CentraleSupélec

Systèmes de décision & préférences

Planification de personnel & affectation de projets

Superviseur: Mousseau Vincent

Mots clés et thèmes

Optimisation multiobjectif, Modèle de préférence, Gurobi.

Répertoire du code

Le code de notre projet se trouve sur GitHub.

Delebassée Robin, Fournier Romain & Michot Albane

6 Février, 2023

Table des matières

1 Introduction

1.1 Description

Le sujet du projet peut être récupéré en s'identifiant avec votre compte de CentraleSupélec à partir de ce lien.

La société *CompuOpti* cherche à **planifier efficacement son personnel sur les projets**. Chaque projet nécessite de staffer un certain nombre de jours/hommes sur des compétences spécifiques (optimisation, gestion de projet, developpement web, ...). Ainsi, un projet peut nécessiter 6 jours/personne de compétences A, 2 jours/personne de compétences B, et 5 jours/personne de compétences C. On considèrera que le problème se déroule sur un horizon de temps donné (on ne considèrera que les jours ouvrés). Chaque membre du personnel de *CompuOpti* possède certaines qualifications, parmi un ensemble donné de qualifications (par exemple A, B, C, D, E), et des jours de congés prédéfinis intervenant durant l'horizon de temps.

Chaque qualification intervenant dans le projet **est associé à un nombre de jours de travail** dédié à cette qualification. Par ailleurs, **chaque projet produit un gain s'il est réalisé**, et **la société cherche à maximiser le bénéfice total** induit par les projets réalisés. Pour chaque projet, une date de livraison a été négociée avec le client; **cette date ne doit pas être dépassée**, sans quoi **une pénalité financière par jour de retard est inscrite** dans le contrat de prestation.

Il s'agit donc de **définir des emplois du temps pour les membres du personnel**, c'est-à-dire d'**affecter chaque jour de travail d'un membre du personnel à une qualification d'un projet** (ou à aucune activité).

Le problème étudié est donc un **problème de planification de personnel et d'affectation de projets**.

1.2 Critères

Outre le bénéfice pour *CompuOpti*, l'entreprise veut prendre en compte d'autres aspects dans l'élaboration du planning. Voici les critères à prendre en compte :

1. **Maximiser le résultat financier** de l'entreprise et donc constituer un planning qui conduit à maximiser le bénéfice (incluant d'éventuelles pénalités).
2. Les collaborateurs n'aient pas à changer trop souvent de projet et, pour ce faire on s'attachera à **minimiser le nombre de projets sur lesquels un quelconque collaborateur est affecté**.
3. Il est important que les projets soient réalisés dans un nombre limité de jours consécutifs, ainsi on cherchera pour cela à **exécuter le projet le plus long en un minimum de jours**.

1.3 Contraintes

Dans la constitution du planning, un certain nombre de contraintes sont bien sûr à respecter :

1. Un membre du personnel ne peut être affecté à une qualification d'un projet que s'il possède cette qualification (*contrainte de qualification du personnel*).
2. A tout instant, un membre du personnel ne peut être affecté qu'à un seul projet et qu'à une seule qualification intervenant dans ce projet (*contrainte d'unicité de l'affectation quotidienne du personnel*).
3. Un membre de personnel ne peut pas être affecté à une qualification de projet un jour de congé (*contrainte de congé*).
4. Un projet n'est considéré réalisé que si tous les jours de travail dédiés à chacune des qualifications intervenant dans le projet ont ét;e couverts par des membres du personnel (*contrainte de couverture des qualifications du projet*).

5. Enfin, un projet ne peut être réalisé qu'une fois sur une période de temps donnée (*contrainte d'unicité de la réalisation d'un projet*).

1.4 Plan mis en place

Afin de répondre au mieux à la problématique, nous résolvons le problème selon la structure suivante:

1. Jeux de données
2. Première modélisation par la programmation linéaire
3. Modèle calculant la surface des solutions non-dominées
4. Modèles de préférence discriminant la surface des solutions non-dominées
5. Conclusion

2 Jeux de données

2.1 Jeux de données fournis

Nous disposons d'un ensemble de **3 instances de tailles variables au format json**. Ces instances comportent des informations sur les membres du personnel et leurs compétences, sur les temps à consacrer à chaque compétence pour chaque projet, sur les deadlines de rendu des projets ainsi que sur les pénalités journalières et le gain associé à chaque projet.

Dans toute la suite, nous utiliserons les notations suivantes pour faire référence à certaines données issues des instances :

- N est le nombre de membres du personnel
- M est le nombre de projets
- T est le nombre de jours dans l'horizon considéré
- K est le nombre de compétences distinctes intervenant dans les projets

Le tableau ci-dessous recense les valeurs de ces 4 données pour les 3 jeux de données fournis :

instance	N	M	K	T
toy	3	5	3	5
medium	5	15	10	22
large	6	28	10	36

2.2 Jeux de données élaborés

Nous avons également créé une **fonction** nous **permettant de générer de nouvelles instances**. Cela pourra notamment nous être utile pour tester l'influence de différents paramètres sur le classement fourni par notre système d'inférence de préférences. Nous pourrons en effet observer son comportement sur des instances créées pour être difficiles à optimiser sur un des critères.

Par exemple, nous avons rendu **difficile l'optimisation sur la durée maximale des projets** en faisant en sorte que ces derniers aient des durées variables et ne soient pas réalisables en seulement 1 jour. Cela est possible en choisissant des projets ayant une demande de jour/consultant supérieure au nombre de membres du personnel.

Néanmoins c'est une approximation assez grossière qui consiste à prendre des projets volontairement très lourds. Si on veut affiner cette approximation en obtenant toujours des projets non réalisables en un jour, on peut s'intéresser à la **capacité de travail de l'entreprise par compétence**. Cela revient, au moment de la création de l'instance, à garder en mémoire le nombre de consultants ayant

chaque capacité. Lors de la création d'un projet, une fois que l'on a choisi quelles compétences allaient intervenir dans celui-ci, on fixe le nombre de jours à travailler en fonction de la somme des capacité de travail pour chaque compétence concernée.

3 Première modélisation par la programmation linéaire

Dans un premier temps, nous souhaitons calculer l'ensemble des solutions non dominées du problème. Commençons donc par formuler le problème sous forme de programme linéaire.

3.1 Données de l'instance

Les données ci-dessous peuvent être extraites d'une instance :

$$c_{ik} = \begin{cases} 1 \text{ si le membre du personnel i a la compétence k} \\ 0 \text{ sinon} \end{cases} \tag{1}$$

$$w_{it} = \begin{cases} 1 \text{ si le membre du personnel i travaille le jour t} \\ 0 \text{ sinon} \end{cases} \tag{2}$$

t_{jk} est le temps à allouer à la compétence k dans le projet j.
g_j est le gain obtenu si le projet j est réalisé.
p_j est la pénalité infligée si la réalisation du projet j a un jour de retard.

$$d_{jt} = \begin{cases} 1 \text{ si l'instant t est postérieur à la date planifiée de livraison du projet j} \\ 0 \text{ sinon} \end{cases} \tag{3}$$

3.2 Variables de décision

On peut alors définir les variables de décision suivantes pour notre problème :

$$x_{ijkt} = \begin{cases} 1 \text{ si la compétence k du projet j est réalisée par i à l'instant t} \\ 0 \text{ sinon} \end{cases} \tag{4}$$

$$r_{jt} = \begin{cases} 1 \text{ si l'instant t est postérieur à la date de terminaison effective du projet} \\ 0 \text{ sinon} \end{cases} \tag{5}$$

$$s_{ij} = \begin{cases} 1 \text{ si le membre du personnel i intervient dans le projet j} \\ 0 \text{ sinon} \end{cases} \tag{6}$$

$$b_{jt} = \begin{cases} 1 \text{ si l'instant t est postérieur à la date de début du projet j} \\ 0 \text{ sinon} \end{cases} \tag{7}$$

Pour linéariser les min/max qui apparaissent lors de l'étape d'expression des objectifs, on introduit de plus les deux variables suivantes :

$y = \max\limits_{1 \leq i \leq N} \sum_{j=1}^{M} s_{ij}$
$z = \max\limits_{1 \leq j \leq M} \sum_{t=1}^{T} (b_{jt} - r_{jt})$

3.3 Contraintes

Enfin, nous pouvons passer à la définition des contraintes de notre programme linéaire :

$$\forall i, \forall j, \forall k, \forall t,\ x_{ijkt} \leq\ c_{ik}\ (\textit{contrainte de qualification du personnel}) \tag{8}$$

$$\forall t, \forall i,\ \sum_{j=1}^{M}\sum_{k=1}^{K} x_{ijkt} \leq 1\ (\textit{contrainte d'unicité de l'affectation quotidienne du personnel}) \tag{9}$$

$$\forall i, \forall j, \forall k, \forall t,\ x_{ijkt} \leq\ w_{it}\ (\textit{contrainte de congé}) \tag{10}$$

$$\forall j, \forall t, \forall k\ \sum_{\tau=1}^{t}\sum_{i=1}^{N} x_{ijk\tau} \geq r_{jt}t_{jk}\ (\textit{contrainte de couverture des qualifications}) \tag{11}$$

$$\forall j, \forall t, \forall k\ \sum_{\tau=1}^{t}\sum_{i=1}^{N} x_{ijk\tau} \leq t_{jk}\ (\textit{contrainte d'unicité de la réalisation d'un projet}) \tag{12}$$

$$\forall j, \forall t, b_{jt} \geq r_{jt}\ (\textit{le début d'un projet est avant sa fin}) \tag{13}$$

$$\forall j, \forall t, NKb_{jt} \geq \sum_{j=1}^{N}\sum_{k=1}^{K} x_{ijkt} \tag{14}$$

(*le projet est commencé si un membre du staff a travaillé sur une compétence de ce projet*)

$$\forall j, \forall t, b_{jt} \leq \sum_{j=1}^{N}\sum_{k=1}^{K} x_{ijkt} + b_{j,t-1} \tag{15}$$

(*si le projet n'était pas commencé à t-1 et que personne n'a travaillé dessus à t, il n'est toujours pas commencé*)

$$\forall i, \forall j, KTs_{ij} \geq \sum_{k=1}^{K}\sum_{t=1}^{T} x_{ijkt} \tag{16}$$

(*une personne est affectée à un projet ssi elle a été affectée à une compétence de ce projet au moins un jour*)

$$\forall i, \forall j, s_{ij} \leq \sum_{k=1}^{K}\sum_{t=1}^{T} x_{ijkt} \tag{17}$$

(*projet non affecté à un membre du staff s'il n'a jamais travaillé dessus*)

$$\forall i, y \geq \sum_{j=1}^{M} s_{ij}\ (\textit{contrainte de linéarisation de l'objectif 2}) \tag{18}$$

$$\forall j, z \geq \sum_{t=1}^{T}(b_{jt} - r_{jt})\ (\textit{contrainte de linéarisation de l'objectif 3}) \tag{19}$$

$$\forall j, \forall t,\ b_{j,t} \leq b_{j,t+1}(\textit{contrainte interne liée à la définition de la variable } b) \tag{20}$$

$$\forall j, \forall t,\ r_{j,t} \leq r_{j,t+1}(\textit{contrainte interne liée à la définition de la variable } r) \tag{21}$$

3.4 Fonctions objectifs

Les 3 fonctions objectifs à optimiser peuvent être formulées de la façon suivante :

$$f_1 = \sum_{j=1}^{M} r_{jT}(g_j - \sum_{t=1}^{T} p_j(1 - r_{jt})d_{jt}) \text{ (\textit{optimisation du résultat financier})} \tag{22}$$

$$f_2 = y \text{ (\textit{optimisation du nombre de projets par employé})} \tag{23}$$

$$f_3 = z \text{ (\textit{optimisation de la durée d'un projet})} \tag{24}$$

On désire maximiser f_1 mais minimiser f_2 et f_3.

4 Modèle calculant la surface des solutions non-dominées

Comme le problème est un **problème de programmation linéaire entière multiobjectif**, nous avons décidé d'**appliquer une méthode de type ϵ-constraint avec des pas entiers**.

Comme il y a 3 objectifs à optimiser, nous avons dû généraliser l'algorithme vu dans le cours pour le cas bi-objectif. Voici la procédure que nous avons employée pour le cas tri-objectif :

Initialisation :

- Calcul du point idéal par résolution des trois problèmes mono-objectifs sans contrainte de type ϵ-constraint. $z^* = (\min_{x \in X} -f_1(x), \min_{x \in X} f_2(x), \min_{x \in X} f_3(x))$
- Choix d'un "pire point" pour fixer la borne supérieure de variation de $f_2(x)$ et $f_3(x)$ (on a choisi respectivement N et T car un membre du staff peut être affecté a au plus N projets et un projet peut durer au plus T jours). $\epsilon \leftarrow (0, N, T)$
- résolution avec Gurobi du problème mono-objectif $\min_{x \in X} -f_1(x)$ avec les contraintes $f_2(x) \leq \epsilon_2$ et $f_3(x) \leq \epsilon_3$. Soit z la solution obtenue, $ND(Y) \leftarrow \{z\}$.

Tant que $\min\{f_1(x); f_2(x) \leq \epsilon_2 \text{ et } f_3(x) \leq \epsilon_3\}$ **est soluble et** $z_1 > z_1^*$ **:**

Tant que $\min\{f_1(x); f_2(x) \leq \epsilon_2 \text{ et } f_3(x) \leq \epsilon_3\}$ **est soluble et** $z_2 > z_2^*$ **:**

Soit z' la solution de $\min\{f_1(x) + \alpha f_3(x);\ f_2(x) \leq \epsilon_2 \text{ et } f_3(x) \leq \epsilon_3\}$ avec α arbitrairement petit.

$ND(Y) \leftarrow ND(Y) \bigcup \{z\}$

$z \leftarrow z'$

$\epsilon_3 \leftarrow z'_3 - 1$

$\epsilon_2 \leftarrow \epsilon_2 - 1$

$\epsilon_3 \leftarrow T$

Filtrage de ND(Y) pour ne conserver que les solutions non-dominées.

En appliquant cette méthode, nous avons pu déterminer les surfaces de pareto des 3 instances à notre disposition. Le tableau ci-dessous résume le nombre de solutions non-dominées trouvé pour chaque instance :

instance	taille surface pareto
toy	10
medium	45
large	100

5 Modèles de préférence discriminant la surface des solutions non-dominées

Soit $A = \{a_1, ..., a_n\}$ l'ensemble des alternatives non dominées et $A^* = \{a_1, ..., a_m\} \subset A$ l'ensemble des alternatives classifiées par le décideur dans des catégories C_1 (solutions satisfaisantes), C_2 (solutions correctes) et C_3 (solutions inacceptables).

5.1 Méthode de surclassement basée sur la comparaison d'alternatives

On note $C(a)$ la catégorie à laquelle une alternative $a \in A$ est affectée. Nous allons considérer la relation de surclassement définie par $a \succsim b \Leftrightarrow \sum_{\{i:g_i(a)\geq g_i(b)\}} w_i \geq \lambda$ où $w_i \in [0;1]$ est le poids normalisé associé au critère i et $\lambda \in [0,5;1]$ est un seuil de majorité.

Plusieurs choix de méthodes d'affectation sont envisageables, mais nous avons retenu le principe d'affectation $C(a) < C(b) \implies b \succ a$ car le même principe avec un $\succsim$ au lieu d'un $\succ$ impliquerait que deux actions incomparables sont dans la même classe.
Ainsi, si deux alternatives sont affectées à deux classes distinctes, mettons $a_1 \in C_1$ et $a_2 \in C_2$, on peut en déduire : $C(a_1) > C_(a_2) \implies a_1 \succ a_2$, ce qui ajoutera deux contraintes aux programme linéaire, à savoir $\sum_{\{i:g_i(a_1)\geq g_i(a_2)\}} w_i \geq \lambda$ $(a_1 \succsim a_2)$ et $\sum_{\{i:g_i(a_2)\geq g_i(a_1)\}} w_i < \lambda$ $(\neg a_2 \succsim a_1)$.

Puisque les actions de C_1 sont préférées à celles de C_2, elles mêmes étant préférées à celles de C_3, le programme linéaire à résoudre est :

$$
\begin{aligned}
& & \min .\ \lambda & \\
s.t. & \sum_{\{i:g_i(a_1)\geq g_i(a_2)\}} & w_i \geq \lambda \ \ \forall a_1 \in C_1,\ \forall a_2 \in C_2 & \\
& \sum_{\{i:g_i(a_2)\geq g_i(a_1)\}} & w_i < \lambda \ \ \forall a_1 \in C_1,\ \forall a_2 \in C_2 & \\
& \sum_{\{i:g_i(a_1)\geq g_i(a_3)\}} & w_i \geq \lambda \ \ \forall a_1 \in C_1,\ \forall a_3 \in C_3 & \\
& \sum_{\{i:g_i(a_3)\geq g_i(a_1)\}} & w_i < \lambda \ \ \forall a_1 \in C_1,\ \forall a_3 \in C_3 & \\
& \sum_{\{i:g_i(a_2)\geq g_i(a_3)\}} & w_i \geq \lambda \ \ \forall a_2 \in C_2,\ \forall a_3 \in C_3 & \\
& \sum_{\{i:g_i(a_3)\geq g_i(a_2)\}} & w_i < \lambda \ \ \forall a_2 \in C_2,\ \forall a_3 \in C_3 & \\
& & 0.5 \leq \lambda \leq 1 & \\
& & \sum_i w_i = 1 & \\
& & w_i \geq 0 \ \ \forall i &
\end{aligned}
\tag{25}
$$

En écrivant les contraintes pour les alternatives déjà classifiées par le décideur, on obtient donc un polyèdre de solutions acceptables qui nous indique que ces affectations ne sont pas incohérentes s'il est non vide. Pour chaque alternative non classifiée, on teste alors l'hypothèse selon laquelle cette alternative appartient à C_1, C_2 ou C_3. Si l'ajout des nouvelles contraintes liées à l'hypothèse rend le programme liénaire insoluble (polyèdre vide), on rejette l'hypothèse. Sinon, si l'optimum λ^* est égal à 0.5, cela signifie que l'hypothèse selon laquelle une nouvelle alternative a appartient à une classe C(a)

n'impose rien sur le seuil de majorité, donc a peut être affecté à la catégorie C(a). Si $0.5 < \lambda^* < 1$, cela signifie que la seule façon de pouvoir affecter l'alternative a à la classe C(a) est de fixer un seuil de majorité supérieur à λ^*. Le décideur peut alors trancher en disant s'il trouve que le seuil à fixer est trop élevé ou pas.

Grâce à cette méthode, on peut donc classer les actions dans des groupements de classes qui peuvent être définis ainsi :

- les actions pouvant appartenir aux 3 classes (peu de sûreté de l'affectation)
- les actions inacceptables et/ou correctes
- les actions correctes et/ou satisfaisantes
- les actions satisfaisantes

5.2 Méthode basée sur la théorie de l'utilité et la somme pondérée

On note g_{i*} et g_i^* les valeurs respectivement minimale et maximale du i-ème critère sur les alternatives de A. On divise alors chaque intervalle $[g_{i*}, g_i^*]$ en $\alpha_i - 1$ intervalles de longueurs égales. Dans l'idéal, on souhaiterait que chaque sous-intervalle contienne le même nombre d'alternatives pour qu'ils encodent la même quantité d'information. Soit h_i le nombre de valeurs distinctes du i-ème critère sur les alternatives de A. Si on suppose que ces valeurs sont distribuées selon une loi uniforme sur $[g_{i*}; g_i^*]$, l'espérance du nombre de points distincts par intervalle est alors $\frac{h_i}{\alpha_i - 1}$. On peut donc choisir α_i comme $\lfloor \frac{h_i}{c_i} \rfloor + 1$ avec c_i le nombre de points par sous intervalle désiré. Or, les nombres de valeurs distinctes par critère peuvent varier grandement d'un critère à l'autre. Pour éviter une grande variation du nombre de degrés de liberté selon les différents critères, on fait le choix de ne pas prendre un c_i constant mais dépendant de h_i : $c_i = \lfloor \sqrt{h_i} \rfloor$.

Chaque g_i^j peut alors être calculé par interpolation :

$$\forall j \in [1; \alpha_i],\ g_i^j = g_{i*} + \frac{j-1}{\alpha_i - 1}(g_i^* - g_{i*}) \tag{26}$$

Supposons que l'évaluation d'une alternative a sur le i-ème critère est telle que $g_i(a) \in [g_i^j, g_i^{j+1}]$, alors l'utilité marginale de l'alternative a vérifie :

$$u_i[g_i(a)] = u_i(g_i^j) + \frac{g_i(a) - g_i^j}{g_i^{j+1} - g_i^j}(u_i(g_i^{j+1}) - u_i(g_i^j)) \tag{27}$$

Pour garantir la monotonicité du critère, la contrainte suivante doit être satisfaite :

$$\forall i \in [1,3], \forall j \in [1; \alpha_i - 1]\ u_i(g_i^{j+1}) - u_i(g_i^j) > 0 \tag{28}$$

En posant $w_{ij} = u_i(g_i^{j+1}) - u_i(g_i^j)$, on peut donc écrire :

$$\begin{cases} \forall i, \forall j,\ w_{ij} > 0 \\ \forall i,\ u_i(g_{i*}) = 0 \\ \forall i \forall j,\ u_i(g_i^j) = \sum_{k=1}^{j-1} w_{ik} \end{cases} \tag{29}$$

On suppose que l'utilité globale d'une alternative a a une forme additive :

$$U(a) = \sum_{i=1}^{3} u_i(g_i(a)) \tag{30}$$

La classification des alternatives est obtenue en comparant l'utilité à des profils limite d'utilité u_1 et u_2 qui permettent de distinguer les classes :

$U(a) \geq u_1 \implies a \in C_1$
$u_2 \leq U(a) < u_1 \implies a \in C_2$
$U(a) < u_2 \implies a \in C_3$

En notant respectivement $\sigma^+(a)$ et $\sigma^-(a)$ les erreurs de surestimation et de sous-estimation d'une alternative a, on obtient le programme linéaire suivant, dont l'objectif est de minimiser la somme de toutes les erreurs :

$$
\begin{aligned}
&\min . \sum_{a \in A^*} \sigma^+(a) + \sigma^-(a) \\
&s.t. \sum_{i=1}^{3} u_i(g_i(a)) - u_1 + \sigma^+(a) \geq 0 \quad \forall a \in C_1 \\
&\sum_{i=1}^{3} u_i(g_i(a)) - u_1 - \sigma^-(a) \leq -\delta \quad \forall a \in C_2 \\
&\sum_{i=1}^{3} u_i(g_i(a)) - u_2 + \sigma^+(a) \geq 0 \quad \forall a \in C_2 \\
&\sum_{i=1}^{3} u_i(g_i(a)) - u_2 - \sigma^-(a) \leq -\delta \quad \forall a \in C_3 \\
&\sum_{i=1}^{3} \sum_{j=1}^{a_i - 1} w_{ij} = \sum_{i=1}^{3} u_i(g_i^*) = 1 \\
&u_1 - u_2 \geq s \\
&w_{ij} \geq 0, \ \sigma^+(a) \geq 0, \ \sigma^-(a) \geq 0
\end{aligned}
\tag{31}
$$

où δ est arbitrairement petit et s est un sueil caractérisant la relation de préférence stricte entre les seuils d'utilité qui distinguent les classes. Gardons également en tête que les utilités peuvent s'obtenir par interpolation en fonction des w_{ij}:

$$u_i(g_i(a)) = \sum_{k=1}^{j-1} w_{ik} + \frac{g_i(a) - g_i^j}{g_i^{j+1} - g_i^j} w_{ij} \text{ si } g_i(a) \in [g_i^j ; g_i^{j+1}] \tag{32}$$

En résolvant ce programme linéaire, on trouve les profils d'utilité, les valeurs des utilités de chaque alternative pour chaque critère ainsi que les erreurs de sur et de sous estimation. Les erreurs permettent d'évaluer la qualité de la classification effectuée par le modèle sur les alternatives déjà classifiées par le décideur tandis que les valeurs d'utilité et les profils vont nous permettre de classifier les alternatives pas encore classifiées par le décideur (par interpolation et comparaison au profil).

Nous avons également essayé de minimiser non pas les erreurs de classifications mais le nombre de mauvaises classifications car cela semblait donner de meilleurs résultats en pratique. Pour cela, il suffit de remplacer les variables continues $\sigma^+(a)$ et $\sigma^-(a)$ par des variables binaires $M^+(a)$ et $M^-(a)$ telles que $M^+(a) = 1$ si a est affecté à une classe plus faible que sa classe effective et $M^-(a) = 1$ si a est affecté à une classe plus forte que sa classe effective.

6 Conclusion

6.1 Problèmes rencontrés

Nous nous sommes aperçus que le calcul de la surface de Pareto devenait vite assez long pour les instances moyenne et large. En investiguant les logs du solveur de Gurobi, nous nous sommes rendus compte que pour certains jeux de contraintes, le solveur arrivait rapidement à un MIPGap de 5% mais penait à arriver à un MIPGap de 0.5% (limite par défaut que le solveur cherche à atteindre). Pour éviter que le calcul de la surface de Pareto ne soit trop long, nous avons décidé de limiter le temps

d'optimisation en fixant une borne supérieure à celui-ci (2 min). On aurait également pu changer le MIPGap de référence à atteindre, mais dans ce cas, on aurait limité la qualité des solutions calculées rapidement (le solveur se serait arrêté à 5% au lieu de 0.5%).
En limitant le temps d'optimisation, on obtient donc une **approximation** de la surface des solutions efficaces, mais nous avons considéré que cela était suffisant dans le cadre de notre problème.

De plus, sur la toy instance, les solutions efficaces ont quasiment toujours une valeur égale à 0 ou 1 sur le troisième critère (il est souvent possible de réaliser un projet en un jour si tout le staff se consacre à ce projet). Dans ce cas, le modèle d'élicitation des préférences basé sur la comparaison des alternatives a du mal à affecter une alternative à une classe avec certitude puisqu'il est facile de trouver un jeu de paramètres compatible avec chacune des clasess.

6.2 Ouverture

La méthode basée sur la relation de surclassement et la comparaison des alternatives a besoin de plus d'exemples que la méthode basée sur la théorie de l'utilité pour inférer des affectations d'alternatives à des classes de manière non ambigue. Cependant, nous avons testé cette méthode sur un exemple simulant l'affectation faite par le décideur par une affectation reposant principalement sur le premier critère. Ainsi, la somme pondérée est capable d'inférer précisément les préférences du décideur car les profils limites sont clairement définis et dépendent beaucoup de l'utilité des alternatives sur le premier critère. Cependant, l'avantage de la méthode basée sur la relation de surclassement est qu'en cas d'affectations plus complexes faites par le décideur (notamment si on avait plus de critères), elle permet d'avoir une marge de sûreté et de mettre de côté les alternatives les moins bonnes. Au contraire, la somme pondérée n'affecte qu'une classe par exemple et est donc plus radicale.

De plus, il faut garder en tête que sur des exemples plus complexes (plus de 3 critères), la complexité calculatoire de la méthode basée sur la théorie de l'utilité peut vite devenir grande si on approxime chaque dimension par un nombre de morceaux linéaires relativement grand. La complexité de la méthode basée sur le surclassement, quant à elle, dépend plus du nombre d'alternatives à classifier que de la dimension de l'espace des critères !

Enfin, la scalabilité du code par rapport à la taille des instances est un point important à considérer pour le déploiement du produit informatique.