Rapport du projet de systèmes de prise de décision

Gillian Keusch, Nicolas Noblot, Antoine Gorceix

16 Décembre 2022

Sommaire

1 Introduction

L'entreprise *CompuOpti* veut affecter son personnel à des projets en fonction de leurs compétences de manière à optimiser plusieurs critères. Le but de ce projet est d'élaborer un algorithme qui produit un emploi du temps convenable.

2 Modélisation et mise sous forme d'un programme linéaire

2.1 Notations

On notera N le nombre de personnes, M le nombre de projets, C le nombre de compétences existantes et T la période considérée en jours. On note $k_{max} \in \{1, \ldots, M\}$ l'indice du projet avec la plus grande charge de travail.

2.2 Instance

Une instance correspond à une entreprise et est composée de :

- une matrice $Comp$ de dimensions (N, C) définie par: $Comp_{i,j} = 1$ si l'employé i a la qualification j et 0 sinon.
- Une matrice $Proj$ de dimensions (M, C) telle que $Proj_{i,j}$ correspond au nombre de jours pour le projet i et la compétence j.
- Une matrice $Cong$ de dimensions (N, T) telle que $Cong_{i,j} = 1$ si l'employé i est en congé le jour j, 0 sinon.
- une liste de deadlines par projet $(d_i)_{1 \leq i \leq M}$.
- une liste de gains $(g_i)_{1 \leq i \leq M}$
- une liste de pénalités par jour de retard $(r_i)_{1 \leq i \leq M}$

2.3 Variables de décision

Les variables de décision utilisées sont :

- un tenseur X de dimensions (N, C, M, T) tel que $X_{i,j,k,l} = 1$ si l'employé i travaille sur le projet j en exploitant la compétence k le jour l et 0 sinon.
- une matrice Y de taille (M, T) définie par : $Y_{i,j} = 1$ si le projet i est dans l'état "fini" le jour j et 0 sinon. Pour un projet i quelconque, $Y_{i,j}$ vaudra 1 si j est supérieur à l'indice du jour auquel le projet est complètement terminé et 0 sinon.
- une matrice S de taille (M, T) définie par $S_{i,j} = 1$ si le projet i est dans l'état "commencé" au jour j et 0 sinon. Pour un projet i quelconque, $S_{i,j}$ vaudra 1 si j est supérieur à l'indice du jour auquel le projet est commencé et 0 sinon.
- une matrice Z de taille (N, M) où Z_{ik} vaut 1 si l'employé i participe au projet k et 0 sinon.
- une variable entière p qui représente le nombre maximal de participations d'un employé à des projets. p varie entre 1 et M.

2.4 Contraintes

2.4.1 Contraintes sur l'emploi du temps

- Contrainte de qualification du personnel :

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,C\}, \forall k \in \{1,\dots,M\}, \forall l \in \{1,\dots,T\}, X_{i,j,k,l} \leq Comp_{i,k}$$

- Contrainte d'unicité de l'affectation quotidienne du personnel :

$$\forall i \in \{1,\dots,N\}, \forall l \in \{1,\dots,T\}, \sum_{j=1}^{M}\sum_{k=1}^{C} X_{i,j,k,l} \leq 1$$

- Contrainte de congé :

$$\forall i \in \{1,\dots,N\}, \forall j \in \{1,\dots,C\}, \forall k \in \{1,\dots,M\}, \forall l \in \{1,\dots,T\}, X_{i,j,k,l} \leq 1 - Cong_{i,l}$$

- Contrainte d'unicité de la réalisation d'un projet :

$$\forall j \in \{1,\dots,C\}, \forall k \in \{1,\dots,M\}, \sum_{i=1}^{N}\sum_{l=1}^{T} X_{i,j,k,l} \leq Proj_{j,k}$$

2.4.2 Relations entre les variables

- Contraintes de couverture des qualifications d'un projet : autrement dit, un projet n'est terminé que si toutes les tâches qui lui sont associées sont terminées. On introduit une constante δ arbitrairement petite.

$$\forall k \in \{1,\dots,M\}, \forall d \in \{1,\dots,T\}, Y_{k,d} \leq \frac{\sum_{i=1}^{N}\sum_{j=1}^{C}\sum_{l=1}^{d} X_{i,j,k,l}}{\sum_{j=1}^{C} Proj_{k,j}}$$

$$\forall k \in \{1,\dots,M\}, \forall d \in \{1,\dots,T\}, \sum_{i=1}^{N}\sum_{j=1}^{C}\sum_{l=1}^{d} X_{i,j,k,l} - \sum_{j=1}^{C} Proj_{k,j} + \delta \leq Y_{k,d}$$

- Contraintes sur le début d'un projet : un projet n'est commencé que si au moins un employé a déjà travaillé dessus. On introduit une constante K_1 arbitrairement grande.

$$\forall k \in \{1,\dots,M\}, \forall d \in \{1,\dots,T\}, S_{k,d} \leq \frac{\sum_{i=1}^{N}\sum_{j=1}^{C}\sum_{l=1}^{d} X_{i,j,k,l}}{\sum_{j=1}^{C} Proj_{k,j}}$$

$$\forall k \in \{1,\dots,M\}, \forall d \in \{1,\dots,T\}, \frac{1}{K_1}\sum_{i=1}^{N}\sum_{j=1}^{C}\sum_{l=1}^{d} X_{i,j,k,l} \leq S_{k,d}$$

- Contraintes sur la participation à un projet : un employé participe à un projet que s'il a travaillé sur une tâche de ce projet. On introduit une constante K_2 arbitrairement grande.

$$\forall i \in \{1,\dots,N\}, \forall k \in \{1,\dots,M\}, Z_{i,k} \leq \sum_{j=1}^{C}\sum_{l=1}^{T} X_{i,j,k,l}$$

$$\forall i \in \{1,\dots,N\}, \forall k \in \{1,\dots,M\}, \frac{1}{K_2}\sum_{j=1}^{C}\sum_{l=1}^{T} X_{i,j,k,l} \leq Z_{i,k}$$

- Contraintes de maximalité du nombre de changements de projet p :

$$\forall i \in \{1, \ldots, N\}, \sum_{k=1}^{M} Z_{i,k} \leq p$$

- Contraintes de maximalité de la durée du projet avec la plus grande charge de travail :

$$\forall k \in \{1, \ldots, M\}, \sum_{d=1}^{T} (S_{k,d} - Y_{k,d}) \leq \sum_{d=1}^{T} (S_{k_{max},d} - Y_{k_{max},d})$$

2.5 Objectifs

Les objectifs sont les suivants :

- Maximiser le résultat financier α :

$$\alpha = \sum_{k=1}^{M} \left(g_k Y_{k,T} - r_k \left((T - d_k + 1) Y_{k,T} - \sum_{d=d_k}^{T} Y_{k,d} \right) \right)$$

  Les projets qui ne sont pas réalisés à la fin de l'horizon de temps n'entraînent aucune pénalité.

- Minimiser le nombre de projets par collaborateur. Cet objectif est équivalent à minimiser la variable p.

- Réaliser le projet le plus long sur un minimum de jours consécutifs. Cet objectif revient à minimiser la variable durée du projet avec la plus grande charge de travail t :

$$t = 1 + \sum_{d=1}^{T} (S_{k_{max},d} - Y_{k_{max},d})$$

2.6 Résolution du problème d'optimisation

On résout le problème d'optimisation de l'emploi du temps à l'aide de la méthode ϵ-contrainte. On utilise le programme linéaire suivant:

$$\max \alpha$$

Les contraintes sont celles de la section 2.4 auxquelles on ajoute deux contraintes sur les deux autres fonctions-objectif:

- $$p \leq \epsilon_1$$

- $$t \leq \epsilon_2$$

Les scalaires ϵ_1 et ϵ_2 sont des entiers qui sont choisis respectivement dans $\{1, \ldots, M\}$ et $\{1, \ldots, T\}$. Pour obtenir toutes les solutions, on exécute le programme linéaire pour tous les couples (ϵ_1, ϵ_2) possibles. On filtre ensuite les solutions trouvées pour ne garder que les solutions non-dominées. La résolution a été faite avec le solveur Gurobi. Dans la pratique, l'algorithme a été exécuté sur des instances de taille moyenne mais qui prennent plusieurs heures pour être calculées. On a donc limité le temps de résolution de Gurobi à 30 secondes afin d'obtenir des solutions dans un temps plus raisonnable. Ceci signifie également que les solutions trouvées ne sont que des **approximations** des solutions optimales.

3 Modèles de préférences pour classifier l'ensemble des solutions

3.1 La méthode UTA

3.1.1 Modélisation du score

La méthode UTA permet de faire une sélection parmi l'ensemble de solutions de la surface de Pareto obtenues dans la partie précédente en résolvant notre modélisation du problème de staffing. L'approche réalisée ici est un peu différente de celle vu en TD dans le cadre du cours de SDP. En effet plutôt que d'établir un classement des solutions entre elles, nous allons les classifier en 3 catégories : *Solution non-acceptable, Solution neutre et Solution satisfaisante.*
Pour cela nous utilisons une modélisation assez simple en construisant un score : $s(j) = \sum_{i=1}^{3} s_i(x_{i,j})$ avec $s_i(x_{i,j})$ l'évaluation d'une de nos solutions j sur le critère i. i varie entre 1 et 3 car les solutions sont sous forme de triplets avec 3 critères : *Profit, Nombre de projets réalisés, Durée du projet le plus long.*

s_i est une fonction monotone non décroissante, telle que $s_i(min_i) = 0$ et $s_i(max_i) = w_i$ où w_i est le "poids" du critère i (avec $\sum_{i=1}^{3} w_i = 1$), et $[min_i, max_i]$ est l'échelle d'évaluation du critère i. Ce score total $s(j)$ est défini de telle sorte que :

- si $s(j) \leq coef_1$ alors la solution j est non-acceptable.
- si $coef_1 < s(j) \leq coef_2$ alors la solution j est neutre.
- si $coef_2 < s(j)$ alors la solution j est satisfaisante.

Les coefficients $coef_1$ et $coef_2$ définissent les bornes des intervalles qui permettent de classifier les différentes solutions. L'idée principale de l'UTA est d'apprendre les différentes fonctions s_i sur un pré-odre établi sur une petite partie des instances, ici donc une pré-classification. Les s_i sont modélisées comme des fonctions affines par morceaux continues. Plus précisément L_i morceaux de fonction affines, L_i étant un paramètre à fixer. L'échelle $[min_i, max_i]$ du critère i décompose l'intervalle L_i en sous-intervalles égaux $[x_i^0, x_i^1]$, $[x_i^1, x_i^2]$, ... $[x_i^{L-1}, x_i^{L_i}]$ avec $x_i^0 = min_i$ et $x_i^L = max_i$. Ainsi, les fonctions $s_i()$ étant affines par morceaux, elles sont entièrement définies par la valeur de la fonction $s_i()$ au niveau des "points d'arrêt", $s_i(x_i^0)$, $s_i(x_i^1)$, ... $s_i(x_i^{L_i})$.

Ainsi d'après ce qui précède, maintenant que l'on a modélisé notre problème, on a l'expression des x_i^k et on peut calculer les $s(x_{i,j})$.
On a $\forall k \in \{0, \ldots, L_i\}$, $x_i^k = min_i + \frac{k}{L_i}(max_i - min_i)$.
Et si $x_{i,j}$ est dans l'intervalle $[x_i^{k+1}, x_i^k]$ alors on a $s(x_{i,j}) = s_i(x_i^k) + \frac{x_{i,j} - x_i^k}{x_i^{k+1} - x_i^k}(s_i(x_i^{k+1}) - s_i(x_i^k))$.

3.1.2 Modélisation du problème d'optimisation

Nous avons établi la forme de notre score. Il s'agit à présent d'établir le problème d'optimisation qui va nous permettre d'apprendre les $s_i(x_i^k)$ sur une petite partie des instances que l'on a pré-classifié. Pour cela on introduit une erreur relative σ_j au score attribué à un de nos triplet solution. On a $s(j)' = \sum_{i=1}^{3} s_i(x_{i,j}) + \sigma_j$ On écrit un programme linéaire qui minimise la somme des erreurs associées à nos triplets solutions. On décompose les σ_j en σ_j^+ et σ_j^- afin de ne pas avoir à introduire des valeurs absolues. Le programme doit inclure des contraintes qui spécifient notre pré-classification sur notre sous ensemble d'instance de solution, la monotonie des fonctions $s_i()$, les contraintes de normalisation $s(j) \in [0, 1]$ ainsi que des contraintes sur nos coefficients $coef_1$ et $coef_2$. On arrive ainsi au problème suivant : Minimizer l'objectif $z = \sum_j \sigma_j^+ + \sigma_j^-$ selon les contraintes :

- $s(j) - \sigma_j^+ + \sigma_j^- \leq coef_1$ si j est une solution non-acceptable
- $\epsilon + coef_1 \leq s(j) - \sigma_j^+ + \sigma_j^- \leq coef_2$ si j est une solution neutre
- $\epsilon + coef_2 \leq s(j) - \sigma_j^+ + \sigma_j^-$ si j est une solution est satisfaisante
- $\epsilon \leq s_i(x_i^{k+1}) - s_i(x_i^k)\ \forall k \in \{0, \ldots, L_i - 1\}$, Monotonie des $s_i()$
- $s_i(x_i^0) = 0\ \forall i$
- $\sum_i s_i(x_i^{L_i}) = 1$
- $\epsilon \ \leq \ coef_1$

Le modèle paraît pertinent en effet si la somme des erreurs est nulle, cela signifie qu'il existe une fonction de valeur additive qui est capable de représenter pleinement notre pré-classification sur notre sous ensemble de solutions. En revanche, si cette fonction objectif croît, nous nous éloignons de la d'une représentation complète de notre pré-classification.

3.1.3 Résolution et classification des solutions

Il y a plusieurs possibilités pour la résolution. Tout d'abord nous commençons avec un cas simplifié : On fixe les valeurs $coef_1 = 0.33$ et $coef_2 = 0.66$ afin de diviser l'ensemble des valeurs possibles $[0, 1]$ en trois segment de même taille. En fonction du segment auquel appartient le score de notre triplet on pourra considérer qu'il s'agit d'une solution satisfaisante ($\in]0.66, 1]$), neutre ($\in]0.33, 0.66]$), non-acceptable ($\in \ [0, 0.33]$). Par ailleurs on considère pour commencer $\forall i \ L_i = 3$ et ainsi que chacune des $s_i()$ est composée de 3 morceaux de fonctions affines.

Comment établir notre pré-classification et le sous-ensemble sur lequel travailler ?

On commence par sectionner toutes les instances dont les critères (Profit, Nombre de projets réalisés, Durée du projet le plus long) prennent les valeurs extrêmes. En effet les $s_i()$ sont définis sur $[min_i, max_i]$, si l'on veut qu'ils soient définis partout, il faut donc inclure dans notre sous-ensemble les valeurs extrêmes de chaque critère i. Cela nous impose ainsi entre 2 et 6 instances dans notre sous-ensemble de départ.

Pour la méthode de classification, nous avons considéré qu'elle était subjective et dépendait des objectifs du décideur. Afin d'établir notre pré-classification, nous avons appliqué les règles de classification suivante : Le profit prime pour déterminer la catégorie, plus il est important plus la solution est satisfaisante. Ensuite lorsqu'une solution est à la frontière entre deux catégories selon le profit, c'est le Nombre de projets réalisés qui va permettre de conclure. Plus celui est important, moins la solution sera satisfaisante. Et enfin de manière moins importante nous prenons en compte la durée du projet le plus long qui elle doit être la plus petite possible pour que la solution soit satisfaisante. En réalisant ce préclassement, on a une hiérarchisé parmi les critères, privilégiant ainsi le profit, puis dans un second temps le Nombre de projets réalisés et enfin la Durée du projet le plus long.

On établit une pré-classification et on résout. On obtient une fonction objectif nulle, la résolution est ainsi optimale car la fonction score que l'on a appris reproduit bien notre pré-classification. Il faut néanmoins faire attention : une classification trop complexe ou aléatoire n'obtiendra pas de solution même en modifiant les paramètres.

Concernant les paramètres, on essaye de faire varier ϵ et les L_i. Ayant 3 critères en entrée, on remarque que si $L_i \ \leq 2$ on a des difficultés à obtenir une solution dû au manque de degrés de

liberté, si l'on considère $3 \leq L_i$ les résultats sont similaires sauf pour des L_i trop important. En effet dans ce cas, l'espace des modèles possibles est trop grand et il faut nourire le modèle avec plus de données en entrée. On prend ainsi $L_i = 3$. Pour ϵ, on remarque que si $0.1 \leq \epsilon$ la résolution est moins fine : on reproduit moins bien l'inégalité stricte par rapport aux valeurs des critères. On choisit ainsi $\epsilon = 0.001$ qui nous permet d'avoir des résultats plus satisfaisants. Par ailleurs en essayant avec des valeurs de ϵ différentes pour caractériser la monotonie et les inégalités stricts on réalise que cela n'a pas beaucoup d'influence sur notre résolution, on les prend donc égaux.

Comparaison coefficients fixés et appris: Lorsque l'on résout notre programme linéaire il y a deux possibilités : on peut fixer les coefficients ou les apprendre. Voici un tableau de comparaison de la répartition dans les différentes classes selon la méthode utilisée avec la même pré-classification en entrée :

Méthode	Solution non-acceptable	Solution neutre	Solution satis-faisante	Coefficients obtenus
Distribution de départ	33%	33%	33%	non défini
Coefficients non-appris	27%	39%	34%	0.33 / 0.66
Coefficients appris	11%	18%	71%	0.02 / 0.13

On remarque une différence importante entre les classes prédites par les deux méthodes. Notamment lorsque l'on regarde les répartitions parmi les classes pour chacune des méthodes. En effet la méthode des coefficients non-appris tend à reproduire la répartition de la distribution de départ et le fait d'avoir donné une taille similaire aux intervalles, reproduit aussi ce phénomène. En revanche, la méthode des coefficients appris engendre une répartition bien différente. À première vue on pourrait penser qu'elle est moins biaisée par la distribution de départ mais lorsque l'on regarde de plus prêt ses résultats, on remarque que la fonction score déterminée pour cette méthode a une forte sensibilité aux variations d'instances donnés en entrée. $coef_2$ étant faible, on obtient ainsi dans la plupart des cas un score élevé et donc l'instance est classée comme une solution satisfaisante. Cela est du à notre éspace de modèles possibles qui est plus important dans le cas de la méthode des coefficients appris (plus de degré de liberté). Ainsi d'optenir un meilleur modèle serait de l'apprendre sur plus de données. Cela est confirmé par l'expérience, lorsque l'on relance la méthode avec plus d'instances, on tend vers les mêmes résultats que ceux obtenus avec la méthode des coefficients non-appris.

Ainsi, dans le cas où notre ensemble de pré-classification contient un nombre raisonnable d'instances, le modèle à coefficients fixés semble le plus efficace : Il a une meilleure capacité de généralisation. Dans le Jupyter notebook de démonstration, on considère un nombre important d'instance dans notre pré-classification et dans ce cas là, les résultats sont très similaires.

3.2 La méthode Xtreme Ranking

Comme pour la méthode UTA, la méthode Xtreme Ranking permet d'opérer une sélection préférentielle des solutions non-dominées de notre problème de staffing. La méthode Xtreme Ranking se base sur le calcul des rangs minimaux et maximaux d'une solution à partir d'un score pondéré calculé à partir du score multicritère.

Nous nous appuierons aussi sur le pré-classement d'un échantillon de solutions en 3 catégories (pour rappel : *Solution non acceptable*, *Solution neutre*, et $Solution\ satisfaisante$.). Nous formaliserons ainsi ces catégories :

- S_1 : ensemble de solutions passées $(X_{1,i})$ non-acceptables
- S_2 : ensemble de solutions passées $(X_{2,i})$ neutres
- S_3 : ensemble de solutions passées $(X_{3,i})$ satisfaisantes

3.2.1 Modélisation du score

Comme précisé précédemment, les solutions de notre surface de Pareto identifiées suite à la résolution réalisée en partie 3 sont caractérisées par un triplet de 3 critères : Le *Profit*, le *Nombre de projets réalisés* ainsi que la *Durée du projet le plus long*. A partir de cela, nous définissons un score pondéré pour chaque solution. Le score $s(j)$ pour un projet j est une combinaison linéaire de ces 3 critères $(x_{i,j})_{1\leq i\leq 3}$, définie ainsi:

$$s(j) = \sum_{i=1}^{3} w_i x_{i,j}$$

où les w_i sont les poids associés à chacun des trois critères. Ce sont des variables continues telles que

- $$\forall i \in \{1,\dots,3\}, 0 \leq w_i \leq 1$$
- $$\sum_{i=1}^{3} w_i = 1$$

Nous allons ainsi pouvoir étudier le classement des solutions entre elles selon un système de poids $(w_i)_{1\leq i\leq 3}$ donné.

3.2.2 Modélisation du problème d'optimisation

Afin d'appliquer la méthode Xtreme Ranking en tant que modèle de préférence, nous allons pouvoir nous appuyer sur le score défini précédemment afin de déterminer un rang minimal (meilleur classement possible) ainsi qu'un rang maximal (pire rang possible) pour chaque solution.
Nous commençons par définir la contrainte de normalisation sur le vecteur de poids booléens $(W_i)_{1\leq i\leq 3}$:

$$\text{(C1)} \ \sum_{i=1}^{3} W_i = 1$$

A partir de l'échantillon pré-classé, nous définissons des contraintes pour chaque paire de solution a, b appartenant à une catégorie différente, telle que le score de la solution appartenant à la catégorie préférée (exemple : S_2 préférée sur S_1) soit supérieur au score de l'autre solution.

Plus formellement, les contraintes liées à l'échantillon sont les suivantes :

$$\text{(C2)} \ \forall a, b \in \{1,2,3\} \ tels \ que \ a < b, \forall X \in S_a, \forall X' \in S_b,$$
$$s(X) - s(X') \geq \epsilon$$

où $\epsilon > 0$.

Pour une solution X_i donnée, nous posons les variables booléennes $(B_j)_{j\neq i}$ telles que

$$\forall j \neq i, \begin{cases} B_j^{(i)} = 1 & \text{si } s(X_i) < s(X_j) \\ B_j^{(i)} = 0 & \text{sinon.} \end{cases}$$

Modélisation pour le rang minimal

Nous nous plaçons d'abord dans la modélisation du problème d'optimisation pour le calcul du rang minimal de la solution X_i. Pour modéliser les $(B_j)_{j\neq i}$ nous ajoutons les contraintes suivantes (avec M une constante suffisamment grande):

$$\text{(C3.a) } \forall j \neq i,\ s(X_i) - s(X_j) + M.B_j^{(i)} > \epsilon$$

Pour calculer le rang minimal de la solution X_i, nous fixons comme objectif la minimisation de la somme $S = \sum_{j\neq i} B_j^{(i)}$ sous les contraintes (C1), (C2) et (C3.a). Ainsi, nous obtenons S_{min} le nombre minimum de solutions classées strictement avant la solution X_i. Cela représente donc le rang minimal (et donc le meilleur rang) de la solution X_i pour tout système de poids $(W_i)_{1\leq i\leq 3}$, que l'on note R^i_{min}.

Modélisation pour le rang maximal

Pour le calcul du rang maximal de notre solution X_j, on pose une contrainte légèrement différente :

$$\text{(C3.b) } \forall j \neq i,\ s(X_j) - s(X_i) + M.B_j^{(i)} > \epsilon$$

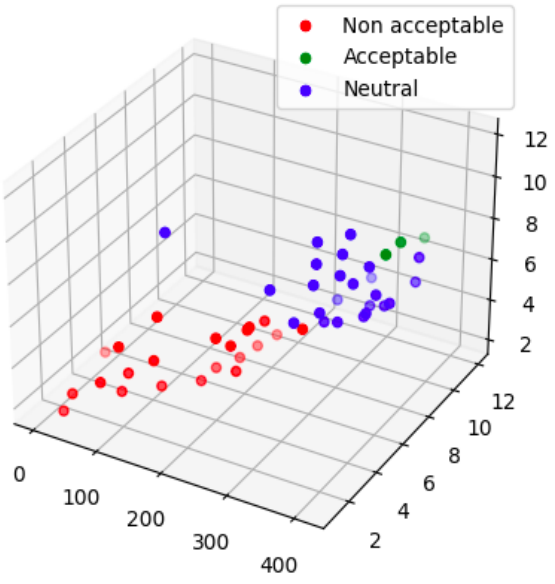
La minimisation de la somme $S = \sum_{j\neq i} B_j^{(i)}$ sous les contraintes (C1), (C2) et (C3.b) nous donne le nombre de solutions classées strictement avant X_i. On en déduit donc facilement le rang maximal de notre solution, qui vaut $N - S$ (N étant la taille de notre instance de solutions non-dominées), et que l'on note R^i_{max}.

3.2.3 Interprétation des résultats

L'objectif de la méthode Xtreme Ranking étant d'obtenir des préférences sur nos solutions en sortie, il convient d'établir une classification de nos solutions d'entrée similaire à celle de notre entrée.
Il suffit donc de poser R_{min} et R_{max} resp. seuil de meilleur et pire rang acceptés. Nous pouvons donc ainsi poser les ensembles résultats :

- Solutions acceptées $Sol_1 = \{X_i, |\ R^i_{min} < R_{min}\}$
- Solutions rejetées $Sol_3 = \{X_i, |\ R^i_{max} > R_{max}\}$
- Solutions neutres $Sol_2 = \{X_i, |\ X_i \notin Sol_1 \cup Sol_3\}$

  La figure suivante montre les résultats pour l'instance medium, appliqué avec un $R_{min} = 10$ et $R_{max} = 20$, sur une surface de Pareto comprenant 49 solutions.



4 Code du projet

Le code Python de l'ensemble du projet est disponible sur github à cette adresse.

5 Conclusion

Dans ce projet, nous avons implémenté un algorithme d'optimisation par programme linéaire pour trouver les solutions non-dominées à un problème de planification sur plusieurs instances. Pour des raisons pratiques, nous avons dû restreindre le temps de résolution de Gurobi. Ainsi nous avons obtenu des **approximations** des solutions non-dominées du problèmes. L'application des méthodes UTA et Xtreme ranking ont ensuite permis de classer les solutions en 3 catégories: *Solutions non-acceptables, Solutions neutres* et *Solutions satisfaisantes*.

Nous avons produit un package python qui à la fois trouve les solutions non-dominées et exécutent les modèles de préférences pour classifier les solutions. Le package est équipé d'une interface en lignes de commandes pour davantage d'interactivité et pour faciliter son utilisation. Une piste d'amélioration du code est de concevoir une interface qui permette à un décideur de rentrer directement ses préférences et de lancer les algorithmes de classification. Dans l'état actuel du code, les préférences doivent être saisies manuellement dans un fichier csv, ce qui est plutôt laborieux.