

Définition : Un problème est dit qu'il appartient à la **classe $\mathcal{P}$** s'il peut être résolu par un algorithme polynômial. Nous dirons que les problèmes de la classe $\mathcal{P}$ sont **faciles**.

Définition : Un problème de **reconnaissance** est un problème dont les résultats ne peuvent prendre que l'une des deux valeurs : *VRAI* ou *FAUX*.

Définition : Etant donné un problème d'optimisation combinatoire :

- Trouver $\hat{s} \in S \; tel \; que \; f(\hat{s}) = \min_{s \in S}[f(s)]$ (resp. $\max_{s \in S}[f(s)]$) et un nombre a, on définit « le problème de reconnaissance associé » :
  - Existe-t-il $\tilde{s} \in S \;\; tel \; que \; f(\tilde{s}) \leq a$ (resp. $f(\tilde{s}) \geq a$)

Théorème : *Si le problème de reconnaissance associé à un problème d'optimisation combinatoire donné est difficile, le problème d'optimisation combinatoire est lui-même difficile (en d'autres termes, le problème de reconnaissance est au moins aussi facile que le problème d'optimisation combinatoire auquel il est associé).*

Définition : On appelle un algorithme **non déterministe** s'il parvient à la solution correcte d'un problème de décision en bénéficiant d'une aide extérieure sous forme de *certificat*. Plus précisément :

- Pour chaque instance I du problème ayant une réponse « oui », il existe un certificat $C(I)$.
- Pour toute instance I du problème ayant une réponse « non », il n'existe pas une certificat $C(I)$ tel que l'algorithme retournera « oui » lorsqu'on lui donne I et $C(I)$.

Définition : Un problème appartient à la **classe $\mathcal{NP}$** s'il peut être résolu en temps polynômial par un algorithme non déterministe lorsque la procédure « choix » est comptée comme une opération élémentaire.

Définition : Soient P_1 et P_2 deux problèmes de reconnaissance. On dit que P_1 **se réduit (en temps polynômial) à** P_2 s'il existe un algorithme pour P_1 qui fait appel (comme un sous-programme) à un algorithme de résolution de P_2 et si cet algorithme de résolution de P_1 est polynômial lorsque la résolution de P_2 est comptabilisée comme une opération élémentaire.

Théorème : *Si le problème P se réduit en temps polynômial à P' et si P' peut être résolu par un algorithme polynômial, il en est de même de P.*

Définition : Un problème de reconnaissance est dit $\mathcal{NP} - \boldsymbol{complet}$ si tout problème de la classe $\mathcal{NP}$ peut se réduire polynômialement à lui.

Théorème:

Si le problème P est $\mathcal{NP} - complet$ et $P' \in \mathcal{NP}$ et si on peut mettre en évidence une réduction polynômiale de Pà P', alors P' est $\mathcal{NP} - complet$.

Définition : Un problème d'optimisation qui est $\mathcal{NP} - \boldsymbol{difficile}$(ou bien $\mathcal{NP} - \boldsymbol{dur}$), sa version en problème de reconnaissance est $\mathcal{NP} - complet$.

❖ **Quelques problèmes classés $\mathcal{NP} - \boldsymbol{complets}$:**

- Le problème du stable
- Le problème de la clique
- Le problème du recouvrement (couverture, transversal)
- Le problème du voyageur du commerce

IV.1. Quelques approches de résolutions

Différentes méthodes exactes existent pour résoudre les problèmes d'optimisation combinatoire dont les plus réputés, qu'on va détailler dans ce paragraphe, sont la méthode du **simplexe** et l'approche par **séparation et évaluation progressive**. La première est utilisée pour la résolution des programmes linéaires d'une façon générale en variables continues dont plusieurs travaux l'ont explicitée tel que [6], [8], [14], [19], [22] et [26], alors que la deuxième méthode est un peu plus générale, elle a été utilisée la première fois pour résoudre les programmes linéaires en nombres entiers après elle a été généralisée pour la majorité des problèmes combinatoires en nombres entiers[1], beaucoup de travaux l'ont détaillée et parmi ceux qui expliquent le concept de base sont [4] et [27].

IV.1.1. La méthode du simplexe

La méthode du simplexe de Dantzig [6] est la méthode la plus ancienne et la plus utilisée dans le domaine de la programmation linéaire, pratiquement très efficace malgré sa complexité algorithmique qu'elle a été prouvé qu'elle est exponentielle par Klee et Minty en 1972 [17].

Soit le programme linéaire suivant :

$$(P): \begin{cases} Z(\max) = cx - \alpha^* \\ Ax = b \\ x \geq 0 \end{cases}$$

(P) est écrit sous forme canonique par rapport à une base réalisable J.

Soit l'application suivante :

$$\begin{aligned} col : \{1, \dots, m\} &\to J \\ i &\mapsto col(i) \end{aligned}$$

Tel que : $A^{col(i)} = e_i$; où : e_i est le i$^{\text{ème}}$ vecteur canonique.

Algorithme du simplexe

1) Initialement, soit la solution réalisable de base initiale :

$$x^*_{col(i)} = b_i\big(i = \overline{1,m}\big);\ x^*_j = 0\ \big(\forall\, j \in \bar{J}\big)$$

Soit l'évaluation initiale α^*, et soit la matrice d'évaluation $M = (A|b)$.

2) Choisir s tel que $c^s > 0$

[1] Linéaires et non-linéaires

a) S'il existe, aller à (3)

b) Sinon, terminer et J est une base optimale, x^* est une solution optimale de base et $Z_{opt} = \alpha^*$)

3) Soit : $I = \{i / A_i^s > 0\}$

a) Si $I \neq \emptyset$, aller à (4)

b) Sinon, terminer(P) n'a pas de solution optimale

4) Soit $L = \left\{ l \ / \frac{b_l}{A_l^s} = \min_{i \in I} \left(\frac{b_i}{A_i^s} \right) \right\}$; et choisir $r \in L$ tel que :$r = \min_{l \in L} l$.

Effectuer le pivotage de la matrice M sur l'élément le r^{ème} ligne et la s^{ème} colonne.

$J' = \{J \cup \{s\}\}/\{col(r)\}$nouvelle base réalisable.

$M' = pivotage(n+1, m+1, r, s; M)$ matrice des coefficients de (P) écrit sous forme canonique par rapport à J'.

$x'_r = \frac{b_r}{A_r^s}$; $x'_{col(i)} = b_i - A_i^s x'_s$; $x'_j = 0, \forall\, j \notin J'$nouvelle solution de base réalisable.

$\alpha' = \alpha^* + c^s \frac{b_r}{A_r^s}$nouvelle évaluation.

Poser : $J \coloneqq J'; col(r) \coloneqq s;\ x^* \coloneqq x';\ \alpha^* \coloneqq \alpha'; M \coloneqq M'$.

Retourner en (2).

Fin de l'algorithme.

La première chose qu'on peut remarquer est que l'algorithme s'applique uniquement dans le cas où la primale réalisabilité est maintenue et en essayant d'avoir la duale réalisabilité en ayant des coûts réduits négatifs ou nuls pour les variables hors base $\left(c^{\bar{J}} \leq 0\right)$.

La deuxième remarque qu'on peut tirer est l'obligation d'avoir une solution de base réalisable de départ. En revanche, ce n'est pas toujours facile d'en tirer une juste en posant (P) sous sa forme canonique, c'est dans ce cas où une autre procédure entre en jeu afin de trouver une solution de base réalisable de départ pour appliquer après la méthode du simplexe, l'ensemble des procédures est appelé **la méthode des deux phases**.

Définition : Soit (P) un programme linéaire, on définit **le programme linéaire auxiliaire** (P_A) tel que :

$$(P_A): \begin{cases} \Psi(\min) = ev \\ Ax + Iv = b \\ x \geq 0; v \geq 0 \end{cases}$$

Avec : I mxm-matrice unité, v m-variables **artificielles** et $e = \{1, \dots, 1\}$.

(P_A) peut-être écrit sous forme canonique par rapport à la base artificielle $J_0 = \{n+1, \dots, n+m\}$ tel que :

$$(P_A): \begin{cases} \Psi(\min) - eb = -eAx \\ Ax + Iv = b \\ x \geq 0; v \geq 0 \end{cases}$$

Avec : $e = (1, \ldots, 1)$

Théorème : *Soit $(\overline{x}, \overline{v})$ une solution optimale de (P_A) alors (P) admet une solution réalisable $\overline{x}$ si et seulement si $\overline{v_i} = 0;\ \forall i = \overline{1, m}$.*

Algorithme de la méthode des deux phases

Phase 1 :

1) Le programme linéaire (P) est écrit sous forme standard.

2) Multiplier par (-1) les équations dont le second membre est négatif.

3) Associer à (P) le programme linéaire auxiliaire en ajoutant un minimum de variables artificielles.

4) Ecrire le programme linéaire auxiliaire sous forme canonique par rapport à une base réalisable.

5) Appliquer l'algorithme du simplexe au programme linéaire auxiliaire en supprimant la variable artificielle qui quitte la base. Soit $\overline{\Psi}$ le minimum de la valeur de la fonction objectif du programme linéaire.

5.1) Si $\overline{\Psi} > 0$ alors terminer ;(P) n'a pas de solution réalisable.

5.2) Si $\overline{\Psi} = 0$ alors faire sortir les variables artificielles de la base en appliquant répétitivement la procédure :

Soit v_r une telle variable artificielle.

Effectuer le pivotage faisant sortir v_r et entrer une variable initiale x_s.

Si cette opération est impossible, on supprime l'équation correspondante.

6) Le programme linéaire (P) est écrit sous forme canonique par rapport à une base réaliable.

Phase 2 :

Appliquer le simplexe à (P) sous la forme obtenue à la fin de la phase 1.

Fin de l'algorithme.

Maintenant dans le cas où (P) n'est pas primal réalisable mais dual réalisable, on ne peut pas appliquer l'algorithme du simplexe mais plutôt le dual du simplexe en maintenant la duale réalisabilité et en essayant d'avoir la primale réalisabilité en ayant $b \geq 0$.

Théorème : *(Théorème fondamental de la dualité)*

i) Si (P) et (D) (le programme linéaire dual de (P)) ont chacun une solution réalisable alors ils ont chacun une solution optimale et les valeurs de la fonction objectif sont égaux.

*ii) Si (P) (resp. (D)) admet un ensemble de solutions réalisables pour lesquelles la fonction objectif n'est pas bornée **sup** (resp. **inf**) alors (D) (resp. (P)) n'a pas de solution réalisable.*

iii) Si (P) (resp. (D)) admet une solution réalisable et (D) (resp. (P)) n'en admet paas alors (P) (resp. (D)) admet un ensemble de solutions réalisables tel que la fonction objectif n'est pas bornée ***sup*** *(resp.* ***inf****).*

iv) Il se peut que ni (P) ni (D) n'a de solution réalisable.

Définition : La base J est dite *primal* (resp. *dual*) si (P) est primal (resp. dual) réalisable.

Algorithme dual du simplexe

1) Initialement, (P) est écrit sous forme canonique par rapport à une base dual réalisable J. Soit la solution réalisable de base initiale :

$$x^*_{col(i)} = b_i\big(i = \overline{1,m}\big);\ x^*_j = 0\ \big(\forall\, j \in \bar{J}\big)$$

Soit l'évaluation initiale α^*, et soit la matrice d'évaluation $M = (A|b)$.

2) Choisir la ligne du pivot r tel que $b_r < 0$

a) S'il existe, aller à (3)

b) Sinon, terminer et J est une base optimale, x^* est une solution optimale de base et $Z_{opt} = \alpha^*$)

3) Soit : $K = \{k / A^k_r < 0\}$

a) Si $K \neq \emptyset$, aller à (4)

b) Sinon, terminer (P) n'a pas de solution réalisable

4) Choisir la colonne du pivot $T = \left\{t \,/ \frac{c^t}{A^t_r} = \min_{k\in K}\left(\frac{c^k}{A^k_r}\right)\right\}$; et choisir $s \in T$.

Effectuer le pivotage de la matrice M sur l'élément le r^{ème} ligne et la s^{ème} colonne.

$J' = \big\{J \cup \{s\}\big\}/\{col(r)\}$ nouvelle base réalisable.

$M' = pivotage(n+1, m+1, r, s; M)$ matrice des coefficients de (P) écrit sous forme canonique par rapport à J'.

$x'_r = \frac{b_r}{A^s_r}$; $x'_{col(i)} = b_i - A^s_i x'_s$; $x'_j = 0, \forall\, j \notin J'$ nouvelle solution de base réalisable.

$\alpha' = \alpha^* + c^s \frac{b_r}{A^s_r}$ nouvelle évaluation.

Poser : $J := J'; col(r) := s;\ x^* := x';\ \alpha^* := \alpha'; M := M'$.

Retourner en (2).

Fin de l'algorithme.

Il existe d'autres versions de la méthode du simplexe tel que le simplexe révisé, la méthode M ou encore le simplexe pour les variables bornées.

IV.1.2. Le principe de la séparation et évaluation

Beaucoup de problèmes concrets appartiennent à la classe NP-dur dont ils ne disposent pas d'algorithmes déterministes polynomiaux pour les résoudre, ce qui a entrainé les

chercheurs à exploiter les méthodes énumératives mais explorer toutes les solutions réalisables d'un domaine est théoriquement exponentiel. En revanche, avoir une méthode qui s'exécute en $\mathcal{O}(2^n)$ mieux qu'une autre qui s'exécute en $\mathcal{O}(n!)$. Ce qui les a poussés à développer des méthodes énumératives moins couteuses en temps d'exécution, c'est-à-dire, avoir la possibilité de ne pas explorer la totalité du domaine des solutions réalisables.

Une de ces méthodes est l'ensemble des méthodes basées sur le principe de la séparation et évaluation développé sur une idée de Descartes qui a dit : « *…le second [précepte], de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait pour les mieux résoudre… Et le dernier de faire partout des dénombrements si entiers et des revues si générales, que je fusse assuré de ne rien omettre…* » [27].

On définit Ω un ensemble fini et f une fonction objectif sur Ω et on note :

$$\max_{\omega\in\Omega} f(\omega) \left(resp. \min_{\omega\in\Omega} f(\omega)\right)$$

Théorème : *Pour n'importe quel ensemble Ω et n'importe quelle fonction réelle f sur Ω on a :*

$$\max_{\omega\in\Omega} f(\omega) = -\min_{\omega\in\Omega}[-f(\omega)]$$

La séparation

Le principe de séparation est basé sur la décomposition du problème en plusieurs sous problèmes tel que l'union de tous les sous problèmes donne le problème initial. En revanche, ce procédé veut dire qu'on va énumérer toutes les solutions réalisables du domaine ce qui n'est pas bénéfique en temps d'exécution (généralement le nombre de solutions réalisables d'un problème est exponentiel), donc l'intégration d'une procédure d'élimination d'une partie du domaine dont on est sûr que la solution optimale ne lui appartient pas est la bienvenue. Généralement, la séparation est faite selon la relaxation combinatoire[1] du problème.

L'évaluation

Le principe de séparation donne une hiérarchie sous forme d'une arborescence tel que la racine est le problème initial et en séparant on aura des sous problèmes qu'il faut évaluer. Selon le problème étudié, les méthodes d'évaluation peuvent être n'importe quelle méthode connue telle que le simplexe pour un programme linéaire en nombres entiers ou la méthode Hongroise pour le problème de voyageur de commerce par exemple. Des fois on se serve juste d'une approximation par défaut pour l'évaluation si on ne connait pas une méthode exacte pour résoudre le problème ou bien la méthode n'est pas polynomiale ou encore elle ne donne pas de bons résultats pratiques.

Ces évaluations servent à sonder quelques branches de l'arborescence sans avoir l'obligation d'énumérer toutes ses solutions réalisables. Si la branche n'est pas vide, on aura deux cas ; soit une solution réalisable qui satisfait la contrainte relaxée alors on évalue cette solution et on sonde sa branche, soit une solution réalisable qui ne satisfait pas la contrainte relaxée alors on évalue la solution et on vérifie si la meilleure solution actuelle, déjà rencontrée dans l'arborescence, a une évaluation meilleure alors on sonde cette branche car

[1] Voir la partie IV.2 du présent chapitre

l'adjonction d'une contrainte qui réduit le domaine ne donnera une solution meilleure que celle qu'on a trouvée, sinon on va séparer sur cette solution et réduit le domaine pour essayer de trouver une solution acceptable.

Cette procédure se déroule jusqu'à la stérilisation de toutes les branches de l'arborescence, et en faisant enregistrer la meilleure solution le long du processus, elle sera la solution optimale en fin de compte.

IV.2. Relaxation combinatoire

La relaxation combinatoire dans la programmation mathématique est le fait de céder une ou plusieurs contraintes pour élargir le domaine des solutions réalisables mais surtout de le rendre plus simple à exploiter. Cette façon de faire est utilisée dans les méthodes qui utilisent le principe de séparation et évaluation, plus précisément, pour la phase d'évaluation, il n'existe pas généralement des méthodes qui résolvent un programme linéaire en nombre entier, alors en le relaxant sur les contraintes d'intégrités, on peut utiliser le simplexe pour faire l'évaluation de la solution réalisable et vérifier après si la solution vérifie ces contraintes-là, il y a aussi le cas du problème du voyageur de commerce qu'on peut formuler son problème en tant que problème d'affectation en ajoutant une contrainte de connexité, alors en faisant la relaxation sur cette dernière contrainte, on utilise la méthode Hongroise pour l'évaluation et on vérifie après si la solution est connexe ou non. Donc, la relaxation combinatoire est l'outil principal des méthodes basées sur le principe de séparation et évaluation et serve à la simplification d'exploitation des problèmes d'une façon générale.

Chapitre II : Programmation linéaire multi-objectif

I. Introduction

De nombreux problèmes rencontrés dans la pratique, requièrent l'optimisation simultanée de plusieurs objectifs qui sont en général conflictuels. Dans les problèmes de production, par exemple, on ne vise pas uniquement un bénéfice maximum mais également des coûts de production minimaux. Des problèmes classiques de l'optimisation combinatoire ont été également étudiés en considérant au moins deux objectifs, on cite entre autres les problèmes : d'affectation, de plus court chemin, de sac à dos, de voyageur de commerce, de flot dans un réseau et d'ordonnancement.

La difficulté principale de tels problèmes est liée à la présence de confits entre les divers objectifs. En effet, les solutions optimales, pour un objectif donné, ne correspondent généralement pas à celles des autres objectifs pris indépendamment. De ce fait, il n'existe, la plupart du temps, aucun point de l'espace de recherche où toutes les fonctions objectifs sont optimales simultanément. On parle dans ce cas de problème d'optimisation multi-objectif qui définit un ensemble de solutions acceptables assurant un compromis entre les objectifs considérés.

II. Définitions et concepts de base

On considère le programme d'optimisation suivant :

$$(PMO)\begin{cases} Max\left(f_1(x), f_2(x), \dots, f_p(x)\right) \\ x \in \Omega \end{cases}$$

(PMO)est un programme d'optimisation multi-objectif caractérisé par :

- Un ensemble de solutions (alternatives ou actions) admissibles (réalisables) Ω,
- Des fonctions objectifs$F: \left(f_1: \Omega \longrightarrow \mathbb{R}, f_2: \Omega \longrightarrow \mathbb{R}, \dots, f_p: \Omega \longrightarrow \mathbb{R}\right)$ associant des valeurs à chaque solution admissible, et consiste à déterminer l'ensemble des solutions $\mathcal{X}^* = \{X_1^*, X_2^*, \dots, X_k^*\} \subset \Omega$ minimisant (ou maximisant) F tel que $X^* = (x_1^*, x_2^*, \dots, x_n^*)$. On note :

$$\max_{X_h \in \Omega}\{F(X_h)\} = \max_{X_h \in \Omega}\{f_1(X_h), f_2(X_h), \dots, f_p(X_h)\}\ ;\ \forall\ h \in \overline{1,k}$$

On va présenter quelques généralités, définitions et théorèmes concernant l'optimisation multi-objectif ([11], [12] et [30]).

II.1. Notion d'efficacité (Optimum de Pareto)

Soit $X = (x_1, x_2, \dots, x_n) \in \Omega$ une solution admissible. Chaque variable $x_i\ ;\ \forall\ i = \overline{1,n}$ est appelé **variable de décision**.

Le schéma de la figure -2- représente l'**espace des décisions,** noté $\mathcal{D}$.

Le schéma de la figure -1- représente l'**espace des critères,** noté $\mathcal{C}$.

Prenons une solution $\overline{X} \in \mathcal{D}$ et soit $X_h \in \mathcal{D}\backslash\{\overline{X}\}$ tel que : $f_j(\overline{X}) \leq f_j(X_h)$; $h = \overline{1,k}$; $\forall\ j = \overline{1,p}$ avec au moins une inégalité stricte.

S'il n'existe aucun indice h qui vérifie cette relation alors on dit que le vecteur $F(\overline{X})$ est **non-dominé** et $\overline{X}$ est une **solution efficace** pour le problème (PMO).
Si $X_1, X_2 \in \mathcal{D}$ tels que : $f_j(X_1) \geq f_j(X_2)$; $\forall\, j$ avec au moins une inégalité stricte alors, on dit que $F(X_1)$ **domine** $F(X_2)$.

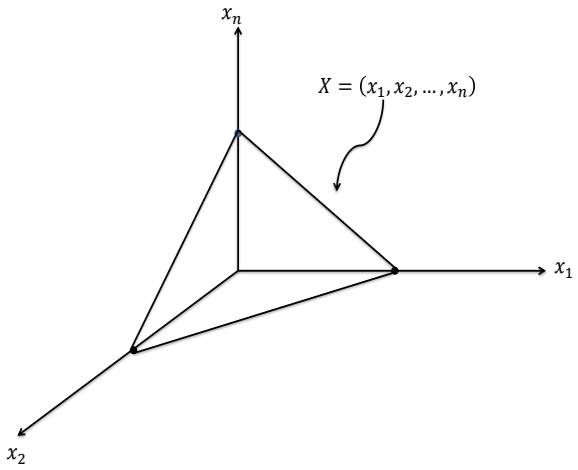


Figure 2 : Espace des décisions

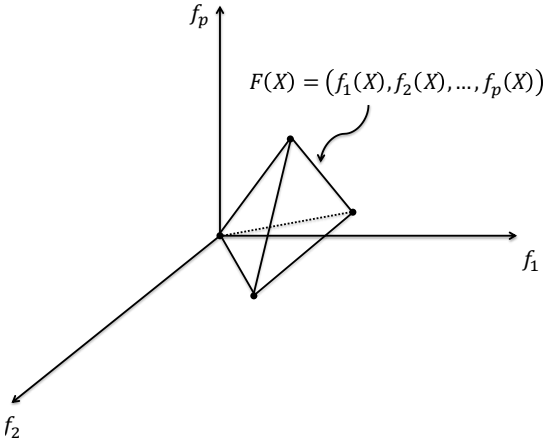


Figure 1 : Espace des critères

L'ensemble des solutions non-dominées est appelé **la frontière de Pareto**.

Définition : On appelle **point idéal,** le vecteur:

$$F(X^*) = \left(\max_{x\in\mathcal{D}}(f_1(x)), \max_{x\in\mathcal{D}}(f_2(x)), \dots, \max_{x\in\mathcal{D}}\left(f_p(x)\right)\right)$$

C'est le point qui prend les composantes des solutions optimales sur chaque fonction objectif. Le vecteur $F(X^*)$ domine toutes les solutions de $\mathcal{C}$, et généralement, $X^* \notin \mathcal{D}$ sinon X^* est l'unique solution efficace du domaine $\mathcal{D}$.

Définition : On appelle **point nadir,** le vecteur :

$$F(\overline{X}) = \left(\min_{x\in D_E}(f_1(x)), \min_{x\in D_E}(f_2(x)), \dots, \min_{x\in eD_E}\left(f_p(x)\right)\right)$$

oùD_E est l'ensemble des solutions efficaces de (PMO).

C'est le point qui prend le minimum des composantes des solutions efficaces sur chaque fonction objectif.

Définition : On appelle **point anti-idéal** le vecteur :

$$F(\underline{X}) = \left(\min_{x\in\mathcal{D}}(f_1(x)), \min_{x\in\mathcal{D}}(f_2(x)), \dots, \min_{x\in\mathcal{D}}\left(f_p(x)\right)\right)$$

II.2. Programmation linéaire d'optimisation multi-objectif

Plus généralement, on parle dans ce paragraphe des problèmes linéaires d'optimisation multi-objectif en nombres entiers. Si on prend un programme linéaire en nombres entiers avec plus d'un objectif, on aura :

$$(P)\begin{cases} Cx = Z_j(\max)\ j = 1..p \\ Ax \leq b \\ x \in \mathbb{N} \end{cases}$$

$A: m \times n$ - matrice des contraintes à composantes entières.
$b: m$ - vecteur colonne (second membre) à composantes entières.
$C: p \times n$ - matrice des objectifs (matrice des coûts).
$x: n$ - vecteur colonne.
(P)est appelé un programme linéaire multi-objectif en nombres entiers dont l'ensemble des solutions réalisables est $D = \{x \in \mathbb{R}^n | Ax \leq b,\ \ x \in \mathbb{N}\}$.

Comme application, nous citons le *problème du stable de cardinalité maximum de poids maximum* qu'on peut formuler comme suit :

$$(S)\begin{cases} (ex = Z_1(\max);\ px = Z_2(\max)) \\ A^t x \leq e \\ x \in \{0,1\}^n \end{cases}$$

où $A: m \times n$ - matrice d'incidence d'un graphe $G = (V, E)$, $e = \{1, \dots, 1\}$ et p est le vecteur des poids des sommets et $e_m = \{1, \dots, 1\}^t$ à m composantes.

II.3. Théorème de Geoffrion [15]

On définit le programme linéaire relaxé de (P) comme suit :

$$(P')\begin{cases} Cx = Z_j(\max) j = 1..p \\ Ax \leq b \\ x \geq 0 \end{cases}$$

Théorème : *Etant donné le programme* $(P_\lambda): Max\{\lambda^t Cx\ | A^t x \leq b \text{ et } x \geq 0\}$ *tel que* λ *vérifie la propriété suivante :* $\left\{\forall\ \lambda \in \mathbb{R}^p \middle| \sum_{i=1}^{p} \lambda_i = 1,\ \lambda_i \geq 0,\ i = \overline{1,p}\right\}$ *; 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, les solutions efficaces de (P') sont celles trouvées en résolvant (P_λ) et en faisant varier λ à chaque fois. Mais la vraie difficulté est quand les variables sont entières car on aura des solutions efficaces pour (P_λ) qui ne sont pas optimales pour (P). Les solutions efficaces de (P') sont appelées les *solutions supportées* tandis que les autres sont les *solutions non-supportées*.

II.4. Le cône dominant

Le cône est l'outil principal utilisé dans les résolutions graphiques des problèmes d'optimisation multi-objectif.

Définition : Soit $c \in C,\ C \subset \mathbb{R}^n, C \neq 0$. Alors C est un cône si et seulement si $\alpha c \in C$ pour tout scalaire $\alpha \geq 0$. L'origine $0 \in \mathbb{R}^n$ est contenu dans chaque cône.

Excepté l'ensemble singleton qui contient uniquement l'origine, tous les cônes sont non bornés.

Définition : Soit $C \subset \mathbb{R}^n$ un cône. On définit le cône polaire non négatif de C (noté $C^{\geq}$), le cône convexe suivant:

$$C^{\geq} = \{y \in \mathbb{R}^n | y^t c \geq 0 \; pour \; tout \; c \in C\}$$

Définition : Le cône polaire semi-positif généré par les gradients des fonctions objectifs de *(P)*,noté $C^{\geq}$, est donné par :

$$C^{\geq} = \{y \in \mathbb{R}^n / Cy \geq 0; Cy \neq 0\} \cup \{0_{\mathbb{R}^n}\}$$

Définition : L'ensemble dominant en $\overline{x} \in D$ est défini par :

$$D_{\overline{x}} = \{\overline{x}\} \oplus C^{\geq}$$
$$= \{x \in \mathbb{R}^n / x = \overline{x} + y; Cy \geq 0; Cy \neq 0\}$$

Cet ensemble contient tous les points $x \in D$dont le vecteur critère domine le vecteur critère de $\overline{x} \in D$.

$D_{\overline{x}}$représente la translation du cône polaire $C^{\geq}$ au point $\overline{x}$.

Théorème : *Soit $D_{\overline{x}}$ l'ensemble dominant en $\overline{x} \in D$. Alors $\overline{x}$ est une solution efficace de (P) si et seulement si $D_{\overline{x}} \cap D = \{\overline{x}\}$.*

Corollaire : *Si $C^{\geq} = \{0_{\mathbb{R}^n}\}$ alors, $\forall x \in D, x$ est une solution efficace de (P).*

II.5. Coupe efficace [5]

Une coupe est dite efficace pour le problème (P), si son adjonction au domaine D supprime au moins une solution réalisable contenue de D, mais ne supprime pas de solutions réalisables entières efficaces de D.

Notons que la définition d'une coupe efficace pour un problème d'optimisation linéaire multi-objectif généralise la définition d'une coupe pour un problème linéaire en nombres entiers (au mono-objectif).

III. Résolution d'un problème d'optimisation multi-objectif

La résolution d'un problème d'optimisation multi-objectif revient à trouver l'ensemble de toutes les solutions efficaces dans l'espace de décision qu'on dénote D_E, ou bien l'ensemble de toutes les solutions non dominées dans l'espace des critères. Des méthodes générales de résolution existent dans la littérature, on peut citer :

- La méthode graphique pour les problèmes à deux variables,
- La méthode de Sylva et Crema [31],
- La méthode du simplexe pour le bi-objectif [12],
- La méthode de la somme pondéré [11],
- La méthode de Benson [11],
- La méthode des ensembles efficaces complets [5],
- La méthode des deux phases généralisée [25],

- La méthode de Özlen et Azizoğlu [23],
- La méthode k-PPM [9].

III.1. La méthode graphique

En utilisant le dernier théorème cité au-dessus, on va chercher les solutions efficaces en faisant des intersections entre le cône dominant et l'espace de décision, tel que si l'intersection est un seul point X^* alors il est efficace. Par exemple :

On considère le programme linéaire suivant :$(P)\begin{cases} Max\ (x_1, -x_1 + x_2) \\ x_1 + 2x_2 \leq 10 \\ x_1 + x_2 \leq 6 \\ x_1 \leq 4 \\ x_1, x_2 \in \mathbb{Z}_+ \end{cases}$

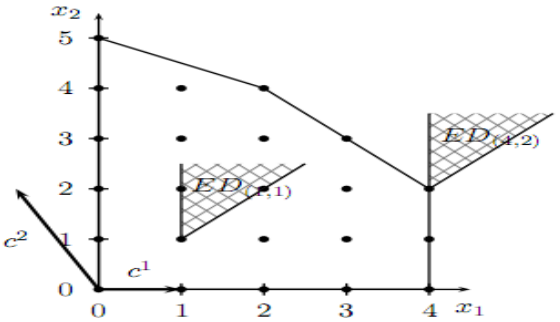


Figure 4 : Espace des décisions de (P)

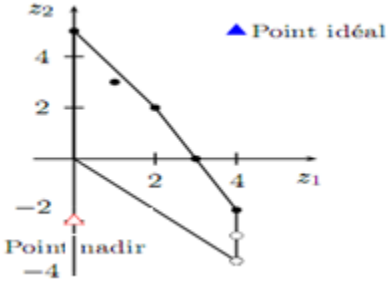


Figure 3 : Espace des critères de (P)

On remarque que la solution (4, 2) est une solution efficace car l'intersection du cône dominant avec l'espace des décisions au point (4, 2) ne donne que le point (4, 2), et c'est la même chose qu'on peut dire sur les solutions (3, 3), (2, 4), (1, 4) et (0, 5) dont on remarque leur évaluation dans l'espace des critères par des points noirs. Par contre, les autres solutions qui restent telle que la solution (1, 1), on remarque que l'intersection comporte plusieurs solutions entières, alors cette solution est non efficace car son image est dominée par les images de l'ensemble des solutions entières qui se trouvent dans l'intersection. La solution (1,4) est efficace car $D_{(1,4)} \cap D = \{(1,4)\}$.

III.2. La méthode des ensembles efficaces complets (EEC) [5]

Dans cette partie, une méthode exacte pour la génération de l'ensemble des solutions efficaces d'un programme linéaire multi-objectif en nombres entiers est décrite et est précisément choisie car elle fait l'objet de cette étude en l'utilisant dans l'approche par séparation et évaluation proposée dans le chapitre IV. Alors que généralement les méthodes publiées commencent par résoudre un programme linéaire en nombres entiers, cette méthode a l'avantage de commencer par une solution optimale d'un programme linéaire continu dont l'objectif est une combinaison positive des critères qui serve à balayer l'espace des décisions, et utilise une procédure de séparation pour générer une solution entière réalisable. Chaque fois qu'une telle solution est trouvée, les directions de croissances des critères sont identifiées et une coupe efficace est construite de manière à supprimer certaines des solutions non efficaces, sans les calculer. Par rapport à certaines méthodes [23] et [31] à chaque étape de la

séparation, les programmes linéaires en nombres entiers considérés sont complétées par de nouvelles contraintes et variables selon le nombre de critères p, cette méthode ne dépend pas de p puisqu'indépendamment du nombre des critères, on ajoute une seule contrainte et une seule variable à chaque étape de la séparation ou lors de l'adjonction de la coupe efficace.

III.2.1. Définitions et notations

On considère le problème linéaire multi-objectif en nombres entiers suivant :

$$(P)\begin{cases}\max Z^1 = c^1 x \\ \max Z^2 = c^2 x \\ \vdots \\ \max Z^p = c^p x \\ x \epsilon S \\ x \ entier\end{cases}$$

où $S = \{x \epsilon \mathbb{R}^n | Ax = b, x \geq 0\}$ et $Z^q = c^q x, q = 1, \dots, p; p \geq 2,$ sont les valeurs réelles de l'objectif, $c^q = (c_j^q)_{j=1,\dots,n}$. On suppose que S est un polyèdre compact non vide, toutes les composantes de la $m \times n$ matrice $A = (a_{ij})_{i=1,\dots,m; j=1,\dots,n}$ et le m-vecteur b sont entiers.

Une solution x est une solution efficace, s'il n'existe pas une autre solution y telle que : $c^q y \geq c^q x$ pour tout $q \epsilon \{1, \dots, p\}$ et $c^q y > c^q x$ pour au moins une fonction objectif $q \epsilon \{1, \dots, p\}$. Autrement, x n'est pas efficace et le vecteur Cy domine le vecteur Cx, quand $C = (c^q)_{q \epsilon \{1,\dots,p\}}$.

Considérons le programme linéaire suivant :

$$(P_l)\begin{cases}\max Z = \sum_{q=1}^{p} \lambda_q c^q x \\ x \epsilon S_l\end{cases}$$

Avec $\lambda_q \geq 0 \ \forall q = 1, \dots, p$; Pour $l = 0$, on a $S_0 = S$ tel que le choix des λ_q est arbitraire.

A chaque étape qu'une solution entière x_l^* est obtenue, on associe les paramètres suivants :

- B_l et N_l sont respectivement, les ensembles des indices des variables de base et des variables hors base de x_i^*,
- $H_l = \{j \in N_l | \exists q \in \{1, \dots, r\};\ \hat{c}_j^q > 0\} \cup \left\{ j \epsilon N_l \middle| \hat{c}_j^q = 0, \forall q \in \{1, \dots, p\} \right\}$ est l'ensemble des directions possibles de croissance des critères, où $\hat{c}_j^q$ est la $j^{ème}$ composante du vecteur coût réduit du critère Z^q,
- Les deux sous-ensembles de S_l sont définis comme suit :

$$S_{l+1} = \{x \in S_l | \sum_{j \in H_l} x_j \geq 1\} \text{ et } T_{l+1} = \{x \in S_l | \sum_{j \in N_l \setminus H_l} x_j \geq 1\}.$$

III.2.2. Principe de la méthode

La méthode est basée sur le concept de séparation et évaluation bien connu en programmation linéaire en nombres entiers. Toutes les opérations décrites ci-dessous sont identifiées par des nœuds et des branches dans une structure arborescente. A chaque nœud, nous avons à résoudre un programme(P_l).

Le nœud l de l'arbre est saturé si le programme correspondant(P_l) n'est pas réalisable ou si $H_l = \emptyset$. Si la solution optimale x_l du programme correspondant (P_l) n'est pas entière, soit $\overline{x_{lj}}$ une coordonnée de x_l fractionnaire. Alors, le nœud de l est séparé en deux nœuds avec les contraintes supplémentaires $x_j \leq \lfloor \overline{x_{lj}} \rfloor$ et $x_j \geq \lceil \overline{x_{lj}} \rceil$ respectivement. Chaque branche correspondante définit un nouveau programme linéaire à résoudre. Le cas correspondant à une solution entière x_l est résolu en utilisant les directions de croissances des critères pour éviter d'explorer les régions non efficaces des solutions possibles du problème (P). Seule la partie du domaine des solutions réalisables dans laquelle au moins l'un des objectifs du problème *(P)* peut être amélioré est traitée. Ceci est rendu possible par l'ajout de la contrainte $\sum_{j \in H_l} x_j \geq 1$ valide appelée coupe efficace, notons que si $H_l = N_l$ on aura la coupe de Dantzig : $\sum_{j \in N_l} x_j \geq 1$.

III.2.3. Algorithme

Etape 1 : (Initialisation)
$S_0 := S$, $l := 0$ et $eff := \emptyset$; (ensemble des solutions efficaces de (P))
Résoudre le programme linéaire (P_0) au nœud 0. Soit $\overline{x_0}$ la solution optimale obtenue.
Si $\overline{x_0}$ n'est pas entière, aller à l'étape 2a, sinon aller à l'étape 2b. On pose $x^* := \overline{x_0}$.

Etape 2 : (Etape générale)
Tant qu'il existe un nœud non sondé dans l'arborescence faire :

- Choisir le nœud l le plus récemment créé non encore sondé et résoudre le programme linéaire correspondant (P_l).
- Si (P_l) est non réalisable, alors le nœud l est sondé.
- Sinon, soit $\overline{x_l}$ une solution optimale de (P_l).
- Si $\overline{x_l}$ n'est pas entière, aller à l'étape 2a.
- Sinon, poser x_l^* la solution entière trouvée et aller à l'étape 2b.

Etape 2a :
Soit $\overline{x_{lj}}$ une coordonnée fractionnaire de $\overline{x_l}$. Séparer le nœud l en deux nouveaux nœuds : ajouter la contrainte $x_j \leq \lfloor \overline{x_{lj}} \rfloor$ au premier nœud et la contrainte $x_j \geq \lceil \overline{x_{lj}} \rceil$ au second nœud et aller à l'étape 2.

Etape 2b :
Si Cx_l^* n'est pas dominé par Cy pour toute solution $y \in eff$, alors $eff := eff \cup \{x_l^*\}$.
S'il existe $y \in eff$ telle que Cy est dominé par Cx_l^*, alors $eff := eff \backslash \{y\} \cup \{x_l^*\}$.
Déterminer les ensembles B_l, N_l et H_l.
Si $H_l = \emptyset$, alors le nœud correspondant est sondé, aller à l'étape 2. Sinon :

- Rajouter la contrainte $\sum_{j\in H_l} x_j \geq 1$ pour obtenir l'ensemble S_{l+1}.
- Résoudre le programme linéaire obtenu (P_{l+1}) par la méthode duale du simplexe et poser $\overline{x_{l+1}}$ la solution optimale trouvée.
- Si $\overline{x_{l+1}}$ est entière, poser x^*_{l+1} la solution entière trouvée et aller à l'étape 2b.
- Sinon, aller à l'étape 2a.

II.2.4. Exemple

On considère l'exemple suivant :

$$(P)\begin{cases} \max Z_1 = 3x_1 - x_2 \\ maxZ_2 = -x_1 + x_2 \\ 4x_1 + 3x_2 \leq 20 \\ x_1 - x_2 \leq 3 \\ x_2 \leq 4 \\ x_1 \geq 0, x_2 \geq 0, entiers \end{cases}$$

Le programme linéaire (P_0) est résolu avec l'objectif Z_1 sans contrainte d'intégrité des variables. La solution optimale (29/7, 8/7) est obtenue dans le tableau du simplexe suivant :

B	b	x_3	x_4
x_2	8/7	1/7	-4/7
x_1	29/7	1/7	3/7
x_5	20/7	-1/7	4/7
$-Z_1$	-79/7	-2/7	-13/7
$-Z_2$	3	0	1

Tableau 2 : Table final du simplexe

La séparation est déclenchée avec création de deux nœuds. Le nœud 1 est sondé car en rajoutant la contrainte $x_1 \geq 5$, le problème est non réalisable. Au nœud 2, la contrainte $x_1 \leq 4$ est rajoutée au tableau 2 et la solution entière optimale (4,1) est obtenue dans le tableau 3 suivant après avoir utilisé le dual du simplexe :

B	b	x_4	x_6
x_2	1	-1	1
x_1	4	0	1
x_5	3	1	-1
x_3	1	3	-7
$-Z_1$	-11	-1	-2
$-Z_2$	3	1	0

Tableau 3 : Résultat final du dual du simplexe au nœud 2

$eff \coloneqq \{(4,1)\}$, $N_2 \coloneqq \{4,6\}$et $H_2 \coloneqq \{4\}$.

La coupe efficace $x_4 \geq 1$ est alors rajoutée et la solution optimale (26/7, 12/7) est donnée dans le tableau 4 :

B	b	x_3	x_7
x_2	12/7	1/7	-4/7

x_1	26/7	1/7	3/7
x_5	16/7	-1/7	4/7
x_6	2/7	-1/7	-3/7
x_4	1	0	-1
$-Z_1$	-66/7	-2/7	-13/7
$-Z_2$	2	0	1

Tableau 4 : Résultat final du dual du simplexe au nœud 2.1

Séparation en deux nouveaux nœuds car la solution n'est pas entière. La contrainte $x_2 \leq 1$ est rajoutée au tableau 4 au nœud 3 et le nœud 4 est créé après adjonction de la contrainte $x_2 \geq 2$ au tableau 4. La solution optimale (3,1) est obtenue au nœud 3 :

B	b	x_7	x_8
x_2	1	0	7
x_1	3	1	7
x_5	3	0	-1
x_6	1	-1	-1
x_4	1	-1	0
x_3	5	-4	-7
$-Z_1$	-8	-3	-2
$-Z_2$	2	1	0

Tableau 5 : Résultat final du dual du simplexe au nœud 3

$eff \coloneqq \{(4,3);(3,1)\}$, $N_3 \coloneqq \{7,8\}$et $H_3 \coloneqq \{7\}$.

On rajoute la coupe efficace $x_7 \geq 1$ et la solution entière optimale (2,1) est obtenue dans le tableau 6 :

B	b	x_8	x_9
x_2	1	7	0
x_1	2	7	1
x_5	3	-1	0
x_6	2	-1	-1
x_4	2	0	-1
x_3	9	-7	-4
x_7	1	0	-1
$-Z_1$	-5	-2	-3
$-Z_2$	1	0	1

Tableau 6 : Résultat final du dual du simplexe au nœud 3.1

$eff \coloneqq \{(4,1);(3,1);(2,1)\}$, $N_4 \coloneqq \{8,9\}$ et $H_4 \coloneqq \{9\}$.

On rajoute la coupe efficace $x_9 \geq 1$, … etc.

L'exploration en profondeur à partir du nœud 3 donne le dernier tableau 7 suivant :

B	b	x_8	x_{12}
x_2	1	7	0
x_1	-1	7	1
x_5	3	-1	0
x_6	5	-1	-1

x_4	5	0	-1
x_3	21	-7	-4
x_7	4	0	-1
x_9	3	0	-1
x_{10}	2	0	-1
x_{11}	1	0	-1
$-Z_1$	4	-2	-3
$-Z_2$	-2	0	1

Tableau 7 : Le résultat final de l'exploration du nœud 3.1

Et le nœud correspondant est sondé car le dual n'est pas réalisable.

$$eff \coloneqq \{(4,1);(3,1);(2,1)\}$$

Au nœud 4, la contrainte $x_2 \geq 2$ est rajoutée au tableau 4 et la solution optimale non entière (7/2, 2) est obtenue. Les nœuds 5 et 6 sont créés. Le nœud 5 est sondé compte tenu de la contrainte $x_1 \geq 4$.

Au nœud 6, la contrainte $x_1 \leq 3$ est rajoutée et la solution entière optimale (3,2) est obtenue.

$eff \coloneqq \{(4,1);(3,1);(3,2)\}$, $N_6 \coloneqq \{8,9\}$ et $H_6 \coloneqq N_6$.

La contrainte $x_8 \geq 1$ est rajoutée.

En procédant de cette manière, dans le tableau 8 on obtient la solution entière optimale (2,4) au dernier nœud 8 :

B	b	x_{11}	x_{12}
x_2	4	0	-1
x_1	2	1	0
x_5	0	0	1
x_6	2	-1	0
x_4	5	-1	-1
x_7	4	-1	-1
x_9	1	-1	0
x_8	2	0	-1
x_3	0	-4	3
x_{10}	1	0	-1
$-Z_1$	-2	-3	-1
$-Z_2$	-2	1	1

Tableau 8 : Le dernier résultat du dual du simplexe au nœud 8

$eff \coloneqq \{(4,1);(3,1);(3,2);(2,4)\}$, $N_9 \coloneqq \{11,12\}$ et $H_9 \coloneqq N_9$.

On rajoute la contrainte $x_{11} + x_{12} \geq 1$ et on obtient le dernier tableau 9 :

B	b	x_3	x_{13}
x_2	5	0	-1
x_1	5/4	¼	¾
x_5	-1	0	1
x_6	11/4	-1/4	-3/4

x_4	27/4	-1/4	-7/4
x_7	23/4	-1/4	-7/4
x_9	7/4	-1/4	-3/4
x_8	3	0	-1
x_{11}	3/4	-1/4	-3/4
x_{10}	2	0	-1
x_{12}	1	0	-1
$-Z_1$	5/4	-3/4	-13/4
$-Z_2$	-15/4	¼	7/4

Tableau 9 : Dernier résultat du dual du simplexe au nœud 8.1

Le dual n'est pas réalisable, donc le nœud 8 est sondé.

Comme tous les nœuds de l'arborescence sont sondés, alors l'algorithme s'arrête et l'ensemble efficace complet est trouvé :

$$eff \coloneqq \{(4,1);(3,1);(3,2);(2,2);(2,3);(2,4);(1,4);(0,4)\}$$

Chapitre III : Problème du stable multi-objectif

I. Introduction

Nombreux sont les problèmes théoriques que pratiques qui se présentent sous forme d'un graphe, on peut citer les plus connus et classiques de la théorie des graphes tels que les problèmes de partitionnement, les problèmes de plus court/long chemin entre deux points, les problèmes d'ordonnancement, les problèmes des canalisations et d'autres problèmes plus particuliers tels que le problème du voyageur de commerce ou celui du postier chinois [18]. Dans notre cas d'étude, on s'intéresse aux problèmes de partitionnement qu'on va détailler.

Définition : Soit $\mathfrak{P}$ une propriété. On dit que l'ensemble S est ***minimal*** pour la propriété $\mathfrak{P}$ si S vérifie $\mathfrak{P}$ et aucun sous ensemble strict de S ne vérifie cette propriété. Un ensemble est dit ***minimum*** pour la propriété $\mathfrak{P}$ si aucun ensemble plus petit (pas nécessairement un sous ensemble) ne vérifie la propriété.

Définition : Soit $\mathfrak{P}$ une propriété. On dit que l'ensemble S est ***maximal*** pour la propriété $\mathfrak{P}$ si S vérifie $\mathfrak{P}$ et aucun ensemble contenant S et différent de S ne vérifie cette propriété. Un ensemble est dit ***maximum*** pour la propriété $\mathfrak{P}$ si aucun ensemble plus grand que S ne vérifie cette propriété.

Définition d'un stable : Soit $G = (V, E)$ un graphe simple. Un sous ensemble de sommets $S \subseteq V$ est dit ***stable*** de G si les sommets de S sont deux à deux non adjacents.

$$\forall\, v, w \in S, (v, w) \notin E$$

- Un stable S_0 est ***maximum*** si : $|S_0| = Max\{|S| \ / S\ stable\ de\ G\}$.
- Un stable S est ***maximal*** si : $\forall\, v \in V \backslash S$, $S \cup \{v\}$ n'est plus un stable.
- Le cardinal maximum d'un ensemble stable de G noté $\alpha(G)$ est appelé le ***nombre de stabilité***.

Le problème du stable de poids maximum est un problème d'optimisation combinatoire dont on peut se modéliser comme suit :

$$x_i = \begin{cases} 1; si\ le\ sommet\ i\ est\ choisi \\ 0; sion \end{cases}$$

$$(P) \begin{cases} wx = Z(\max) \\ A^t x \leq e \\ x \in \{0,1\}^n \end{cases}$$

$A : n \times m$ - matrice d'incidence du graphe $G = (V, E)$ donné,
w : vecteur de poids à n compsantes.
$e = \{1, \dots, 1\}^t$

Problématiques liées au stable :

- *Problème du stable de cardinalité maximum dans un graphe* ([10] et [28])
- *Problème du stable de poids maximum dans un graphe* ([20])
- *Problème de l'énumération de tous les stables maximaux d'un graphe* ([16] et [32])

Problématiques duales liées aux problématiques liées au stable :

Définition d'un transversal : Un transversal T est un sous ensemble de sommets dans un graphe tel que toute arête de ce graphe ait au moins une de ses extrémités dans T.

- *Problème du transversal de cardinalité minimum dans un graphe*
- *Problème du transversal de poids minimum dans un graphe*
- *Problème de l'énumération de tous les transversaux minimaux d'un graphe*

Remarque : $S \subset V$ est un stable si et seulement si $V \backslash S$ est un transversal.

II. Problème du stable multi-objectif

La programmation linéaire multi-objectif a connu beaucoup de développements ces derniers temps, et différentes méthodes sont mises au point pour améliorer l'efficacité de la résolution de ce type de problèmes. Aussi, il n'existe pas encore une méthode telle que la méthode du simplexe pour les problèmes de la programmation linéaire, qui est théoriquement exponentielle [17] mais très avantageuse pratiquement. En revanche, si on voit du côté des problèmes d'optimisation combinatoire multi-objectif, on trouve que les propriétés de quelques problèmes particuliers de l'optimisation combinatoire, dans le cas général, sont intéressantes et exploitables telles que la structure de la matrice des contraintes et du vecteur du second membre dans le problème du stable. Cette voie peut donner de nouvelles opportunités pour la résolution de ce type de problèmes en examinant soigneusement les structures du problème qu'on a entre les mains.

Pour que les idées soient claires, dans ce travail, on va prendre le cas du « *problème du stable multi-objectif* », on redéfinit le problème du stable de poids maximum sous une forme d'un programme linéaire (P):

$$x_i = \begin{cases} 1; si\ le\ sommet\ i\ est\ choisi \\ 0; sion \end{cases}$$

$$(P)\begin{cases} wx = Z(\max) \\ A^t x \leq e \\ x \in \{0,1\}^n \end{cases}$$

A: $n \times m$ - matrice d'incidence du graphe $G = (V, E)$ donné,
w: vecteur de poids à n compsantes.
$e = \{1, \dots, 1\}^t$

Ce problème d'optimisation est classé $\mathcal{NP} - difficile$, il a beaucoup d'importance théorique que pratique. On peut utiliser une méthode par séparation et évaluation pour résoudre le problème. A partir de là, on cherche à appliquer un aspect similaire de résolution pour le même problème en multi-objectif défini comme suit :

$$(MOISP)\begin{cases} w_j x = Z_j(\max)\ ;\ \forall j = \overline{1,p} \\ A^t x \leq e \\ x \in \{0,1\}^n \end{cases}$$

w_j : $j^{\text{ème}}$ vecteur de poids du à n composantes.

$e = \{1, \dots, 1\}^t$

$(MOISP)$ est la forme du programme linéaire du problème du stable multi-objectif, donc à première vue, on voit que le problème peut être résolu en tant qu'un programme linéaire multi-objectif en nombres entiers par la méthode des ensembles efficaces complets par exemple. En revanche, le temps de la résolution du problème est extrêmement grand même pour des petites instances, sachant que le problème est classé $\mathcal{NP} - difficile$.

Comme application, nous citons le *problème du stable de cardinalité maximum de poids maximum* qu'on peut formuler comme suit :

$$(S)\begin{cases}\left(\sum_{i=1}^{n} x_i = Z_1(\max);\ \sum_{i=1}^{n} p_i x_i = Z_2(\max)\right)\\ A^t x \leq e\\ x \in \{0,1\}^n\end{cases}$$

où $A : m \times n$ - matrice d'incidence d'un graphe $G = (V, E)$, p_i est le poids du sommet i et $e = \{1, \dots, 1\}^t$.

De ce point de vue, le but de ce mémoire est de pouvoir contribuer cette méthode, et de mettre au point un algorithme qui prend en considération la particularité du problème. On va le tester sur une batterie d'instances générées aléatoirement, et on compare le temps d'exécution avec celui de la méthode des ensembles efficaces complets qui va être notre référence. En plus, on va adapter une métaheuristique de type algorithme génétique connue dans la littérature nommée NSGA-II.

Chapitre IV : Une Méthode par séparation et évaluation

I. Introduction

Le problème du stable de poids maximum multi-objectif est un problème classé NP-difficile. À ce moment, le problème n'a jamais été traité à notre connaissance, d'où, il n'existe pas une méthode dédiée à la résolution du problème, sauf les méthodes générales qui sont connues pour la résolution des programmes linéaires multi-objectif en nombres entiers, entre autres celles mentionnées dans le chapitre II, paragraphe II.

Rappelons que le modèle mathématique du stable de poids maximum multi-objectif s'écrit comme suit :

$$(MOISP)\begin{cases} w_j x = Z_j(\max) & ; \forall j = \overline{1,p} \\ A^t x \leq e & ; e = \{1, \dots, 1\}^t \\ x \in \{0,1\}^n & \end{cases}$$

w_j est le $j^{\text{ème}}$ vecteur de poids à n composantes et A est la matrice d'incidence du graphe $G = (V, E)$ donné.

Le problème a une structure assez particulière, la matrice des contraintes est creuse, chaque ligne contient exactement deux un (01), et le vecteur du second membre est égal à un (01) pour chacune de ses composantes.

II. Concepts de base

On définit un **stable efficace** toute solution efficace de $(MOISP)$. Le principe de la méthode réside dans l'exploitation de cette structure et l'utilisation du principe de la séparation et évaluation afin de réduire la taille du problème pour utiliser la méthode des ensembles efficaces complets sur chaque feuille de l'arborescence dont sa profondeur est fixée au préalable. Initialement, on supprime tous les sommets ayant leurs poids négatifs ou nuls avec au moins une composante strictement négative car ce type de sommets ne peut appartenir à aucun stable efficace.

Opération de tri

Initialement, on fait un tri sur l'ordre de traitement des sommets du graphe G. Pour cela, plusieurs façons sont envisageables et nous en proposons cinq dans les paragraphes suivants.

Séparation

Une fois que le tri des sommets est réalisé, la séparation est faite sur la base de la règle ''retenir'' ou ''ne pas retenir'' un sommet i dans les solutions ''stables efficaces'' ($x_i = 1$ou $x_i = 0$). En outre, le fait de retenir un sommet aura un effet de propagation de contraintes sur ses voisins dans ce sens que ; tous ses voisins seront automatiquement rejetés et beaucoup de contraintes du programme *MOISP* deviennent saturées, ce qui a pour premier effet bénéfique la diminution de la taille de *MOISP*. Suite à cet ''effet domino'', le deuxième effet bénéfique réside dans l'apparition possible de sommets isolés dans le sous graphe obtenu.

Le cas correspondant à ''ne pas retenir ''un sommet entraine sa suppression du graphe courant et dans ce cas aussi, le sous graphe obtenu peut admettre des sommets isolés.

La règle générale de traitement d'un sommet *i* isolé est la suivante :

- Si tous les poids w_i^j du sommet isolé *i* sont positifs ou nuls alors, $x_i = 1$,
- S'il existe un poids w_i^j négatif du sommet *i*, alors on aura deux cas à traiter : retenir le sommet i ($x_i = 1$) ou rejeter le sommet i ($x_i = 0$).

Evaluation

L'évaluation est faite en approximant le point idéal local par rapport au sous problème obtenu sur chaque nœud. Initialement, l'approximation $\overline{Z_J}$ du maximum de chaque critère $j$est faite par défaut sur le vecteur $X = (x_i)_{i=1..n}$ avec $x_i = 1$ si le poids $w_i^j \geq 0$ et $x_i = 0$ si le poids $w_i^j < 0$, c'est-à-dire $\overline{Z_J} = \sum_{i \in \{1,\dots,n\};\, w_i^j \geq 0} w_i^j$.

Lors de la séparation, si un sommet *i* est retenu dans une branche de l'arborescence, la mise à jour de $\overline{Z_J}$ est faite par l'ajout des poids négatifs associés au sommet i à $\overline{Z_J}$. Si le sommet *i* n'est pas retenu, la mise à jour de $\overline{Z_J}$ est faite par la soustraction des poids positifs associés au sommet i à $\overline{Z_J}$.

Principe général

Initialement, on supprime tous les sommets dont le vecteur poids est négatif ou nul, avec au moins une composante strictement négative, pour ensuite évaluer le premier nœud de l'arborescence. Le processus va se poursuivre par la séparation et l'évaluation de chaque nœud de l'arborescence, sachant qu'on a fixé au préalable le nombre de niveaux mesurant la profondeur de l'arborescence. L'effet domino a une incidence directe sur la réduction considérable de la taille du problème. Dans le pire des cas, pour réduire la matrice des contraintes à une matrice nulle, on aura 2^n feuilles dans l'arborescence quand la densité du graphe devient nulle (dans ce cas le graphe est un ensemble de sommets isolés). Donc, la fixation de la profondeur de l'arborescence a pour effet d'éviter l'exploration de 2^n feuilles. Ainsi, il suffit d'exécuter, sur chaque feuille correspondant à un sous graphe de *G* de densité non nulle, donc un programme de taille réduite par rapport au programme initial, une méthode connue dans la littérature pour la résolution des programmes linéaires multi-objectif en nombres entiers.

D'autre part, on a dit que la séparation a pour conséquence un effet domino sur la réduction de la matrice courante *A* des contraintes, correspondant à un sous-graphe *G*, cet effet est détaillé dans ce qui suit :

- Lors de la retenue d'un sommet *i*, sa colonne correspondante et toutes les lignes dans lesquelles le sommet *i* apparait, sont supprimées de la matrice *A*. De plus, tous ses sommets adjacents dans *G* seront automatiquement supprimés et donc, toutes les colonnes correspondantes sont aussi supprimées de *A*.
- Lors du rejet (suppression) d'un sommet, on va supprimer sa colonne correspondante dans la matrice *A*. De plus, chaque ligne qui le contient dans *A* ne comportera plus qu'un seul un correspondant à son sommet adjacent dans *G*.

En procédant ainsi, on peut avoir une ligne dans *A* avec un seul un, ce qui correspond à un sommet isolé *i* dans *G* obtenu après l'opération de réduction de la matrice *A*. Si tous ses poids sont positifs ou nuls avec au moins un poids strictement positif, alors on retient le sommet *i*.

III. Méthodes de tri des sommets

Les sommets doivent être triés initialement, et cela pour essayer d'accélérer le processus global d''exécution en exploitant la réduction de la taille du problème et la domination du point idéal.

Il existe plusieurs façons de trier les sommets, on va présenter cinq façons possibles :

III.1. Tri par adjacence descendant :

On va trier les sommets selon le cardinal de l'ensemble des voisins de chaque sommet du plus grand au plus petit. Cette façon permet de réduire la taille des problèmes rapidement. L'inconvénient est que très souvent les premiers stables trouvés sont d'une faible pondération ce qui n'exploite pas vraiment la domination du point idéal.

Exemple :

Soit le graphe *G* suivant :

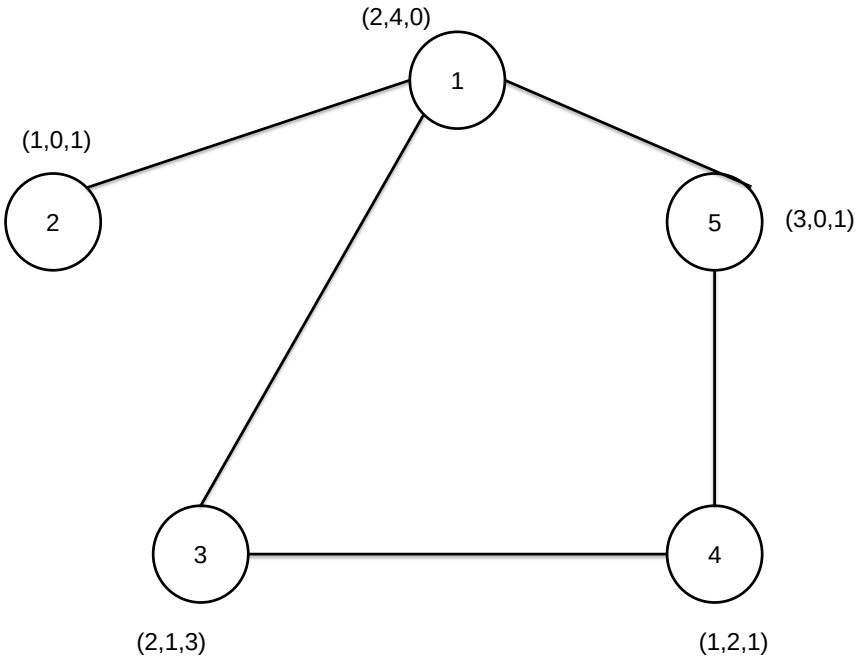


Figure 5 : Graphe de l'exemple d'exécution des méthodes du tri

La liste des adjacents est :

1 : 2, 3, 5
2 : 1
3 : 1, 4

4 : 3, 5
5 : 1, 4

Donc le tri sera comme suit : 1, 3, 4, 5, 2.

III.2. Tri par adjacence ascendant :

On va trier les sommets selon le cardinal de l'ensemble des voisins de chaque sommet du plus petit au plus grand. Cette façon permet de prendre les sommets qui ont le plus petit degré dans les stables puisqu'ils ont plus de probabilité d'y être que les autres sommets. En revanche, cette façon ne permet pas une réduction rapide de la taille du problème et on ne peut rien dire sur la domination des sommets.

Exemple :

Pour le même exemple, on a la même liste d'adjacence, donc le tri est : 2, 3, 4, 5, 1

III.3. Tri par domination des poids des sommets :

On va trier les sommets en les surclassant selon la notion de l'optimum de Pareto. Cette façon permet d'exploiter la notion de domination inter-sommets qui va aider à une utilisation bénéfique de la notion de domination du point idéal local. Par contre, on ne peut rien dire sur la rapidité de la réduction de la taille du problème.

Exemple :

Pour le même exemple précédent, la liste de domination des poids des sommets est :

1 : -
2 : -
3 : 2
4 : 2
5 : 2

Donc le tri est : 3, 4, 5, 1, 2

III.4. Tri par domination et adjacence :

On va trier les sommets en les surclassant selon la notion de l'optimum de Pareto calculés en sommant tous les poids et en retranchant pour chaque sommet les poids de ses voisins. Cette façon ne donne pas nécessairement un résultat identique au troisième tri présenté mais ils partagent les mêmes avantages et inconvénients.

Exemple :

Pour le même exemple précédent on a la liste des poids correspondant de chaque sommet avec l'ensemble des sommets qui les dominent :

1 : (3, 6, 1) domine : -
2 : (7, 3, 6) domine : 3, 5

3 : (6, 1, 5) domine : -
4 : (4, 6, 2) domine : 1
5 : (6, 1, 5) domine : -

Donc les meilleurs sommets sont ceux qui dominent le plus de sommets, d'où le tri : 2, 4, 1, 3, 5.

III.5. Tri combiné :

Afin d'essayer d'améliorer la réduction de la taille du problème et la domination du point idéal en même temps, on peut combiner les tris déjà exposés (le premier ou le deuxième avec le troisième ou le quatrième).

En prenant par exemple le premier et le quatrième tri, on va surclasser les sommets selon leur utilité. On la calcule en calculant initialement $\alpha = |\mathcal{N}(v)|$ pour chaque sommet v tel que $\mathcal{N}(v)$ est l'ensemble des voisins du sommet v, après $\beta = |H|$, tel que H est l'ensemble des sommets dominés par le sommet v dans le quatrième tri. Donc l'utilité de chaque sommet v est $u(v) = \alpha + \beta$.

Exemple :

Pour le même exemple précédent, on va combiner le premier et le quatrième tri en sommant la cardinalité de l'ensemble des voisins avec la cardinalité de l'ensemble des sommets dominés par chaque sommet, et on aura la liste :

1 : 3
2 : 3
3 : 2
4 : 3
5 : 2

Donc le tri est : 1, 2, 4, 3, 5.

Remarques :

- On peut aussi considérer le tri aléatoire, mais on ne peut rien dire sur son efficacité ou pas.
- On parle de l'efficacité des tris selon l'exploration de l'arborescence qu'on fera en fixant les sommets dans cet ordre, et cela est confirmé dans les résultats obtenus dans l'étude expérimentale dans le septième paragraphe du présent chapitre.

IV. Algorithme

Données

Soit $G = (V, E)$ un graphe valué avec V l'ensemble des sommets, E l'ensemble des arêtes, $C = \left(w_i^j\right)\ i = 1..n; j = 1..r$, une matrice réelle des poids sur les sommets de G avec les w_i^j pas tous nuls, et A sa matrice d'incidence.
N la profondeur de l'arborescence fixées.

Initialiser

$ESP \coloneqq \emptyset$; $SP \coloneqq \emptyset$; et $SD \coloneqq \emptyset$; (ensemble des ensembles des sommets retenus, ensemble des évaluations des ensembles des sommets retenus et ensemble des sommets isolés respectivement)

$Eff \coloneqq \emptyset$; et $SND \coloneqq \emptyset$; (ensemble des solutions efficaces et l'ensemble des solutions non-dominées respectivement).

Niveau actuel $niv \coloneqq 1$ et nœud actuel $l \coloneqq 1$;

$A_l \coloneqq A; C_l \coloneqq C; V_l \coloneqq V; T_l \coloneqq \emptyset$; ($T_l$ Liste des sommets initialement vide)

$Eff^l \coloneqq \emptyset; SND^l \coloneqq \emptyset$;

${}^I_J A_l$ est la matrice A_l avec I l'ensemble des indices des lignes et J l'ensemble des indices des colonnes.

***Etape* 1**

- Supprimer tous les sommets ayant tous leurs poids négatifs ou nuls avec au moins un poids strictement négatif.
- Supprimer de A_l et C_l les colonnes des sommets supprimés.
- Si A_l est vide alors Terminer : Tout sommet dont le vecteur poids est non dominé constitue un stable efficace.
  Sinon :
  - Calculer l'évaluation de la racine $e_l \coloneqq e_1 = \left(\sum_{\substack{i \in V_l \\ w_i^1 > 0}} w_i^1, \ldots, \sum_{\substack{i \in V_l \\ w_i^r > 0}} w_i^r \right)$.
  - Trier les sommets dans la liste T_l.
  - On retient le sommet $T_l[1]$.
  - ${}^I_J A_l \coloneqq {}^I_J A_l \backslash_{T_l[1]} {}^I_J A_l$; (supprimer la colonne du sommet $T_l[1]$)
  - ${}^I_J A_l \coloneqq {}^I_J A_l \backslash^{I(T_l[1])}_{\quad J} A_l$; (supprimer les lignes contenant le sommet $T_l[1]$).
  - Construire l'ensemble $\mathcal{N}(T_l[1])$; (ensemble des sommets voisins de $T_l[1]$)
  - Construire l'ensemble :
    $$K = \left\{ k \in I\left(\mathcal{N}(T_l[1])\right) \; et \; {}^k_J A_l = 1 \setminus {}^{I\{k\}}_{\quad J} A_l \; contient \; au \; moins \; deux \; uns \right\}$$
  - ${}^I_J A_l \coloneqq {}^I_J A_l \backslash {}^K_J A_l$;
  - ${}^I_J A_l \coloneqq {}^I_J A_l \backslash_{\mathcal{N}(T_l[1])} {}^I_J A_l$; (supprimer les colonnes des voisins du sommet $T_l[1]$)
  - ${}^I_J C_l \coloneqq {}^I_J C_l \backslash_{T_l[1]} {}^I_J C_l$; (supprimer la colonne du sommet $T_l[1]$)
  - ${}^I_J C_l \coloneqq {}^I_J C_l \backslash_{\mathcal{N}(T_l[1])} {}^I_J C_l$; (supprimer les colonnes des voisins du sommet $T_l[1]$)
  - Supprimer de la liste T_l son premier élément ainsi que ses voisins.
  - $l \coloneqq l + 1$; $niv \coloneqq niv + 1$;

  Fin de Si ;

***Etape* 2**

- Si A_l est vide alors aller à Etape 3
  Sinon :
  - Si $niv > N$ alors
    Si $e_l \prec SND_i$; $\forall i$ alors sonder le nœud l
    Sinon :

Utiliser la méthode EEC [5] sur A_l et C_l pour avoir Eff^l et SND^l;
$ESP \coloneqq \emptyset$; $EV \coloneqq \emptyset$;
Pour chaque stable k de Eff^l on a :
$e_k \coloneqq e_l$;
$SP' \coloneqq SP$;
$SP' \coloneqq SP' \cup Eff^l(k)$;
$e_k^i \coloneqq \begin{cases} e_k^i + w_i^j \text{ ; si } w_i^j < 0 \text{ avec } i \in Eff^l(k) \\ e_k^i - w_i^j \text{ ; sinon} \end{cases}$;
$ESP \coloneqq ESP \cup \{SP'\}$;
$EV \coloneqq EV \cup \{e_k\}$;
Fin Pour ;
Aller à Etape 3 ;
Fin Si ;

- Sinon :

$_J^I A_l \coloneqq {}_J^I A_l \backslash_{T_l[1]} {}_J^I A_l$; (effacer la colonne du sommet $T_l[1]$)
$_J^I A_l \coloneqq {}_J^I A_l \backslash^{I(T_l[1])} {}_J A_l$; (effacer les lignes correspondantes au sommet $T_l[1]$)
$_J^I A_l \coloneqq {}_J^I A_l \backslash {}_J^K A_l$; tel que :
$K = \left\{ k \in I\left(\mathcal{N}(T_l[1])\right) et\ {}_J^k A_l = 1 \setminus {}_J^{I\{k\}} A_l \ contient\ au\ moins\ deux\ 1 \right\}$
$_J^I A_l \coloneqq {}_J^I A_l \backslash_{\mathcal{N}(T_l[1])} {}_J^I A_l$; (effacer les colonnes des voisins du sommet $T_l[1]$)
$_J^I C_l \coloneqq {}_J^I C_l \backslash_{T_l[1]} {}_J^I C_l$; (effacer la colonne du sommet $T_l[1]$)
$_J^I C_l \coloneqq {}_J^I C_l \backslash_{\mathcal{N}(T_l[1])} {}_J^I C_l$; (effacer les colonnes du sommet $T_l[1]$)
Effacer les voisins du premier sommet de T_l.
Effacer le premier sommet de T_l.
$e_l \coloneqq e_l + w_{T_l[1]}$; $w_{T_l[1]} < 0$;
$e_l \coloneqq e_l - w_{\mathcal{N}(T_l[1])}$; $w_{\mathcal{N}(T_l[1])} > 0$;
$l \coloneqq l + 1$; $niv \coloneqq niv + 1$;
Aller à Etape 2 ;
Fin Si ;

Fin Si ;

***Etape* 3**

- Soit $\mathcal{P}(SD)_l$ l'ensemble des parties de SD constituant chacun un stable non dominé relativement à $\mathcal{P}(SD)_l$.
- Pour chaque ensemble k de EV on a :
  Pour chaque ensemble p de $\mathcal{P}(SD)_l$ on a :
  $e_v \coloneqq e_k + e_p$;
  S'il n'existe aucun élément qui domine e_v dans SND alors
  $SND \coloneqq SND \cup \{e_v\}$;
  $Eff \coloneqq Eff \cup \{ESP(k) \cup p\}$;
  S'il existe un élément t de SND dominé par e_v alors enlever t de SND et sa solution correspondante de Eff ;

Fin Si ;

Fin Pour ;

Fin Pour ;

- Le nœud l est sondé ;

Fin de l'algorithme.

V. Exemple

Soit le graphe G suivant :

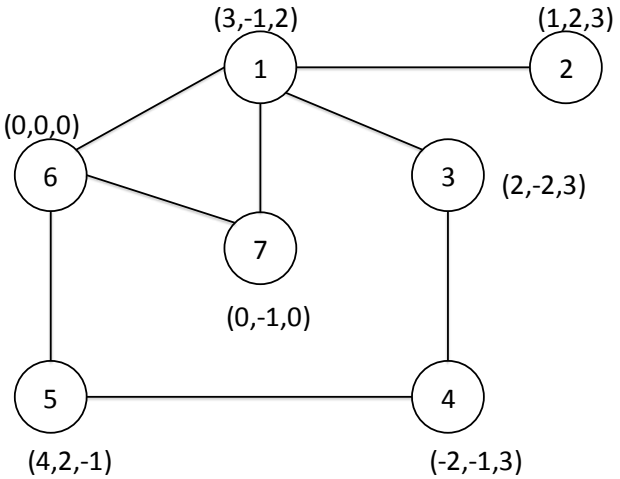


Figure 6 : Graphe de l'exemple d'exécution de la méthode exacte

- Initialement, on remarque que les poids du sommet « 7 » sont tous négatifs ou nuls avec le deuxième poids qui est strictement négatif alors on va éliminer ce sommet du problème.
- On fixe le tri des sommets : 1, 3, 4, 5, 6, 2.
- On fixe la profondeur de l'arborescence à 3.
- On commence l'arborescence par la création de la racine ou le *nœud 1* et en calculant son évaluation qui est égale à (10, 4, 11) et niveau = 0.
- Le nœud 1 est non-sondé et niveau < 3 alors on sépare en deux nœuds, le *nœud 2* prend le premier sommet de la liste trié (donc le sommet 1) et élimine ses adjacents et le *nœud 3* ne prend pas le sommet 1. On pose : niveau =1.
- *Traitement du nœud 2* : le sommet 1 est pris $SP \coloneqq \{1\}$et les sommets 2, 3 et 6 sont éliminés à cause de l'adjacence, l'évaluation du nœud est à (7, 1, 6). Le nœud n'est pas vide et niveau < 3 alors on sépare le nœud en deux nouveaux nœuds, le *nœud 4* en prenant le sommet 4 et le *nœud 5* en ne le prenant pas. On pose niveau = 2.
- *Traitement du nœud 4* : le sommet 4 est pris $SP \coloneqq SP \cup \{4\}$et le sommet 5 est éliminé à cause de l'adjacence, l'évaluation du nœud est à (1, -2, 5). Le nœud et vide alors $Eff \coloneqq \{SP\}$ et $SND \coloneqq \{(1,-2,5)\}$. On sonde le nœud 4 et on retourne au nœud 2 et niveau = 1 et $SP \coloneqq SP\backslash\{4\}$.
- *Traitement du nœud 5* : On pose niveau = 2. Le sommet 4 est éliminé et le sommet 5 devient un sommet isolé dans le graphe alors $SD \coloneqq \{5\}$ car il contient des poids négatifs. L'évaluation du nœud est à (7, 1, 5). Le nœud est vide alors on construit la partition de SD tel que $\mathcal{P}(SD) \coloneqq \{\emptyset, \{5\}\}$. Donc on teste tous les sous-ensembles de $\mathcal{P}(SD)$ union SP d'où : $Eff \coloneqq Eff \cup \{1\} \cup \{1,5\}$ et $SND \coloneqq SND \cup \{(3,-1,2)\} \cup \{(7,1,1)\}$. On sonde le

nœud 5 et on retourne au nœud 2 et niveau = 1 et $SD := SD \backslash \{5\}$. Le nœud 2 n'a pas d'autres branches alors on retourne au nœud 1 et niveau = 0 et $SP := SP \backslash \{1\}$.

- *Traitement du nœud 3* : On pose niveau = 1. Le sommet 1 est éliminé et le sommet 2 devient un sommet isolé dans le graphe alors $SP := \{2\}$ car tous les poids du sommet sont positifs ou nuls. L'évaluation du nœud est égale à (7, 4, 9). Le nœud n'est pas vide et niveau < 3 alors on sépare en deux nœuds, le *nœud 6* prend le sommet 3 et le *nœud 7* qui ne le prend pas. On pose niveau = 2.
- *Traitement du nœud 6* : le sommet 3 est pris $SP := \{2, 3\}$ et le sommet 4 est éliminé à cause de l'adjacence, l'évaluation du nœud est à (7, 2, 6). Le nœud n'est pas vide et niveau < 3 alors on sépare le nœud en deux nouveaux nœuds, le *nœud 8* en prenant le sommet 5 et le *nœud 9* en ne le prenant pas. On pose niveau = 3.
- *Traitement du nœud 8* : le sommet 5 est pris $SP := \{2, 3, 5\}$ et le sommet 6 est éliminé à cause de l'adjacence, l'évaluation du nœud est à (7, 2, 5). Le nœud est vide alors $Eff := \{2, 3, 5\}$ et $SND := \{(7, 2, 5)\}$. On sonde le nœud 8 et on retourne au nœud 6 et niveau = 2 et $SP := SP \backslash \{5\}$.
- *Traitement du nœud 9* : On pose niveau = 3. Le sommet 5 est éliminé et le sommet 6 devient un sommet isolé dans le graphe alors $SD := \{6\}$ car tous ses poids sont nuls. L'évaluation du nœud est à (3, 0, 6). Le nœud est vide alors on construit la partition de SD tel que $\mathcal{P}(SD) := \{\emptyset, \{6\}\}$. Donc on teste tous les sous-ensembles de $\mathcal{P}(SD)$ union SP d'où : $Eff := \{\{2, 3, 5\}, \{2, 3\}, \{2, 3, 6\}\}$ et $SND := \{(7, 2, 5), (3, 0, 6)\}$. On sonde le nœud 9 et on retourne au nœud 6 et niveau = 2 et $SD := SD \backslash \{6\}$. Le nœud 6 n'a pas d'autres branches alors on retourne au nœud 3 et niveau = 1 et $SP := SP \backslash \{3\}$.
- *Traitement du nœud 7* : On pose niveau = 2. Le sommet 3 est éliminé, l'évaluation du nœud est à (5, 4, 6). Le nœud n'est pas vide et niveau < 3 alors on sépare le nœud en deux nouveaux nœuds, le *nœud 10* en prenant le sommet 4 et le *nœud 11* en ne le prenant pas. On pose niveau = 3.
- *Traitement du nœud 10* : le sommet 4 est pris $SP := \{2, 4\}$et le sommet 5 est éliminé à cause de l'adjacence et le sommet 6 devient isolé, l'évaluation du nœud est à (-1, 1, 6). Le nœud est vide on construit la partition de SD tel que $\mathcal{P}(SD) := \{\emptyset, \{6\}\}$. Donc on teste tous les sous-ensembles de $\mathcal{P}(SD)$ union SP d'où :
  $Eff := \{\{2, 3, 5\}, \{2, 3\}, \{2, 3, 6\}, \{2, 4\}, \{2, 4, 6\}\}$et
  $SND := \{(7, 2, 5), (3, 0, 6), (-1, 1, 6)\}$. On sonde le nœud 10 et on retourne au nœud 7 et niveau = 2 et $SP := SP \backslash \{4\}$ et $SD := SD \backslash \{6\}$.
- *Traitement du nœud 11* : On pose niveau = 3. Le sommet 4 est éliminé. L'évaluation du nœud est à (5, 4, 3). Le nœud n'est pas vide et niveau = 3 alors on remarque qu'il n'y a aucune solution non-dominée dans SND qui domine l'évaluation du nœud alors on va utiliser la méthode des ensembles efficaces complets sur le sous-problème restant (qui est l'arête (5,6)). Alors la solution de la méthode est : $Eff_{11} := \{\{5\}, \{6\}\}$. Alors les solutions du nœud 11 sont l'union des sous-ensembles Eff_{EEC} avec SP d'où on a à tester {2}, $\{2, 5\}$ et $\{2, 6\}$. Donc $Eff := Eff \cup \{2, 5\}$ et $SND := SND \cup \{(5, 4, 2)\}$. On sonde le nœud 11.
- Tous les nœuds sont sondés alors terminer. L'ensemble des solutions efficaces est :

$$Eff := \{\{2, 3, 5\}, \{2, 3\}, \{2, 3, 6\}, \{2, 4\}, \{2, 4, 6\}, \{2, 5\}\}$$

Et l'ensemble des solutions non-dominées respectives est :

$$SND := \{(7,2,5),(3,0,6),(-1,1,6),(5,4,2)\}$$

Remarque : Les sommets sont choisis à partir de la liste triée qui est mise à jour à chaque itération.

VI. Justification de l'algorithme

***Lemme*1 :** *Si tous les vecteurs poids des sommets de G sont nuls alors, tous les stables de G sont efficaces.*

Preuve :

Le cône polaire $C^{\geq}$ est réduit à $\{0_{\mathbb{R}^n}\}$ et dans ce cas, toutes les solutions réalisables sont efficaces [26].

***Lemme*2 :** *Si tous les vecteurs poids des sommets de G sont négatifs ou nuls, avec au moins un poids strictement négatif pour chacun des vecteurs poids alors, tout sommet dont le vecteur poids est non dominé constitue un stable efficace.*

Preuve : évident.

***Lemme*3 :** *Considérant qu'il existe des sommets dont les vecteurs poids ne sont pas tous négatifs, alors chaque sommet dont le vecteur poids est négatif ou nul, avec au moins un poids strictement négatif, peut être supprimé du graphe, n'appartient à aucun stable efficace.*

Preuve :

Soit S_0 est un stable qui contient un sommet s tel que le vecteur colonne $C(s) \leq 0$ avec au moins $C_i(s) < 0$. En utilisant le programme mathématique *MOISP* (voir chapitre II) :

$$Z(S_0) = Z(S_0 \backslash \{s\}) + C(s)$$

$\Leftrightarrow Z(S_0) \leq Z(S_0 \backslash \{s\})$ car $C(s) \leq 0$ avec $Z_i(S_0) < Z_i(S_0 \backslash \{s\})$ car $C_i(s) < 0$.

D'où : $Z(S_0 \backslash \{s\})$ domine $Z(S_0)$.

Donc S_0 n'est pas un stable efficace.

***Lemme* 4 :** *Considérant qu'il existe des sommets dont les vecteurs poids ne sont pas tous négatifs, alors pour n'importe quel critère j, les sommets d'un graphe Gconstituant un stable de poids maximum Z_j^*, sont tous de poids positifs ou nuls.*

Preuve :

Supposons que S soit un stable de poids maximum avec $s \in S$ un sommet de poids w_s^j<0.
Alors $S\backslash\{s\}$est aussiun stable de poids : $Z_j(S\backslash\{s\}) = Z_j^* - w_s^j > Z_j^*$.
Contradiction avec S stable de poids maximum.

***Théorème* 1 :** *L'évaluation de chaque nœud l de l'arborescence est une approximation par excès du point idéal local du problème MOISP à cette étape.*

Preuve :

Soit :

Γ l'unique chemin de la racine au nœud *l* de l'arborescence,

E l'ensemble des sommets i de G pour lesquels $x_i = 1$ suivant Γ.

Soit e_l l'évaluation du nœud *l* de l'arborescence tel que :

$$e_1 = \left(\sum_{\substack{i \in V_1 \\ w_i^1 > 0}} w_i^1 , \dots , \sum_{\substack{i \in V_1 \\ w_i^r > 0}} w_i^r \right)$$

et

$$e_l = \left(\sum_{i \in V_l \cup E / w_i^1 > 0} w_i^1 , \dots , \sum_{i \in V_l \cup E / w_i^r > 0} w_i^r \right)$$

Soit le point idéal I_l du programme *MOISP* au nœud *l*tel que :

$$I_l = \left(\sum_{i \in S_l^1} w_i^1 , \dots , \sum_{i \in S_l^r} w_i^r \right)$$

tel que S_l^j est le stable de poids maximum par rapport au critère $j, j \in \{1..r\}$, du programme *MOISP* au nœud *l*.

Comme on a :

$$\begin{cases} E \subset S_l^j \\ et \\ S_l^j \subset V_l \cup E \end{cases}$$

alors, pour tout critère $j, j \in \{1..r\}$, on a la relation suivante entre e_l et I_l:

$$\sum_{i \in (V_l \cup E) \, / w_i^j > 0} w_i^j = \sum_{k \in S_l^j} w_k^j + \sum_{h \in \left(V_l \setminus S_l^j\right) / w_i^j > 0} w_h^j$$

$$\Leftrightarrow \sum_{i \in (V_l \cup E) \, / w_i^j > 0} w_i^j \geq \sum_{k \in S_l^j} w_k^j$$

Donc, e_l *domine* I_l.

***Théorème* 2 :** *L'algorithme génère toutes les solutions efficaces et converge en un nombre fini d'itérations.*

Preuve :

L'ensemble des solutions réalisables D étant un ensemble compact, il contient un nombre fini de solutions entières. Le processus de séparation engagé en premier lieu dans l'algorithme a pour effet de se ramener, au niveau des feuilles de l'arborescence, à des programmes *MOISP* de taille réduite. D'autre part, le lemme 4 nous assure qu'aucune solution efficace n'est supprimée lors du test de dominance des solutions réalisables générées avec l'approximation du point idéal.
De plus, l'appel à la méthode des EEC au niveau de chaque feuille de l'arborescence, nous assure de l'obtention de toutes les solutions efficaces localement [3]. La complétion de ses solutions efficaces locales par les sommets déjà retenus dans l'unique chemin issu de la source jusqu'à une feuille de l'arborescence permet d'obtenir des stables potentiellement efficaces pour le programme initial *MOISP*. Donc, aucune solution efficace n'est perdue pendant le processus.

VII. Expérimentation numérique

La méthode des ensembles efficaces complets (EEC) et la méthode par séparation et évaluation dédiée au problème du stable multi-objectif (BBMOISP) ont été mises en œuvre sous le langage de programmation MATLAB, en utilisant un PC DualCore, processeur 1.80 GHz, 1 Go de RAM. Il a été testé sur m contraintes (c'est-à-dire m arête) générées aléatoirement, $m \in \{100, 120, 150, 250, 350, 500, 1000, 1200\}$ et k fonctions objectifs, $k \in \{3, 7\}$. Les coefficients sont à valeurs bivalentes $\{0,1\}$ tel que chaque contrainte contient exactement deux un, et pour les fonctions objectifs, on a des coefficients entiers non corrélés répartis uniformément dans l'intervalle $[0, 9]$. Pour chaque contrainte la valeur du second membre est fixée à un. Des problèmes avec n variables (c'est-à-dire n sommets), $n \in \{20, 25, 30, 35, 40, 50, 60\}$, sont considérés. Pour chaque instance (n, m, k) une séquence de vingt jeux de données sont résolus et l'ensemble des solutions efficaces a été généré pour chaqu'un. La colonne « MOY » indique le nombre moyen des stables efficaces détectés par rapport aux jeux de données générés pour chaque instance et la colonne « MAX » le nombre maximal de stables efficaces détectés. Sous les colonnes « EEC » et « BBMOISP », les colonnes « MIN », « MAX » et « MOY » concernent le CPU de chaque méthode.

Instances	MOY	MAX	EEC			BBMOISP		
			MIN	MAX	MOY	MIN	MAX	MOY
(20,100,3)	6,9	11	2,859	6,899	5,037	0,292	1,457	0,651
(25,100,3)	6,8	15	11,532	78,743	47,788	0,716	11,626	4,09
(30,120,3)	8,5	22	163,561	829,919	453,369	1,916	101,503	38,873
(20,150,3)	6,6	11	3,822	6,969	5,298	0,333	0,721	0,58
(25,150,3)	9,6	15	22,869	65,187	42,262	1,28	7,388	2,864
(25,250,3)	7	12	29,729	62,451	43,992	1,372	2,105	1,74

(30,120,7)	64,8	100	523,903	1076,817	842,913	11,042	198,689	74,306
(35,150,3)	13,3	23	1188,9	4140,509	2196,917	69,491	795,678	273,002
(35,350,3)	9,3	15	772,239	1451,653	978,521	21,296	57,268	30,591
(35,500,3)	8,6	16	755,14	1143,351	869,434	10,352	20,718	14,941
(40,150,3)	22,7	38	-	-	-	8,81	2533,444	796,677
(40,350,3)	13,8	26	-	-	-	39,314	120,71	59,34
(50,250,3)	27,3	39	-	-	-	55,875	2618,37	989,122
(50,650,3)	18,5	28	-	-	-	36,029	65,816	47,192
(50,1000,3)	12,1	20	-	-	-	18,251	32,914	24,221
(60,1200,3)	17,6	31	-	-	-	72,128	151,754	100,893

Tableau 5 : Résultats généraux des tests de la méthodes exactes

La première chose qu'on teste ici est le temps d'exécution (CPU), on remarque que le CPU de la méthode BBMOISP est nettement meilleur quel que soit l'instance prise. Cela dit, la méthode BBMOISP offre une bonne contribution de la méthode EEC. En plus, dans les cases vides de la méthode EEC le temps d'exécution a dépassé 2 heures, ce qu'on a jugé non intéressant de terminer ces instances-là.

Densité	CPU(s) EEC	CPU(s) BBMOISP
d≤1/3	3588,197	362,678
1/3<d<2/3	3085,164	28.127
d≥2/3	3063,7448	28.026

Tableau 6 : Résultats généraux moyens des tests par rapport à la densité des graphes

Maintenant, en voyant les résultats par rapport à la densité des graphes, on trouve que ceux avec une faible densité sont plus lents à trouver par rapport aux autres densités, car si la densité est faible le nombre de stable d'une façon générale augmente. On remarque aussi que la méthode EEC n'a pas une différence significative entre la moyenne et forte densité, par contre la méthode BBMOISP trouve les stables efficaces de forte densité plus rapidement que ceux de moyenne densité malgré que la différence soit très petite.

On conclue que, ce qui concerne la comparaison entre la méthode EEC et la méthode BBMOISP, la seconde est nettement meilleure que la première et a donnée des améliorations en temps d'exécution clairement importante. Maintenant, on va se concentrer sur les différents paramètres de la méthode BBMOISP, et essayer de fixer les meilleures valeurs pour elles concernant la méthode du tri et le nombre de niveau fixé à préalable.

Niveau	CPU(s)
3	515,211
6	367,254
10	131,376

Tableau 7 : Temps d'exécution des instances par rapport au nombre de niveau fixé

On remarque que le temps d'exécution s'améliore à chaque fois qu'on augmente le nombre de niveau, car l'augmentation du nombre de niveau va permettre de fixer plus de sommet avant le dernier nœud ce qui permet la réduction de la taille du sous-problème qui va être en entré de la méthode EEC, ce qui signifie une exécution rapide par rapport aux autres niveaux inférieurs, de cette façon, on aura beaucoup de nœud qui seront sondés avant le dernier niveau à cause de la réduction de la taille du problème à l'ensemble vide. Il faut préciser que si on augmente beaucoup le nombre de niveau pour un problème de grande de dimension, généralement les sous-problèmes peuvent ne pas être réduits à l'ensemble vide, ce qui peut générer dans le pire des cas 2^{niveau} sous-problèmes à traiter avec la méthode EEC ce qui est très gênent. Avec les expériences numériques, on a remarqué que la borne $\left\lceil \frac{n}{2} \right\rceil$ donne de bons résultats, et c'est ce qui est appliqué à partir de la $11^{ème}$ catégorie des instances.

La méthode BBMOISP utilise une méthode de tri initialement, et comme on a vu dans le deuxième paragraphe du présent chapitre, il y en a plusieurs façon pour trier ces sommets, alors on a choisi les trois première méthodes cités dans le même paragraphe afin de tester l'efficacité de la méthode en fonction de la méthode choisie.

Instances	Tri par domination des poids			Tri par adjacence croissant			Tri par adjacence décroissant		
	MIN	MAX	MOY	MIN	MAX	MOY	MIN	MAX	MOY
(20,100,3)	0,388	1,457	0,619	0,656	1,158	0,866	0,292	0,793	0,469
(25,100,3)	1,135	6,98	3,384	4,112	11,626	7,538	0,712	2,152	1,348
(30,120,3)	11,936	58,829	32,074	43,619	101,503	71,389	1,916	24,94	13,156
(20,150,3)	0,494	0,721	0,594	0,561	0,714	0,641	0,333	0,657	0,506
(25,150,3)	1,316	7,388	3,261	2,557	6,175	3,347	1,28	2,769	1,985
(25,250,3)	1,426	2,063	1,699	1,6	2,1	1,875	1,376	1,875	1,649
(30,120,7)	43,081	92,771	63,22	92,111	198,686	131,794	11,045	58,208	27,903
(35,150,3)	96,881	421,45	232,413	210,511	795,678	472,035	69,491	277,215	114,55
(35,350,3)	29,238	37,841	33,063	21,601	33,947	22,601	21,296	57,268	31,17
(35,500,3)	12,787	19,233	15,821	12,921	15,252	14,031	10,352	20,712	14,97
(40,150,3)	61,471	1430,5	606,5	784,2	2533,4	1675,7	8,81	307,906	107,7
(40,350,3)	45,555	95,382	66,195	39,314	72,762	53,201	40,381	120,71	58,621
(50,250,3)	313,3	1378,9	705,07	1467,8	2618,3	1982	55,87	999,4	280,2
(50,650,3)	36,02	65,81	48,98	38,69	54,07	44,7	40,255	55,59	47,88
(50,1000,3)	19,013	32,914	24,81	18,25	25,853	22,27	21,555	31,837	25,573
(60,1200,3)	86,926	125,98	103,35	72,44	110,75	86,62	72,128	151,75	113,7

Tableau 8 : CPU de la méthode BBMOISP relatif à chaque tri

Le temps d'exécution favorise le tri par adjacence décroissant, globalement, et cela est lié aux différents paramètres qu'on va examiner dans ce qui suit.

VII.1. Comparaison de la réduction de la taille

Instances	Tri par domination des poids			Tri par adjacence croissant			Tri par adjacence décroissant		
	MIN	MAX	MOY	MIN	MAX	MOY	MIN	MAX	MOY
(20,100,3)	0	23	6,678	0	19	5,2	0	21	8,33
(25,100,3)	0	25	5,958	0	19	3,833	0	28	9,133
(30,120,3)	0	33	6,095	0	22	3,5	0	50	10,8
(20,150,3)	2	15	6	0	18	5,766	0	18	7,1
(25,150,3)	2	25	7,7	0	30	6,866	0	28	9,766
(25,250,3)	0	17	6,2	0	25	5,566	1	19	7,4
(30,120,7)	0	62	17,5	0	50	8,8	0	124	24,166
(35,150,3)	0	33	6,233	0	9	11,633	0	47	11633
(35,350,3)	0	26	5,966	0	25	5,6	0	29	9,4
(35,500,3)	0	18	5,8	0	39	8	0	20	7,2
(40,150,3)	19	88	46,818	17	86	42,909	37	115	79,272
(40,350,3)	14	64	35,272	2	62	35	19	49	35,454
(50,250,3)	56	97	78,3	46	171	85,2	70	225	129,1
(50,650,3)	21	57	36,8	31	69	45,1	33	84	58,4
(50,1000,3)	10	31	21,3	20	39	31,2	16	55	35,9
(60,1200,3)	18	70	38,6	13	63	38	34	84	60,5

Tableau 9 : Résultats généraux de la réduction de la taille par rapport à la méthode du tri choisi

On remarque dans ce tableau que la réduction de la taille du problème à un ensemble vide dans la méthode BBMOISP met le tri par adjacence décroissant comme le meilleur tri, car il donne des réductions plus que les autres tris sachant que la réduction à l'ensemble vide permet de ne pas utiliser la méthode EEC, en plus, elle peut sonder le nœud avant le dernier niveau ce qui raccourcit l'arborescence. En outre, après la ligne doublée, on a les niveaux fixés à $\left\lceil \frac{n}{2} \right\rceil$ qui donnent des réductions bien meilleures que les premières instances dont le nombre de niveaux est fixé à 3, 6 ou 10. Pour mieux visualiser ça, on présente les détails dans le graphe suivant :

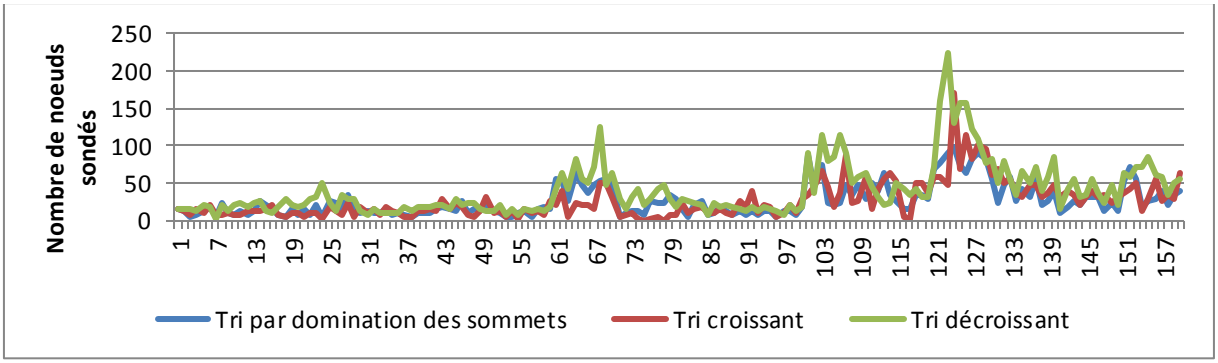


Figure 7 : Réduction de la taille du problème

VII.2. Comparaison de la domination du point idéal

Instances	Tri par domination des poids			Tri par adjacence croissant			Tri par adjacence décroissant		
	MIN	MAX	MOY	MIN	MAX	MOY	MIN	MAX	MOY
(20,100,3)	0	37	9,233	0	49	12,033	0	10	2,366
(25,100,3)	0	111	31,123	0	318	69,3	0	38	10,3
(30,120,3)	0	445	88,15	0	636	128,7	0	140	30
(20,150,3)	0	12	3,933	0	20	5,5	0	3	1,233
(25,150,3)	0	58	14,233	0	127	26,633	0	21	6,2
(25,250,3)	0	17	5	0	19	6,566	0	7	2,3
(30,120,7)	0	191	30,566	0	186	44,533	0	65	13,4
(35,150,3)	0	442	107,8	0	441	62,966	0	231	42,966
(35,350,3)	0	76	21	0	142	33,366	0	23	7,233
(35,500,3)	0	47	8,8	0	61	12,066	0	12	3,133
(40,150,3)	1489	8045	4295,909	9595	16280	13581	620	3838	1244,363
(40,350,3)	216	431	321,727	371	1016	641,727	55	138	104,818
(50,250,3)	10141	71724	33126,4	90378	169671	128790,6	3241	9434	5874,6
(50,650,3)	229	465	332,4	240	450	319,7	129	259	185,7
(50,1000,3)	36	70	49,4	21	49	37,9	17	42	28,2
(60,1200,3)	121	315	205,9	84	171	146,3	80	164	118,5

Tableau 10 : Résultats généraux de la domination du point idéal par rapport à la méthode du tri choisi

On remarque dans ce tableau que la domination du point idéal des sous- problèmes dans le dernier niveau dans la méthode BBMOISP met le tri par adjacence croissant comme le meilleur tri, car il donne des réductions plus que les autres tris, malgré que le tri par domination des poids lui aussi donne des résultats similaires sachant que la domination du point idéal permet de ne pas utiliser la méthode EEC. De plus, elle sonde le nœud dans le dernier niveau ce qui raccourcit l'arborescence. Pour mieux visualiser les résultats, on présente les détails dans le graphe suivant :

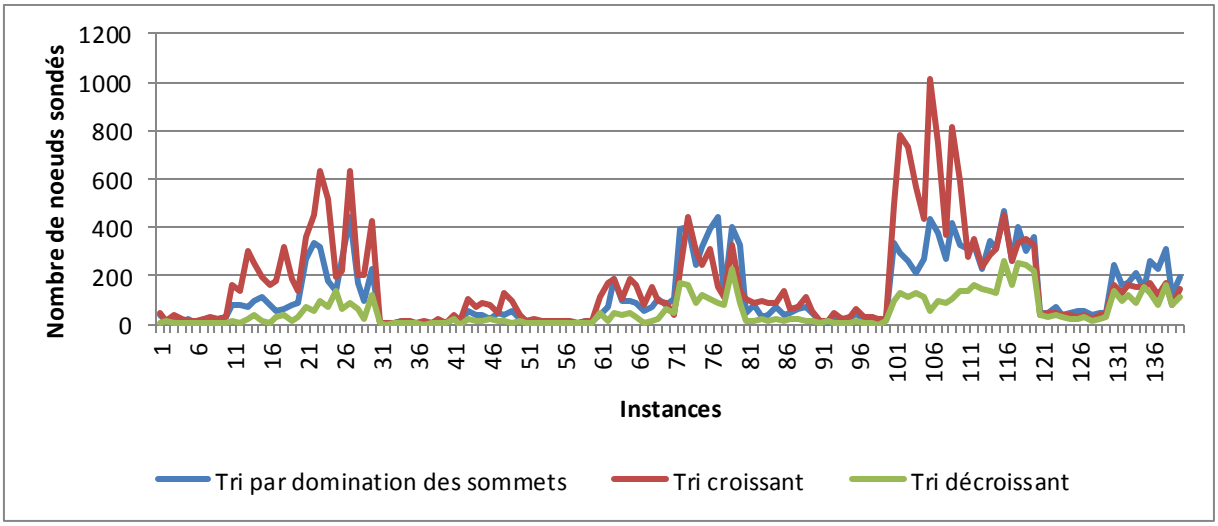


Figure 8 : Domination par le point idéal

Remarque : Pour la clarté du graphe, on a omis les résultats de grandes grandeurs qui sont à la faveur du 2ème tri en rouge.

VII.3. Le gain dans la réduction de la taille

On a dit que la réduction de la taille à un ensemble vide permet d'éviter l'utilisation de la méthode EEC, cela si le nœud est sondé au dernier niveau, par contre si le nœud est sondé dans l'avant dernier niveau alors on va éviter d'utiliser deux fois la méthode EEC, et si le nœud est sondé avant-avant le dernier niveau alors on va éviter d'utiliser quatre fois la méthode EEC et ainsi de suite. Donc, le nombre de nœud (gagné) au total est : $2^{\mathcal{N}-nv}$ tel que $\mathcal{N}$ est le nombre de niveau fixé et nv est le nombre du niveau actuel dont on a sondé le nœud. Maintenant on va voir trois graphiques concernant le gain total de la réduction de la taille par rapport à chaque méthode de tri :

Remarque : Pour la clarté du graphe, on a omis les résultats de grandes grandeurs qui sont à la faveur du 3ème tri dans le deuxième et le troisième graphe.

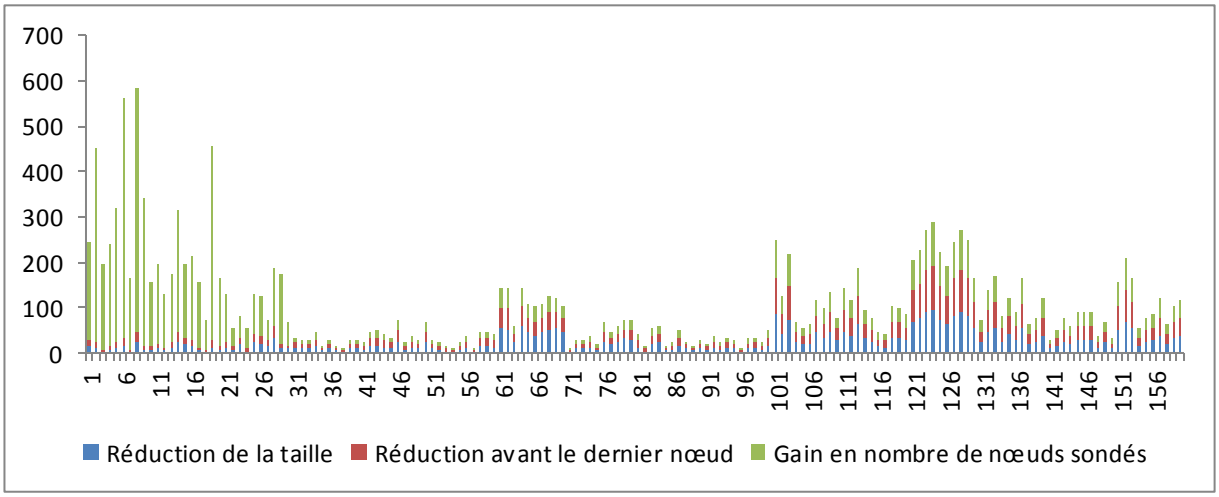


Figure 9 : Gain des sommets réduits par rapport à la méthode du tri de la domination

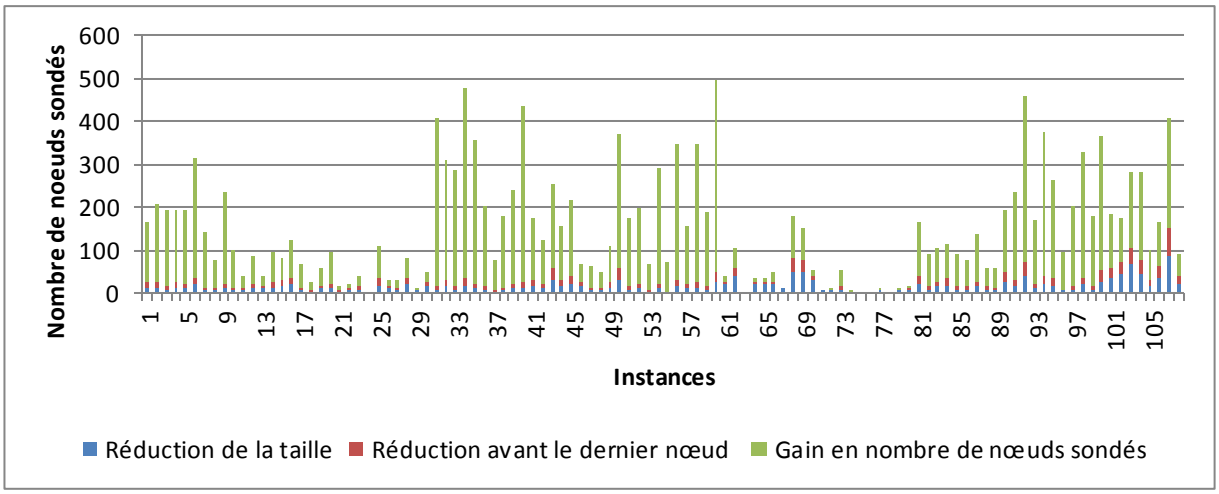


Figure 10 : Gain des sommets réduits par rapport à la méthode du tri par adjacence croissant

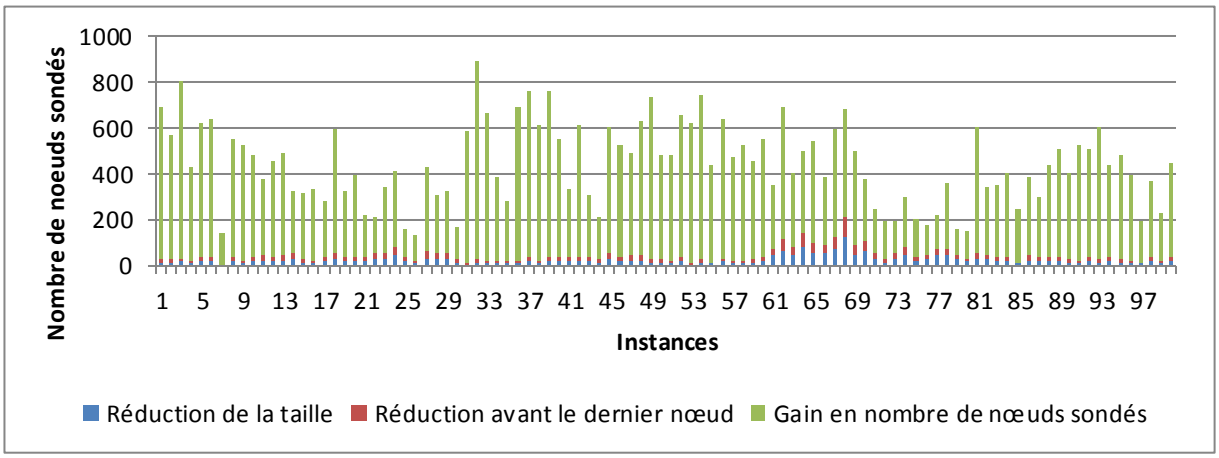


Figure 11 : Gain des sommets réduits par rapport à la méthode du tri par adjacence décroissant

On remarque les battons en vert qui représente le nombre de nœuds gagnés par rapport à chaque méthode de tri. Le gain dans le premier et le second tri ne vaut rien devant le troisième tri ce qui explique le temps d'exécution de la méthode en utilisant ce tri-là.

VII.4. Le gain dans la réduction de la taille vs la réduction par la domination du point idéal

On va comparer dans ce paragraphe les deux réductions précédentes par rapport à chaque tri.

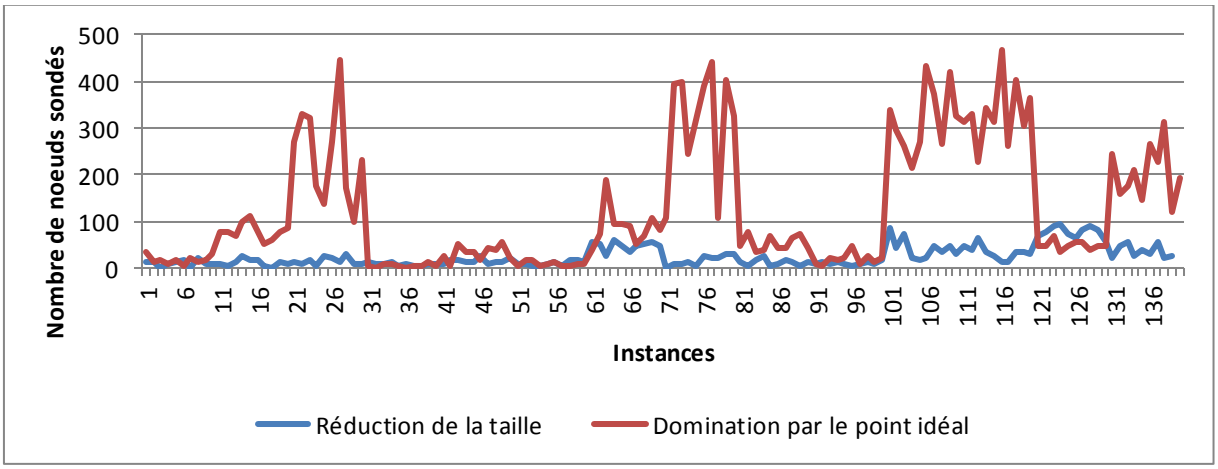


Figure 12 : Comparaison de la réduction du sommet par rapport au point idéal selon le premier tri

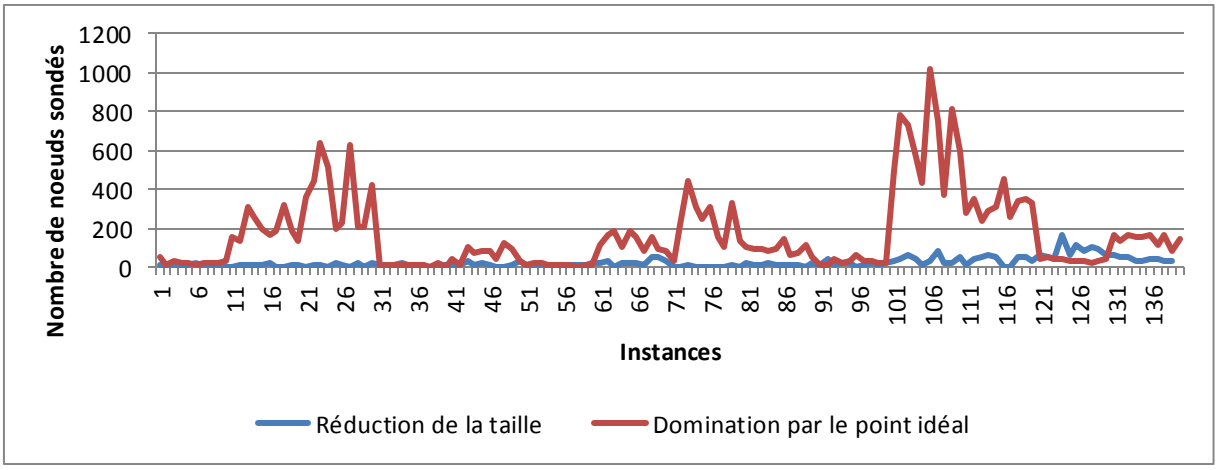


Figure 43 : Comparaison de la réduction du sommet par rapport au point idéal selon le deuxième tri

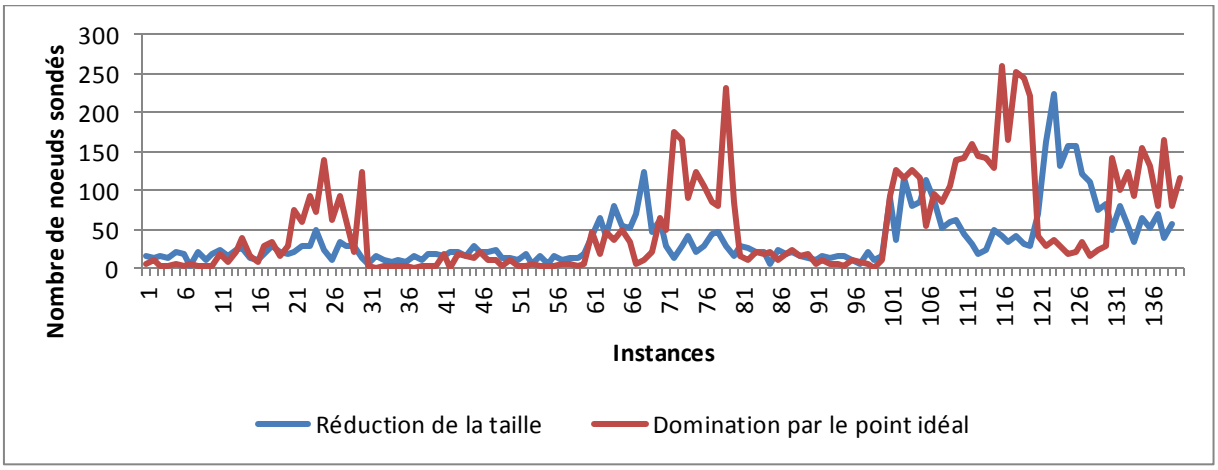


Figure 14 : Comparaison de la réduction du sommet par rapport au point idéal selon le troisième tri

On remarque que le premier tri fait des réductions du point idéal plus que la réduction de la taille, ce qui est au contraire du troisième tri, par contre, le deuxième tri tient un certain équilibre mais les piques ne sont pas très hausses par rapport au troisième tri. Le but de cette partie est d'essayer de trouver une idée pour développer un nouveau tri qui combine les propriétés des deux réductions en même temps espérant que cela va permettre de réduire encore plus le temps d'exécution de la méthode.

A première vue, on pense à combiner le troisième et le deuxième tri, mais cela ne marche pas car c'est le même tri, un décroisant et l'autre croissant. Alors on va combiner le premier et le troisième tri pour essayer d'améliorer les résultats obtenus puisque chacun présente de bons avantages dans l'une des réductions qui n'existe pas chez l'autre.

VII.5. Nombre de nœuds sondés

Dans ce paragraphe, on va présenter le nombre de nœuds total sondés par chaque tri.

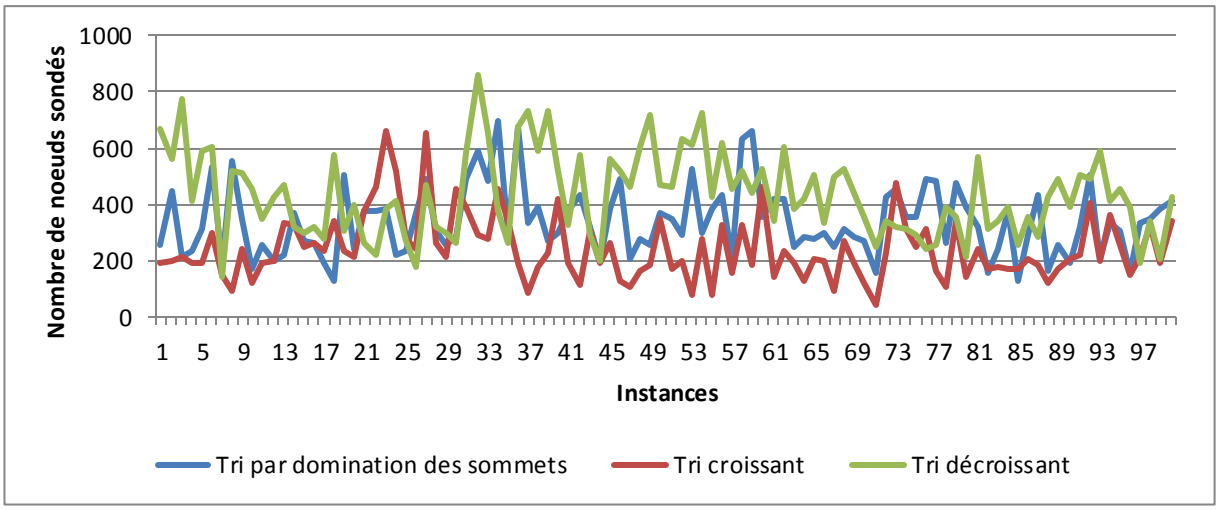


Figure 15 : Le nombre total sondé

On remarque que la majorité des instances favorisent le troisième tri qui permet de sonder plus de nœuds que les deux autres et c'est ce qu'on a déjà dit auparavant.

Chapitre V : Adaptation de la métaheuristique NSGA-II

I. Introduction

Le domaine des métaheuristiques a toujours été très réceptif aux propositions sur la façon de structurer les algorithmes afin de résoudre efficacement les problèmes d'optimisation. L'innovation des approches de solution a toujours été l'un des traits du champ, et les paradigmes de conception ont réussi comme source d'inspiration pour les concepteurs de l'algorithme: inspiration de la nature, l'amélioration de la recherche locale, de la logique et les probabilités, … etc.

Malgré les contributions apportées come vu dans le chapitre précédent, on a toujours besoin d'une réponse dans un temps raisonnable pour des instances de grandes tailles dont la méthode proposée ne fournit pas, alors on s'intéresse aux métaheuristiques et on se contente d'une solution approchée.

I.1. Heuristiques

Une heuristique est une méthode pour la résolution d'un problème défini. La solution obtenue est une solution réalisable mais pas nécessairement optimale. Il existe des idées de base qu'on peut utiliser pour construire une heuristique pour un problème donné, par exemple :

- **Heuristiques gloutonnes [21]:**
  Les algorithmes gloutons sont des algorithmes qui construisent des solutions réalisables en se basant sur le principe des algorithmes sans retour arrière, c'est-à-dire, qu'à chaque étape, l'algorithme fait un choix optimal localement et continue de considérer uniquement ce sous-problème.
- **Heuristiques par exploration locale [21]:**
  Le principe de ces algorithmes est de fixer une solution de départ s et de définir une fonction de voisinage $V(.)$ et on cherche à chaque itération une solution voisine de s ($V(s)$) qui est meilleure que s (en terme d'objectif). Si elle existe alors on la remplace dans s.
  On peut tester l'efficacité de chaque heuristique par différents moyens, tels que :
- Critère du plus mauvais cas
- Borne générale pour les algorithmes gloutons
- Schémas d'approximation
- Analyse probabiliste

I.2. Métaheuristiques

Une métaheuristique est une heuristique qui peut s'adapter à plusieurs problèmes différents. On peut citer quelques métaheuristiques telles que :

- Méthode de voisinage
- Recuit simulé
- Recherche tabou
- Colonie de fourmi
- Algorithmes évolutionnaires

On va exposer quelques métaheuristiques brièvement :

- **Recuit simulé [21]**

La méthode de recuit simulé s'inspire du processus de recuit physique. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide. En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température.

Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût.

A chaque nouvelle itération, un voisin $S' \in V(S)(V(S)$ est l'ensemble des voisins de S) de la configuration courante Sest généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté.

Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, *i.e.*, $f(s') \leq f(s)$, il est systématiquement retenu. Dans le cas contraire, S'est accepté avec une probabilité $\theta(\Delta E, T)$qui dépend de deux facteurs : d'une part l'importance de la dégradation $\Delta f = f(S') - f(S)$(les dégradations plus faibles sont plus facilement acceptées), d'autre part un paramètre de contrôle T, la température (une température élevée correspond à une probabilité plus grande d'accepter des dégradations).

La condition d'arrêt de cette procédure est quand on n'améliore plus la fonction de coût après un nombre maximum d'itération fixé au préalable, ou quand la température T tend vers zéro.

- **Recherche tabou [21]**

Le principe de base de la recherche tabou est de poursuivre la recherche des solutions même lorsqu'un optimum local est rencontré en permettant des déplacements qui n'améliorent pas la solution ou en utilisant le principe de mémoire pour éviter les retours en arrière (mouvements cycliques).

On note les différents paramètres utilisés comme suit :

- S : la solution actuelle
- S' : la prochaine solution atteinte (meilleure solution voisine)
- $V(S)$: l'espace de solutions voisines à S. En général, $card(V(S)) \approx 1/2$ taille du problème
- m : mouvement de S à S'
- S^* : la meilleure solution rencontrée

Et donc, jusqu'à présent, nous avons:

• *Mouvement non améliorateur*: un mouvement qui nous sortirait d'un minimum local S^* en nous amenant à une solution voisine S' pireque l'actuelle.

La méthode taboue permet un mouvement non améliorateur, comme le permet le recuit simulé. La différence entre les deux méthodes est que la recherche tabou choisira le meilleur S'dans $V(S)$, l'ensemble des solutions voisines.

• *Mouvement tabou*: un mouvement non souhaitable, comme si on redescendait à un minimum local d'où on vient juste de s'échapper.

Le mouvement est considéré tabou pour un nombre prédéterminé d'itérations k. k représente l'index des itérations (l'itération actuelle).

• T : *liste des mouvements tabous*. Il peut exister plusieurs listes simultanément. Les éléments de la liste sont $t(S,m)$.

Une liste T avec trop d'éléments peut devenir très restrictive. Il a été observé que trop de contraintes (tabous) forcent le programme à visiter des solutions voisines peu alléchantes à la prochaine itération.

Une liste T contenant trop peu d'éléments peut s'avérer inutile et mener à des mouvements cycliques. On utilise souvent une liste de taille 7. Les éléments de la liste sont gérés suivant le principe FIFO : first in first out.

• $a(S,m)$: *critères d'aspiration*. Déterminent quand il est avantageux d'entreprendre m, malgré son statut tabou ; si une solution S' tabou (obtenue par un mouvement tabou) a pour ''coût'' $f(S')$ meilleur qu'une valeur à laquelle on aspire, alors S' perd son statut tabou et devient donc, une solution voisine normale de $V(S)$. Entre autres, si $f(S')$ est plus petite que $f(S^*)$ alors, $S^* := S'$ et S' n'a plus le statut tabou.

- **Algorithmes évolutionnaires [21] et [33]**

Beaucoup de grandes inventions sont les fruits des principes de la biologie ou bien l'évolution naturelle pour l'étude et la conception des systèmes humains. L'évolution naturelle des espèces peut être vue en tant qu'un processus d'apprentissage comme pour adapter des environnements et optimisation des formes des espèces. Alors on peut imiter le point de vue de la génétique moderne, c'est-à-dire, le principe « la survie au plus fort » dans l'optimisation des designs ou les algorithmes d'apprentissage. Ici, les algorithmes évolutionnaires entrent en jeu.

Les algorithmes évolutionnaires sont des algorithmes qui performent l'optimisation ou la tâche d'apprentissage avec une capacité d'évoluer. Ils ont trois principales caractéristiques :

a- Base de population : Les algorithmes évolutionnaires maintiennent un groupe de solutions appelé *population*, pour optimiser ou apprendre le problème d'une façon parallèle. La population est un principe de base du processus évolutionnaire.

b- **Orientation de fitness :** Toute solution dans la population est appelée *individu*. Tout individu a une représentation génétique appelée *code*, et sa performance d'évaluation appelée *la valeur fitness*. Les algorithmes évolutionnaires préfère les individus à la plus grande valeur fitness qui la fondation de l'optimisation et la convergence des algorithmes.

c- **Conduction des variations :** Les individus passent par un nombre d'opérations de variation pour imiter les changements naturelles qui sont fondamentales pour la recherche dans l'espace des solutions.

Les algorithmes génétiques, les stratégies d'évolution et les programmes d'évolution construisent le cœur des algorithmes évolutionnaires.

II. Algorithme génétique

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et des mécanismes d'évolution des espèces : croisements, mutations, sélections, etc... Ils appartiennent á la classe des algorithmes évolutionnaires [33].

Dans les années 60, John Holland étudie les systèmes évolutifs et, en 1975, il introduit le premier modèle formel des algorithmes génétiques (*the canonical genetic algorithm AGC*) dans son livre « *Adaptation in Natural and Artificial Systems* ». Ce modèle servira de base aux recherches ultérieures.

En 1989, David Goldberg publie un ouvrage de vulgarisation des algorithmes génétiques.

Dans les années 90, il y a eu la programmation d'une panoplie d'algorithmes génétiques transcris en C++, appelée GAlib. Cette librairie contient des outils pour des problèmes d'optimisation en utilisant les algorithmes génétiques. Elle est conçue pour servir de support de programmation.

On va présenter le procédé général qu'on trouve plus en détail dans [21] et [33].

II.1. Procédé général

Dans un algorithme génétique simple, on maintient plusieurs « *individus* » dans la « *population* ». Chaque individu a deux propriétés : Son emplacement ou encore la représentation de la solution réalisable appelé « *chromosome* » et sa qualité appelé « *fonction objectif ou fonction fitness* ». Après avoir la qualité de chaque individu, on commence le processus de sélection des individus pour générer le « *mating pool ou l'ensemble des parents* ». Les individus avec une grande qualité doivent avoir une grande probabilité pour être sélectionnés dans l'ensemble des parents, comme ça, les meilleurs individus restent et les mauvais sont rejetés. Généralement, deux parents sont choisis aléatoirement de l'ensemble des parents pour offrir deux « *offspring ou enfants* » sans remplacement (cette étape est appelé « *croisement ou* hybridation ») et chaque enfant doit passer par quelques changements pour devenir un nouvel individu (cette étape est appelée « *mutation* »). Donc, la nouvelle « *génération* » remplace l'ancienne génération et autre génération commence à être générée.

On détaille quelques points importants dans ce qui suit :

- Il faut toujours avoir une population au départ du procédé ou bien un ensemble de solutions réalisables de départ.
- Les chromosomes doivent avoir une certaine codification qui facilite le croisement ou encore la mutation. On peut avoir une codification binaire, en nombre réel ou bien une codification établie par l'utilisateur.
- La sélection des individus dans l'ensemble des parents nécessite d'avoir une probabilité de sélection pour chaque individu liée à sa fonction fitness. La sélection peut être effectuée par plusieurs méthodes : sélection par roulette, sélection par rang, sélection par tournoi …etc. Il y en a quelques sélections un peu particulière qui sert à garder les meilleurs individus d'une génération à une autre comme la sélection de steady-state qui conserve les meilleurs individus d'une ancienne génération et les insèrent dans la nouvelle génération, une autre méthode hors des sélections appelée « *élitisme* » qui conserve les meilleurs individus de la population avant l'hybridation et les meilleurs enfants obtenus de l'hybridation avant la mutation en tant qu'individus.
- Le croisement ou l'hybridation se déroule en choisissant un point de croisement aléatoirement dans les chromosomes des parents, le premier fils aura les mêmes informations que le premier parent jusqu'au point du croisement, le reste des informations du premier fils sont les même que le second parent à partir du point du croisement. Pour le second fils, la première partie est celle du second parent jusqu'au point du croisement et la dernière partie est celle du premier parent à partir du point de croisement. Le croisement peut avoir plusieurs points de croisement. Il existe aussi d'autres façons pour effectuer le croisement comme le choix de deux points de croisement par exemple.
- La mutation se fait en choisissant aléatoirement une position dans le chromosome de l'enfant et on change sa valeur. Il se peut avoir plusieurs points de mutation.

II.2. Algorithme

Etape 0 : Initialisation

(0) Allouer en entré les variables qu'on a besoin dans l'algorithme génétique tels que : *popsize* (taille de la population), *maxgen* (nombre maximum de génération générée), p_c (probabilité de croisement entre deux parents), p_m (probabilité de mutation) … etc.
(1) Générer *popsize* aléatoirement pour former la population initiale et calculer les fonctions fitness des individus. On met *gen*= 0.

Etape 1 : Boucle main. Répéter les instructions qui suivent jusqu'au *gen>maxgen*

(0) Sélectionner *popsize* individus de la population pour générer l'ensemble des parents.
(1) Répéter les opérations qui suivent jusqu'au l'obtention de la nouvelle population avec *popsize* individus.
Sélectionner deux individus de l'ensemble des parents sans remplacement pour effectuer un croisement avec une probabilité p_c et performer une mutation pour chaque

enfant obtenu du croisement avec une probabilité p_m. Après, insérer les mutants dans la population.

(2) Evaluer la fonction fitness pour chaque nouvel individu dans la population.

(3) Remplacer la population courante avec la nouvelle population. *gen=gen*+ 1.

Etape 2 : Soumettre les finaux *popsize* individus comme résultat de l'algorithme génétique.

II.3. Méthodes de sélection

Il existe plusieurs méthodes de sélection pour l'algorithme génétique dont les plus importants sont :

1. Sélection par roulette [33]

Les parents sont sélectionnés en fonction de leur performance. Meilleur est le résultat codé par un chromosome, plus grandes sont ses chances d'être sélectionné. Il faut imaginer une sorte de roulette russe sur laquelle sont placés tous les chromosomes de la population, la place accordée à chacun des chromosomes étant en relation avec sa valeur de la fonction fitness.

Ensuite, la bille est lancée et s'arrête sur un chromosome. Les meilleurs chromosomes peuvent ainsi être tirés plusieurs fois et les plus mauvais ne jamais être sélectionnés. Cela peut être simulé par l'algorithme suivant :

1. On calcule la somme S_1de toutes les fonctions d'évaluation d'une population.

2. On génère un nombre $r$entre 0 et S_1.

3. On calcule ensuite une somme S_2des évaluations en s'arrêtant dès que $r$est dépassé.

4. Le dernier chromosome dont la fonction d'évaluation vient d'être ajoutée est sélectionné.

Exemple

Maximiser $f(x) = 3x + 1$où $x$est un entier dans [0,10]**.**

Population : $3, 5, 1, 7$**.**

$S_1 = 10 + 16 + 4 + 22 = 52,$

$r = 27,$

$S_2 = 10 + 16 + 4 = 30 > 27,$

Donc, la solution sélectionnée est $x = 5$.

2. Sélection par rang [33]

La sélection précédente rencontre des problèmes lorsque la valeur d'adaptation des chromosomes varie énormément. Si la meilleure fonction d'évaluation d'un chromosome

représente 90% de la roulette alors les autres chromosomes auront très peu de chance d'être sélectionnés et on arriverait à une stagnation de l'évolution.

La sélection par rang trie d'abord la population par fitness. Ensuite, chaque chromosome se voit associé un rang en fonction de sa position. Ainsi le plus mauvais chromosome aura le rang 1, le suivant le rang 2, et ainsi de suite jusqu'au meilleur chromosome qui aura le rang m (pour une population de m chromosomes). La sélection par rang d'un chromosome est la même que par roulette, mais les proportions sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. Le tableau suivant fournit un exemple de sélection par rang. Avec cette méthode de sélection, tous les chromosomes ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution. Ceci est dû au fait que les meilleurs chromosomes ne diffèrent pas énormément des plus mauvais.

Exemple

Chromosomes	1	2	3	4	5	6	Total
Probabilité initiale	89 %	5 %	1 %	4 %	3 %	2 %	100 %
Rang	6	5	1	4	3	2	21
Probabilité finale	29 %	24 %	5 %	19 %	14 %	9 %	100 %

Tableau 11 : Exemple de sélection par rang pour 6 chromosomes

3. Sélection par tournoi [33]

La sélection par rang et la sélection par roulette considèrent des informations globales, c'est-à-dire, la valeur relative de fitness ou bien le rang. Parfois, on ne possède que des informations locales, c'est-à-dire, le meilleur dans un petit groupe. Ici, la sélection par tournoi est utile.

Pour implémenter la sélection par tournoi, on a juste besoin de piquer k individus aléatoirement (pour $k = 2$ on dit qu'on a un *tournoi binaire*) avec remplacement et comparer les valeurs de fitness de ces k individus d'où on aura le tournoi. Le meilleur gagne le tournoi et est sélectionné dans le « mating pool ». On répète le processus jusqu'au avoir $popsize$ individus sélectionné.

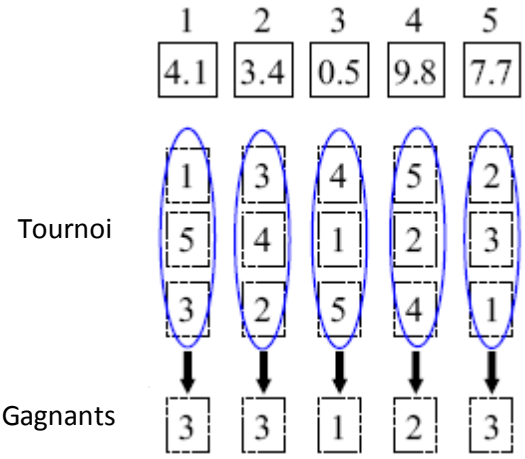
Cette sélection souffre d'une sélection biaisée, c'est-à-dire, si le meilleur individu n'a pas de chance, il ne sera pas sélectionné et vice versa.

Pour dépasser cette lacune de biais, Sokolov et Whitley ont proposé une sélection par tournoi sans biais. Pour un tournoi de taille k, tout individu sera comparé k fois dans une sélection par tournoi sans biais, ce qui assure que le meilleur individu aura k copies dans le « mating pool » et le mauvais ne sera pas sélectionné.

Exemple

Soit une population de 5 individus, leurs valeurs de fitness sont :{4.1,3.4,0.5,9.8,7.7}.

On choisit $k = 3$, alors on génère trois permutations dans l'intervalle [1,5]. On prend aléatoirement : (13452), (54123) et (32541).



III. Algorithmes génétiques pour les problèmes d'optimisation multi-objectif

Un bon algorithme génétique pour les problèmes d'optimisation multi-objectif doit satisfaire au-moins les trois suggestions suivantes [33] :

a. **Convergence :** Un algorithme génétique pour le problème d'optimisation multi-objectif doit avoir un mécanisme de convergence d'une façon à pouvoir atteindre la frontière de Pareto le plus tôt possible. Pour un seul objectif, la direction de la recherche est claire, mais en multi-objectif, un bon algorithme doit chercher l'espace des objectifs de différentes façons.
b. **Distribution :** Un algorithme génétique pour le problème d'optimisation multi-objectif doit essayer de distribuer les individus régulièrement que possible sur le long de la frontière de Pareto pour produire plus de solutions non-dominées. Pour un seul objectif, le processus de sélection la transmission génétique de l'information force la convergence de la population à une seule solution optimale. Alors il faut définir une façon pour maintenir la distribution au problème multi-objectif.
c. **Elitisme :** Un algorithme génétique pour le problème d'optimisation multi-objectif doit définir quels individus qui doivent être archivés ? et combien ? et est-ce qu'il y a un mécanisme d'insertion/remplacement[1] ?

Pour les problèmes d'optimisation combinatoire multi-objectif, la métaheuristique NSGA-II a connu une grande réputation, et afin de pouvoir bien la décrire, on définit [29]:

- **Rang de Pareto :**
  Goldberg a suggéré une méthode pour ranger la population, c'est la méthode de rang de Pareto. Tous les individus ont besoin d'être comparés avec les autres en utilisant le concept de dominance de Pareto pour déterminer les solutions non-dominées dans la population actuelle. Ces individus ont le rang 1. Après, on élimine tous les individus de rang 1 de la population, et on cherche à nouveau les solutions non-dominées dans la sous-population, et

[1] Insertion des individus dans la population sans/avec remplacement des individus de la génération précédente

on leurs donne le rang 2. On refait la procédure jusqu'à donner à tous les individus un rang.

- **Partage de fitness :**
  Le concept de partage de fitness est introduit par Goldberg et Richardson en 1987. Ils ont pensé aux hauteurs de pic dans une fonction. Les individus de la même espèce résident dans une fonction en partageant les ressources en diminuant leurs valeurs fitness selon la complexité de l'espèce. Un standard partage fitness est effectué avant le processus de sélection des parents.
- **Niche radius :**
  Une niche est une métaphore pour désigner l'ensemble partiel des solutions dont un seul pic qui réside (uni-modal). Niche radius ou bien le rayon d'une fonction désigne la région de la niche ou bien le périmètre ou se trouve.
- **Distance de la foule « Crowding method » :**
  Le standard de la foule est introduit la première fois par De Jong en 1975 en tant qu'une façon d'ajustement de la diversité de la population en calculant ce qu'on appelle la distance de la foule. A part le partage fitness, la distance de la foule a une autre perspective. On pense que l'ensemble des solutions est groupé avec les individus. Alors la nouvelle génération a besoin de compléter les anciens pour remplacer les plus mauvais anciens individus proches d'eux.
- **Croisement binaire simulé (SBX) :**
  Deb et Agrawal ont cherché sur les croisements dans un seul point pour les chromosomes codés en binaire et ils ont trouvé que les fils ont le même centre que les parents. Ils ont suggéré le croisement binaire simulé (SBX) pour simuler cette propriété dans un croisement pour des chromosomes avec des codes réels.

III.1. Non-dominated Sorting Genetic Algorithm-II (NSGA II)

Srinivas et Deb ont suggéré leur NSGA en 1994 [29], en combinant la méthode du rang de Pareto de Goldberg et le partage de fitness. Après le classement de Pareto, tout individu de même rang r prend la même valeur fitness f_r. De cette façon, les individus séparés de rang inférieur (ceux de bonne qualité pour la notion de dominance de Pareto) gagnent un avantage de sélection ce qui pousse NSGA à atteindre la frontière de Pareto avec une bonne distribution.

Deb, Pratap et Agarwal ont suggéré NSGA II en 2002 [7] pour améliorer son prédécesseur de trois façons :

- Améliorer la méthode de rang de Pareto en réduisant la complexité de l'algorithme.
- Ajouter le mécanisme d'élitisme.
- Eliminer le besoin du rayon d'une fonction (" niche radius ") et le remplacer par la distance de la foule (" crowding method ").

Pour une seule génération, le processus de solution dans NSGA II est comme suit :

Etape 0 : Assigner le rang de Pareto et la distance de la foule

(0) Combiner la population $P(t)(popsize)$ et l'archive[1] $A(t)(popsize)$ pour avoir $2 \times popsize$ individus.
(1) Assigner à chaque individu son rang de Pareto.
(2) Calculer la distance de la foule pour chaque individu.

Etape 1 : Générer le nouvel archive $A(t+1)$

(0) Insérer les individus dans $A(t+1)$. Les individus de rang 1 doivent être insérer premièrement, après ceux de rang 2 … Si les individus de rang $r$ne peuvent pas être insérer entièrement dans $A(t+1)$ alors insérer les individus dans un ordre décroissant selon leurs distance de la foule jusqu'au avoir l'archive $A(t+1)$ plein avec $popsize$ individus.

Etape 2 : Générer la nouvelle population $P(t+1)$

(0) Sélectionner de $A(t+1)$ en utilisant la sélection par tournoi binaire pour former le « mating pool ». Si deux individus dans $A(t+1)$ ont différent rang, celui avec le plus bas rang gagne le tournoi. Ou celui avec le même rang mais avec une distance de la foule plus grande gagne le tournoi.
(1) Générer la nouvelle population $P(t+1)$ par croisement et mutation du « mating pool ». Dans un NSGA II standard l'opérateur de croisement est SBX et l'opérateur de mutation est la mutation polynômiale.

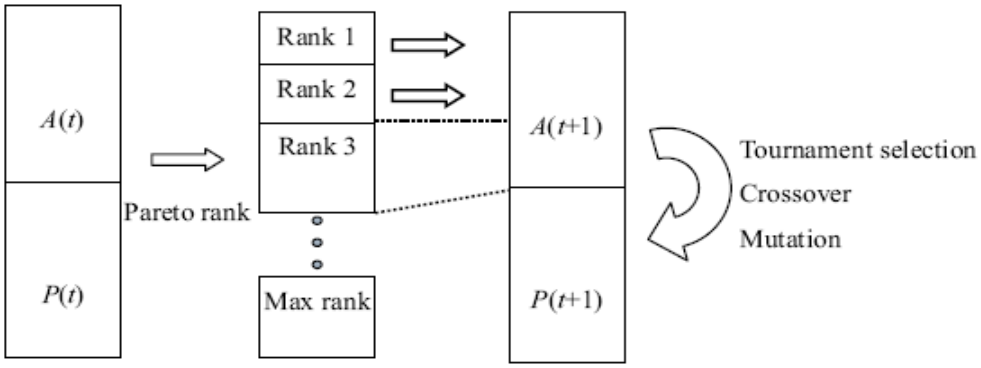


Figure 16 : Le processus d'évolution pour une génération de NSGA II [2]

On suppose qu'un individu j réside sur la frontière de Pareto et loin des autres. A partir du processus de la solution et la figure -16- au-dessus, on peut voir que l'individu j ne sera jamais perdu. Il restera toujours dans le rang 1 avec une large distance de la foule, ce qui signifie qu'il restera dans l'archive $A(t+1)$. Pour le croisement et la mutation, ils ne peuvent pas le détruire, puisque à la prochaine génération, $A(t+2)$ sera sélectionné sur la base de l'union de $A(t+1)$ et $P(t+1)$ et l'individu j est dans $A(t+1)$.

La convergence, la distribution et le mécanisme d'élitisme dans NSGA-II sont le rang de Pareto, sélection par tournoi, la distance de la foule, et l'introduction à l'archive A

[1] Ensemble des individus stocké dans la procédure d'élitisme, et sa taille est : popsize (La taille de la population)
[2] Cette figure est prise de :(Xinjie, et al., 2010) p.211 [33]

respectivement. A ce stade-là, NSGA-II établie une balance minutieuse dans les trois besoins et ne nécessite aucun paramètre additionnel en dehors de l'algorithme génétique simple.

IV. Résultats expérimentaux

Bien que la méthode NSGA-II a prouvé son efficacité dans le cas général pour les problèmes d'optimisation multi-objectif [7], on va tester dans ce paragraphe son efficacité dans notre cas. La méthode de NSGA-II a été mise en œuvre sous le langage de programmation C, en utilisant un PC DualCore, processeur 1.80 GHz, 1 Go de RAM. Il a été testé sur m contraintes (c'est-à-dire m arête) générées aléatoirement, $m \in \{100, 120, 150, 250, 350, 500, 650, 1000, 1200\}$ et $k = 3$ fonctions objectifs. Les coefficients sont des bivalents $\{0,1\}$ tel que chaque contrainte contient exactement deux uns, et pour les fonctions objectifs, on a des coefficients entiers non corrélés répartis uniformément dans l'intervalle $[0,9]$. Pour chaque contrainte la valeur du second membre est fixé à un. Des problèmes avec n variables (c'est-à-dire n sommets), $n \in \{20, 25, 30, 35, 40, 50, 60\}$, sont considérés. Pour chaque instance (n, m, k) une séquence de vingt problèmes est résolue et l'ensemble des solutions efficaces a été généré pour chaque problème. La taille de population est fixé à $popsize = 200$ pour 300 générations effectués. La probabilité de croisement $p_c = 0.9$ et la probabilité de mutation $p_m \in [0.05, 0.2]$ selon l'instance utilisée et en la changeant puisque chaque instance est testé au moins trois fois. La colonne « MOY » indique le nombre moyen des stables efficaces détectés pour l'instance et la colonne « MAX » le nombre maximal de stables efficaces détectés. Sous la colonne « NSGA-II », les colonnes « MAX » et « MOY » concernent les stables efficaces détectés ainsi que l'efficacité de la méthode en calculant le rapport des solutions efficaces trouvés par NSGA-II sur le nombre des stables efficaces qui existent.

Instances	**MOY**	**MAX**	**NSGA-II**		
			MOY	**MAX**	**Efficacité (%)**
(20,100,3)	6,9	11	6,9	11	100
(25,100,3)	6,8	15	6,5	15	94,39
(30,120,3)	8,5	22	7	18	88,02
(20,150,3)	6,6	11	6,6	11	100
(25,150,3)	9.6	15	9,2	15	96,52
(25,250,3)	7	12	6,8	12	97,50
(35,150,3)	13,3	23	12,4	21	94,53
(40,150,3)	22,7	38	21,8	35	92,54
(40,350,3)	13,8	26	13,5	26	90,12
(50,250,3)	27,3	39	26,4	39	92,55
(50,650,3)	18,5	28	17,5	25	96,58
(50,1000,3)	12,1	20	11,8	19	98,75
(60,1200,3)	17,6	31	15,8	31	93,25

Tableau 12 : Efficacité de NSGA-II

On remarque une efficacité convaincante de la méthode malgré que les instances testées ne reflètent pas vraiment cette déclaration car ce n'est pas une mesure d'efficacité bien argumentée pour donner un tel jugement, mais la complexité du problème ne donne pas une chance à d'autres choix. En plus, le temps d'exécution n'est pas discuter puisque la méthode n'a pas dépassé les deux minutes lors de l'exécution pour chaque instance.

Pour tester l'efficacité de la méthode sur des instances plus importantes, on va tester le front de Pareto sur une instance de 100 sommets, 500 contraintes et 2 objectifs (on remarque que la densité est si faible, ce qui va rendre la résolution du problème très difficile). Le teste est fait en fixant la taille de la population à 200 sur 300 générations, la probabilité de croisement $p_c = 0.9$ et la probabilité de mutation $p_m = 0.1$. On va présenter les résultats sur les graphiques suivants :

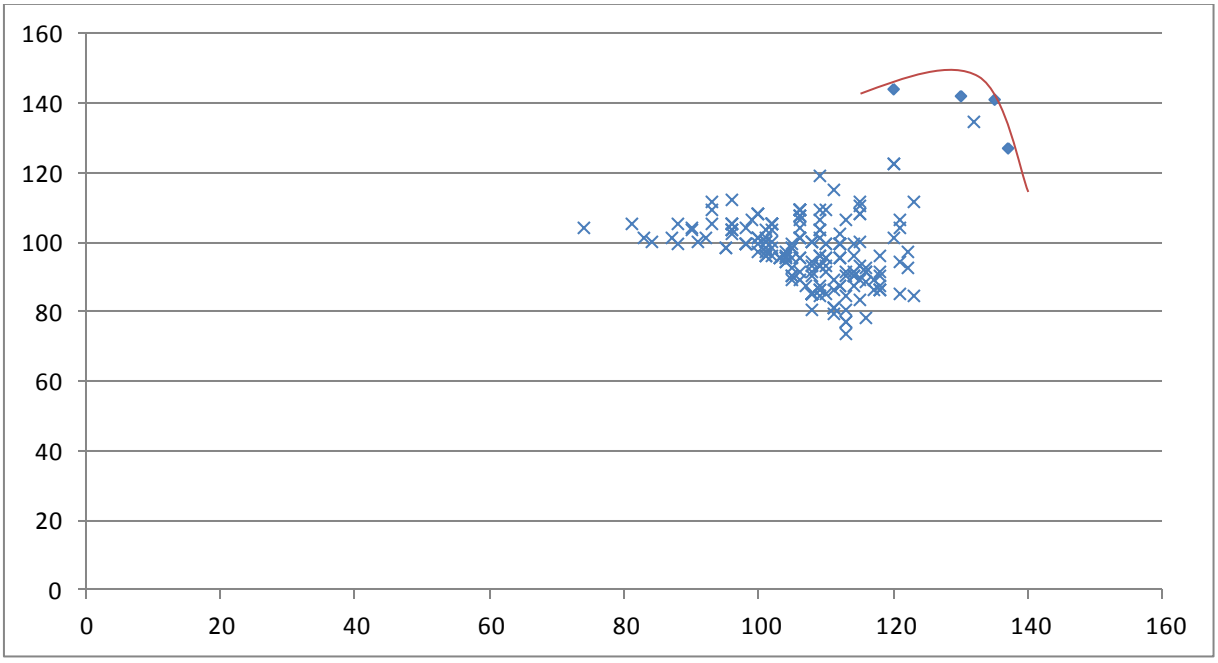


Figure 17 : Espace des critères de l'instance testée

On remarque le front de Pareto en rouge, et quatre solutions potentiellement efficaces (les petits losanges bleus). Le reste des solutions (en croix bleu) représentent l'ensemble des solutions réalisables détectées par la méthode NSGA-II. On remarque dans le graphe suivant la population initiale dont toutes les solutions sont non-réalisables :

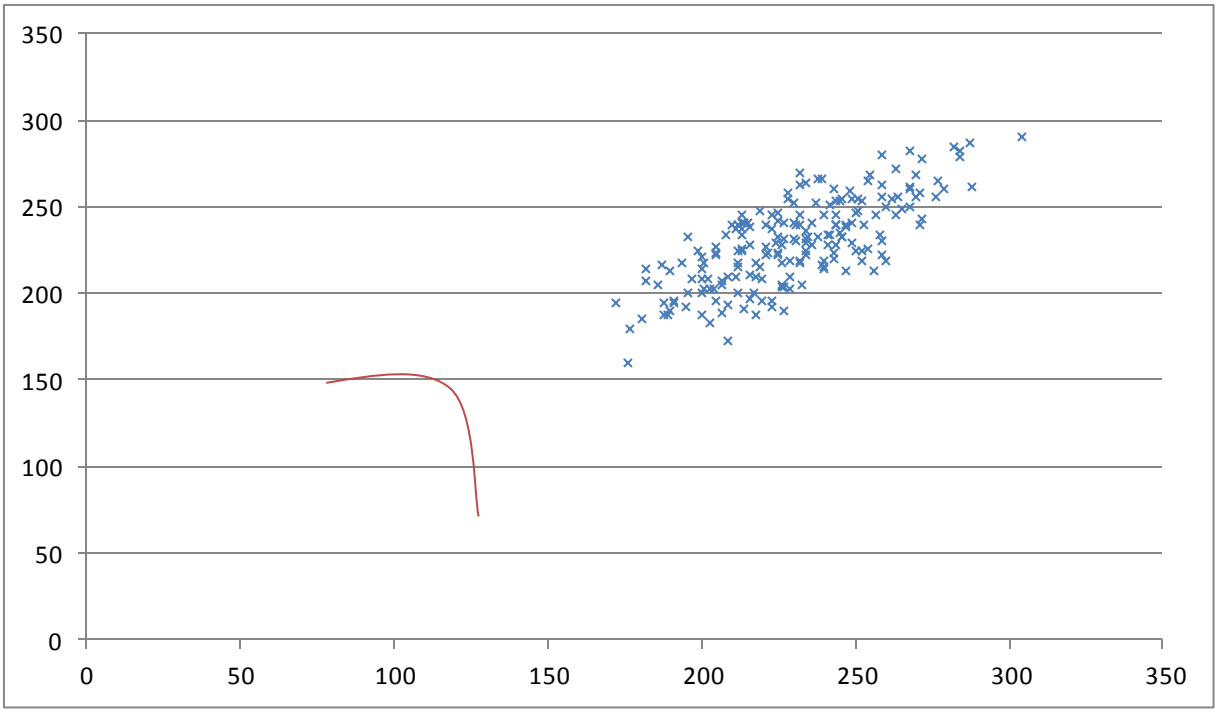


Figure 18 : Population initiale de l'instance testée

Au début du processus, les solutions générées étaient bien loin du front de Pareto, même elles étaient non-réalisables, ce qui veut dire qu'on a une violation des contraintes[1], chaque itération, on préserve les bonnes solutions réalisables (potentiellement efficaces), s'il n'y en a plus on garde ceux avec un minimum de mesure de violation.

La méthode NSGA-II a bien prouvé son efficacité en testant sur plusieurs instances, ainsi que le teste du front de Pareto sur une grande instance, ce qui donne une réponse pour les problèmes de taille importante dans les cas réels.

[1] C'est-à-dire qu'une ou plusieurs contraintes ne sont pas vérifiées.

Conclusion Générale

Le présent travail qu'on a étalé tout au long de ce mémoire, consistait à comprendre et à corréler les deux grands domaines de la recherche opérationnelle : théorie des graphes et programmation linéaire multi-objectif, en intensifiant notre étude sur le problème du stable multi-objectif.

Sans excès d'optimisme, on estime qu'on a atteint l'objectif de cette étude dans le sens où on a pu exploiter les caractéristiques d'un ensemble stable pour contribuer à la mise en œuvre d'une méthode exacte pour le problème du stable multi-objectif, alors qu'à notre connaissance, aucune méthode exacte n'est relatée dans la littérature. En outre, l'utilisation de la métaheuristique NSGA-II pour notre problème, a prouvé son efficacité compte tenu des bons résultats obtenus.

Le système mis en place, demeure malgré tout, une base de travail non négligeable. En fait, les points positifs de cette étude sont :

- Amélioration du temps d'exécution de la résolution du problème du stable multi-objectif par rapport à la méthode EEC.
- Amélioration de la taille du problème qu'on peut résoudre en temps acceptable.
- La souplesse de la méthode, puisqu'on peut utiliser n'importe quelle méthode à la place de la méthode EEC qui peut se valoriser meilleure en termes de temps d'exécution

A travers notre étude, nous pouvons envisager d'autres voies possibles pouvant améliorer les résultats que nous avons obtenus :

- Explorer d'autres façons du tri des sommets qui peuvent améliorer encore l'exécution de la méthode exacte ?
- Doper la procédure d'une autre coupe basée sur le point nadir pour affiner encore plus le domaine des solutions réalisables ?
- Voir la contribution qu'on pourra obtenir si on applique le procédé à d'autres problèmes ?
- Envisager la contribution qu'on pourra obtenir si on remplace la méthode EEC par d'autres méthodes à condition qu'elles soient encore plus efficaces qu'elle en termes de temps d'exécution ?
- Quel serait l'apport du processus de la séparation et évaluation que nous avons développé pour la méthode exacte dans la métaheuristique NSGA-II?

Enfin, on espère que ce modeste travail contribuera positivement dans les recherches qui vont suivre dans ce contexte.

TABLE DES FIGURES

LISTE DES TABLEAUX

Bibliographie

[1] Berge Claude Graphe et Hypergraphe [Ouvrage]. - [s.l.] : Dunod, 1973.

[2] Cayley Arthur On the Colouring of maps [Revue]. - [s.l.] : Coll. Math Papers, 1879. - 11. - pp. 7-8.

[3] Cayley Arthur On the theory of groups [Revue]. - [s.l.] : Coll. Math Papers, 1889. - 12. - pp. 639-656.

[4] Charon Irène et Hudry Olivier Branch-and-Bound Methods [Section] // Concepts of Combinatorial Optimization / auteur du livre Paschos Vangelis Th.. - Great Britain & USA : ISTE & WILEY, 2010.

[5] Chergui Mohamed El-Amine Thèse de doctorat d'état. - Alger : USTHB, 2010.

[6] Dantzig George Bernard Maximization of a linear function of variables subject to linear inequalities [Revue] / éd. Allocation Activity Analysis of Production and. - New York : Wiley, 1951. - pp. 339-347.

[7] Deb Kalyanmoy [et al.] A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II [Revue] // IEEE. - 2002. - pp. 182-197.

[8] Della Croce Frédérico et Grasso Andrea Simplex Algorithms for Linear Programming [Section] // Concepts of Combinatorial Optimization / auteur du livre Paschos Vangelis Th.. - Great Britain & USA : ISTE & WILEY, 2010.

[9] Dhaenens Clarisse, Lemesre J. et Talbi El-Ghazali K-PPM: A new exact method to solve multi-objective combinatorial optimization problems [Revue]. - [s.l.] : Elsevier, 2009.

[10] Dharwadker Ashay The Independent Set Algorithm [Online] // dharwadker. - 2006. - http://www.dharwadker.org/independent_set.

[11] Donoso Yezid and Fabregat Ramon Multi-Objective Optimization in Computer Networks Using Metaheuristics [Book]. - [s.l.] : AUERBACH PUBLICATIONS, 2007. - ISBN 0-8493-8084-7.

[12] Ehrgott Matthias Multicriteria Optimization [Book]. - [s.l.] : Springer, 2005. - ISBN 3-540-21398-8.

[13] Eiselt H. A. et Sandblom C.-L. Linear Programming and its applications [Ouvrage]. - [s.l.] : Springer, 2007. - ISBN 978-3-540-73670-7.

[14] Faure Robert Précis de recherche opérationnelle [Ouvrage]. - [s.l.] : Dunod, 1976.

[15] Geoffrion A.M. Proper efficiency and the theory of vector maximization [Revue]. - [s.l.] : Journal of Mathematical Analysis and Applications, 1968. - 22. - pp. 618-630.

[16] Johnson David S., Yannakakis Mihalis et Papadimitriou Christos H. On generating all maximal independents sets [Article] // Information Processing Letters. - 1988. - North Holland. - 119-123 : Vol. 27.

[17] Klee Victor et Minty George J. How good is the simplex algorithm ? [Revue] / éd. Inequalities. - New York : Academic Press, 1972. - Vol. III. - pp. 159-175.

[18] Kocay William and Kreher Donald L. Graphs, Algorithms, and Optimization Discrete Mathematics and Its Applications [Book]. - [s.l.] : CRC Press, 2005. - ISBN 1-58488-396-0.

[19] Korte Bernhard and Vygen Jens Combinatorial Optimization [Book]. - [s.l.] : Springer, 2006. - ISBN 3-540-43154-3/ISSN 0937-5511.

[20] D. Warrier, W. E. Wilhelm, J. S. Warren, and I. V. Hicks A Branch-and-Price Approach for the Maximum Weight Independent Set Problem [Report]. - Netw., vol. 46, no. 4, pp. 198–209, 2005

[21] Maniezzo Vittorio, Stützle Thomas and Vob Stefan Metaheuristics [Book]. - [s.l.] : Springer, 2009. - ISBN 978-1-4419-1305-0/ISSN 1934-3221.

[22] Nocedal Jorge et Wright Stephen J. Numerical Optimization [Ouvrage]. - New York : Springer, 2006. - 0-387-30303-0.

[23] Özlen Melih et Azizoglu Mera Multi-objective integer programming: A general approach for generating all non-dominated solutions [Revue]. - [s.l.] : Elsevier, 2008.

[24] Paschos Vangelis Th. Basic Concepts in Algorithms and Complexity Theory [Section] // Concepts of Combinatorial Optimization. - Great Britain & USA : ISTE & WILEY, 2010.

[25] Przybylski Anthony, Gandibleux Xavier et Ehrgott Matthias A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives [Revue]. - [s.l.] : Elsevier, 2010.

[26] Sakarovitch Michel Techniques mathématiques et algorithmiques de la recherche opérationnelle: 1- Programmation linéaire [Ouvrage]. - [s.l.] : E.N.S.I.M.A.G., 1983.

[27] Sakarovitch Michel Techniques mathématiques et algorithmiques de la recherche opérationnelle: 2- Optimisation combinatoire [Ouvrage]. - [s.l.] : E.N.S.I.M.A.G., 1983.

[28] Sharieh Ahmed [et al.] An Algorithm for Finding Maximum Independent Set in a Graph [Journal]. - [s.l.] : EuroJournals Publishing, 2008. - 4 : Vol. 23. - ISSN 1450-216X.

[29] Srinivas N. et Deb K. Multiobjective Optimization using Nondominated Sorting in Genetic Algorithms [Revue]. - [s.l.] : Evolutionary Computation, 1994. - 3 : Vol. 2.

[30] Steuer Ralph E. Multiple Criteria Optimization: Theory, Computation and Application [Ouvrage]. - New York : John Wiley and Sons, 1986. - 0271-6232.

[31] Sylva John et Crema Alejandro A method for finding the set of non-dominated vectors for multiple objective integer linear programs [Revue]. - [s.l.] : Elsevier, 2003.

[32] Tomita Etsuji, Tanaka Akira and Takahashi Haruhisa The worst-case time complexity for generating all maximal cliques and computational experiments [Article] // Theoretical Computer Science. - JAPON : Elsevier, 2006. - Elsevier. - 28 – 42 : Vol. 363.

[33] Xinjie Yu and Mitsuo Gen Introducion to Evolutionary Algorithms [Book]. - [s.l.] : Springer, 2010. - ISBN 978-1-84996-128-8/ISSN 1619-5736.

Résumé :

Le problème du stable multi-objectif est un problème difficile à résoudre sachant qu'au mono-objectif l'est déjà. Le présent travail propose une méthode par séparation et évaluation dont la complexité est de l'ordre 2^n en faisant des évaluations sur chaque nœud de l'arborescence en approximant le point idéal local du sous-problème obtenu après la fixation des variables (les sommets dans le graphe correspondant) en tenant compte de la propagation des contraintes encourues, et en fixant une profondeur de l'arborescence pour utiliser sur chaque feuille une méthode dédiée à la résolution des MOILP (pour notre cas, on a la méthode EEC). Cette dernière est testée sur une batterie d'instances pour prouver son efficacité par rapport à la méthode EEC. Etant la méthode proposée est exponentielle, et étant toujours besoin d'une réponse dans un temps raisonnable, on a adapté la métaheuristique NSGA-II pour notre problème et tester son efficacité sur une batterie d'instances et en cherchant le front de Pareto sur une instance de grande taille.

Mots clé : Problème du Stable Multi-Objectif, Méthode par Séparation et Evaluation, Méthode des Ensembles Efficaces Complets (EEC), NSGA-II.

Abstract :
The multi-objective independent set problem is a hard problem for resolving which is the same think in the mono-objective. This paper proposes a branch and bound method which the complexity is about 2^n by assessments on each node of the tree approximating this ideal point of the local sub-problem obtained after setting variables (vertices in the corresponding graph), taking in the account the propagation of constraints encountered, and setting the tree's depth to use on each leaf a method for solving MOILP (in our case, the method is EEC method). The method is tested on a battery of instances to proof its effectiveness relative to EEC method. Since the prosed method is exponential, and because we always need an answer within reasonable time, we adapted the metaheuristic NSGA-II for our problem and test its effectiveness on a battery of instances and seeking the Pareto front on a large instance.

Key words : Multi-Objective Independent Set Problem, Branch and Bound Method, EEC Method, NSGA-II.