Projet MADMC
Sélection bi-objectifs avec coefficients intervalles

Ariana Carnielli

1 Introduction

Le projet a été implémenté en Python et le code correspondant est fourni dans le fichier annexe `projet_madmc.py`. Un fichier Jupyter notebook `tests.ipynb` contient les codes de génération de tous les exemples traités dans ce rapport et des figures présentées ici.

On rappelle brièvement le cadre de ce projet. On considère que l'on dispose de n objets et d'une valuation $c^i = (c_1^i, c_2^i) \in \mathbb{R}^2$ pour chaque objet $i \in \{1,\dots,n\}$. On dispose aussi d'un entier $k \in \{0,\dots,n\}$ donnant le nombre d'objets à être sélectionnés et on représente une sélection de k objets par un vecteur $x = (x_1,\dots,x_n) \in \{0,1\}^n$ avec $\sum_{i=1}^n x_i = k$, où $x_i = 1$ représente que l'objet i a été pris. L'espace de solutions est dénoté par X et représente ainsi l'ensemble de tous les vecteurs x de $\{0,1\}^n$ tels que $\sum_{i=1}^n x_i = k$.

Étant donné un vecteur $x \in X$, la valuation de l'ensemble d'objets représenté par x est définie par $y = c(x) = (\sum_{i=1}^n c_1^i x_i, \sum_{i=1}^n c_2^i x_i)$. On considère alors que l'espace des objectifs est $\mathbb{R}^2$ et on dénote par Y l'image de X dans l'espace des objectifs, c'est-à-dire que $Y = \{c(x) \mid x \in X\}$.

On fixe également un intervalle $I = [\alpha_{\min}, \alpha_{\max}] \subseteq [0,1]$, avec $0 \le \alpha_{\min} < \alpha_{\max} \le 1$. L'objectif est de trouver une sélection de k objets parmi les n disponibles qui minimise sur $y \in Y$ le critère

$$f_I(y) = \max_{\alpha \in I} [\alpha y_1 + (1-\alpha) y_2].$$

2 Résultats préliminaires

Question 1. On considère les vecteurs et l'intervalle suivants :

$$y = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad y' = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \qquad y'' = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad I = [0,1].$$

On calcule :

$$\begin{aligned} f_I(y) &= \max_{\alpha \in I} [\alpha + (1-\alpha)] = 1 \\ f_I(y') &= \max_{\alpha \in I} 2(1-\alpha) = \max_{\alpha \in I} [2 - 2\alpha] = 2 \\ f_I(y + y'') &= \max_{\alpha \in I} [3\alpha + (1-\alpha)] = \max_{\alpha \in I} [1 + 2\alpha] = 3 \\ f_I(y' + y'') &= \max_{\alpha \in I} [2\alpha + 2(1-\alpha)] = 2 \end{aligned}$$

On observe donc que $f_I(y) < f_I(y')$ et $f_I(y' + y'') < f_I(y + y'')$, et alors le principe d'optimalité n'est pas vérifié : ajouter un même vecteur à deux vecteurs donnés peut changer leur ordre selon f_I.

Question 2. La réponse à cette question a été implémentée dans la fonction `gen_vecteur`, qui crée une liste de vecteurs, chaque vecteur étant représenté comme un *tuple* de Python à 2 éléments. La génération aléatoire a été faite avec la fonction `gauss` de la bibliothèque `random`.

Question 3. Le principe de l'algorithme naïf est le suivant : pour chaque vecteur de la liste, on le teste contre tous les autres vecteurs. S'il n'est dominé par aucun des autres vecteurs, on le rajoute à la liste qui contiendra la solution. Cet algorithme a été implémenté dans la fonction `non_domine_p_naif`, qui prend en argument la liste de vecteurs et retourne la liste de ceux qui ne sont pas dominés.

Question 4. Comme indiqué à l'énoncé, la première étape de cet algorithme (que l'on appellera "algo lex" dans la suite par simplicité) est de trier la liste de vecteurs de façon lexicographique. Ensuite, l'algorithme procède comme suit : on introduit une variable `min2`, initialisée à $+\infty$, qui contient, à chaque itération i, la plus petite valeur de la deuxième composante des vecteurs qui sont entre les positions 0 et $i-1$ de la liste triée. À l'itération i, on regarde le i-ème vecteur de la liste triée : si sa deuxième composante est strictement plus petite que `min2`, alors on le rajoute à la liste des vecteurs non-dominés et on met à jour la valeur de `min2`, sinon on ne fait rien. On retourne à la fin la liste des vecteurs non-dominés.

Cela marche car, dans la liste triée, le vecteur $y^i = (y_1^i, y_2^i)$ traité à l'itération i est tel que $y_1^i \geq y_1^j$ pour tout $j \in \{0,\dots,i-1\}$ et que, si $y_1^i = y_1^j$ pour un tel j, alors forcément $y_2^i \geq y_2^j$. Donc il n'y a que deux façons pour que ce vecteur ne soit pas dominé : soit il est identique à un vecteur qui vient avant lui dans la liste triée (auquel cas il n'est pas nécessaire de le rajouter à la liste de vecteurs non-dominés), soit $y_2^i < y_2^j$ pour tout $j \in \{0,\dots,i-1\}$, ce qui arrive si et seulement si $y_2^i < \min_{j\in\{0,\dots,i-1\}} y_2^j$.

Cela a été implémenté dans la fonction `non_domine_p`, qui prend en argument la liste de vecteurs et retourne la liste de ceux qui ne sont pas dominés.

Pour tester les fonctions `non_domine_p_naif` et `non_domine_p`, on a exécuté les deux sur une même instance, de taille $n = 100$ et tirée aléatoirement selon le principe de la Question 2 avec $m = 100$. Le résultat est présenté sur la Figure 1, dans laquelle les vecteurs générés sont représentés par des cercles bleus, sauf les vecteurs non-dominés retournés par les fonctions, qui sont marqués par un cercle orange.

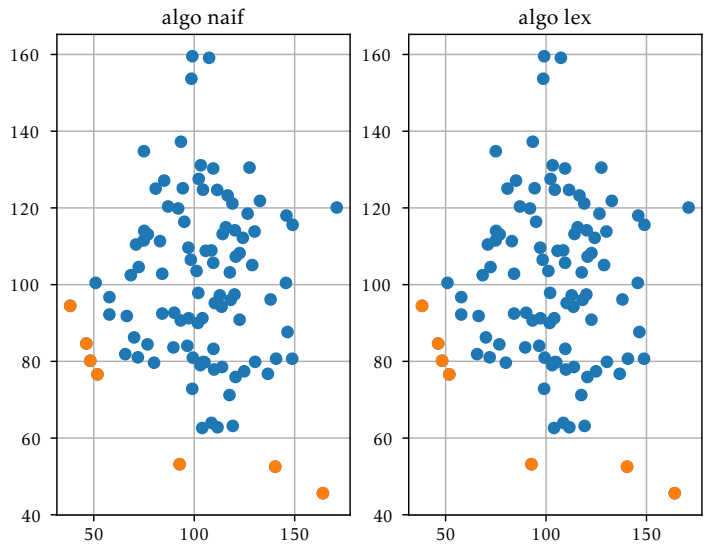

FIGURE 1 – Test de l'implémentation des algorithmes naif et lex

Question 5. L'algorithme naïf teste toutes les paires de vecteurs, deux fois chaque paire, et donc sa complexité est en $O(n^2)$. L'algorithme "lex" de la Question 4 fait d'abord un tri lexicographique de la liste des vecteurs, qui s'exécute en $O(n \log n)$, et ensuite il fait un seul parcours de la liste, en $\Theta(n)$, donc sa complexité est en $O(n \log n)$.

La Figure 2 représente les courbes correspondant aux temps d'exécution des deux algorithmes avec les spécifications données à l'énoncé, obtenues à l'aide de la fonction `tester_temps_n` qui utilise la fonction `process_time` du module `time` de Python pour mesurer les temps d'exécution. On observe effectivement que l'algorithme naïf s'exécute plus lentement que l'algorithme "lex" et que son temps d'exécution a une croissance bien plus vite que linéaire, alors que le temps d'exécution de l'algorithme "lex" croît de façon presque linéaire avec la taille de l'instance.

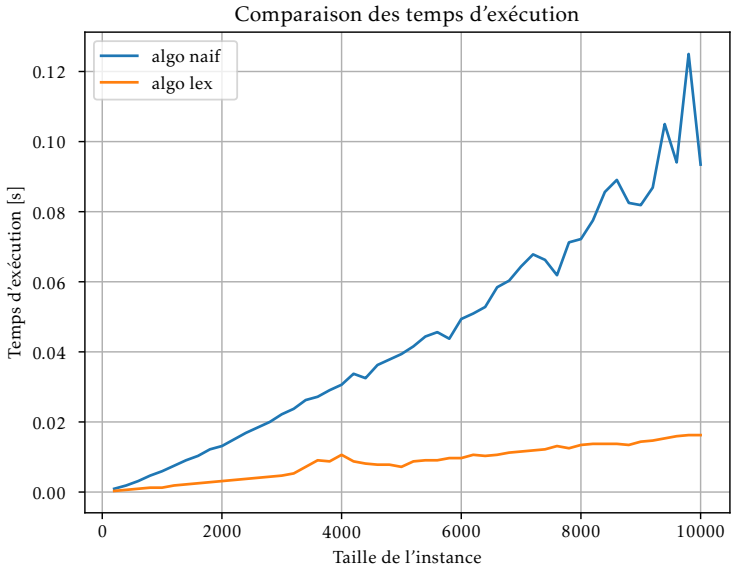

FIGURE 2 – Temps d'exécution des algorithmes naif et lex

3 Une première procédure de résolution

Question 6. On dénote la dominance de Pareto dans $\mathbb{R}^2$ par $\prec_P$ et la dominance faible de Pareto dans $\mathbb{R}^2$ par $\precsim_P$. On montre d'abord que, pour tous $y, z \in \mathbb{R}^2$, on a l'implication

$$z \precsim_P y \implies f_I(z) \leq f_I(y). \tag{1}$$

En effet, si $z \precsim_P y$, on a $z_1 \leq y_1$ et $z_2 \leq y_2$. Alors, pour tout $\alpha \in I \subseteq [0,1]$, on a

$$\alpha z_1 + (1-\alpha) z_2 \leq \alpha y_1 + (1-\alpha) y_2.$$

Soit $\alpha_* \in I$ la valeur qui maximise $f_I(z)$, c'est-à-dire $f_I(z) = \alpha_* z_1 + (1-\alpha_*) z_2$. Alors, en utilisant l'inégalité précédente avec $\alpha = \alpha_*$, on a

$$f_I(z) = \alpha_* z_1 + (1-\alpha_*) z_2 \leq \alpha_* y_1 + (1-\alpha_*) y_2 \leq \max_{\alpha \in I} [\alpha y_1 + (1-\alpha) y_2] = f_I(y),$$

et donc on a bien $f_I(z) \leq f_I(y)$, ce qui conclut la preuve de (1).

Pour passer à la preuve du résultat de l'énoncé, on introduit quelques notations. Soit Y_I l'ensemble des points minimax, Y_{IP} l'ensemble des points Pareto non-dominés lorsque l'on ne considère que les points de Y_I, et Y_P l'ensemble des points Pareto non-dominés dans Y, c'est-à-dire :

$$\begin{aligned} Y_I &= \{z \in Y \mid \forall y \in Y, f_I(z) \leq f_I(y)\} \\ Y_{IP} &= \{z \in Y_I \mid \nexists y \in Y_I \text{ tel que } y \prec_P z\} \\ Y_P &= \{z \in Y \mid \nexists y \in Y \text{ tel que } y \prec_P z\} \end{aligned}$$

On remarque que ces trois ensembles sont non-vides et que $Y_{IP} \subseteq Y_I$. Montrer la propriété donnée à l'énoncé revient à montrer que $Y_I \cap Y_P \neq \emptyset$ et, pour montrer cela, on montrera le résultat légèrement plus fort que $Y_{IP} \subseteq Y_P$.

Supposons, pour obtenir une contradiction, que $Y_{IP} \not\subseteq Y_P$ et soit $z \in Y_{IP}$ tel que $z \notin Y_P$. Comme $z \in Y_{IP}$, on a en particulier $z \in Y_I$, et donc, pour tout $y \in Y$, on a

$$f_I(z) \leq f_I(y). \tag{2}$$

Comme $z \notin Y_P$, alors il existe $y \in Y$ tel que $y \prec_P z$. En particulier, $y \precsim_P z$ et, en appliquant (1), on en déduit que $f_I(y) \leq f_I(z)$. En combinant avec (2), on obtient que $f_I(z) = f_I(y)$. Comme $z \in Y_I$, cela dit aussi que $y \in Y_I$. Mais alors $y \in Y_I$ est tel que $y \prec_P z$, ce qui est une contradiction avec le fait que $z \in Y_{IP}$. Cette contradiction établit donc la preuve que $Y_{IP} \subseteq Y_P$, et donc que $Y_I \cap Y_P \neq \emptyset$: il existe au moins un point minimax qui est aussi Pareto non-dominé.

Question 7. On dénote par $T(i,j)$ l'ensemble des images dans l'espace des objectifs $\mathbb{R}^2$ des solutions Pareto non-dominées du problème $P(i,j)$, i.e., du problème dans lequel on sélectionne j objets parmi $\{1,\dots,i\}$.

Pour chaque solution Pareto non-dominée x de $P(i,j)$, on a deux cas : soit l'objet i a été pris dans x, soit il ne l'a pas été. Dans le premier cas, si on supprime i de x, on obtient une solution de $P(i-1,j-1)$ qui est forcément Pareto non-dominée, car si elle était dominée, on pourrait rajouter i à la solution qui la domine et on obtiendrait ainsi une solution de $P(i,j)$ dominant x, ce qui est une contradiction. Dans ce cas, le coût $c(x)$ appartient à $T(i-1,j-1)+c^i$. Dans le deuxième cas, la solution x est aussi Pareto non-dominée pour le problème $P(i-1,j)$, et donc $c(x)$ appartient à $T(i-1,j)$. Alors

$$T(i,j) = \mathcal{M}[(T(i-1,j-1)+c^i) \cup T(i-1,j)], \qquad \forall i \in \{2,\dots,n\}, \forall j \in \{1,\dots,k\}, \tag{3}$$

où $\mathcal{M}(A)$ dénote les éléments Pareto non-dominés de l'ensemble A.

Pour initialiser T, on peut observer que, si $j = 0$, alors on ne prend aucun objet et le coût de la solution est donc $(0,0)$, d'où $T(i,0) = \{(0,0)\}$ pour tout $i \in \{1,\dots,n\}$. On a aussi que $T(1,1) = \{c^1\}$ et que $T(1,j) = \emptyset$ pour $j \in \{2,\dots,k\}$. Ces relations permettent donc d'initialiser la procédure.

Le tableau T à calculer contient $\Theta(nk)$ cases. Soit M une borne supérieure sur la valeur de chaque composante de $c(x)$, et on suppose ici que $c(x)$ est un vecteur d'entiers positifs. Alors un ensemble Pareto-optimal a au plus M éléments, et en particulier chaque case du tableau a au plus M éléments. Pour calculer une case $T(i,j)$, il nous faut calculer l'ensemble d'images de points Pareto non-dominés dans un ensemble de $O(M)$ éléments, ce qui se fait, en utilisant l'algorithme de la Question 4, en $O(M \log M)$. La complexité du calcul de tous les $T(i,j)$ est donc en $O(nkM \log M)$, ce qui donne un algorithme pseudo-polynomial.

Cette procédure de programmation dynamique a été implémentée dans la fonction `pareto_dyn`, qui prend en argument les coûts c^i des objets (une liste de taille n avec en chaque case un `tuple` de taille 2) et l'entier k. La fonction calcule le tableau T selon la procédure décrite ci-dessus ; le tableau est représenté comme un tableau `numpy` d'objets avec, en chaque case, une liste contenant les images des points non-dominés de la case correspondante. La fonction retourne la liste contenue dans la dernière case de ce tableau, c'est-à-dire $T(n,k)$.

Question 8. Soit $y \in \mathbb{R}^2$ et considérons la fonction $\varphi(\alpha) = \alpha y_1 + (1-\alpha)y_2$ définie pour $\alpha \in I$. Alors $\varphi(\alpha) = a\alpha + b$ avec $a = y_1 - y_2$ et $b = y_2$ et donc φ est une fonction affine. Ainsi, son maximum est atteint pour $\alpha = \alpha_{\max}$ si $y_1 > y_2$ et pour $\alpha = \alpha_{\min}$ si $y_1 < y_2$. En plus, si $y_1 = y_2$, φ est constante et donc son maximum est atteint pour tout α. On a donc que pour un y fixé, $\max_{\alpha \in I} \varphi(\alpha)$ est réalisé pour $\alpha = \alpha_{\min}$ ou $\alpha = \alpha_{\max}$.

On peut alors calculer $f_I(y)$ par la formule :

$$f_I(y) = \max\{\alpha_{\max} y_1 + (1-\alpha_{\max})y_2,\ \alpha_{\min} y_1 + (1-\alpha_{\min})y_2\}$$

L'algorithme pour déterminer un vecteur minimax dans un ensemble de vecteurs est alors de parcourir cet ensemble de vecteurs, calculer f_I sur chacun d'entre eux avec la formule ci-dessus et garder celui avec la plus petite valeur de f_I. Cet algorithme a été implémenté dans la fonction `vec_mininmax` qui prend en argument une liste de vecteurs de taille 2 chacun et un `tuple` I avec les valeurs de $\alpha_{\min}$ et $\alpha_{\max}$, en utilisant la fonction auxiliaire `f_i` qui implémente la formule de f_I ci-dessus.

Question 9. La procédure a été implémentée dans la fonction `pareto_solver`, qui prend en argument les coûts c^i des objets (une liste de taille n avec en chaque case un `tuple` de taille 2), l'entier k et un `tuple` I avec les valeurs de $\alpha_{\min}$ et $\alpha_{\max}$. Elle ne fait qu'appeler `pareto_dyn` pour calculer les vecteurs Pareto non-dominés et applique ensuite `vec_minimax` dans ces vecteurs pour trouver un point minimax.

Pour illustrer les fonctions implémentées dans cette partie on a généré une instance de taille petite du problème dans laquelle on arrive à calculer tous les points dans l'espace des objectifs. On a fixé $n = 10$ et généré des valuations avec la fonction gen_vecteur en prenant $m = 30$. On a calculé les images des solutions Pareto non-dominées pour $k = 3$ avec la fonction `pareto_dyn` et le point minimax à l'aide de la fonction `pareto_solver` avec un intervalle $I = [\frac{1}{3}, \frac{2}{3}]$. Ces points sont représentés respectivement en orange et vert sur la Figure 3 où on représente aussi en bleu les images des $\binom{10}{3} = 120$ solutions réalisables.

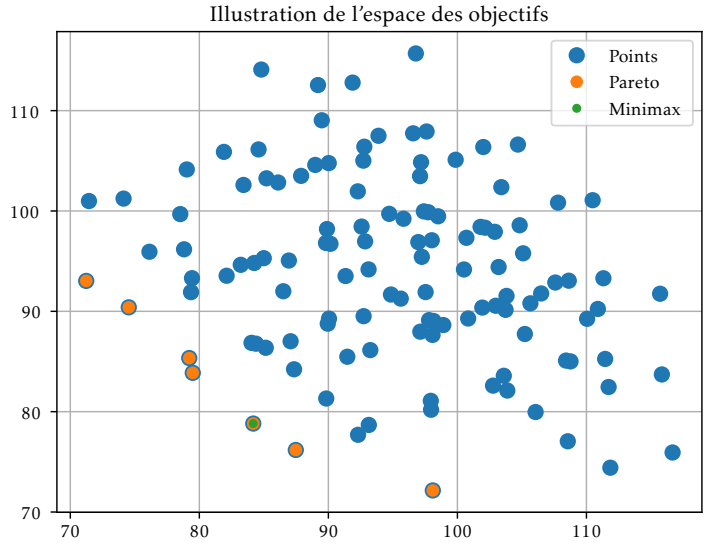

FIGURE 3 – Images des points réalisables, Pareto non-dominés et minimax dans l'espace des objectifs

4 Une seconde procédure de résolution

Question 10. Dans la suite on écrit $z \prec_I y$ pour dire que z I-domine y. Si $NI \subseteq ND$ alors pour tout $y \notin ND$ on a aussi $y \notin NI$. On montre donc ce dernier fait en prenant $y \notin ND$. Dans ce cas, il existe $z \in Y$ tel que $z \prec_P y$, c'est-à-dire $z_1 \leq y_1$, $z_2 \leq y_2$ et au moins une de ces inégalités est stricte. Pour toute $\alpha \in I \subseteq [0,1]$ en multipliant la première inégalité par α et la deuxième par $1-\alpha$ on déduit que :

$$\alpha z_1 + (1-\alpha) z_2 \leq \alpha y_1 + (1-\alpha) y_2$$

et l'inégalité est stricte si $\alpha \neq 0$ et $\alpha \neq 1$. Ainsi $z \prec_I y$ et alors $y \notin NI$. Cela conclut la preuve que $NI \subseteq ND$.

Pour montrer qu'un point minimax est inclus dans NI, on suit une stratégie très similaire à celle de la Question 6. On utilise le même ensemble Y_I de la Question 6 pour dénoter l'ensemble des points minimax et on introduit l'ensemble :

$$Y_{II} = \{z \in Y_I \mid \nexists y \in Y_I \text{ tel que } y \prec_I z\}$$

On remarque que $Y_{II} \subseteq Y_I$ et $Y_{II} \neq \emptyset$. On va montrer que $Y_{II} \subseteq NI$.

Supposons, pour obtenir une contradiction, que $Y_{II} \not\subseteq NI$ et soit $z \in Y_{II}$ tel que $z \notin NI$. Comme $z \in Y_{II}$, on a en particulier $z \in Y_I$, et donc, pour tout $y \in Y$, on a

$$f_I(z) \leq f_I(y). \tag{4}$$

Comme $z \notin NI$, alors il existe $y \in Y$ tel que $y \prec_I z$. En particulier, pour tout $\alpha \in I$ on a $\alpha y_1 + (1-\alpha)y_2 \leq \alpha z_1 + (1-\alpha)z_2$ et donc en prenant comme α la valeur $\alpha_* \in I$ qui maximise $f_I(y)$, c'est-à-dire $f_I(y) = \alpha_* y_1 + (1-\alpha_*)y_2$ on a

$$f_I(y) = \alpha_* y_1 + (1-\alpha_*)y_2 \leq \alpha_* z_1 + (1-\alpha_*)z_2 \leq \max_{\alpha \in I}[\alpha z_1 + (1-\alpha)z_2] = f_I(z),$$

D'où $f_I(y) \leq f_I(z)$ et en combinant avec (4), on obtient que $f_I(z) = f_I(y)$. Comme $z \in Y_I$, cela dit aussi que $y \in Y_I$. Mais alors $y \in Y_I$ est tel que $y \prec_I z$, ce qui est une contradiction avec le fait que $z \in Y_{II}$. Cette contradiction établit donc la preuve que $Y_{II} \subseteq NI$, et donc il existe au moins un point minimax qui est aussi non I-dominé.

Question 11. Soient $y, z \in Y$ et supposons que $y \prec_I z$. On veut montrer que

$$\forall \alpha \in \{\alpha_{\min}, \alpha_{\max}\} \quad \alpha y_1 + (1-\alpha)y_2 \leq \alpha z_1 + (1-\alpha)z_2 \tag{5a}$$
$$\exists \alpha \in \{\alpha_{\min}, \alpha_{\max}\} \quad \alpha y_1 + (1-\alpha)y_2 < \alpha z_1 + (1-\alpha)z_2 \tag{5b}$$

Selon la règle de I-dominance on sait que pour tout $\alpha \in I$, $\alpha y_1 + (1-\alpha)y_2 \leq \alpha z_1 + (1-\alpha)z_2$ et donc plus particulièrement cette inégalité est vraie aussi pour $\alpha_{\min}$ et $\alpha_{\max}$ ce qui montre (5a).

Pour montrer (5b) on définit d'abord la fonction $\varphi(\alpha) = \alpha z_1 + (1-\alpha)z_2 - \alpha y_1 - (1-\alpha)y_2$ qui est une fonction affine en α. D'après la règle de I-dominance on a donc $\forall \alpha \in I, \varphi(\alpha) \geq 0$ et $\exists \alpha \in I$, $\varphi(\alpha) > 0$. Si (5b) n'est pas vraie, alors $\varphi(\alpha_{\min}) = \varphi(\alpha_{\max}) = 0$ et comme φ est affine cela implique que $\varphi(\alpha) = 0$ pour tout $\alpha \in I$, ce qui contredit la I-dominance. Ainsi, on a bien (5b).

Supposons maintenant que (5a) et (5b) sont vraies et on veut montrer que $y \prec_I z$. C'est immédiat à partir de (5b) que $\exists \alpha \in I$, $\alpha y_1 + (1-\alpha)y_2 < \alpha z_1 + (1-\alpha)z_2$. En utilisant (5a) on a $\varphi(\alpha_{\min}) \geq 0$ et $\varphi(\alpha_{\max}) \geq 0$ et comme φ est une fonction affine alors $\varphi(\alpha) \geq 0 \, \forall \alpha \in I$, ce qui montre que $y \prec_I z$.

Pour la transformation de Π en Π' on définit la fonction

$$\pi(y) = \begin{pmatrix} \alpha_{\min} y_1 + (1-\alpha_{\min})y_2 \\ \alpha_{\max} y_1 + (1-\alpha_{\max})y_2 \end{pmatrix}$$

D'après le résultat que l'on vient de montrer, on a $\pi(y) \prec_P \pi(z)$ si et seulement si $y \prec_I z$. Si on a une instance Π du problème de détermination des points non I-dominés on peut donc appliquer la fonction π à tous les vecteurs de l'instance Π pour trouver l'instance Π' du problème de détermination des points non-dominés au sens de Pareto. Une fois qu'on trouve une solution de ce problème, on peut appliquer la fonction π^{-1} aux point Pareto non-dominés pour retrouver les points non I-dominés de l'instance de départ.

Pour donner l'expression de π^{-1} on peut d'abord remarquer que

$$\pi(y) = \begin{pmatrix} \alpha_{\min} & 1-\alpha_{\min} \\ \alpha_{\max} & 1-\alpha_{\max} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

donc π^{-1} se calcule en inversant la matrice, ce qui donne

$$\pi^{-1}(w) = \frac{1}{\alpha_{\min} - \alpha_{\max}} \begin{pmatrix} 1-\alpha_{\max} & -1+\alpha_{\min} \\ -\alpha_{\max} & \alpha_{\min} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

On a implémenté une fonction `non_domine_i` qui prend en argument une liste de vecteurs ainsi que les extrémités de l'intervalle I et qui retourne la liste des vecteurs non I-dominés parmi les vecteurs passés en argument. Pour cela, elle transforme les vecteurs à l'aide de la fonction π, calcule les vecteurs Pareto non-dominés en utilisant la fonction `non_domine_p` et applique la fonction π^{-1} aux vecteurs trouvés pour obtenir la solution.

Pour résoudre le problème qui nous intéresse on doit implémenter une fonction `i_solver` qui prend en argument la liste des valuations de chaque objet, le nombre k d'objets à choisir et les extrémités de l'intervalle I et qui calcule d'abord les points non I-dominés dans l'espace des objectifs et retourne un point minimax parmi ceux-ci. Pour cela, on a deux méthodes possibles pour le calcul des points non I-dominés :

- Implémenter une fonction similaire à `pareto_dyn` qui utilise `non_domine_i` à la place de `non_domine_p` dans le calcul des cases du tableau de la procédure de programmation dynamique.
- Transformer les valuations des objets à l'aide de la fonction π et utiliser la procédure de programmation dynamique déjà codée dans `pareto_dyn` en appliquant π^{-1} à la solution retournée.

Ces deux méthodes sont équivalentes car π est une fonction linéaire et la procédure de programmation dynamique ne fait intervenir que des sommes des valuations des objets. L'équivalence provient donc du fait que $\pi(y)+\pi(z)=\pi(y+z)$. On a choisi la deuxième méthode car on applique π et π^{-1} moins de fois par rapport à la première.

Pour illustrer la solution implémentée dans cette partie on utilise la même instance que dans la Question 9 avec $n=10$, $m=30$, $k=3$ et $I=[\frac{1}{3},\frac{2}{3}]$. La Figure 4 montre le même nuage de points que la Figure 3 mais avec les points non I-dominés à la place des points Pareto non-dominés. On observe qu'on trouve la même solution minimax et que comme montré à la Question 10 on a bien $NI \subseteq ND$ avec une inclusion stricte dans ce cas.

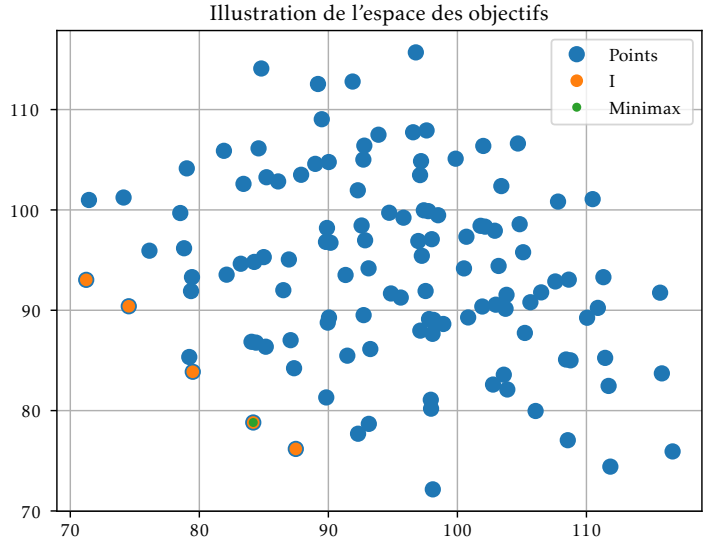

FIGURE 4 – Images des points réalisables, non I-dominés et minimax dans l'espace des objectifs

Question 12. Les tests de temps d'exécution des deux procédures ont été implémentés dans la fonction `tester_temps_alpha` qui suit les consignes de l'énoncé à part le fait que, pour obtenir des courbes plus "lisses", on fait des moyennes sur 500 instances tirées aléatoirement. Ce choix d'augmenter la quantité d'instances a été fait car le temps de calcul avec 50 instances était très faible (environ 3 secondes) et ainsi faire cette augmentation n'empêche pas les calculs dans un temps raisonnable. Les résultats obtenus sont montrés dans la Figure 5.

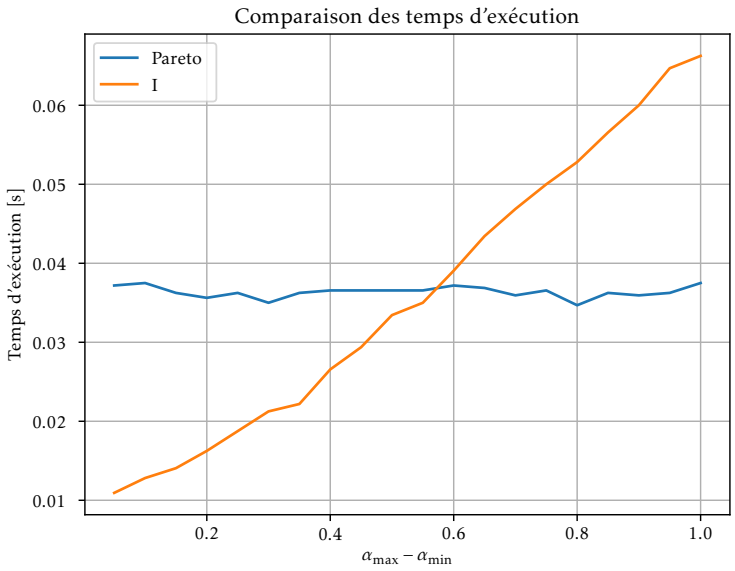

FIGURE 5 – Temps d'exécution des procédures basées sur la dominance de Pareto et la I-dominance en fonction de $\alpha_{\max} - \alpha_{\min}$

Comme on peut voir dans la figure le temps d'exécution de la procédure avec les points Pareto non-dominés reste constant et ne dépend pas de I. Cela est dû au fait que le calcul des points Pareto non-dominés par programmation dynamique ne prend pas en compte I, qui est utilisé uniquement dans le calcul d'un point minimax parmi ceux-ci dans la fonction `vec_minimax` dont le temps d'exécution est indépendant de I.

En ce qui concerne la procédure avec les points non I-dominés, on voit que le temps d'exécution croît avec la taille de I et il est plus petit que celui de l'autre procédure si I est petit, mais plus grand si I est grand. Pour expliquer cela on regarde ce qui se passe dans les cas extrêmes. Dans le cas limite $I = [\frac{1}{2}, \frac{1}{2}]$, en utilisant le résultat de la Question 11, y I-domine z si et seulement si $\frac{y_1+y_2}{2} < \frac{z_1+z_2}{2}$, ce qui donne un ordre total des points. Cela donne un très petit nombre de points non I-dominés ce qui diminue la quantité de points dans chaque case du tableau de programmation dynamique et donc diminue le temps de calcul de la procédure. Dans le cas $I = [0, 1]$ on voit, d'après le résultat de la Question 11, que la I-dominance coïncide avec la dominance de Pareto et donc le temps de calcul du tableau de programmation dynamique est le même que pour la première procédure. Comme on fait les étapes d'appliquer les transformations π et π^{-1} en plus, le temps de calcul est alors naturellement plus grand que celui de la première procédure. Ainsi, l'utilisation de la I-dominance vaut la peine lorsque l'intervalle I est petit mais perd l'intérêt quand I est grand.