Travaux d'Etude et de Recherche

Stagiaires : Arthur Gontier[1], Sanjy Andriamiseza[2]
Encadrants : Anthony Przybylski[3], Xavier Gandibleux[4]

2019

UNIVERSITÉ DE NANTES

Optimisation multiobjectif : Méthode du simplexe multicritère pour V-Opt solver

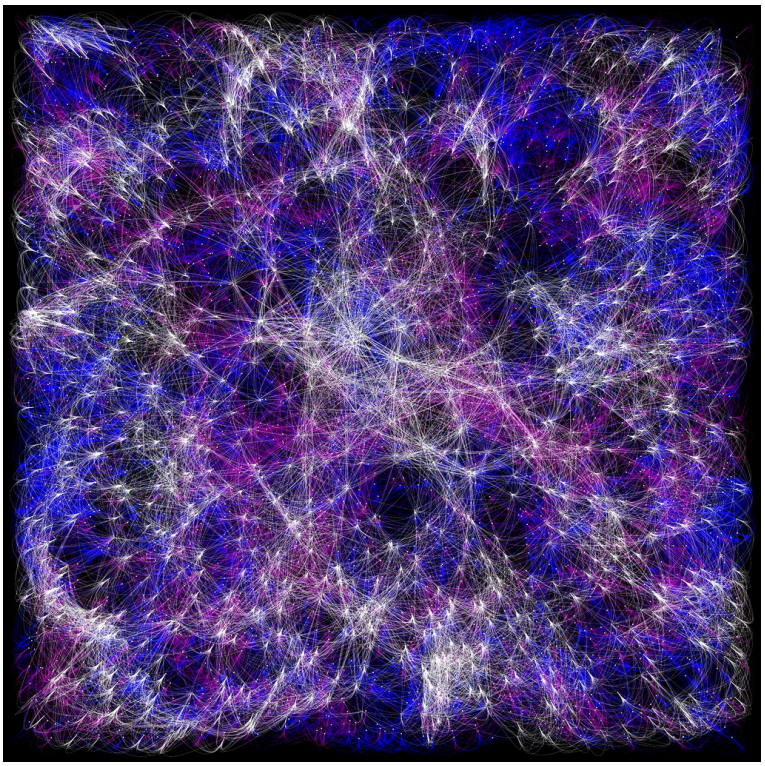

Table des matières

1 Introduction **4**

1.1 Motivations 4

1.2 Objectifs 4

2 Rappel de programmation mathématique **4**

2.1 La Programmation Linéaire 4

2.1.1 Définitions et notations 4

2.2 Simplexe 6

2.2.1 Tableau simplexe 6

2.2.2 L'algorithme du simplexe 7

2.2.3 L'algorithme du simplexe : Phase 2 7

2.2.4 L'algorithme du simplexe : Phase 1 10

2.2.5 La dégénérescence 11

2.3 Programme dual 13

2.3.1 La dualité 13

2.3.2 Construire le dual d'un PL 14

2.3.3 Propriété importante de la dualité 15

3 Forme révisée du simplexe **15**

3.1 Point de vue algébrique 16

3.2 Forme révisée et décomposition LU 17

3.2.1 Comment trouver la variable entrante : 17

3.2.2 Comment trouver la variable sortante : 18

3.2.3 Décomposition LU : 18

4 Optimisation multiobjectif **19**

4.1 Introduction à l'optimisation multiobjectif 19

4.2 Dominance et efficacité 20

4.3 Différentes approches 21

4.4 Les propriétés importantes en optimisation linéaire multiobjectif : 21

4.5 Principe 22

4.6 Phase 1 23

4.7 Phase 2 23

4.8 Phase 3 23

4.8.1 Un problème auxiliaire 23

4.9 Exemple d'application du simplexe multicritère : 24

4.9.1 Incorporation de la forme révisé LU 26

5 Concernant l'implémentation **26**

5.1 Implémentation efficace et Structure de donnée 27

5.1.1 Structure des liste de Bases 27

5.2 Décomposition LU en pratique 27

5.3 Instances 27

5.3.1 Générateur d'instances et parseur 27

5.4 Structuration d'un MOLP en machine 28

5.5 Julia et JuMP 29

5.6 La dégénérescence 30

5.6.1 La récupération des variables en base 30

5.6.2 La suppression des combinaisons linéaires 30

6 Limites du simplexe multicritère **32**

7 Objectifs repoussés **32**

7.1 Matrices creuses 32

7.2 Interface avec voptgeneric 32

8 Conclusion **32**

1 Introduction

1.1 Motivations

L'optimisation multiobjectif est un large domaine en voie de développement qui prend de plus en plus d'ampleur dans notre quotidien. Que ce soit dans le monde des entreprises (industrie, supply chain, management...) ou dans le monde scientifique (mathématique, physique, informatique...), elle revêt de nombreuses formes et la maîtrise de cette discipline devient, dans le cadre de la recherche opérationnelle, une nécessité. Les points de vue sont variés mais dans une optique évidemment scientifique la problématique sera toujours d'exposer au mieux des outils et des solutions pour traiter des situations s'y référant.
Il existe donc pléthores d'approches et de méthodes pour aborder l'optimisation multi-critères en passant par des métaheuristiques (SEMO Simple Evolutionary Multiobjective Optimizer, NSGA Nondominated Sorting Genetic Algorithm ...) à des algorithmes d'approximation (HypE - Hypervolume Estimation Algorithm for Multiobjective Optimization) ou bien à la méthode du simplexe multicritère pour une résolution exacte. L'optimisation multiobjectif peut se subdiviser en plusieurs branches c'est dans une optique linéaire et de résolution exacte que nous avons traiter la thématique. Cela s'intitule, MOLP : MultiObjective Linear Programming.

1.2 Objectifs

L'objectif était de ce fait de fournir une implémentation de la méthode du simplexe multicritère pour un solver OpenSource qui est Vopt-Solver. Bien que comme nous l'ayons dit, la demande est forte, malheureusement dans le panel d'outils et de solver qui sont sur le marché, les possibilités sont déjà moindres si l'on veut se restreindre à des instruments gratuits ou open-source. Et même dans le domaine du monde propriétaire des solvers commerciaux le choix n'est pas large. C'est pourquoi ce TER pourrait déjà fournir une première brique à une demande grandissante.

2 Rappel de programmation mathématique

2.1 La Programmation Linéaire

2.1.1 Définitions et notations

La programmation linéaire ou l'optimisation linéaire se traduit par le fait d'avoir un objectif (linéaire) à minimiser ou à maximiser selon certaines contraintes elles aussi linéaires.
Plus mathématiquement on aura donc une fonction souvent notée z exprimée en fonction de variables $(x_1, x_2, ..., x_n)$ le tout contraint avec un ensemble d'inégalités ou d'égalités. Dans la suite nous ferons des raisonnements inductifs et partirons de l'exemple pour comprendre les mécaniques.

Par exemple : Supposons que nous fabriquions deux produits de consommation devant passer successivement par deux usines disctinctes. Pour une journée, le produit $x1$ rapporte 3 euros de bénéfices et le produit $x2$ en rapporte 2. L'usine 1 en produisant le produit 1 et 2 consomme respectivement 3kWh et 1kWh. Puis l'usine 2 en finalisant le produit 1 et 2 consomme respectivement 2kWh et 2kWh. Si les usines ne peuvent malheureusement fournir une consommation d'électricité supérieure à 12 Kwh à la journée. Le but étant d'optimiser le plan de production des produits et d'avoir le maximum de bénéfice à la journée. On aura la formulation suivante en reprenant les unités citées :

$$\begin{array}{lrcl} \text{Max (bénéfice)} & 3x_1 + 2x_2 & & \\ \text{s.t.} & 3x_1 + x_2 & \leq & 12 \text{ (usine 1)} \\ & 2x_2 + 2x_2 & \leq & 12 \text{ (usine 2)} \\ & x_1, x_2 & \geq & 0 \text{ (intégrité variables)} \end{array}$$

Si nous représentons graphiquement le problème nous obtenons ceci :

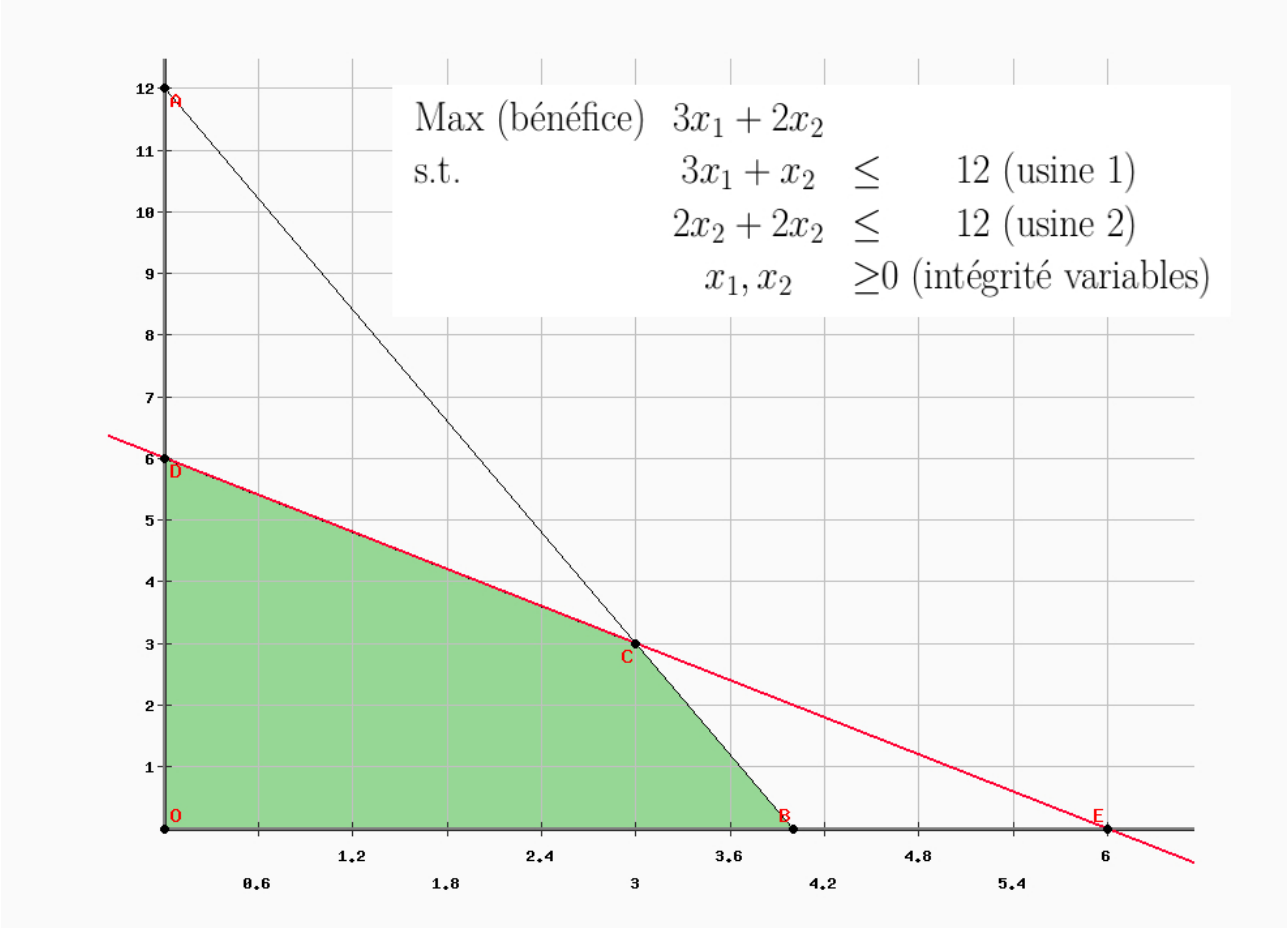

FIGURE 1 – Résultat obtenus via solveur du modèle de LP

Dans le graphique, l'ensemble convexe construit par les contraintes forme une figure appelée polytope. La région décrite par le polytope correspond à l'ensemble admissible des solutions aussi appelé région admissible notée X représentée en vert dans le graphique. Une solution optimale est une solution qui appartient à l'ensemble admissible et dont aucune autre solution admissible n'est strictement meilleure qu'elle. On constate bien visuellement que la ou les solutions optimales se trouveront sur une extrémité du polyèdre, soit sur une arrête, soit sur un sommet.
Remarque : La région admissible correspond à un polytope ou à un polyèdre mais ces deux termes semblent ambigus, en effet différentes définitions circulent. De notre point de vue : un polytope correspond à l'enveloppe convexe d'un nombre fini de points et un polyèdre correspond à l'intersection de demi-espaces fermés. Logiquement un polytope est bien un polyèdre borné.

Les variables dont on souhaite déterminer les valeurs qui optimisent la fonction objectif sont appelées les variables de décisions. On notera c^T le vecteur ligne des couts réduits qui correspond aux coefficients que multitplient les variables dans la fonction objectif. A sera la matrice de contraintes. Et b un vecteur colonne qui possédera les seconds membres donc la partie droite des inégalités et égalités.

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad c^T = (3, 2) \quad A = \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \quad b = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

La forme générale des programmes linéaires ressemblent donc à cela :

$$\begin{array}{lll} \text{Max} & c^T x & \\ \text{s.t.} & Ax & \leq \text{ b} \\ & x & \geq 0 \end{array}$$

La forme présentée ci-dessus correspond à la forme canonique d'un programme linéaire.
Il existe aussi une autre forme qui s'appelle la forme standard. Dans cette forme on introduit une variable d'écart pour remplacer chaque contrainte d'inégalité par une contrainte d'égalité. Soit k un vecteur de taille de x. Dans le cas où nous avons une inégalité inférieure, on ajoute la variable d'écart pour compenser.

$$k^T x \leq b$$
$$k^T x + e = b$$

Dans le cas où nous avons une inégalité supérieure, on aura une variable d'excèdent e (qu'importe le nom de variable) et donc on aura à retirer et non à ajouter.

$$k^T x \geq b$$
$$k^T x - e = b$$

Les deux formes sont strictement équivalentes sauf que la forme standard mais l'algorithme du simplexe se repose sur la forme standard. Nous le verrons dans la suite. Pour bien visualiser la forme standard voici la forme standard du problème linéaire des usines.

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 + 0x_3 + 0x_4 & & \\ \text{s.t.} & 3x_1 + x_2 + x_3 & = & 12 \\ & 2x_1 + 2x_2 + x_4 & = & 12 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

2.2 Simplexe

2.2.1 Tableau simplexe

L'algorithme du simplexe est un algorithme générique pour résoudre et trouver la solution optimale d'un programme linéaire. Il se fait en 2 phases. La première phase consiste à se placer sur une première solution admissible et la seconde consiste à se déplacer de solutions en solutions en vue d'atteindre la solution optimale. Le déplacement de solutions en solutions correspond à un déplacement de sommet en sommet sur le polytope. Bien qu'en pire cas, la complexité de l'algorithme soit exponentielle, en pratique et en moyenne le simplexe est très efficace. Pour appliquer l'algorithme du simplexe, il faut mettre notre problème sous forme standard et tracer le tableau simplexe correspondant. Reprenons toujours le problème formulé auparavant sous forme standard.

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 + 0x_3 + 0x_4 & & \\ \text{s.t.} & 3x_1 + x_2 + x_3 & = & 12 \\ & 2x_1 + 2x_2 + x_4 & = & 12 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

Voyons par la suite à quoi correspond un tableau simplexe. Le tableau simplexe initial correspondant à un programme linéaire est une traduction quais immédiate de la forme standard.

- La première ligne correspond aux coûts réduits des variables (les coefficients de la fonction objectif)
- Il y a par la suite autant de lignes que de contraintes et le tableau se remplit avec les coefficents présents dans les contraintes
- La dernière colonne correspond aux seconds membres b

	x_1	x_2	x_3	x_4	
c	3	2	0	0	0
x_3	3	1	1	0	12
x_4	2	2	0	1	12

Les variables fixées à 0 sont appelées variables hors base et les autres variables en base. Nous expliquerons ultérieurement pourquoi nous fixons des variables à 0. Attention une variable hors base est forcément nulle mais cela ne signifie pas qu'une variable de base est non-nulle (nous en reparlerons aussi dans la suite). Ici x_1 et x_2 ont un coût réduit de (3,2) et valent (0,0). Les variables de base sont indiquées dans la première colonne ici nous avons x_3 et x_4 qui sont en base et ont obligatoirement un coût réduit de (0,0), aussi leur valeur est indiquée dans la dernière colonne qui est (12,12). x_3 et x_4 sont égales à 12, x_1 et x_2 sont égales à 0.

c	3	2	0	0	0
x_3	3	1	1	0	12
x_4	2	2	0	1	12

Voici un autre exemple de tableau simplexe qui possède la variable de décision x_1 en base.

c	0	1	-1	0	-12
x_1	1	$\frac{1}{3}$	$\frac{1}{3}$	0	4
x_4	0	$\frac{4}{3}$	$-\frac{2}{3}$	1	4

On a x_1 et x_4 en base et qui ont comme valeur 4. Celles qui ne sont pas indiquées dans la ligne sont hors bases et valent donc 0. De ce fait les variables de décisions x_2 et x_3 sont nulles (car hors bases).

x_1 et x_4 sont égales à 4, x_2 et x_3 sont égales à 0. La décision de prendre le couple $x_1, x_2 = (4,0)$ comme plan de production est une solution admissible dite solution de base admissible avec les valeurs d'écarts ayant comme valeurs $x_3, x_4 = (0,4)$.
La praticité du tableau simplexe est convaincante puisque nous pouvons aussi lire la valeur du bénéfice réalisée par le plan de production correspondant. Comme nous l'avons dit $x_1, x_2 = (4,0)$ cela signifie que la valeur de la fonction objectif $z(x)$ est égale à
$z(x) = 3x_1 + 2x_2$
$z(x) = 12$
On note dans le tableau simplexe que dans la dernière colonne de la première ligne, il y a la valeur -12, en effet c'est l'opposé de la valeur de la fonction objectif.
Pour conclure, il est possible avec un tableau simplexe de savoir quelle est la valeur de la fonction objectif, de connaître le plan de production associé avec les valeurs des variables d'écarts. Toutes ces informations nous permettrons de comprendre l'algorithme du simplexe.

2.2.2 L'algorithme du simplexe

On se place toujours dans le cadre d'un problème écrit sous forme standard. L'algorithme du simplexe se déroule en deux phases. La première phase consiste à se placer sur une première solution de base admissible et la deuxième consiste à se déplacer de solutions en solutions pour approcher une solution optimale.
Nous verrons d'abord la phase 2 puis la phase 1 car la mécanique algorithmique de la phase 2 est celle utilisée dans la phase 1.

2.2.3 L'algorithme du simplexe : Phase 2

La phase 2 de l'algorithme part du tableau simplexe associé à une solution admissible d'un problème linéaire. Puis en somme : il choisit de faire rentrer et sortir de la base des variables pour atteindre une solution optimale. Un sommet du polytope est aussi appelé un point extrême et ce que l'algorithme opère c'est justement un déplacement de point extrême en point extrême. Or la propriété même d'un point extrême est de vérifier autant de contraintes à l'égalité qu'il n'a de variables. C'est cela qui justifie le fait qu'il doit toujours y avoir des variables hors bases qu'on fixe à 0 dans la forme canonique pour que l'on puisse se placer sur un point extrême.

Comment déterminer la variable entrante ?
Pour savoir quelle variable rentrer en base, il faut comprendre la signification du coût réduit. Le coût réduit représente la variation dans la fonction objectif de l'augmentation d'une unité de la variable associée. Pour notre

tableau simplexe, cela implique que la variable à faire entrer en base correspond à la variable qui a la valeur de coût réduit la plus positive ou négative selon qu'on est en min ou max.
Attention ce choix cependant est une décision gloutonne, n'importe quelle variable autorisée à entrer en base fait l'affaire. C'est un choix heuristique qui semble avoir fait ses preuves mais il est tout à fait possible de faire autrement.

Comment déterminer la variable sortante ?

c	3	2	0	0	0
x_3	3	1	1	0	12
x_4	2	2	0	1	12

On a les variables x_3 et x_4 en base, on veut faire rentrer la variable x_1
Faire rentrer n'importe quelle variable peut mener à violer les contraintes. Il s'agira de regarder quelle variable sera la première à poser problème lorsque la variable entrante augmente. Récrivons les variables en bases en fonction de la variable entrante :

$$x_3 = 12 - 3x_1 \ (1)$$
$$x_4 = 12 - 2x_1 \ (2)$$

$$x_3 = 12 - 3 * 4 \equiv x_3 = 0 \ (1)$$
$$x_4 = 12 - 2 * 6 \equiv x_3 = 0 \ (2)$$

x_3 s'annule lorsque x_1 est à 4 .
x_4 s'annule lorsque x_1 est à 6 .

En conclusion la variable à faire sortir est la première variable à s'annuler lorsque la variable entrante augmente.

Nous avons maintenant toutes les briques pour expliciter l'algorithme du simplexe. L'algorithme du simplexe réalise plusieurs itérations. Une itération consiste à faire rentrer et sortir une variable. On appelle l'intersection entre la colonne de la variable entrante et la ligne de la variable sortante le pivot. Comme dans une méthode de pivot de Gauss : on va chercher à annuler tous les coefficients de la colonne pivot (colonne de la variable entrante) et mettre le pivot à 1.
Exécutons l'algorithme du simplexe sur l'exemple des usines pour y voir plus clair.

Voilà le programme linéaire de notre problème de production.

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 + 0x_3 + 0x_4 & & \\ \text{s.t.} & 3x_1 + x_2 + x_3 & = & 12 \\ & 2x_1 + 2x_2 + x_4 & = & 12 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

Le premier tableau simplexe associé à celui-ci ayant les variables d'écart en base.

	x_1	x_2	x_3	x_4	
c	3	2	0	0	0
x_3	3	1	1	0	12
x_4	2	2	0	1	12

On veut passer à l'itération suivante, dans la ligne des coûts réduits x_1 est à 3 et x_2 est à 2. x_1 sera donc la variable entrante.
Dans la ligne de la variable en base x_3 on a $\frac{12}{3} = 4$ et pour la variable en base x_4 on a $\frac{12}{2} = 6$. Le minimum des deux est 4 donc x_3 est la variable sortante.
Au final x_1 entre et x_3 sort, 3 sera notre pivot :

	x_1	x_2	x_3	x_4		
c	3	2	0	0	0	L_1
x_3	3	1	1	0	12	L_2
x_4	2	2	0	1	12	L_3

On souhaite annuler la colonne pivot, pour cela on a réalisé les opérations suivantes : $L_1 = L_1 - L2$; $L_2 = \frac{L_2}{3}$; $L_3 = L_3 - L2\frac{2}{3}$

	x_1	x_2	x_3	x_4		
c	0	1	-1	0	-12	L_1
x_1	1	$\frac{1}{3}$	$\frac{1}{3}$	0	4	L_2
x_4	0	$\frac{4}{3}$	$-\frac{2}{3}$	1	4	L_3

On obtient :

c	0	1	-1	0	-12
x_1	1	$\frac{1}{3}$	$\frac{1}{3}$	0	4
x_4	0	$\frac{4}{3}$	$-\frac{2}{3}$	1	4

On peut faire rentrer la variable x_2 hors base en base car dans la ligne des coûts réduits $1 > 0$. et nous avons vu auparavant que faire rentrer en base une variable au coût réduit positif est rentable pour la fonction objectif. Donc x_2 entre en base. Regardons la variable en base qui s'annule la première :

$$Min\{4 \div \tfrac{1}{3}, 4 \div \tfrac{4}{3}\}$$

$$Min\{12, 3\}$$

$Min\{12, 3\} = 3$ Cela correspond à la variable x_4, donc x_2 rentre et x_4 sort.

c	0	1	-1	0	-12	$L_1 = L1 - \frac{3}{4}L3$
x_1	1	$\frac{1}{3}$	$\frac{1}{3}$	0	4	$L_2 = L2 - \frac{1}{4}L3$
x_4	0	$\frac{4}{3}$	$-\frac{2}{3}$	1	4	$L_3 = \frac{3}{4}L3$

Après avoir réalisé les opérations pour chaque ligne on obtient :

c	0	0	$-\frac{1}{2}$	$-\frac{3}{4}$	-15
x_1	1	0	$\frac{1}{2}$	$-\frac{1}{4}$	3
x_2	0	1	$-\frac{1}{2}$	$\frac{3}{4}$	3

Une fois cette itération finie, on peut constater que les coûts réduits sont strictement négatifs et comme nous sommes en maximisation cela implique qu'il n'est plus rentable de faire rentrer en base une quelconque variable. C'était la dernière itération et nous obtenons la solution optimale de notre problème de production qui est $x = (x_1, x_2) = (3, 3)$ avec un rendement de 15 pour la fonction objectif.

Dans le cas que nous avons traité, il existait une solution optimale à notre problème et l'algorithme du simplexe nous l'a exhibée. Cependant ce n'est pas toujours le cas, en effet le problème peut aussi ne pas avoir de solution optimale. Deux cas sont à différencier, soit le problème est non bornée et une variable peut tendre vers l'infini sans être contraint (par exemple si les usines possèdent de l'énergie infinie on pourrait produire une infinité de produits sans perte), soit il n'existe pas de solution optimale et la région admissible est l'ensemble vide (par exemple si une contrainte nous oblige à produire une certaine quantité de produits mais que produire cette quantité dépasse le seuil d'énergie autorisée d'une usine).
Si après une itération du simplexe, il est possible de faire entrer en base une variable mais que les coefficients de la colonne pivot sont négatifs ou nuls cela implique que le problème est non bornée. Effectivement si les coefficients sont négatifs cela signifie qu'aucune variable en base ne peut sortir du fait qu'elle ne s'annule pas lorsque la variable entrante augmente.

Nous prendrons les notations suivantes pour définir le pseudo-code de la phase 2 de l'algorithme. Pour différencier les valeurs des variables mises à jour de leur valeurs d'origine, on choisit de mettre une barre sur

la variable. Soit un problème linéaire ayant n variables et m contraintes. $i \in \{1, m\}; j \in \{1, n\}$. On note $\bar{c}_j$ les coûts réduits associées à un tableau simplexe : $\bar{c}_4$ correspond au coût réduit dans le tableau de la variable 4. Les $\bar{a}_{ij}$ sont les coefficients dans la matrice des contraintes et $\bar{b}_i$ sont les seconds membres. x_s est la variable entrante et x_r est la variable sortante. s et r correspondante aux indices respectivement de la variable entrante et sortante.

tant que *il existe* $\bar{c}_j > 0$, $j \in$ *{1,n}* **faire**
  $s \leftarrow \text{argmax}(\bar{c}_j) \mid j \in \{1,\text{n}\}$
  si $\forall k \in \{1, m\}, \bar{a}_{ks} \leq 0$ **alors**
    STOP : le problème est non bornée
  sinon
    $r \leftarrow \text{argmin}\{ \frac{\bar{b}_k}{\bar{a}_{ks}} \mid \text{k} \in \{1,\text{m}\} \text{ et } \bar{a}_{ks} \neq 0 \}$
  fin
  Pivot(x_s,x_r)
fin

2.2.4 L'algorithme du simplexe : Phase 1

La phase 1 consiste à se placer sur un premier point admissible soit sur une première solution de base admissible. Pour savoir si oui ou non la phase 1 est à appliquer il faut pré-traiter le problème en multipliant par -1 toutes les contraintes ayant un second membre négatif. Multiplier une contrainte par -1 change simplement le sens de l'inégalité de la contrainte. Une fois que l'on a tous les seconds membres ≥ 0, il faut sélectionner les contraintes qui sont soit des égalités soit des inégalités supérieures car ce sont celles qui empêchent la solution triviale où les variables de décisions sont nulles d'être admissible. Pour résoudre le problème il nous faudra rajouter des variables artificielles pour ces contraintes en particulier et définir un problème auxiliaire qui une fois résolu nous fournira des valeurs pour les variables de décisions permettant de commencer la phase 2.
Soit un problème de minimisation ou maximisation arbitraire en forme standard.

$$\begin{array}{lll} Min/Max & c^T x & \\ \text{s.t.} & Ax & = \text{b} \\ & x & \geq 0 \end{array}$$

On le mettra sous forme standard mais pour chaque contrainte à l'égalité ou en supérieure ou égale nous leur rajouterons une variable artificielles v.

$$\begin{array}{lll} Min/Max & c^T x & \\ \text{s.t.} & Ax + v & = \text{b} \\ & x,\, v & \geq 0 \end{array}$$

Ce qui nous intéresse, c'est de récupérer une valeur des variables artificielles qui rend notre problème de départ démarrable avec la phase 2. Pour ce faire on change la fonction objectif telle qu'au lieu de vouloir minimiser/maximiser la fonction objectif on va vouloir minimiser les variables artificielles. C'est notre problème auxiliaire.

$$\begin{array}{lll} Min & \sum v_1 + v_2 + ... + vm & \\ \text{s.t.} & Ax + v & = \quad \text{b} \\ & x,\, v & \geq \quad 0 \end{array}$$

Maintenant nous résolvons ce problème auxiliaire telle que nous l'avions vu en faisant la phase 2 de l'algorithme du simplexe. En fin de phase 1, nous devrions avoir les variables artificielles hors base et nous pouvons prendre connaissance de la valeur des variables en base pour la phase 2. Et si le problème auxiliaire fournit une solution dont la valeur de la fonction objectif est 0 alors le problème d'origine a une solution admissible sur laquelle on pourra démarrer. Dans le cas contraire si tous les v_i ne sont pas nulles alors le problème est impossible. Pour conclure l'ensemble des étapes de la phase 1 :

1. Transformer le problème de sorte à avoir b ≥ 0
2. Standardiser le programme linéaire
3. Ajouter les variables artificielles
4. Résoudre le problème auxiliaire
5. Récupérer les variables de base
6. Déduire l'impossibilité du problème ou commencer la phase 2 (il existe un autre cas que nous verrons ci-après)

2.2.5 La dégénérescence

Lorsqu'une solution de base admissible a au moins une de ses variables de base nulle : c'est la dégénérescence, la solution est dite donc dégénérée. Lorsqu'intervient la dégénérescence cela implique plusieurs choses. Il est possible qu'en fin de phase 1 : il existe des variables artificielles en base.
Deux cas sont possibles, soit il est possible de faire sortir la/les variables artificielles de la base et de faire rentrer des variables initiales de notre problème d'origine. Dans ce cas nous savons le faire, il s'agit simplement de réaliser des pivots. Si cela est possible, nous réobtenons une base avec nos variables d'origine et pouvons commencer la phase 2. S'il n'est pas possible de faire rentrer des variables de notre problème d'origine en base (car pivot nul) cela signifie qu'il existe une relation de dépendance linéaire entre les contraintes. La contrainte qui correspond à la ligne possèdant la valeur artificielle en base peut être supprimée.

Traitons des exemples pour illustrer la dégénérescence.
Cas 1 :

On se donne le problème suivant :

$$\begin{array}{lll} \text{Max} & 2x_1 + x_2 & \\ \text{s.t.} & x_1 - x_2 & = 2 \\ & x_1 - 3x_2 & \geq 1 \\ & x_1 - x_2 & = 3 \\ & x_1, x_2 & \geq 0 \end{array}$$

Nous avons d'abord une phase 1 à exécuter, on obtient le problème auxiliaire suivant.

$$\begin{array}{llllllll} \text{Min} & v_1 + v_2 + v_3 & & & & & & \\ \text{Max} & 2x_1 + x_2 & & & & & & \\ \text{s.t.} & x_1 - x_2 & + v_1 & & & = & 2 \\ & x_1 - 3x_2 \text{ -}x_3 & & + v_1 & & = & 1 \\ & x_1 - x_2 & & & + v_1 & = & 3 \\ & x_1, x_2, x_3, v_1, v_2, v_3 & \geq & 0 & & & \end{array}$$

On a le tableau simplexe associé suivant :

$\bar{c}^a$	-3	6	1	0	0	0	-6
$\bar{c}$	2	1	0	0	0	0	0
v_1	1	-2	0	1	0	0	2
v_2	1	-3	-1	0	1	0	1
v_3	1	-1	0	0	0	1	3

Nous sommes en min, on fait donc rentrer x_1 et v_2 sort de la base. On obtient :

$\bar{c}^a$	0	-3	-2	0	3	0	-3
$\bar{c}$	0	7	2	0	-2	0	-2
v_1	0	1	1	1	-1	0	1
x_1	1	-3	-1	0	1	0	1
v_3	0	2	1	0	-1	1	2

x_2 entre en base.
On a bien un cas dégénéré puisque deux variables s'annulent en même temps, on choisira de respecter la règle de Bland en faisant sortir v_1.

$\bar{c}^a$	0	0	1	3	0	0	0
$\bar{c}$	0	0	-5	-7	5	0	-9
x_2	0	1	1	1	-1	0	1
x_1	1	0	2	3	-2	0	4
v_3	0	0	-1	-2	1	1	0

On se rend compte que le problème auxiliaire est résolu puisque les variables artificielles ont été annulées. Le problème étant que v_3 est encore en base. Pour autant il est possible de faire entrer x_3 en base (son coût réduit importe peu) et de faire sortir v_3. x_3 aura simplement une valeur nulle.

$\bar{c}^a$	0	0	0	1	1	1	0
$\bar{c}$	0	0	0	3	0	-5	-9
x_2	0	1	0	-1	0	0	1
x_1	1	0	0	-1	0	0	4
x_3	0	0	1	2	-1	-1	0

Au final, on a une solution dégénérée admissible sur laquelle on peut démarrer la phase 2. $(x_1, x_2 x_3) = (4, 1, 0)$

Cas 2 :

On se donne de résoudre le problème suivant :

$$\begin{array}{lllllll} \max & 6x_1 + & 5x_2 + & 7x_3 + & 4x4 & & \\ \text{s.t} & x_1 + & x_2 + & & & = & 20 \\ & & & x_3 & +x4 & = & 10 \\ & x_1 & & +x_3 & & = & 13 \\ & & x_2 & & +x_4 & = & 17 \\ & x_1, & x_2, & x_3, & x_4 & \geq & 0 \end{array}$$

Appliquons la phase 1 :

$$\begin{array}{lllllllllll} \max & 6x_1 + & 5x_2 + & 7x_3 + & 4x4 & & & & & & \\ \text{s.t} & x_1 + & x_2 + & & & +v_1 & & & & = & 20 \\ & & & x_3 & +x4 & & +v_2 & & & = & 10 \\ & x_1 & & +x_3 & & & & +v_3 & & = & 13 \\ & & x_2 & & +x_4 & & & & +v_4 & = & 17 \\ & x_1, & x_2, & x_3, & x_4 & \geq & 0 & & & & \end{array}$$

On a le tableau simplexe de phase 1 suivant :

$\bar{c}^a$	-2	-2	-2	-2	0	0	0	0	-60
$\bar{c}$	6	5	7	4	0	0	0	0	0
v_1	0	1	0	0	1	0	0	0	20
v_2	1	0	1	1	0	1	0	0	10
v_3	0	0	1	0	0	0	1	0	13
v_4	0	1	0	1	0	0	0	1	17

x_1 entre et v_3 sort

$\bar{c}^a$	0	-2	0	-2	0	0	2	0	-34
$\bar{c}$	0	5	1	4	0	0	-6	0	-78
v_1	0	1	-1	0	1	0	-1	0	7
v_2	0	1	-1	0	1	0	-1	0	10
x_1	1	0	1	0	0	0	1	0	13
v_4	0	1	0	1	0	0	0	1	17

x_2 entre et v_1 sort

$\bar{c}^a$	0	0	-2	-2	2	0	0	0	-20
$\bar{c}$	0	0	6	4	-5	0	-1	0	-113
x_2	0	1	-1	0	1	0	-1	0	7
v_2	0	0	1	1	0	1	0	0	10
x_1	1	0	1	0	0	0	1	0	13
v_4	0	0	1	1	-1	0	1	1	10

x_3 entre et v_2 sort

$\bar{c}^a$	0	0	0	0	2	2	0	0	0
$\bar{c}$	0	0	0	-2	-5	-6	-1	0	-173
x_2	0	1	0	1	1	1	-1	0	17
x_3	0	0	1	1	0	1	0	0	10
x_1	1	0	0	-1	0	-1	1	0	3
v_4	0	0	0	0	-1	-1	1	1	0

Comme nous l'avons dit la contrainte la présence de v_4 implique que la contrainte associée peut s'exprimer comme une combinaison linéaire des autres contraintes. On peut tout simplement retirer intégralement la contrainte du tableau et on se retrouve avec un tableau simplexe correcte pour commencer la phase 2.

Dans certains cas de dégénérescence cela est très rare mais on peut avoir un cyclage et revenir indéfiniement sur les mêmes points extrêmes déjà visités. Pour résoudre ce problème, on applique la règle de Bland en choisissant simplement la variable de plus petit indice parmi les variables candidates.

La dégénérescence posent aussi encore un autre problème lors de l'implémentation mais nous y reviendrons ultérieurement dans la partie concernée.

Source : Exemples tirés du cours de Master 1 ORO dispensé par Anthony Przybylski, Xavier Gandibleux

2.3 Programme dual

2.3.1 La dualité

Il est naturel de penser que différents programmes linéaires peuvent présenter une même solution optimale. Aussi on sait qu'en informatique il est possible de réduire un problème à un autre. La dualité en optimisation est une notion plus ou moins similaire, les problèmes d'optimisation peuvent adoptés plusieurs formes et être appréhendés sous plusieurs angles. En optimisation linéaire ou non, on peut définir un problème primal et son dual.
Obtenir une borne inférieure de la valeur optimale à un problème de maximisation quelconque est triviale puisque n'importe quelle solution admissible est une borne inférieure de la solution optimale d'un problème. Cependant exhiber une borne supérieure est plus intéressant car cela peut être une bonne indication de la qualité de la solution. Il existe notamment des bornes de qualités pour des problèmes spécifiques comme pour le Knapsack avec une heuristique etc. Rien de toutes ces méthodes n'est générique et c'est pourquoi l'importance de la dualité est capitale dans le domaine de l'optimisation continue (la dualité dans le cas discret est une autre affaire). En définissant un dual générique, l'idée est de pouvoir fournir une borne supérieure à notre problème
Pour un problème d'optimisation quelconque, on peut déterminer pour tout problème son dual. Le problème d'origine est qualifié lui de primal. Cependant le problème dual et primal ne sont pas réellement équivalents, il existe un écart entre les valeurs des solutions optimales de ces derniers. Cet écart s'appelle le saut de dualité mais il ne s'applique pas dans le cas l'optimisation convexe, fort heureusement dans le cas linéaire, les polyèdres construits sont convexes. Il n'existe donc pas de saut de dualité et l'écart entre nos solutions optimales primales et duales pour nos programmes linéaires sera inexistant. C'est ce qu'on appelle la dualité forte. Cette propriété nous servira par la suite, pour une de nos phases de la méthode du simplexe multicritère. La dualité faible quant à elle ne nous sera pas utile, elle établit une relation de grandeur entre les valeurs des solutions admissibles du primal et dual.

2.3.2 Construire le dual d'un PL

On appelle le primal le problème de départ. Pour construire le dual d'un programme linéaire, il faut revoir la composition d'un programme linéaire.
Dans un PL on a : une fonction objectif, des variables, des contraintes, un domaine sur nos variables. Le dual du primal respecte les propriétés suivantes :

1. Si un problème primal est en minimisation son problème dual est maximisation et inversement
2. A toute contrainte primale est associée une variable duale
3. A toute variable primale est associée une contrainte duale
4. Les coûts réduits du dual correspondent aux seconds membres du primal (c'est la raison pour laquelle les seconds membres sont parfois appelées coûts duaux ou coûts ombres)
5. Les seconds membres du dual correspondent aux coûts réduit du primal
6. La matrice de contraintes du dual correspond à la transposé de la matrice des contraintes du primal
7. Les inégalités ou égalités des contraintes du dual sont données par le domaine des variables du primal. De même inversement : le domaine des variables du dual sont données par les égalités ou inégalités des contraintes du primal :

Pour éviter les ambiguïtés entre dual et primal, la fonction objectif du dual est noté w, et les variables dans le dual sont nommées u et non x. Pour le reste les notations sont inchangées.

Tableaux récapitulatifs pour le point 7

primal en min	dual en max
contrainte $\geq b_i$	variable $u_i \geq 0$
contrainte $\leq b_i$	variable $u_i \leq 0$
contrainte $= b_i$	variable u_i libre
variable $x_j \geq 0$	contrainte $\leq c_j$
variable $x_j \leq 0$	contrainte $\geq c_j$
variable x_j libre	contrainte $= c_j$

primal en max	dual en min
contrainte $\geq b_i$	var $u_i \leq 0$
contrainte $\leq b_i$	var $u_i \geq 0$
contrainte $= b_i$	var u_i libre
var $x_j \geq 0$	contrainte $\geq c_j$
var $x_j \leq 0$	contrainte $\leq c_j$
var x_j libre	contrainte $= c_j$

Exemple illustratif
Reprenons encore le cas de production de produits vu en première partie.

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 & & \\ \text{s.t.} & 3x_1 + x_2 & \leq & 12 \\ & 2x_2 + 2x_2 & \leq & 12 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Déterminons son dual :
Par 1 : Le primal est en max donc le dual est en min.
Par 2.3 : Le dual possède 2 contraintes et 2 variables.
Par 4 : Les seconds membres du primal sont 12 et 12 donc la fonction objectif du dual sera $w(u) = 12u_1 + 12u_2$
Par 5 : Les coûts réduits du primal sont 3 et 2 donc les seconds membres du dual seront $b = (3, 2)$
Par 6 : On fait simplement la transposée de A.
Par 7 : comme $x_1, x_2 \geq 0$ alors les 2 contraintes du dual sont des $\geq$
Par 7 : Les contraintes du primal sont des $\leq$ alors on a les variables du dual $u_1, u_2 \geq 0$

En résultat nous construisons le dual suivant :

$$\begin{array}{lrcl} \text{Min} & 12u_1 + 12u_2 & & \\ \text{s.t.} & 3u_1 + 2u_2 & \geq & 3 \\ & u_2 + 2u_2 & \geq & 2 \\ & u_1, u_2 & \geq & 0 \end{array}$$

Exemple complémentaire :

Primal

$$\begin{array}{lllll} \text{Max} & 3x_1 + & 2x_2 + & 5x_3 & \\ \text{s.t.} & x_1 + & x_2 + & -3x_3 & = \ 2 \\ & -x_1 + & 0 + & 2x_3 & \geq \ 7 \\ & 0 & -x_2 + & 3x_3 & \leq \ 1 \\ & 5x_1 + & x_2 + & x_3 & \geq \ 3 \\ & x_1 \geq 0 & x_2 \leq 0 & x_3 \in \mathbb{R} & \end{array}$$

Dual

$$\begin{array}{lllll} \text{Min} & 2u_1 + & 7u_2 + & u_3 & + 3u_4 \\ & u_1 & -u_2 + & 0 & + 5u_4 \ \geq \ 3 \\ & 0 + & u_2 & -u_3 & + u_4 \ \leq \ 2 \\ & -3u_1 + & 2u_2 & 3u_3 & + u_4 \ = \ 5 \\ & u_1 \in \mathbb{R} & u_2 \leq 0 & u_3 \geq 0 & + u_4 \leq 0 \end{array}$$

Plus génériquement :

Primal PL_1

$$\begin{array}{lll} \text{Max} & c^T x & \\ \text{s.t.} & Ax & \leq \ \text{b} \\ & x & \geq \ 0 \end{array}$$

Dual PL_1

$$\begin{array}{lll} \text{Min} & b^T u & \\ \text{s.t.} & A^T u & \geq \ \text{c} \\ & u & \geq \ 0 \end{array}$$

Primal PL_2

$$\begin{array}{lll} \text{Max} & c^T x & \\ \text{s.t.} & Ax & = \ \text{b} \\ & x & \geq \ 0 \end{array}$$

Dual PL_2

$$\begin{array}{lll} \text{Min} & b^T u & \\ \text{s.t.} & A^T u & \geq \ \text{c} \end{array}$$

2.3.3 Propriété importante de la dualité

Le dual du dual est le primal.

Dualité faible :
Soit x une solution réalisable du primal (en min) et u une solution réalisable du duale (et inversement) alors on a :
$b^T u \leq c^T x$

Si le primal est non bornée alors le dual est infaisable et inversement.

Il est possible que le primal et le dual soit tous les deux infaisables.

Dualité forte :
Si le primal et le dual ont tous les deux un ensemble de solutions réalisables non vide alors :
$\min c^T x = \max b^T u$ x et u sont des solutions réalisables.
Aussi si u et x sont des solutions optimales alors
$b^T u = c^T x$

3 Forme révisée du simplexe

L'algorithme du simplexe demande de calculer une suite de tableaux simplexe. En théorie cela ne pose aucun problème mais dans la pratique ces calculs peuvent générer des imprécisions numériques qui se propagent de tableau en tableau. Et si le problème est assez long à résoudre, on peut finir avec un tableau complètement erroné. C'est pourquoi nous nous intéressons à la forme révisée du simplexe.

La forme révisée du simplexe permet de calculer un tableau simplexe dans n'importe quelle base à partir des données d'origine du problème. Tous les calculs réalisés sont faits de manière algébrique et matriciels. Le tableau simplexe est une méthode mécanique et ne prend pas en compte le caractère creux des matrices. Or sur des problèmes de grandes tailles la matrice des contraintes étant creuse, on en retire une rapidité dans les multiplications de matrice et la stabilité numérique.

3.1 Point de vue algébrique

Avant d'attaquer la forme révisée, il est pertinent de voir comment appréhender un programme linéaire de manière algébrique. En effet l'ensemble d'un PL peut être résumé sous forme de vecteur et de matrices et il donc possible de résoudre le problème en faisant uniquement des calculs matriciels. Si on reprend l'exemple des usines on se souvient qu'on avait la solution optimale avec le tableau simplexe qui suit :

c	0	0	$-\frac{1}{2}$	$-\frac{3}{4}$	-15
x_1	1	0	$\frac{1}{2}$	$-\frac{1}{4}$	3
x_2	0	1	$-\frac{1}{2}$	$\frac{3}{4}$	3

On peut retrouver facilement le tableau final sans faire de pivot.
Si on considère le problème d'origine sous forme standard, notons :
_ x_B la valeur des variables en base.
_ x_N la valeur des variables hors base.
_c_B la valeur des coûts réduits en base.
_ c_N la valeur des coûts réduits hors base.
_ A_N la matrice dans les contraintes associée au variables hors base
_ A_B la matrice dans les contraintes associée au variables en base

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 + 0x_3 + 0x_4 & & \\ \text{s.t.} & 3x_1 + x_2 + x_3 & = & 12 \\ & 2x_1 + 2x_2 + x_4 & = & 12 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

On a par exemple en reprenant les notations ci dessus

$$A = \begin{pmatrix} 3 & 1 & 1 & 0 \\ 2 & 2 & 0 & 1 \end{pmatrix}$$

x_B = (12,12) x_N = (0,0) c_B = (0,0) c_N = (3,2)

$$A_B =$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$A_N =$$

$$\begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$$

Ce qui nous intéresse c'est de trouver la valeur de x_B et de z sans faire de pivot.
On sait que :

$Ax = b$ et $A = (A_B, A_N)$ aussi $x = ((x_B)^T, (x_N)^T)^T$ on en déduit
$(A_B x_B) + (A_N x_N) = b$
$(A_B x_B) = b - (A_N x_N)$
or x_N est nulle car hors base donc
$(A_B x_B) = b$
$x_B = A_B^{-1} b$

Si on reprend dans l'exemple on veut la solution associée à la base (1,2). x_1 et x_2 en base.

$$\text{On a } A_B = \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$$

$$A_B^{-1} = \text{ on retrouve les coefficients du tableau simplexe final } \begin{pmatrix} \frac{1}{2} & -\frac{1}{4} \\ -\frac{1}{2} & \frac{3}{4} \end{pmatrix}$$

et

$$A_B^{-1}b = \begin{pmatrix} \frac{1}{2} & -\frac{1}{4} \\ -\frac{1}{2} & \frac{3}{4} \end{pmatrix} \begin{pmatrix} 12 \\ 12 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \tag{1}$$

Au final on a la valeur de nos variables en base

$$(x_1, x_2) = x_B = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \tag{2}$$

Mais comme nous avons la valeur des variables en base on peut en déduire la valeur de z :

$$z = c_B^T x_B$$
$$z = c_B^T A_B^{-1} b$$

$$z = c_B^T A_B^{-1} b = (3, 2) \begin{pmatrix} 3 \\ 3 \end{pmatrix} = 15 \tag{3}$$

En conclusion on peut d'un point de vue algébrique, obtenir pour toute base la solution associée. Cependant en pratique c'est fastidieux car on calcule l'inverse d'une matrice creuse qui a de ce fait de forte chance d'être dense. En terme de complexité cela peut être coûteux. De plus on ne fait que calculer des solutions on ne trouve pas la ou les solutions optimales.

3.2 Forme révisée et décomposition LU

La forme révisée reprend là le point de vue algébrique s'arrête et tente en plus d'intégrer la mécanique du simplexe à savoir faire déterminer les variables entrantes et sortantes et trouver la solution optimale. Cependant tout ces calculs se font sans structure de tableau mais bien de manière matricielle et dans un tableau simplexe tout le tableau est mise à jour en forme révisée seules les informations qui nous sont utiles sont pris en compte.

3.2.1 Comment trouver la variable entrante :

Nous avons vu que l'on pouvait pratiquement retrouver toutes les informations du tableau simplexe en faisant des calculs matriciels.
Trouver la variable sortante consiste à retrouver les coûts réduits des variables hors base qui sous forme algébrique est donnée par :

$\bar{c}_N^T = c_N^T - c_B^T A_B^{-1} A_N$ On peut se débarasser du calcul de la matrice inverse en posant :
$y^T = c_B^T A_B^{-1}$ on trouve que $c_B^T = y^T A_B$
En réinjectant c_B^T dans $\bar{c}_N^T$:
$\bar{c}_N^T = c_N^T - y^T A_B A_B^{-1} A_N$
$\bar{c}_N^T = c_N^T - y^T A_N$

On considère la seconde itération de la résolution du PL suivant :

$$\begin{array}{lrcl} \text{Max} & 3x_1 + 2x_2 + 0x_3 + 0x_4 & & \\ \text{s.t.} & 3x_1 + x_2 + x_3 & = & 12 \\ & 2x_1 + 2x_2 + x_4 & = & 12 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

On a B = $\{x_1, x_4\}$, x_B = (4,4) , N=$\{x_2, x_3\}$

$$A_B = \begin{pmatrix} 3 & 0 \\ 2 & 1 \end{pmatrix} \tag{4}$$

Trouvons le vecteur y^T,
$y^T A_B = c_B^T$

$$(y_1, y_2) \begin{pmatrix} 3 & 0 \\ 2 & 1 \end{pmatrix} = (3, 0) \tag{5}$$

On trouve donc $y^T = (1, 0)$, substituons sa valeur dans $\bar{c}_N^T = c_N^T - y^T A_N$

$$(\bar{c}_2, \bar{c}_3) = (2, 0) - (1, 0) \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix} = (1, -1) \tag{6}$$

Une fois qu'on a les coûts réduits on peut choisir la variable entrante.

3.2.2 Comment trouver la variable sortante :

Notons a la colonne dans A de la variable entrante. Précédemment si on choisit la variable x_2 comme entrante alors a = $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ Il faudrait pouvoir calculer la première variable qui s'annule lorsque la variable entrante augmente. Pour cela il faut calculer la colonne de la variable entrante. Notons cette colonne d, elle est égale à : $d = A_B^{-1} a \implies A_B d = a$. En résolvant ce système on trouve la valeur de d. Une fois la colonne d trouvée, on peut diviser d par la valeur x_B des variables en base pour déterminer la variable sortante.

Si on reprend l'exemple là où on l'avait laissé avec B = $\{x_1, x_4\}$, x_B = (4,4) , N=$\{x_2, x_3\}$.
Faisons rentrer x_2 :
$a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ $A_B = \begin{pmatrix} 3 & 0 \\ 2 & 1 \end{pmatrix}$

$$A_B d = a = \begin{pmatrix} 3 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{7}$$

On trouve donc la colonne d= $\begin{pmatrix} \frac{1}{3} \\ \frac{4}{3} \end{pmatrix}$ Puis on peut diviser la valeur des variables (4,4) par d on a

$$Min\{4 \div \tfrac{1}{3}, 4 \div \tfrac{4}{3}\}$$

$$Min\{12, 3\}$$

$Min\{12, 3\} = 3$ Cela correspond à la variable x_4, donc x_2 rentre et x_4 sort.

On a pu maintenant trouver la variable sortante et entrante, on a simplement à mettre à jour la base avec la solution associée.

3.2.3 Décomposition LU :

La décomposition LU permet d'accélérer les calculs matriciels du simplexe révisé. En effet en décomposant en LU la matrice A_B, on simplifie les systèmes linéaires à résoudre, ceux ci deviennent quasiment immédiat à résoudre. La matrice A_B est décomposée en deux matrices L et U qui sont respectivement des matrices triangulaires inférieure et triangulaires supérieure. Le principe reste quasiment le même mais plutôt que de résoudre $A_B d = a$ on aura $LUd = a$.
Pour retrouver la valeur de d il faudra donc d'abord résoudre un système triangulaire supérieure intermédiaire $Ly = a$ (en ayant posé bien entendu $y = Ud$).
Puis on résout le système triangulaire inférieure : $Ud = y$

La décomposition LU est une méthode qui permet d'avoir une implémentation moins encline à générer des imprécisions numériques. Elle est très utilisée dans les solveurs et a de nombreuses applications et c'est d'ailleurs pourquoi elle est présente dans de nombreuses bibliothèque mathématiques pour les langages de haut niveau.

4 Optimisation multiobjectif

4.1 Introduction à l'optimisation multiobjectif

Il arrive bien souvent que l'on souhaite optimiser un problème en prenant en compte un ou plusieurs autres objectifs. Dans le cas mono-objectif cela ne pose pas de problème vu que l'ensemble des solutions est un ensemble muni d'un ordre total. En effet il est possible de comparer de manière cohérente chacune des solutions entre elles. Dans le cas à plusieurs objectifs cela n'est plus possible. Certaines solutions ne sont pas comparables entre elles.
Par exemple on pourrait vouloir acheter un ordinateur en considérant sa durée de vie notons le z_1 et sa puissance z_2. Supposons un ensemble d'ordinateurs (z_1, z_2) :
$\{(1,1);(2,3);(3,2);(7,2)\}$
Les solutions (2,3) et (7,2) sont incomparables et impossible de les départager sans faire de supposition sur les objectifs. Revenons sur la notation, en optimisation multiobjectif linéaire, nous considèrons les MOLP sous cette forme ci :

$$\begin{array}{lll} \text{Min} & C^T x & \\ \text{s.t.} & Ax & = \text{b} \\ & x & \geq 0 \end{array}$$

C est une matrice $M_{p\times n}$, c'est à dire qu'il y a p objectif et n variables. Comme l'on note x une solution, on note y l'image de x par f. f étant une application linéaire qui donne la valeur des objectifs. $f_i(x)$ correspond à la valeur sur l'objectif i de la solution x. Si x est une solution réalisable alors son image par f y est appelé un point réalisable dans l'espace des objectifs. Comme nous sommes dans le cas linéaire, nous pouvons nous permettre d'utiliser les exposants en tant qu'indice. x^5 correspond à une solution et y^5 à son vecteur associée dans l'espace objectif.
Et notamment $y^i = (z_1(x^i), z_2(x^i), ..., z_p(x^i)) = (y_1^i, y_2^i, ..., y_p^i)$

Avant de commencer toute autre réflexion, il serait pertinent de reconsidérer au préalable l'ensemble des points réalisables. En effet en programmation linéaire, les régions construites et étudiées jusqu'ici ont la particularité d'être des polytopes ou des polyèdres. Si ce n'était plus le cas dans l'espace objectif nombreux théorèmes et principes tomberait à l'eau. Commençant donc par prouver le caractère polytopique de l'ensemble des points réalisables.

Un polytope correspond à l'enveloppe convexe d'un nombre fini de points dans un espace de dimension quelconque. Ainsi posons $d \in \mathbb{R}^n$ (d pour décision) l'ensemble des solutions de base réalisable. d correspond aux sommets du polytope. Le polytope P se décrit donc comme suit :

$$P = \{d \in \mathbb{R}^n | d = \textstyle\sum_i \lambda_i d_i, \sum_i \lambda_i = 1\}$$

Soit x une solution réalisable appartenant à P, on a : $x \in P$. Appliquons l'application linéaire z à x :

$$z(x) = z(\textstyle\sum_i \lambda_i x_i)$$

Par linéarité de z :
$$z(x) = \textstyle\sum_i \lambda_i z(x_i)$$

Si on pose $z(x_i) = y_i$, $z(X) = Y$ et alors on a :
$Y = \sum_i \lambda_i y_i$ et on a toujours nécessairement $\sum_i \lambda_i = 1$

Or on retombe simplement sur la définition d'un polytope de ce fait l'image Y de X par z est aussi un polytope.

Montrons que c'est aussi le cas pour les polyèdres. Décrivons un polyèdre comme suit :

$$L = \{l \in \mathbb{R}^n | l = \sum_i \lambda_i l_i + c_j \vec{v_j}, \sum_i \lambda_i = 1\}$$

Un polyèdre étant un polytope non borné, on a $\vec{v_j}$ la direction vers laquelle la solution peut croître infiniement et c_j le coefficient entre $[0, +\infty]$

Soit x une solution réalisable appartenant à L, on a : $x \in L$. Appliquons l'application linéaire z à x :

$$z(x) = z(\sum_i \lambda_i x_i + u_j \vec{v_j})$$

Par linéarité de z :
$$z(x) = \sum_i \lambda_i z(x_i) + \sum_i z(u_j \vec{v_j})$$

Si on pose $z(x_i) = y_i$, $z(X) = Y$ et $z(c_j \vec{v_j}) = a_i \vec{u_i}$ alors on a :
$Y = \sum_i \lambda_i y_i + a_i \vec{u_i}$ et on a toujours nécessairement $\sum_i \lambda_i = 1$

Or on retombe comme attendu sur la définition d'un polyèdre avec $a_i \vec{u_i}$ des nouvelles directions pour le polyèdre construit.

Nous avons donc bien montré que l'image d'un polytope ou d'un polyèdre par une application linéaire est un polytope ou un polyèdre.

Maintenant nous pouvons aussi se demander comment départager les solutions et comment sélectionner celles qui sont convaincantes ? Tout ceci nous implique de revoir la notion d'optimalité en plus de définir les notations.

4.2 Dominance et efficacité

La notion de dominance est celle qui nous permettra de déterminer si une solution est meilleure qu'une autre. Toutes les notations explicitées seront reprises par la suite.

Résoudre un problème d'optimisation multiobjectif consiste donc à récupérer l'ensemble des solutions non dominées. Ces solutions sont qualifiées de solutions efficaces et à toute solution efficace correspond sa base efficace. On ne parlera plus donc de solution optimale mais d'un ensemble de solutions efficaces et de points qui seront non-dominées. La notion de dominance est explicité juste ci-dessous.
L'ensemble des solutions efficaces est notée : X_E et l'ensemble des points non dominé est notée : Y_N.
Dans le cas de minimisation :
Dominance stricte :
Si on a $y_k^1 < y_k^2 \mid \forall k \in \{1, ..., p\}$ alors on dit que y^1 domine strictement y^2 et on note $y^1 < y^2$

Dominance faible :
Si on a $y_k^1 \leq y_k^2 \mid \forall k \in \{1, ..., p\}$ alors on dit que y^1 domine faiblement y^2 et on note $y^1 \leqq y^2$

Dominance :
Si on a $y_k^1 \leqq y_k^2 \mid \forall k \in \{1, ..., p\}$ et $y^1 \neq y^2$ alors on dit que y^1 domine y^2 et on note $y^1 \leq y^2$

Soit $\hat{x}$ une solution réalisable de notre MOLP et posons $\hat{y} = C\hat{x}$.
Faiblement efficace :
Si $\nexists$ x $\in X | Cx < C\hat{x}$ alors on dit que :
$\hat{x}$ est dit faiblement efficace
$\hat{y}$ est dit faiblement non dominée ($\hat{y} = C\hat{x}$)

Efficace :
Si $\nexists$ x $\in X | Cx \leq C\hat{x}$ alors on dit que :
$\hat{x}$ est dit efficace
$\hat{y}$ est dit non dominée ($\hat{y} = C\hat{x}$)

variables efficaces : Soit une base efficace B, une variable hors base est dite efficace si en entrant dans la base B alors la nouvelle base obtenue est efficace.

Notations complémentaires :
$\mathbb{R}^P_> = \{y \in \mathbb{R}^p : y > 0\}$
$\mathbb{R}^P_\geq = \{y \in \mathbb{R}^p : y \geq 0\}$
$\mathbb{R}^P_\geqq = \{y \in \mathbb{R}^p : y \geqq 0\}$

4.3 Différentes approches

On peut vouloir transformer un MOLP en un LP. En scalairisant les objectifs : cela implique d'attribuer un poids aux fonctions objectifs. Généralement on note ce vecteur de λ. Pour un MOLP en $min\{Cx : Ax \geq b; x \geq 0\}$ on peut par exemple définir une scalairisation : $min\{\lambda^T Cx : Ax \geq b; x \geq 0\}$. Le problème de cette approche avec une somme pondérée est qu'il existe une infinité de possibilités de fixer λ. Soit il est possible grâce à un expert métier de définir un poids sur nos objectifs mais rien ne peut à priori nous permettre de fixer de tels poids sans connaissances spécifiques du problème en question.

La paramétrisation des objectifs est à oublier dans le cas où il y a plus de 2 objectifs mais cette notion est importante car elle nous permettra néanmoins de récupérer des solutions efficaces, nous y reviendrons ultérieurement.

On peut aussi définir un ordre lexicographique sur les objectifs et ainsi maximiser ou minimiser en priorité les objectifs prioritaires. Le problème de cette approche est comme pour la scalairisation. On ne peut définir un ordre lexicographique de manière rigoureuse sans faire de supposition sur le problème.

4.4 Les propriétés importantes en optimisation linéaire multiobjectif :

Lemme : l'efficacité d'une solution :
Une solution réalisable x^0 est efficace si et seulement si le PL suivant a une solution optimale $(\hat{x}, \hat{z})$ avec $\hat{z} = 0$:
Ce PL s'appelle le primal de Benson :

$$\begin{array}{lrcl} \text{Min} & e^T z & & \\ \text{s.t.} & Ax & = & \text{b} \\ & Cx + Iz & = & Cx^0 \\ & x, z & \geqq & 0 \end{array}$$

e^T correspond à la scalairisation des objectifs où tous les objectifs sont considérés de même importance et de même poids 1. $e^T = (1, ..., 1)$
I est une matrice identité de taile $p \times p$. z et x correspondent aux variables de décisions.

Preuve du lemme : l'efficacité d'une solution :
Soit $(x, z)|x \in X$ et $z \in \mathbb{R}^P_\geqq$ une solution réalisable. Alors $Cx + Iz = Cx^0$ et donc $z = Cx^0 - Cx \geqq 0$ grâce à la non-négativité de z. Si $(\hat{x}, \hat{z})$ une solution efficace alors il n'y a pas de $x \in X$ telle que $Cx \leqq C\hat{x}$, nécessairement $\hat{z} = 0$. D'autre part si $\hat{x}$ n'est pas efficace il doit y avoir un $x \in X$ telle que $Cx \leq Cx^0$. Mais alors cela implique qu'il existe un z telle que pour au moins un k $z_k > 0$ ce qui contredit l'optimalité de $(\hat{x}, 0)$.

Autre formulation du lemme :
Une solution réalisable x^0 est efficace si et seulement si le PL suivant a une solution optimale $u^T b + w^T Cx^0$.

$$\begin{array}{lrcl} \text{Max} & u^T b + w^T Cx^0 & & \\ \text{s.t.} & u^T A + w^T C & \geqq & 0 \\ & w & \geqq & \text{e} \end{array}$$

Ce PL correspond au dual du précédent et s'appelle conséquemment le dual de Benson, l'utilité est qu'ici on récupère le vecteur w qui correspond à une scalairisation sur nos objectifs. On s'en servira ultérieurement dans la méthode du simplexe multicritère.

Preuve du lemme :
Comme le PL correspond au dual du précédent, on sait par la dualité forte que si le primal est faisable et

possède une solution optimale alors celle ci correspondra à la solution optimale du dual. Nécessairement $e^T\hat{z} = \hat{u}^T b + \hat{w}^T C x^0 = 0$.

L'adjacence des bases efficaces :
A une solution efficace est associée bien entendu une base efficace. Comme nous l'avons évoqué, trouver les bases efficaces est l'objectif en MOLP. Cependant quelle garantie avons nous de pouvoir toutes les énumérer ? En effet rappelons nous que pour un programme linéaire, on se déplace de point extrême en point extrême et on en retire l'optimalité. En MOLP rien ne garantit à priori que les bases efficaces soient connectés mais c'est heureusement le cas. $Y_N = z(X_E)$ est un polytope ou un polyèdre comme nous l'avons déjà montré auparavant. Donc si x est une solution de base admissible alors son image par z est aussi point réalisable non dominé dans l'espace objectif.
Le fait que les bases efficaces soient connectés est le point essentiel qui permet le fonctionnement du simplexe multicritère. Comme pour le simplexe normal, le simplexe multicritère se déplacera de bases efficaces en bases efficaces. Une fois toutes énumérées nous aurons naturellement résolu notre MOLP.

Théorème : efficacité et somme de poids positifs :
Une solution optimale d'une scalairisation d'un MOLP est une solution efficace.

Théorème (Isermann (1973)) :
Soit le MOLP : Min $\{Cx\ ,\ Ax = b,\ \mathrm{x} \geqq 0\}$
Une solution réalisable $x^0 \in X$ de ce MOLP est une solution efficace si et seulement $\forall x \in X$ il existe un $\lambda \in \mathbb{R}^P_>$:

$$\lambda^T C x^0 \leq \lambda^T C x$$

Preuve : Faisons la preuve dans les deux sens.

($\Longleftarrow$) Le théorème précédent rend triviale la preuve dans ce sens. En effet on a bien quelque soit $x \in X$, $\lambda^T C x^0 \leq \lambda^T C x$ cela correspond à la définition même de l'efficacité d'une solution et x^0 est une solution réalisable efficace.

($\Longrightarrow$) En partant du dual de benson, on sait que celui a une solution optimale $(\hat{u}, \hat{w})$ telle que $\hat{u}^T b = -\hat{w}^T C x^0$. Ce même $\hat{u}$ est aussi la solution optimale du LP1 : max $\{u^T b : u^T A \geqq -\hat{w}^T C\}$ avec $w = \hat{w}$ fixé. Donc il existe aussi une solution optimale pour le dual du LP1, notons le LP2 : max $\{-\hat{w}^T C x : Ax = b, x \geqq 0\}$
Par la dualité faible on sait que toute solution réalisable de LP1 est une solution réalisable de LP2 et en plus on sait par le dual de benson que $\hat{u}^T b = -\hat{w}^T C x^0$. On en déduit que x^0 est une solution optimale de LP2.
On peut constater au final que LP2 est équivalent à min $\{\hat{w}^T C x : Ax = b, x \geqq 0\}$.
Donc x^0 est une solution optimale de notre scalairisation avec $\lambda = \hat{w}$ comme vecteur de poids.

Ces propriétés sont importantes car elles justifient de façon rigoureuse la mécanique du simplexe multicritère.

Source : Les preuves sont extraites du livre "Multicriteria Optimization" de Matthias Ehrgott

4.5 Principe

L'algorithme du simplexe multicritère se déroule en 3 phases distinctes.

La **phase 1** détermine si oui ou non le MOLP admet un ensemble de solutions réalisables non vide. Si X est non vide on aimerait trouver une première base réalisable $x^0 \in X$ pour commencer la phase 2.

En **phase 2,** il y a deux étapes, la première est de savoir si l'ensemble des solutions efficaces n'est pas un ensemble vide. Dans ce cas $X_E = \{\varnothing\}$ et donc il n'y a pas à continuer puisqu'il n'y a pas de bases efficaces et que donc le MOLP n'a pas de solutions satisfaisantes. Si $X_E \neq \{\varnothing\}$ alors dans un deuxième temps on aimerait exhiber une première base efficace B_0

La **phase 3** de l'algorithme consiste donc à énumérer l'ensemble des bases efficaces en se déplaçant de bases efficaces en efficaces. Elle part de la première base efficace B_0 trouvée en phase 2.

On récupère l'ensemble des bases efficaces et leurs solutions assiciées à la fin de l'algorithme.

4.6 Phase 1

On aimerait exhiber une solution triviale pour savoir si oui ou non le MOLP est impossible. Pour cela on peut annuler la fonction objectif et simplement trouver une solution x qui respecte les contraintes du MOLP.

Algorithme 1 : Phase 1 : Problème impossible ?

```
x_0 ← Solution optimale du LP1 : min{0 : Ax = b; x ≥ 0}
si LP1 est impossible alors
    Le MOLP est impossible
sinon
    retourner x_0
fin
```

4.7 Phase 2

En résolvant le $LP2_1$ suivant nous avons une scalairisation des objectifs qui nous permet de déterminer si oui ou non l'ensemble $X_E = \{\varnothing\}$

$$max\{u^T b + w^T C X_0 : u^T A^T + w^T C^T \geq 0; w \geq 1\}$$

Ce $LP2_1$ a déjà été évoqué dans les propriétés importantes.

Une fois les poids trouvés, on souhaite récupérer la première base efficace. Pour cela, on fait une résolution de la somme pondérée du $LP2_2$ qui suit :

$$min\{w^T C x : Ax \geq b; x \geq 0\}$$

Algorithme 2 : Phase 2 : Ensemble efficace et première base efficace

```
w ← Poids calculés sur LP2_1 : min{u^T b + w^T C X_0 : u^T A^T + w^T C^T ≥ 0; w ≥ 1}
si le LP2_1 est impossible alors
    Le MOLP est non-borné et donc X_E = {∅}
sinon
    B_1 ← Première base efficace donnée par LP2_2 : min{w^T C x : Ax ≥ b; x ≥ 0}
    L[1] ← B_1 : On met la première base dans une liste
    retourner Liste de base L
fin
```

4.8 Phase 3

4.8.1 Un problème auxiliaire

On a maintenant une première base efficace. On aimerait savoir quelle variable ajouter et laquelle retirer pour avoir une autre base efficace.
Pour savoir si il serait intéressant selon les objectifs de faire entrer une variable x_j en base, on construit un problème auxiliaire avec les fonction objectifs de la base courante. On met les valeurs des objectifs des variables hors base dans une matrice R. On note r^j la $j$ème colonne de la matrice R.

On construit alors les contraintes et la fonction objectif du programme auxiliaire comme suit :

- → $R - r^j$: la concaténation de R et de moins la colonne r_j (la colonne de la variable que l'on veut faire entrer dans la base).
- → c_{aux} : Le vecteur dont chaque valeur est égal à la somme d'une colonne de la matrice $R - r^j$.

On note le problème auxiliaire comme suit :

$$min\{c_{aux}^T (y + \delta) : Ry - r^j \delta \geq 0; y, \delta \geq 0\}$$

Si la résolution de ce problème est bornée, alors introduire la variables x_j dans la base mènera vers une base

efficace. C'est surtout le caractère bornée du problème qui nous intéresse. Cette partie sera plus explicitée en détail dans l'exemple d'application du simplexe multicritère.

Algorithme 3 : Phase 3 énumération des bases efficaces

```
1  i ← 1
2  tant que i ≤ |L| faire
3  |  B ← L[i]
4  |  R ← Matrice objectifs dans la base B calculée par la forme révisée LU
5  |  pour chaque variable entrante x_j faire
6  |  |  On résout problème auxiliaire : min{c_aux^T (y + δ) : Ry − r^j δ ≥ 0; y, δ ≤ 0}
7  |  |  si La variable x_j est efficace alors
8  |  |  |  x_s ← variable sortante calculée avec la forme révisée LU
9  |  |  |  si une variable peut sortir alors
10 |  |  |  |  B_j ← B_/x_s ∪ {x_j}
11 |  |  |  |  si B_j ∉ L alors
12 |  |  |  |  |  L ← On ajoute B_s au bout de la liste de base
13 |  |  |  |  fin
14 |  |  |  fin
15 |  |  fin
16 |  |  i ← i + 1
17 |  fin
18 |  retourner L
19 fin
```

4.9 Exemple d'application du simplexe multicritère :

Considérons le MOLP suivant :

$$\begin{array}{llllll}
\min & -x_1 & -2x_2 & & & \\
\min & -x_1 & & +2x_3 & & \\
\min & x_1 & & -x_3 & & \\
\text{s.c} & x_1 & +x_2 & & \leq & 1 \\
 & & +x_2 & & \leq & 2 \\
 & x_1 & -x_2 & +x_3 & \leq & 4
\end{array}$$

On introduira les variables d'écart x_4, x_5, x_6 pour obtenir notre MOLP sous forme standard $Ax = b$.

Phase 1 : Le MOLP possède t-il une solution admissible ?
Oui, clairement la solution triviale $(x_1, x_2, x_3) = (0, 0, 0)$ est admissible.
Notons $x^0 = (0, 0, 0, 1, 2, 4)$ une solution de base admissible et sa base $B = \{4, 5, 6\}$.

Phase 2 : L'ensemble des solutions efficace est-il vide ?
Résolvons le dual de benson associé au MOLP.

$$min\{u^T b + w^T C x_0 : u^T A^T + w^T C^T \geq 0; w \geq 1\}$$

$$\begin{array}{llllllll}
\min & u_1 & + 2u_2 & +4u_3 & & & & \\
\text{s.c} & u_1 & & +u_3 & -w_1 & -w_2 & +w_3 & \geq 0 \\
 & u_1 & +u_2 & -u_3 & -2w_1 & & & \geq 0 \\
 & & & u_3 & & +2w_2 & -w_3 & \geq 0 \\
 & u_1, & u_2, & u_3 & & & & \geq 0 \\
 & & & & w_1, & w_2, & w_3 & \geq 1
\end{array}$$

La solution du dual de benson nous donne une solution optimale avec des poids $w = (1, 1, 1)$.

Résolvons maintenant en reprenant la valeur de w le LP qui nous permettra de trouver la première base efficace.

$$min\{w^T C x : Ax \geq b; x \geq 0\}$$

$$\begin{array}{llllll} \min & -x_1 & \text{-}2x_2 & +x_3 & & \\ \text{s.c} & x_1 & +x_2 & & \leq & 1 \\ & & +x_2 & & \leq & 2 \\ & x_1 & \text{-}x_2 & +x_3 & \leq & 4 \end{array}$$

On aura en solution optimale de LP la base $B^1 = \{2, 5, 6\}$ et sa solution $x^1 = (0, 1, 0, 0, 1, 5)$

Phase 3 :
L_1 correspond à la liste des bases que l'on doit traiter. L_2 correspond à toutes les bases visitées et donc enregistré comme base efficace. Soit εN l'ensemble des variable hors base.
L_1 correspond aux bases que nous n'avons pas traité et qui nous reste à visiter. L_1 correspond aux bases que nous avons visités, nous les stockons pour éviter de retomber sur une base déjà visitée. Nous partons de la base $B^1 = \{2, 5, 6\}$ et ainsi $L_1 = \{\varnothing\}$ et $L_2 = \{\{2, 5, 6\}\}$
Voici le tableau simplexe associé à la base {2,5,6}

$\bar{c}^1$	1	0	0	2	0	0	2
$\bar{c}^2$	-1	0	2	0	0	0	0
$\bar{c}^3$	1	0	-1	0	0	0	0
x_2	1	1	0	1	0	0	1
x_5	-1	0	0	-1	1	0	1
x_6	2	0	1	1	0	1	5

Regardons lesquelles des variables hors bases sont des variables efficaces. Rappelons que nous devons pour chaque variable vérifier le programme auxiliaire $min\{c_{aux}^T(y + \delta) : Ry - r^j\delta \geq 0; y, \delta \leq 0\}$ si celui ci est borné alors la variable est efficace.
Variable x_1 : explicitation de R,r et v dans le tableau simplexe pour x_1
Les variables hors base sont εN = {1,3,4}. Pour construire R, on extraira les colonnes des coûts des objectifs des variables 1,3,4.

La colonne 1 de R est (1,-1,1) pour x_1. La colonne 3 de R est (0,2,-1) pour x_3 et la colonne 3 de R est (2,0,0) pour x_4 Ainsi R= $\begin{pmatrix} 1 & 0 & 2 \\ -1 & 2 & 0 \\ 1 & -1 & 0 \end{pmatrix}$.

$-r^j$ correspond simplement à l'opposé de la colonne de la variable à tester. Ici c'est x_1 que l'on vérifie sa colonne dans au niveau des coûts des objectifs est (1,-1,1) et donc $-r^j = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$

Au niveau des coûts réduits dans le tableau simplexe on fera simplement la somme des coefficient de la colonne qui correspond à la variable (excepté pour v qui sont des variables d'écarts).

Le tableau simplexe associé à la variable x_1 est donc :

R			$-r^j$	v			
1	1	2	-1	0	0	0	0
1	0	2	-1	1	0	0	0
-1	2	0	1	0	1	0	0
1	-1	0	-1	0	0	1	0

Il existe un pivot, le problème est borné donc x_1 est une variable efficace.

Variable x_3 : Le tableau simplexe associé à la variable x_1 :

R			$-r^j$	v			
1	1	2	-1	0	0	0	0
1	0	2	0	1	0	0	0
-1	2	0	-2	0	1	0	0
1	-1	0	1	0	0	1	0

Il existe un pivot, le problème est borné donc x_3 est une variable efficace.

Variable x_4 : Le tableau simplexe associé à la variable x_4 :

R			$-r^j$	v			
1	1	2	-2	0	0	0	0
1	0	2	-2	1	0	0	0
-1	2	0	0	0	1	0	0
1	-1	0	0	0	0	1	0

Il n'existe pas de pivot, le problème est non borné donc la variable x_4 n'est pas efficace.

Les variables 1 et 3 sont les variables efficaces pour la base {2,5,6}. En regardant le tableau simplexe associé à la base {2,5,6} si x_1 rentre alors x_2 sort ce qui nous donne une nouvelle base {1,5,6}.
Si x_3 rentre alors x_6 sort ce qui nous donne une nouvelle base {2,5,3}.
Ces deux nouvelles bases sont rajoutés dans $L1$ la liste des bases à visiter. Bien entendu tant qu'il reste des bases à visiter l'algorithme continue. Pour chaque base dans $L1$ il va trouver les variables hors bases puis calculer les variables efficaces et réactuliser les bases à visiter et les bases visitées etc...
Trivia :
La solution de ce MOLP correspond aux 3 bases {{2,5,6},{1,5,6},{2,3,5}}, continuer l'algorithme n'aurait pas fourni plus de bases.

source : exemple issu du livre "Multicriteria optimization" de Matthias Ehrgott.

4.9.1 Incorporation de la forme révisé LU

Comme dit précédemment, On souhaite souhaite incorporer à cet algorithme la décomporstion LU. Celle-ci nous permet dans un premier temps de calculer la matrice R dans chaque base puis si une variable est efficace, elle permet de trouver la variable sortante. Dans les algorithmes suivants, t, y, et d sont des variables des systèmes d'équation. La forme révisée avec la décomposition LU nous permet donc de savoir pour chaque base la solution associée mais recontruire partiellement les parties qui nous conviennent dans le tableau simplexe.

Algorithme 4 : Calcul de la matrice objectif R avec la décomposition LU

$N \leftarrow$ Variables hors bases
L,$U \leftarrow$ décompositionLU (A_B)
for *Tous les objectifs k* **do**
  $t \leftarrow$ solution du système d'équation : $t^T U = C_B[k]$
  $y \leftarrow$ solution du système d'équation : $y^T L = t^T$
  $R[k] \leftarrow (C_N[k])^T - y^T A_N$
end

Algorithme 5 : Calcul de la variable sortante avec la décomposition LU

$X_B \leftarrow$ Solution optimale associée à la base B
$y \leftarrow$ solution du système d'équation : $Ly = X_B$
$d \leftarrow$ solution du système d'équation : $Ud = y$
$s \leftarrow$ trouver le min positif de $\{\frac{X_B}{d}\}$

5 Concernant l'implémentation

L'implémentation de l'algorithme précédemment décrit a été réalisée en Julia, version 1.02.
Liste des library en Julia utilisées :
LinearAlgebra, JuMP, MathProgBase, ClpSolver, GurobiSolver, GLPKSolver, CPLEX

5.1 Implémentation efficace et Structure de donnée

La principale faiblesse de l'algorithme est l'ajout successif de bases dans une liste en vérifiant à chaque fois que cette base n'y est pas déjà. Si le problème possède un grand nombre de solution paréto-optimal ce test d'appartenance sera de plus en plus coûteux. C'est pourquoi une structure de donnée performante est nécessaire.

Pour coder les données du problème, on utilise simplement les matrices et les vecteurs de Julia. Ces objets sont indicés à partir de 1 en Julia.

Pour résoudre la phase un du simplexe, le dual de benson et la première somme pondérée, on utilise la modélisation JuMP et un solveur classique, par exemple GLPK.

5.1.1 Structure des liste de Bases

Si les choix d'implémentations ci-dessus paraissent évident, il n'est pas de même pour le choix concernant l'encodage des Bases et des listes de Bases. En effet, L'algorithme tel que décrit dans [1] propose deux listes L1 et L2. Les bases que l'on a trouvées mais pas encore explorées sont dans la liste L2 et quand on a détecté toutes les variables efficaces de cette base, on la retire de L2 pour la mettre dans L1.

On a donc ici une représentation ensembliste des bases et il existe en julia une telle classe, les set, qui on été testés dans les premières implémentations. On avait alors des bases qui étaient des ensembles de variables et des liste qui étaient des ensembles de bases.

Cependant, l'implémentation s'est dirigée vers une structure plus simple de tableaux. Cette implémentation a deux avantages par rapport aux set, on connais sa complexité et elle permet de ne manipuler qu'une seule liste. En effet comme un tableau est indicé, cela nous permet de passer simplement sur la base suivante du tableau et d'ajouter les nouvelles bases au bout de celui-ci. Pour savoir si une base appartient à la liste, on est obligé de regarder une à une les variables de chaque bases de la liste. On a donc une complexité de $\mathcal{O}(\#(B)^2 \times \#(L))$.

Une deuxième idée pour gagner en complexité serait de trier les bases. De cette manière, on peut savoir plus simplement si elles appartiennent à la liste de bases. Cette structure a théoriquement une complexité de $\mathcal{O}(\#(B \times log(B)) + \#(B) \times \#(L))$ Même si dans la pratiques les deux approches sont équivalentes en terme de temps d'exécution car la principale complexité de l'algorithme est dans la résolution des problèmes auxiliaires et des systèmes d'équations de la décomposition LU.

Même si ces deux structures en tableaux sont efficaces, elles souffrent aussi de la faiblesse de l'algorithme. En effet, on doit toujours énumérer les bases en évitant les retours sur des bases connues. Une nouvelle manière intelligente de parcourir les bases serait nécessaire pour réduire la complexité du test d'appartenance de la liste. Ce nouveau parcourt intelligent demanderait une refonte complète de l'algorithme.

5.2 Décomposition LU en pratique

Lors de l'implémentation, une erreur dans la décomposition LU a forcé la réécriture de la phase trois avec la matrice inverse de la forme révisé du simplexe pour vérifier les calculs. Comme dit précédemment, la décomposition LU permet de simplifier les calculs et d'accumuler moins d'imprécisions numériques. Cette observation fut flagrante lors des tests. Bien évidemment sur de grandes instances, la décomposition LU n'est pas infaillible et peut tout de même générer quelques imprécisions numériques.

5.3 Instances

5.3.1 Générateur d'instances et parseur

Pour avoir plus d'instance de test, nous avons entrepris de faire un générateur d'instances multiobjectif. La génération de problèmes aléatoires est dirigée par plusieurs paramètres qui peuvent êtres modifiés comme suit :

Notation :
— $[a : b]$ signifie toutes les valeurs entières entre a et b inclus.
Paramètres :
— [3 : 9] objectifs,

— [2 : 20] variables,
— [1 : 30] contraintes,
— $[-9 : 9]$ pour les valeurs dans A et C avec une forte probabilité pour la valeur 0
— [1 : 20] pour les valeurs de b
— une chance sur 6 d'avoir une contrainte d'égalité et le reste des chances réparties entre les inégalités

On remarque que 70% des instances générés sont impossibles, 25% des restantes ne passent pas la phase 2 et il est rare de ne pas avoir plusieurs rayons infini lors de l'exploration des bases. On en déduit qu'il est très difficile de générer des instances intéressantes de manières aléatoires. En soit c'est assez logique. Quelques exemples générés intéressants sont dans le fichier resultats.jl Pour tester d'autres instances, un parser pour les instances .mop a été programmé.
Exemple d'instances qui peuvent être parsé : "data/molp_10_779_10174_entropy.mop" Les instances utilisés pour les test sont tirées des livres suivants :[1], [2], [3], ainsi que quelques instances générées.

5.4 Structuration d'un MOLP en machine

Avant de commencer toute implémentation, il serait pertinent de savoir sous quelles formes traiter les MOLP. Subséquemment, la notation et la formulation utilisées dans le livre "Multicriteria Optimization" de Matthias Ehrgott ont été reprises.

$$\begin{array}{lrl} \text{Min} & -x_1 - 2x_2 & \\ \text{Min} & -x_1 + 2x_3 & \\ \text{Min} & x_1 - x_3 & \\ \text{s.t.} & x_1 + x_2 & \leq 1 \\ & x_2 & \leq 2 \\ & x_1 - x_2 + x_3 & \leq 4 \end{array}$$

Cela peut paraître contre-intuitif (des inférieur ou égal avec des Min) mais dans la formulation on a bien :

— Fonction objectif : tout est en minimisation

Nous ramènerons donc tout MOLP (MultiObjective Linear Programm) sous cette forme dans un souci de cohérence.

Transformation :

Les fonctions à maximiser deviennent des minimisation en étant multiplié par -1.

Max$f(X)$ devient Min $-f(X)$

Exemple :

On a le MOLP suivant :

$$\begin{array}{lrl} \text{Max} & 3x_1 - 2x_2 & \\ \text{Max} & x_1 + 2x_2 & \\ \text{s.t.} & 3x_1 + 2x_2 & \geq 1 \end{array}$$

Celui ci est équivalent à :

$$\begin{array}{lrl} \text{Min} & -3x_1 + 2x_2 & \\ \text{Min} & -x_1 - 2x_2 & \\ \text{s.t.} & 3x_1 + 2x_2 & \geq 1 \end{array}$$

++Cinq informations sont donc importantes pour la représentation et la formalisation de nos MOLP.

— Les coefficients des fonctions objectives (Tableau de tableau de flottants intitulé "C" dans le code)
— L'indication de minimisation ou maximisation des fonctions (tableau d'entiers intitulé "MM" dans le code)

— Les coefficients dans les contraintes (appelé "A" dans le code)
— Les seconds membres (appelé "b" dans le code)
— L'indication d'inéquation ou d'égalité (tableau d'entiers intitulé "equ_const" dans le code)

Pour illustrer concrètement cette représentation, faisons un raisonnement inductif en s'aidant d'un exemple. Reprenons le MOLP suivant :

$$\begin{array}{lrl} \text{Min} & -x_1 - 2x_2 & \\ \text{Min} & -x_1 + 2x_3 & \\ \text{Min} & x_1 - x_3 & \\ \text{s.t.} & x_1 + x_2 & \leq 1 \\ & x_2 & \leq 2 \\ & x_1 - x_2 + x_3 & \leq 4 \end{array}$$

On aura :
Les objectifs :

C=[[-1.0 , -2.0 , 0.0] , [-1.0 , 0.0 , 2.0], [1.0 , 0.0 , -1.0]]

Max ou Min :

MM=[O,O,O]
MM[i]=0 signifie que le ième objectif est une minimisation
MM[i]=1 signifie que le ième objectif est une maximisation

Matrice de contraintes :

constraint=[1.0 1.0 0.0 ; 0.0 1.0 0.0 ; 1.0 -1.0 1.0]

Membre de droite :

b=[1.0,2.0,4.0]

Equation ou inéquation :

equ_const=[1,1,1]
equ_const[i]= 1 signifie que la ième contrainte est une inéquation inférieure ou égale
equ_const[i]= 2 signifie que la ième contrainte est une inéquation supérieure ou égale
equ_const[i]= 3 signifie que la ième contrainte est une égalité

Si on a MM qui correspond à une suite de 0 et equ_const qui correspond à une suite de 1 cela signifie que le format est bon et qu'on a bien que des minimisation et des inférieures ou égales.

Le MOLP décrit auparavant est introduit comme suit sous Julia :

```
MM=[0,0,0]
C=[ -1.0 -2.0 0.0 ; -1.0 0.0 2.0 ; 1.0 0.0 -1.0 ]
#matrice contraintes
A=[ 1.0 1.0 0.0 ; 0.0 1.0 0.0; 1.0 -1.0 1.0]
equ_const=[1,1,1]
#membre de droite
b=[1.0,2.0,4.0]
```

5.5 Julia et JuMP

JuMP qui signifie Julia Mathematical Programming est la passerelle entre un solver et le langage de programmation julia. C'est grâce à celui-ci qu'il est possible de définir très facilement un programme linéaire sous Julia. Voici un petit exemple de modèle extrait de la documentation officielle du problème de maximisation suivant :

$$\begin{array}{ll} \text{Max} & 5\text{x} + 3\text{y} \\ \text{s.t.} & \text{x} + \text{y} \leq 3 \end{array}$$

```
m = Model(solver = ClpSolver())
@variable(m, x)
@variable(m, y )

@objective(m, Max, 5x + 3*y )
@constraint(m, 1x + 5y <= 3.0 )
status = solve(m)
```

Nous pouvons en sortie du solver récupérer la valeur des variables de décision et la valeur de la fonction objectif. Cependant nous n'avons pas l'accès direct aux valeurs des variables d'écarts.

5.6 La dégénérescence

5.6.1 La récupération des variables en base

Le problème avec la manipulation de solver pour résoudre des LP est que ce sont de vraies boites noires. En effet les données sont d'abord formalisées sous JuMP puis encore retransformées par le solver. Après la résolution d'un LP nous n'avons aucune information sur les variables d'écarts et les variables en base et hors base. Bien entendu comme nous le savons si la solution n'est pas dégénérée et que donc toutes les variables de décision ne sont pas nulles, alors on peut facilement calculer les valeurs des variables d'écarts ou déduire les variables hors base et de base. Reprenons une partie de l'exemple de notre MOLP étudié dans les parties antérieures.

$$\begin{array}{lrl} \text{Min } 0 & & \\ \text{s.t.} & x_1 + x_2 & \leq 1 \\ & x_2 & \leq 2 \\ & x_1 - x_2 + x_3 & \leq 4 \end{array}$$

Comme on le sait la solution triviale (x_1, x_2, x_3)=(0,0,0) est une solution admissible avec les variables de décisions hors bases. Si on a (x_1, x_2, x_3)=(1,2,4), on en déduit facilement que les variables de décisisons sont toutes en bases. En programmation linéaire le cas de cyclage est rare mais l'obtention de solution dégénérée est extrêment répandue. Une solution autre que regarder la valeur nulle ou non des variables des décisions devaient être apportée sans quoi on ne pouvait commencer la phase 3 du simplexe multicritère qui doit avoir connaissance de quelles variables sont en bases pour démarrer.

Le problème ici n'est pas réellement un problème de reflexion mais un problème technique. Pour récupérer des informations plus bas niveau, il était possible de regarder au niveau du solver lui même ou tenter de comprendre comment se fait la transition entre JuMP et le solver.
Ce petit problème en apparence a nécessité plus de temps que prévu. Malheureusement les solvers sont peu documentés quant à leur manière de procéder et même si l'on arrivait à récupérer des données bas niveau celles ci correspondaient à un amas de pointeur, de nombres et de variables dans un désordre incompréhensible et inutilisable.

Bien heureusement il s'est avéré qu'il existe sous MathProgBase une fonction permettant justement pour certain solver de récupérer les informations sur les variables (en base, bornes etc...). Les solver concernaient sont ClpSolver, GurobiSolver, CPLEX et MOSEK.

Le problème a été résolu cependant en pleine transition avec JuMP 0.19, MathProgBase est maintenant obsolète (bien que toujours utilisable) et a été remplacé par MathOptInterface pour lequelle cette astuce ne fonctionne pas.

5.6.2 La suppression des combinaisons linéaires

Nous avons vu qu'en cas de dégénérescence et de combinaisons linéaires entre les contraintes il pouvait y avoir une suppression de contraintes. Une suppression de contraintes implique un nombre de variables en base différents or lorsque le problème est donné au solver, cette suppression n'est pas forcement indiquée. Supprimer au préalable les contraintes qui ont une dépendance linéaire permet d'éviter d'autres complications.
Seules les contraintes à l'égalité sont concernées car pour les inégalités une variable d'écart spécifique et unique lui est associée et donc la contrainte en question ne peut être linéairement dépendante avec une autre.

Ce problème est en réalité un problème d'algèbre linéaire et consiste à retirer la plus grande famille libre d'une famille de vecteurs. De façon plus parlante, le problème consiste à échelonner une matrice et à retirer les lignes qui sont toutes nulles après échelonnage. Aussi le nombre de contraintes redondantes correspond au nombre de contraintes moins le rang de la matrice associée.
Les étapes de l'implémentation sont donc les suivantes :

1. Extraire les contraintes qui sont des contraintes d'égalité.
2. Construire une matrice associée
3. Echelonner la matrice
4. Retirer les lignes nulle
5. Réinjecter les contraintes non redondantes au problème initial

Reprenons l'exemple vu dans le cadre de la dégénérescence.

$$\begin{array}{llllllll}
\max & 6x_1 + & 5x_2 + & 7x_3 + & 4x4 & & \\
\text{s.t} & x_1 + & x_2 + & & & = & 20 \\
 & & & x_3 & +x4 & = & 10 \\
 & x_1 & & +x_3 & & = & 13 \\
 & & x_2 & & +x_4 & = & 17 \\
 & x_1, & x_2, & x_3, & x_4 & \geq & 0
\end{array}$$

Construction de la matrice associée :

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 20 \\ 0 & 0 & 1 & 1 & 10 \\ 1 & 0 & 1 & 0 & 13 \\ 0 & 1 & 0 & 1 & 17 \end{pmatrix}$$

Echelonnage :

On échange la ligne 2 et 4.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 1 & 17 \\ 1 & 0 & 1 & 0 & 13 \\ 0 & 0 & 1 & 1 & 10 \end{pmatrix}$$

$$L3 = L3 - L1$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 1 & 17 \\ 0 & -1 & 1 & 0 & -7 \\ 0 & 0 & 1 & 1 & 10 \end{pmatrix}$$

$$L3 = L3 + L2$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 1 & 17 \\ 0 & 0 & 1 & 1 & 10 \\ 0 & 0 & 1 & 1 & 10 \end{pmatrix}$$

$$L4 = L4 - L3$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 1 & 17 \\ 0 & 0 & 1 & 1 & 10 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

On obtient une matrice de rang 3, il y a une contrainte qui est combinaison linéaire des autres et on peut la supprimer cette contrainte correspondait à la ligne 2 (au départ avant la permutation). De ce fait la contrainte 2 peut être supprimée. Il y a plusieurs possibilités de suppression, si par exemple la contrainte 1 s'exprime comme la combinaison linéaire de la contrainte 3 et 5. Supprimer n'importe quelle contrainte parmi 1,3,5 est envisageable. Il s'agira de bien veiller à supprimer les contraintes concernées.

6 Limites du simplexe multicritère

Le simplexe multicritère un simplexe pédagogique qui certe fournit des solutions satisfaisantes mais la mécanique a bien sur ses défauts. Nous avons vu qu'il était ardu de traiter les problèmes d'imprécisions numériques et cela peut arriver très vite sachant que beaucoup de programmes linéaires intermédiaires et de systèmes d'équations sont résolus au sein même de la méthode pour pouvoir solutionner le MOLP.
De plus l'énumération des bases efficaces dans la phase 3 fait exploser la complexité de l'algorithme. Rien que pour 100 variables et 3 contraintes cela nous donne 3 parmi 100 combinaisons de bases possibles en pire cas. D'un point de vue théorique il serait raisonnable d'abandonner toute résoltion de MOLP de grandes instances. Il n'y a à ce jour pas de stratégie notable et connu pour savoir dans quel ordre visiter les bases efficaces et si cela peut alléger ou non la lourdeur de la phase.

7 Objectifs repoussés

7.1 Matrices creuses

L'utilisation des matrices creuses est une piste à creuser pour la décomposition LU qui peut se reposer dessus. Sous julia, il existe une structure de données pour manipuler des matrices creuses mais implémenter tout le processus nous demanderait un certain temps qui nous fait défaut.

7.2 Interface avec voptgeneric

En raison de la conjoncture actuelle, Vopt-solver devra attendre avant de pouvoir intégrer l'implémentation du simplexe multicritère qui a été faite. La raison principale en est qu'il y a récemment de nombreux changements dans l'ensemble des outils utilisés. En effet JuMP est passé à sa version supérieure 0.19 et souffre de problème de compatibilité avec Vopt. De plus la bibliothèque nous permettant de récupérer le modèle interne du solver est devenue dépréciée. En pleine phase de transition : MathProgBase est maintenant obsolète et remplacée par MathProgInterface qui n'a pas fini de réviser toutes les possibilités offertes par son prédecesseur.

8 Conclusion

Nous avons tout d'abord cherché à implémenter l'algorithme tel que décrit dans le livre "Multicriteria Optimization" [1]. Nous avons remarqué que celui-ci pouvait être légèrement optimisé en modifiant les structures de données qu'il utilise.
Puis nous avons amélioré l'algorithme avec la forme révisée du simplexe et la décomposition LU. Et nous avons obtenu une réduction conséquente des imprécisions numériques.
Nous avons observé qu'un générateur d'instance peine à créer des problèmes intéressants. La version du langage de modélisation de JuMP n'est pas encore stable. C'est pour cette raison que l'intégration de l'algorithme à vOptSolver n'a pas été faite.
La détection des facettes non-dominées et l'interface graphique dans l'espace des objectifs de dimension 3 pourraient être intéressantes mais ces fonctionnalités ne paraissent pas vitales pour cet algorithme car il ne sera pas utilisé en tant que tel. Nous ne traitons que des problèmes multiobjectif linéaires et continus mais peut être que l'utilisation de cet algorithme serait plus utile dans la relaxation d'un problème multiobjectif combinatoire. De ce fait, l'algorithme se doit d'être le plus optimisé possible. Et pour ceci, les trois pistes que nous avons sont la reprogrammations de la décomposition LU, la résolution des systèmes d'équations triangulaires, et le remplacement de JuMP par des appels directs aux solveurs.

D'une manière générale, nous sommes assez satisfaits du travail qui nous a été demandé et du travail réalisé. De plus le sujet en lui même était très enrichissant et nous as permis de nous familiariser avec des notions assez poussées encore inconnues jusque là pour nous. Nous sommes fiers d'avoir pu contribuer et pu apporter ne serait qu'un petit peu au monde de la recherche opérationnelle.
Et fort de cette expérience, nous restons à l'écoute de toutes remarques, améliorations et contributions au code et au travail qui a été réalisé.

Références

[1] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[2] Ralph E Steuer. Multiple criteria optimization. *Theory, Computation and Applications*, 1986.

[3] Milan Zeleny and James L Cochrane. *Multiple criteria decision making*. University of South Carolina Press, 1973.