

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et Technologie de Houari Boumedienne



Faculté de Mathématiques

Département de Recherche Opérationnelle

Mémoire

En vue de l'obtention du Diplôme de Licence en Recherche Opérationnelle

THÈME

*Méthode d'agrégation pour le
problème d'U sac à des bi-objectifs.*

Présenté par :

Rayan AITMECHEDAL

Mahmoud BENAMMAR

Encadré par :

Nadia BABOU

Table des matières

Table des matières	i
Dédicace Remerciements	1
Introduction générale	1
1 Problème d'optimisation étudié	3
1.1 Optimisation combinatoire	3
1.1.1 optimisation combinatoire mono-objectifs :	4
1.1.2 optimisation combinatoire multi-objectifs(MOCP) :	4
1.1.3 Exemple (Steuer 1985) :	5
1.2 Problème étudié	5
1.3 Programme linéaire associé	6
1.3.1 sac a dos mono-objectif	6
1.3.2 Sac a dos multi-objectif	7
1.3.3 Sac a dos bi-objectif	7
1.4 Méthodes de résolution	7
1.4.1 méthode exacte :	8
1.4.2 méthodes approchée :	8
2 Méthode de résolution	10
2.1 Présentation de la méthode	10

2.1.1	Méthode d'agrégation	10
2.1.2	Dominance de Pareto	11
2.1.3	Optimalité de Pareto	11
2.1.4	Front de Pareto	11
2.1.5	Agrégation pondérée conventionnelle pour MOCP	12
2.1.6	Algorithme de résolution	12
3	Implémentation	14
3.1	Langage de programmation utiliser et sa motivation.	14
3.1.1	Les langages de programmations	14
3.1.2	Python	15
3.2	Manuel d'utilisation	15
3.2.1	Les bibliothèque utilise	15
3.2.2	Les fonctions élémentaires	16
3.2.3	La fonction Pricipale :	19
3.2.4	Un exemple d'utilisation :	19
3.3	Résultats des simulations.	20
3.4	Conclusion :	23
	Bibliographie	I
	Bibliographie	I

Dédicace Remerciements

Rayan :

En tout premier, je remercie le bon dieu, tout puissant, de m'avoir donné la force pour survivre . Je tiens à remercier tout d'abord mon encadreur madame BABOU pour sa patience, et surtout pour ses conseils.

Je dédie ce modest travail a mon chère père qui m'a beaucoup soutenu et encouragé Pour arriver à cette réussite, Dieu le protège . sans oublier ma chère mère qu'elle m'a aidé de près et de loin, et je peux pas oublier mes soeurs Asma et Meriem et mon frère Oussama.

Mahmoud :

C'est avec une grande joie qu'on dedie ce mémoire a nos cher parents pour leur affection inépuisable durant le mois sacré du ramadan , et a notre chere encadreute madame BABOU, et nos amis Skender, nilly et hadjer pour le soutien tout au long de la realisation de ce projet de fin d'étude.

Introduction générale

La Recherche opérationnelle (RO) qui a commencé comme une activité interdisciplinaire pour résoudre des problèmes dans l'armée pendant la Seconde Guerre mondiale, depuis la (RO) c'est et elle comprend un grand nombre de disciplines comme l'Optimisation Linéaire, l'Optimisation non Linéaire, la programmation dynamique, La théorie des files d'attente, La théorie des graphes, etc.

Le but de la recherche opérationnelle est de trouver la meilleure solution d'un problème. Il existe beaucoup de méthodes pour élaborer la meilleure décision et la quand ca vient à l'optimisation linéaire qu'on utilise pour modéliser les problèmes et construire le problème mathématique (Programme Linéaire) associé à ce problème .

On collecte des données associées à ce problème et résoudre ce problème mathématique avec un des algorithmes pour trouver le meilleur résultat possible .

parmi les problèmes qu'on utilise l'optimisation linéaire pour le résoudre est le problème de sac dos.

Le problème de sac a dos note KP (en anglais , Knapsack Problem) Ce problème est inspiré par le mathématicien "tobias dantzig(1884-1956)", Le problème est de remplir un sac à avec des objets ayant un poids et une valeur . Le but est de choisir quel objet à prendre pour maximiser la valeur totale des objets prêts, et sans dépasser le poids maximum des sac .

Ce problème est utilisé dans des différents domaines en tant que sous-problème dans le chargement d'avion : tous les bagages doivent être amenés, sans avoir de surcharge . dans les problèmes de découpe : dans les « matériaux » de dimensions données, découper

(ou disposer) un nombre maximal d'éléments de dimensions plus petites ? Ou encore, comment découper (ou disposer) un nombre donné d'éléments de manière à utiliser une quantité minimale du « matériau » de base ? (1).

Le problème appartient à la classe de problème np-difficile, Il existe différentes variantes de ce problème . qui ne font pas du tout appel aux mêmes méthodes de résolution deux grands types de méthode pour les résoudre : les méthodes exactes et les méthodes approchées. l'un des méthodes approchées de résolution de ce problème est la méthode d'agrégation que nous le verrons dans le chapitre 2

ce memoire est composé de deux chapitres le premier consacré a l'optimisation combinatoire et ses différents methodes de résolution et le problème étudié dans ce memoire qui est le problème du sac a dos. et dans second chapitre on détaillera la methode de résolution qui est la methode d'agregation.

1

Problème d'optimisation étudié

1.1 Optimisation combinatoire

L'optimisation combinatoire est un domaine de la recherche opérationnelle lié généralement à la théorie des algorithmes. elle consiste à trouver un "meilleur" choix parmi un ensemble fini (souvent très grand) de possibilités, C'est une branche de la « Programmation Mathématique » qui recouvre les méthodes qui servent à déterminer l'optimum d'une fonction sous des contraintes données. Il s'agit donc de minimiser ou à maximiser une fonction Z sur un ensemble fini, mais éventuellement très grand, et dont les propriétés mathématiques ne sont pas facilement caractérisables. En d'autres termes, un Problème d'optimisation combinatoire (POC) peut être défini pour une variable de décision x de la manière suivante :

$$(POC) \left\{ \begin{array}{ll} \max Z(x) \\ \text{s.c.} & g_i(x) \leq b_i; \quad i = 1, \dots, n; \quad (1) \\ & h_j(x) = c_j \quad j = 1, \dots, m; \quad (2) \\ & x \in X \end{array} \right.$$

1.1.1 optimisation combinatoire mono-objectifs :

Lorsqu'un seul objectif (critère) est donné, le problème d'optimisation est mono-objectif. Dans ce cas la solution optimale est facile à définir, c'est celle qui a le coût optimal (minimal, maximal).

1.1.2 optimisation combinatoire multi-objectifs(MOCP) :

La plupart des problèmes d'optimisation réels sont décrits à l'aide de plusieurs objectifs ou critères souvent contradictoires. Ces problèmes sont caractérisés par une multiplicité de solutions, c'est-à-dire qu'il n'y a pas une seule meilleure solution (l'optimum global), mais un ensemble de solutions. Alors que, pour les problèmes n'incluant qu'un seul objectif(mono-objectif), l'optimum recherché est facile à définir. les solutions réalisables particulières (pour MOCP), appelées ensemble de Pareto ou solutions non dominées, qui sont supérieures aux autres lorsqu'on les compare à d'autres solutions. solutions non dominées, qui sont supérieures aux autres lorsqu'on considère tous les objectifs. La multiplicité des solutions s'explique par le fait que les objectifs sont contradictoires. et l'optimisation combinatoire bi-objectif est un cas particulier de l'optimisation combinatoire multi-objectif. (2).

Exemple :

Dans le cas de deux objectifs à minimiser, toute amélioration de l'un des 2 objectifs(problème d'optimisation combinatoire bi-objectif) se fait au détriment de l'autre et que la solution optimale ou proche de l'optimum est un compromis entre les deux. Dans l'achat d'une voiture d'occasion, la voiture idéale est celle qui est peu chère (critère économique) avec peu de kilomètres (critère qualitatif), il n'est pas évident de pouvoir regrouper en un seul objectif ces deux critères non commensurables.

Ainsi il n'existe plus une solution optimale unique mais un ensemble de solutions. Nous

allons donc devoir identifier les meilleurs compromis possibles suivant notre budget. Les problèmes bi-objectifs ont la particularité d'être un peu plus difficiles à traiter que leur équivalent mono-objectif. La difficulté réside dans l'absence d'une relation d'ordre total entre les solutions. Une solution peut être meilleure qu'une autre sur certains objectifs et moins bonne sur les autres. Donc il n'existe généralement pas une solution unique qui procure simultanément la solution optimale pour les 2 objectifs. Voilà pourquoi le concept de solution optimale devient moins pertinent en optimisation bi-objectif. Dans ce cas la solution optimale ou de bonne qualité n'est plus une solution unique mais, un ensemble de solutions compromis entre les 2 objectifs à optimiser.

1.1.3 Exemple (Steuer 1985) :

$$\begin{array}{ll}
 \min & Z_1 - 3x_1 + x_2 \\
 \min & Z_2 - x_1 - 2x_2 \\
 \text{sujet à} & x_2 \leq 3 \\
 & 3x_1 - x_2 \leq 6 \\
 & x \geq 0
 \end{array}$$

1.2 Problème étudié

Le problème de sac à dos est un problème d'optimisation combinatoire appartenant à la classe des problèmes NP-complets modélisant une situation dans laquelle un décideur dispose d'un ensemble d'objets, chaque objet est numéroté par l'indice i qui est associé un poids p_i et une valeur v_i pour $i \in (1..n)$ parmi lesquels il doit faire une sélection, en respectant une contrainte de capacité. L'objectif est de maximiser le profit apporté par les objets choisis.

La contrainte exprime le fait que la somme des poids des objets pris doit tenir dans la capacité du sac à dos w . Selon un objet i est sélectionné ou non.

-la variable associée x_i qui prend 1 si le i ème objet est sélectionné ou 0 c'est s'il est laissé de côté.

Il ya une seule contrainte pour ce problème il faut que la somme des poids des objets sélectionnés ne dépasse pas la capacité maximum w de sac à dos : $\sum_{i \in I} x_i p_i \leq W$.

Notation :

- . $z(x)$ est la fonction objective
- . tout vecteur x vérifiant : $\sum_{i \in I} x_i p_i \leq W$ est un vecteur réalisable
- . si la valeur de $z(x)$ est maximal alors x est dit optimal

Dans sa forme la plus simple, dénommée : sac à dos unidimensionnel en variables binaires .
Et nous verrons tous les types de Programme lineaire associe a ce problem.

Appart sa form classique mono-objectif, Il existe d'autres variants du problème de sac à dos, si on ajoute les profits associer à chaque objet on aura plusieurs profits, il devient un problème multi-objectif on s'intéresse dans le prochain chapitre au cas du bi-objective.

exemple pour multi-objective :

Le pays organisateur de la coupe du monde a décidé d' envoyer un avion pour récupérer toutes les équipes . l'avion a un poid maximum de 40,000 kg (ça sera la valeur W) .

chaque joueur a un poid p_i

chaque joueur a une compétence c_i^1

chaque joueur a un salaire c_i^2

on veut avoir un maximum de jouer sur l'avion c_i^3

donc on a 3 fonction a optimiser :

— maximiser les compétence des joueurs prêt $\max z^1(X) = \sum_{i=1}^n x_i p_i^1$

— minimiser le salaire demande $\max z^2(X) = \sum_{i=1}^n x_i p_i^2$

— maximiser les jouer prêt $\max z^3(X) = \sum_{i=1}^n x_i p_i^3$

en général on a une famille de fonctions objectifs : $\max z^j(X) = \sum_{i=1}^n x_i p_i^j, \forall j \in \{1, \dots, m\}$

1.3 Programme linéaire associé

1.3.1 sac a dos mono-objectif

pour le problem de sac a dos mono-objectif(explique preceddement) on peut le formuler ainsi :

$$(KP) \left\{ \begin{array}{ll} \max z(x) = \sum_{i=1}^n v_i x_i \\ \text{s.c.} & \sum_{i=1}^n p_i x_i \leq w; \quad i = 1, \dots, n; \\ & x_i \in \{0, 1\} \end{array} \right. \quad (1)$$

1.3.2 Sac a dos multi-objectif

Pour le problem de sac a dos multi-objectif :est une variante du problème ou plusieurs objectifs sont à maximiser simultanément. on peut le formulu ainsi :

$$(MOKP) \left\{ \begin{array}{ll} \max z_j(x) = \sum_{i=1}^n v_i^j x_i; \\ \text{s.c.} & \sum_{i=1}^n p_i x_i \leq w; \quad i = 1, \dots, n; \\ & x_i \in \{0, 1\} \end{array} \right. \quad (1)$$

1.3.3 Sac a dos bi-objectif

Le problem de sac à dos bi-objectif : c'est un cas particulier du sac a dos multi-objectif ou j=2 c'est tout mais les méthodes de résolution divergent du mono,bi,multi-objectif bien sur .

$$(MOKP) \left\{ \begin{array}{ll} \max z_j(x) = \sum_{i=1}^n v_i^j x_i; \quad j \in \{1, 2\} \\ \text{s.c.} & \sum_{i=1}^n p_i x_i \leq w; \quad i = 1, \dots, n; \\ & x_i \in \{0, 1\} \end{array} \right. \quad (1)$$

1.4 Méthodes de résolution

Il existe deux grandes catégories de méthodes de résolution de problèmes d'optimisation combinatoire : les méthodes exactes et les méthodes approchées. Les méthodes exactes permettent d'obtenir la solution optimale à chaque fois, mais le temps de calcul peut être long si

le problème est compliqué à résoudre. Les méthodes approchées, encore appelées heuristiques, permettent d'obtenir rapidement une solution approchée, donc pas nécessairement optimale.

1.4.1 méthode exacte :

Pour trouver la solution optimale, et être certain qu'il n'y a pas mieux, il faut utiliser une méthode exacte, qui demande un temps de calcul beaucoup plus long (si le problème est difficile à résoudre). Il n'existe pas une méthode exacte universellement plus rapide que toutes les autres. Chaque problème possède des méthodes mieux adaptées que d'autres. Parmi ces méthodes : Programmation dynamique, méthode de séparation et évaluation.

— Méthode de séparation et évaluation (branch and bound) :

La méthode de séparation et évaluation est une procédure très connue pour résoudre les problèmes d'optimisation combinatoire. Elle consiste à décomposer le problème en sous-problèmes.

On représente un ensemble de solutions par un nœud et on étudie ce ensemble de solutions et si ce ensemble ne peut pas fournir une solution optimale ou réalisable on élimine ce ensemble et on étudie les autres ensembles.

— Programmation dynamique :

On introduit un paramètre supplémentaire tel que le problème à résoudre corresponde à la dernière valeur du paramètre. Ce paramètre est choisi de façon que la résolution du problème pour une valeur quelconque ne dépende que de la solution pour les valeurs plus petites.

1.4.2 méthodes approchées :

Une méthode approchée a pour but de trouver une solution avec un bon compromis entre la qualité de la solution et le temps de calcul. Parmi ces méthodes : méthode de recherche en profondeur variable, méthode de recherche Tabou, Algorithme glouton, Algorithme génétique et la méthode d'aggrégation qu'on va expliquer en détails dans le chapitre 2.

— Méthodes de recherche en profondeur variable (variable depth search) :

L'idée de départ de ces méthodes est simple. Tant qu'un optimum local n'a pas été atteint,

chaque itération consiste se déplacer (comme dans l'algorithme du meilleur voisin) vers la meilleure solution contenue dans le voisinage de la solution courante

— Méthode de recherche Tabou :

l'idée de cette méthode est très simple, supposons que partant d'une solution initiale x_0 , On applique une méthode classique de descente du type meilleur voisin. on engendre ainsi une suite de solutions $x_0, x_1, x_2, \dots, x_p$ de valeurs strictement décroissantes et on s'arrête en un minimum local x_p

— Algorithme génétique :

le principe est de faire Évoluer une population de solutions en sélectionnant les meilleures solutions à chaque Étape l'Évolution sera obtenue par croisement des solutions sélectionnées, qui pourront également subir des perturbations aléatoires.

— Algorithmes gloutons (greedy algorithms) :

le principe des algorithmes dits "gloutons" est simple : on part d'une solution partielle non réalisable du problème et on fixe à chaque étape une (ou plusieurs) variables jusqu'à déterminer une solution réalisable

— Méthode d'agrégation :

c'est une méthode approchée pour résoudre des problèmes d'optimisation combinatoires développé par ARTHUR M. Geoffrion et publié en 1968 dans Journal of mathematical analysis and application.

2

Méthode de résolution

2.1 Présentation de la méthode

2.1.1 Méthode d'agrégation

La méthode d'agrégation est une approche directe de l'optimisation multi-objectifs. elle consiste à transformer un problème multiobjectif en un problème qui combine les différentes fonctions objectif du problème en une seule fonction f de façon linéaire :

$$f(x) = \sum_{i=1}^k \lambda_i Z^i(x)$$

Et selon le théorème de Geoffrion , des solutions efficaces peuvent être obtenues en résolvant des problèmes mono-objectif (P). Une seule solution efficace est obtenue par chaque agrégation linéaire des fonctions objectives.

Comme cite au par avant les solutions réalisables particulières (pour MOCP), appelées ensemble de Pareto ou solutions non dominées.

2.1.2 Dominance de Pareto

Un vecteur $u = (u_1, \dots, u_k)$ est dit dominer si et seulement si $u_i - v_i; i = 1; 2, \dots, k$ et il existe au moins un élément avec $u_i - v_i$ tel que : $v = (v_1, \dots, v_k)$

2.1.3 Optimalité de Pareto

On dit d'une solution x qu'elle est optimale au sens de Pareto si et seulement s'il n'existe pas une autre solution x_0 pour que $f(x)$ soit dominée par $f(x_0)$. Toutes les solutions optimales au sens de Pareto pour un problème d'optimisation problème d'optimisation multi-objectif donné sont appelées ensemble optimal de Pareto (P^*).

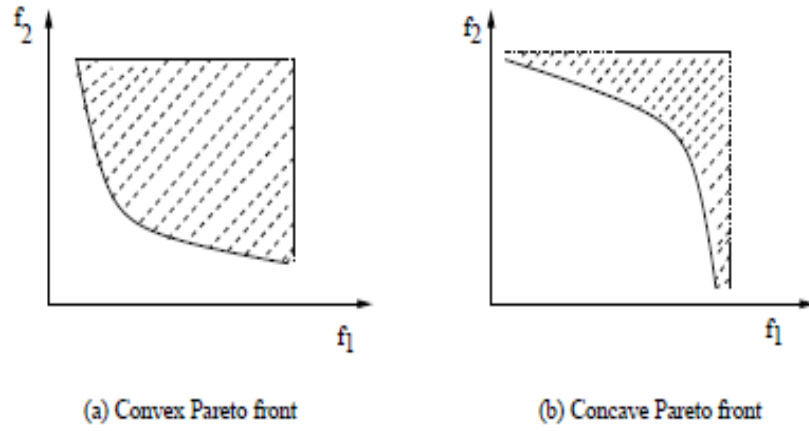
2.1.4 Front de Pareto

Pour un problème d'optimisation multi-objectif donné et son ensemble optimal de Pareto P^* front de Pareto (PF^*) est défini comme suit : $\mathcal{PF}^* = \{f(x) = (f_1(x), \dots, f_k(x)) \mid x \in P^*\}$

Un front de Pareto (PF^*) est dit convexe si et seulement si $\forall \mathbf{u}, \mathbf{v} \in \mathcal{PF}^*, \forall \lambda \in (0, 1), \exists \mathbf{w} \in \mathcal{PF}^* : \lambda \|\mathbf{u}\| + (1 - \lambda) \|\mathbf{v}\| \geq \|\mathbf{w}\|$

on dit qu'un front de Pareto est concave si et seulement si $\forall \mathbf{u}, \mathbf{v} \in \mathcal{PF}^*, \forall \lambda \in (0, 1), \exists \mathbf{w} \in \mathcal{PF}^* : \lambda \|\mathbf{u}\| + (1 - \lambda) \|\mathbf{v}\| \leq \|\mathbf{w}\|$

Par exemple, la figure 1(a) est un front de Pareto convexe et la figure 1(b) un front de Pareto concave. Fig.1(b) est un front de Pareto concave. Bien sûr, un front de Pareto peut être partiellement convexe et partiellement concave.



2.1.5 Agrégation pondérée conventionnelle pour MOCP

L'agrégation pondérée conventionnelle (CWA) est une approche directe de l'optimisation multi-objectifs. Dans cette méthode, les différents objectifs sont additionnés en un seul scalaire avec un poids prescrit.

$$F = \sum_{i=1}^k w_i f_i(\mathbf{x})$$

où w_i est le poids non négatif pour l'objectif $f_i(x)$; $i = 1, \dots, k$. En général, une connaissance a priori est nécessaire pour spécifier les poids. Pendant l'optimisation, les poids sont axés dans la méthode classique d'agrégation pondérée. En utilisant cette méthode, une seule solution optimale de Pareto peut être obtenue avec une seule exécution de l'algorithme d'optimisation. En d'autres termes, si l'on souhaite obtenir des solutions de Pareto différentes, il faut exécuter l'optimiseur plusieurs fois. Cela n'est bien sûr pas possible dans de nombreux problèmes du monde réel, car cela prend généralement trop de temps pour exécuter l'optimisation plus d'une fois.

2.1.6 Algorithme de résolution

On présente ici un exemple d'algorithme 2.1

Algorithme 2.1 : algorithme de l'agrégation

```
1 begin
2   Choisir un pas  $p \in ]0, 1]$  d'incrémentation de  $\lambda$ ;
3   Initialiser  $\lambda \leftarrow \epsilon$ , avec  $\epsilon \approx 0$ ;
4   while  $\lambda \leq 1$  do
5     Résoudre  $(P_\lambda)$  :
6       
$$(P_\lambda) \begin{cases} \max f(x) = \lambda Z^1(x) + (1 - \lambda)Z^2(x) \\ \text{s.c.}, & x \in \Omega \end{cases}$$

7     Incréments  $\lambda \leftarrow \lambda + p$ ;
8   Renvoyer les  $\frac{1}{p}$  solutions obtenus;
```

3

Implémentation

3.1 Langage de programmation utiliser et sa motivation.

3.1.1 Les langages de programmations

Les ordinateurs avec lesquels nous travaillons aujourd’hui, ne sont que des machines qu’il exécutent les choses que nous leur disons. Et nous leur parlons en utilisant un langage de programmation.

Un langage de programmation est un ensemble d’instructions(commandes) et de règles que nous utilisons pour nous interfacer avec une machine.

Il existe de très nombreux langages de programmation différents. Vous pouvez même créer le vôtre. Quelques exemples ? JavaScript. Python. Matlab.

Chacun de ces langages résout un problème dans un domaine différent ou est particuliè-

rement bien adapté dans certaines situations. et on va utiliser python.

3.1.2 Python

Python est un langage de programmation interprété de haut niveau célèbre pour pour ça simple syntaxe et est sans doute le langage le plus populaire au monde car il est facile à apprendre mais pratique pour les grands projets.

python est le langage de choix pour l'analyse de donnée volumineuse .

Il est créé en 1991 par Guido van Rossum

3.2 Manuel d'utilisation

Dans cette partie, nous avons utiliser le langage de programmation python pour implémenter La méthode d'agrégation pour le le problème de sac à dos bi-objectifs.

Nous avons écrit une fonction principale nommée "agregation " qui applique la méthode d'agrégation . pour avoir un programme lisible et bien structre et facile a améliorer on le décompose on plusieurs fonctions élémentaires, et une fonction principale. Cette dernière fait appel a d'autres fonctions élémentaires qu'on va les voir par la suite .

3.2.1 Les bibliothèque utilise

on va tout d'abord parler des bibliothèques utilisé dans notre programme .

OR-Tools :

OR-Tools est une bibliothèque de programmation par contraintes elle disponible dans plusieurs langages comme python Elle a été conçu pour résoudre les problèmes de programmation en nombres entiers et linéaires et de programmation par contraintes.

pour utiliser cette bibliothèque il faut que vous installez or-tools sur votre machine par cette commande :

```
python -m pip install --upgrade --user ortools
```

après vous pouvez l'importer dans votre code comme suit :

```
from ortools.sat.python import cp_model
```

3.2.2 Les fonctions élémentaires

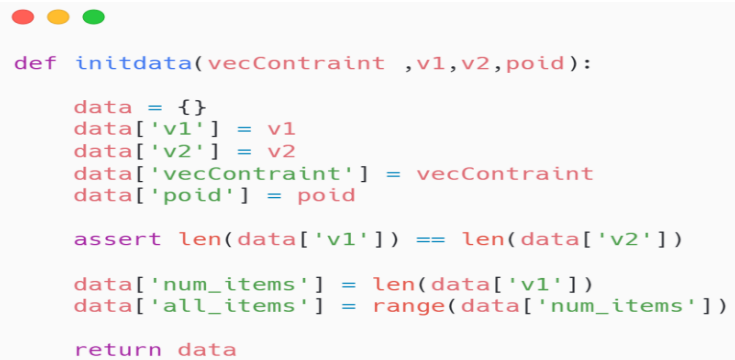
— La fonction **initData** :

Cette fonction reçoit comme donnée : vectContraint, v1, v2 et poid tel que :

- vectContraint : represente la contrainte du probleme. (poid des obejts)
- v1 : Le vecteur associe a z1 (la premier fonction objectif).
- v2 : Le vecteur associe a z2 (la deuxieme fonction objectif).
- poid : est le poid max du sac à dos.

Et qui renvoi comme resultat un objet "data" qu'il contient tout ces données

La figure ci-dessous représente le fonctionnement de cette fonction :



```
def initData(vecContraint ,v1,v2,poid):  
    data = {}  
    data['v1'] = v1  
    data['v2'] = v2  
    data['vecContraint'] = vecContraint  
    data['poid'] = poid  
  
    assert len(data['v1']) == len(data['v2'])  
  
    data['num_items'] = len(data['v1'])  
    data['all_items'] = range(data['num_items'])  
  
    return data
```

— La fonction **addConstraints** :

Cette fonction qui reçoit comme donnée un objet data qui est retourné par la fonction précédente et qui crée apartir de ces donne un modèle mathématique et elle est associée a ce modele une variable de décision x_i et elle est aussi associée a la contrainte donner par l'utilisateur

et elle renvoi 2 objets :

- x : est un obejt qui contient la variable de décision.
- model : est le modèle mathematique associe a ce problem.

La figure ci-dessous représente le fonctionnement de cette fonction :

```

def addConstraints(data):
    model = cp_model.CpModel()

    x = {}

    for i in data['all_items']:
        x[i] = model.NewBoolVar(f'x_{i}')

    model.Add(
        sum(x[i] * int(data['vecContraint'][i])
            for i in data['all_items']) <= data['poid'] )

    return (x,model)

```

— La fonction **optimize** :

qui reçoit comme entre opt, data, x, model, objective et elle renvoie la valeur optimal de la fonction objective donne tel que :

- opt : le but soit maximizer ou minimiser.
- data : les données associe le programme linéaire.
- model : est le modèle mathématique.
- x :la variable de décision.
- objective : la fonction objective pour optimiser.

La figure ci-dessous représente le fonctionnement de cette fonction :

```

def optimize(opt,data,x,model,objective):

    if opt == 'Maximize' :
        model.Maximize(cp_model.LinearExpr.Sum(objective))
    else:
        model.Minimize(cp_model.LinearExpr.Sum(objective))

    return objective

```

— La fonction **afficheResultat** :

qui reçoit comme entre data, x, model et qui renvoi les resultat de ce model comme : la valeur optimal de f et la valeur optimal de la premier fonction obejtive et la valeur optimal de la 2eme fonction obejtive et le poid des objet pret

La figure ci-dessous représente le fonctionnement de cette fonction :

```
def afficheResultat(x,data,model):
    solver = cp_model.CpSolver()
    status = solver.Solve(model)
    if status == cp_model.OPTIMAL:
        print(f'La Valeur optimal de f est : {solver.ObjectiveValue()}')

        poid_total = 0
        valeur_total = 0
        for i in data['all_items']:
            if solver.Value(x[i]) > 0:
                print(
                    f"L'objet {i} de valeur1: {data['v1'][i]} valeur2: {data['v2'][i]}"
                )
                poid_total += int(data['v2'][i])
                valeur_total += int(data['v1'][i])
        print(f"Donc on a dans le sac une valeur de : {valeur_total}")
        print(f"Donc on a dans le sac un poid de : {poid_total}\n")

        return solver.ObjectiveValue()

    else:
        print('The problem does not have an optimal solution.')
```

— La fonction **resoudre P λ** :

Cette fonction résout les programmes linéaires P λ qui est représenté dans le chapitre precedent. La figure ci-dessous représente le fonctionnement de cette fonction :

```
def resoudrePλ(vecContraint,v1,v2,lmda,poid):

    data = initdata(vecContraint,v1, v2, poid)
    x,model = addConstraints(data)

    objectivev1 = []
    for i in data['all_items']:
        objectivev1.append(
            cp_model.LinearExpr.Term(x[i], lmda * int(data['v1'][i])))

    objectivev2 = []
    for i in data['all_items']:
        objectivev2.append(
            cp_model.LinearExpr.Term(x[i], (lmda - 1 ) * int(data['v2'][i])))

    objective = objectivev1 + objectivev2

    optimize('Maximize', data, x, model,objective)

    afficheResultat(x, data, model)
```

3.2.3 La fonction Principale :

aggregation

qui resout le problem du sac a dos bi-objective avec la methode d'aggregation et qui recoit comme entré vectContraint, v1, v2. tel que : Choisir un pas $p \in]0, 1]$ d'incrémentation de λ ;
Initialiser $\lambda \leftarrow \epsilon$, avec $\epsilon \approx 0$;

et elle fait $1/p$ itterations et a chaque fois elle renvoi une solition de $p\lambda$ qui est meilleur que la précédente.

La figure ci-dessous représente le fonctionnement de cette fonction :



```
def aggregation(vecContraint, v1, v2, lmda, p, poid):  
  
    counter = 0  
    while lmda < 1:  
        counter += 1  
  
        resoudrePλ(vecContraint, v1, v2, lmda, poid)  
        lmda += p  
  
    print(counter, 'iteration pour resoudre ce problem.')
```

3.2.4 Un exemple d'utilisation :

Par exemple :

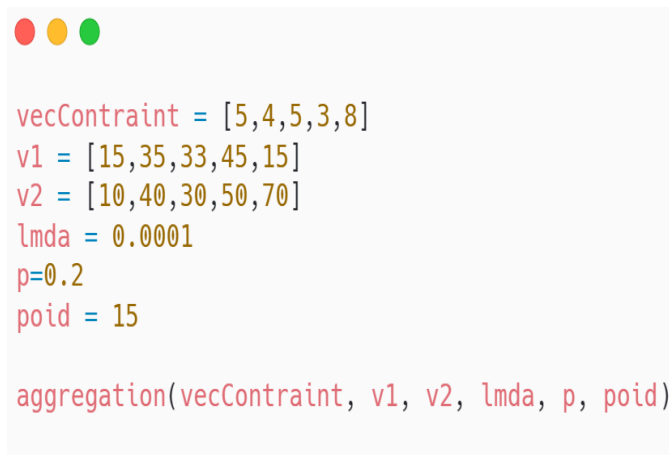
On prend les données suivant :

- Nombre des objets : $n=5$
- Capacité de sac : $w=15$
- les poids des 5 objet : $p_1=5, p_2=4, p_3=6, p_4=3, p_5=8$
- le premier objective : $v_{11}=15, v_{12}=35, v_{13}=33, v_{14}=45, v_{15}=60$
- le deuxieme objective : $v_{21}=10, v_{22}=40, v_{23}=30, v_{24}=50, v_{25}=70$

pour résoudre ce problem on tape à la fonction principale "aggregation" avec les données suivantes :

- $\text{poid} = 15$
- $\text{vectContraint} = [5,4,5,3,8]$
- $\text{v1} = [15,35,33,45,15]$
- $\text{v2} = [10,40,30,50,70]$
- $\lambda \approx \epsilon$
- $p \in]0, 1]$

La figure ci-dessous représente le fonctionnement de cette fonction :



```

vecContraint = [5,4,5,3,8]
v1 = [15,35,33,45,15]
v2 = [10,40,30,50,70]
lmda = 0.0001
p=0.2
poid = 15

aggregation(vecContraint, v1, v2, lmda, p, poid)

```

3.3 Résultats des simulations.

Nous avons créer une interface pour facilité l'utilisation de l'outil de programmation. La figure ci-dessous représente la forme de cette interface :

cette page represante l'ensemble des entrées où l'utilisateur doit identifiées. pour resoudre ce problem.

Le bouton qui apparait au milieu de l'interface nommé " solver " permet d'appliquer la Méthode d'agrégation pour le problème du sac à dos bi-objectifs. en utilisant les données stockées par l'utilisateur.

Sac a Dos

Bienvenu

La 1er fonction objective

La 2eme fonction objective

donner le vecteur de votre contraint

donner Le poid maximum de sac :

Après donner les deux fonction objectives et les constraints et le poids maximum du sac comme suit :

Sac a Dos

Bienvenu

La 1er fonction objective

La 2eme fonction objective

donner le vecteur de votre contraint

donner Le poid maximum de sac :

On appuye sur Le bouton " solver " et en voit ce resultat :

les valeur de xi :

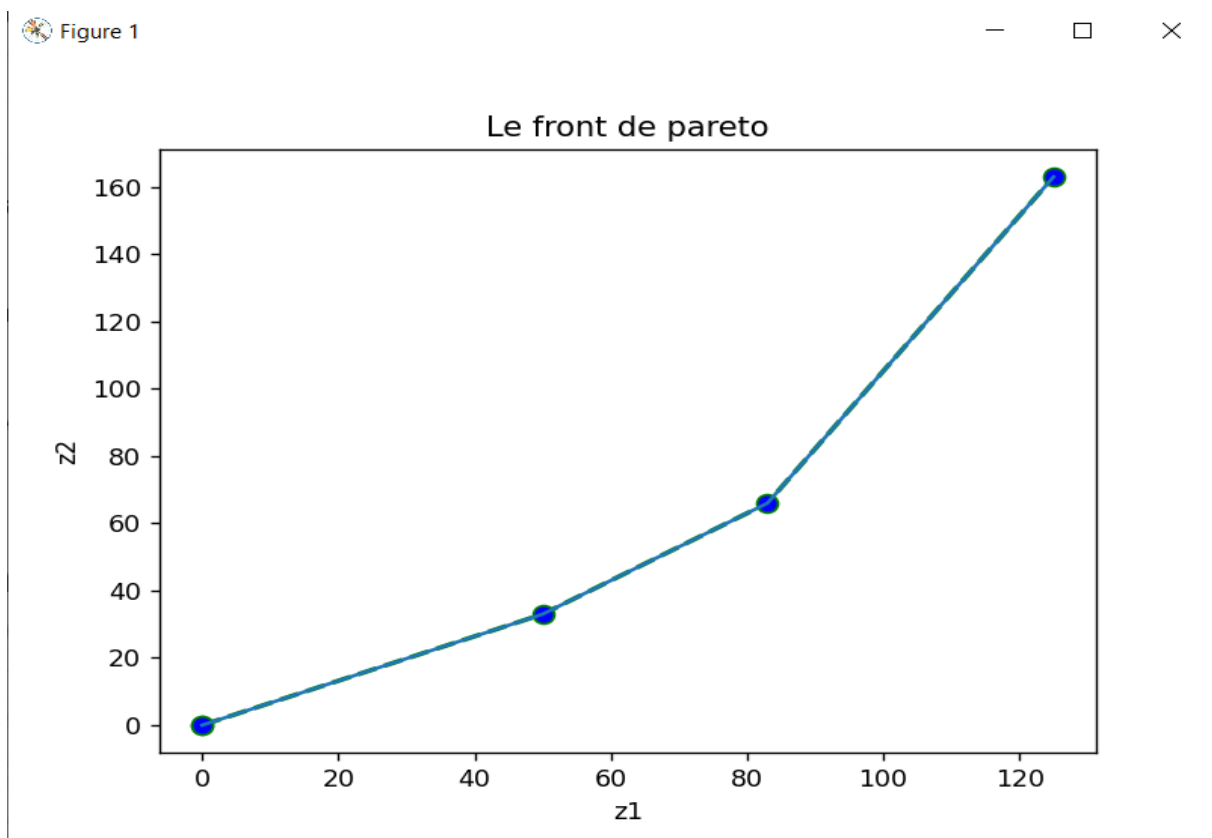
la valeur optimal de f apres la premier iteration :

la valeur optimal de f apres la 2eme iteration :

la valeur optimal de f apres la 3eme iteration :

la valeur optimal de f apres la 4eme iteration :

Voici les resultats du front de Pareto tel que Z1 et Z2 sont les deux fonctions objectifs(Le graphe) :



On remarque que les objet pret sont : le 2ème, 4ème et le 5ème avec 3 résultat et la valeur maximum de f c'est 110

Pour teste l'efficacité de ce programme on va le tester avec des differents nombre d'objet et voir combien de temps pour il renvoi le resultat on a le tableaux suivant :

tell que :

n : est le nombres d'objets,

T : est le temp pour résoudre ce problem avec n obejts

n	T
100	0,15 s
500	0.33 s
1000	0.75 s
10000	6 s
50000	43 s
100000	182 s
300000	23 min
500000	plus de 1 h

Remarque : ces résultat ce différent d'une machine a autres.

3.4 Conclusion :

Nous devons avouer que retrospectivement,nous sommes plutot satisfait de ce mémoire puisque nous avons atteint de nouveaux objectifs.

En effet,ce mini projet nous a permis de comprendre et apprendre les principaux concepts de l'optimisation combinatoire ,ainsi l'une des méthodes de résolution qui est la méthode d'agrégation,pour cela nous avons présenté a travers notre travail,les propriétés fondamentales de cette dernière,sans oublier son algorithme. nous poursuivant notre étude par l'implémentation de la méthode d'agrégation sous le langage de programmation PYTHON.

Notre recherche,nous a permis de voir la complexité et la richesse du sujet,tout en sachant que ce dernier est très important et efficace dans le domaine universitaire.

Bibliographie

- [1] Formalisation et résolution des problèmes de découpes linéaires, *RAIRO. Recherche opérationnelle*, tome 16, no 1 (1982), p. 65-8.
- [2] Alaya, I. Optimisation multi-objectif par colonies de fourmis : cas des problèmes de sac à dos, 2009.