CentraleSupélec | université PARIS-SACLAY

Systèmes de Décision Préférence

# Projet 2022

## CompuOpti planning


***Étudiants :***
Benitah Noam
Favoli Thomas
Verwaerde Alice

***Enseignants :***
Mousseau Vincent



17 février 2023

# I Introduction

## Motivation

Les employés sont l'actif le plus important pour les entreprises du secteur des services. L'affectation de la main-d'œuvre et la planification du personnel traitent de l'affectation des équipes de travail afin de couvrir la demande de ressources qui varie dans le temps.

## Contexte

Le problème étudié est un problème de planification de personnel et d'affectation de projets. On considèrera que le problème se déroule sur un horizon de temps donné (on ne considèrera que les jours ouvrés).

Chaque membre du personnel 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 projet fait appel à des qualifications parmi l'ensemble des qualifications (un sous-ensemble de  A, B, C, D, E  ). 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. Pour chaque projet, une date de livraison à 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.

## Objectifs

Nous devons 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). Les objectifs sont multiples :

- Maximiser le résultat financier de l'entreprise
- Minimiser le nombre de projets sur lesquels un quelconque collaborateur est affecté
- Executer le projet le plus long en un minimum de jours.

Il conviendrait d'utiliser une approche d'optimisation multi-objectifs afin de réaliser ces objectifs.

# II Notre modèle

La programmation mathématique est une approche déclarative dans laquelle le modélisateur formule un problème d'optimisation mathématique qui capture les principales caractéristiques d'un problème de décision complexe. Un modèle d'optimisation mathématique comporte habituellement les sections suivantes :

## Paramètres

**JOBS** : L'ensemble des projets
**STAFF** : L'ensemble du personnel
**QUALIFICATION** : L'ensemble des qualifications
**HORIZON** : L'horizon de temps
**EMPLOYEE-QUALIFICATION** : Les qualifications de l'ensemble des employés
**EMPLOYEE-VACATION** : Les vacances de l'ensemble des employés
**PROJECT-QUALIFICATION-REQUIREMENT** : Les qualifications requises pour l'ensemble des projets
**PROJECT-QUALIFICATION-DAYS** : Le décompte par job du nombre de jours par compétence necessaires à la réalisation du job
**PROJECT-GAIN** : L'ensemble des gains des projets
**PROJECT-PENALITY** : L'ensemble des pénalités des projets
**PROJECT-DUE-DATE** : L'ensemble des due dates des projets

## Variables de décision

$\mathbf{X}_{i,j,k,t}$ : Renvoi 1 si le salarié i travail sur la compétence k du projet j le jour t et 0 sinon
$\mathbf{Y}_j$ : Renvoi 1 si le projet j est réalisé totalement et 0 sinon
$\mathbf{L}_j$ : Renvoi le nombre de jours de retard pour le projet j
$\mathbf{E}_j$ : Renvoi la date de fin de réalisation du projet j
$\mathbf{Z}_{i,j}$ : Renvoie 1 si le salarié i à travaillé sur le projet j
$\mathbf{B}_j$ : Renvoie la date de début du projet j

## Contraintes

### Contrainte d'unicité de l'affectation quotidienne du personnel

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.

$\forall i \in \textbf{STAFF}, \forall t \leq h, \qquad \sum_{\substack{j \in \textbf{JOBS} \\ k \in \textbf{QUALIFICATION}}} X_{i,j,k,t} \leq 1$

### Contrainte de congé

Un membre de personnel ne peut pas être affecté à une qualification de projet un jour de congé.

$\forall i \in \textbf{STAFF}, \forall t \in \textbf{VACATION(i)}, \qquad \sum_{\substack{j \in \textbf{JOBS} \\ k \in \textbf{QUALIFICATION}}} X_{i,j,k,t} = 0$

#### Contrainte de qualification du personnel

Un membre du personnel ne peut être affecté à une qualification d'un projet que s'il possède cette qualification.

$\forall i \in$ **STAFF**, $\forall j \in$ **JOBS**, $\forall k \in$ **QUALIFICATION** $\mid$ $k \notin$ **EMPLOYEE-QUALIFICATION(i)** $\cup$ $k \notin$ **PROJECT-QUALIFICATION-REQUIREMENT(j)**, $\forall t \leq h, \qquad X_{i,j,k,t} = 0$

#### Contrainte de couverture des qualifications du projet

Un projet n'est considéré réalisé que si tous les jours de travail dédiés à chacune des qualifications intervenant dans le projet ont été couverts par des membres du personnel.

$\forall j \in$ **JOBS**, $\forall k \in$ **PROJECT-QUALIFICATION-REQUIREMENT(j)**,
$\mathbf{Y}_j \times$ **PROJECT-QUALIFICATION-DAYS (j)(k)** $\leq \sum_{\substack{i \in \textbf{STAFF} \\ t \leq h}} X_{i,j,k,t}$

#### Contrainte de réalisation

La date de fin de projet intervient après le dernier jour de travail sur le projet.

$\forall i \in$ **STAFF** ,$\forall j \in$ **JOBS** ,$\forall k \in$ **QUALIFICATION** ,$\forall t \leq h, \qquad X_{i,j,k,t} \times t \leq E_j$

#### Contrainte de pénalité

On compte le nombre de jour de retard en soustrayant la due date à la date de réalisation du projet.

$\forall j \in$ **JOBS** $\qquad E_j - PROJECT - DUE - DATE(j) \leq L_j$

## Objectifs

#### Maximiser le résultat financier de l'entreprise

$max \ \sum_{j \in \textbf{JOBS}} (Y_j \times PROJECT - GAIN[j]) - (L_j \times PROJECT - PENALITY[j])$

#### Minimiser le nombre de projets par salariés

$\forall i \in$ **STAFF** ,$\forall j \in$ **JOBS** ,$\forall k \in$ **QUALIFICATION** ,$\forall t \leq h, \qquad X_{i,j,k,t} \leq Z_{i,j}$

$\forall i \in$ **STAFF** , $\qquad \sum_{j \in \textbf{JOBS}} Z_{i,j} \leq max - nb - projects - per - employee$

#### Minimiser la durée de réalisation du projet le plus long

$\forall i \in$ **STAFF** ,$\forall j \in$ **JOBS** ,$\forall k \in$ **QUALIFICATION** ,$\forall t \leq h, \qquad B_j \times X_{i,j,k,t} \leq t$

$\forall j \in$ **JOBS** , $\qquad E_j - B_j \leq max - duration - project - 1$

# III Nos résultats

## Analyse - Génération des solutions possibles

Pour rappel nous avons trois objectifs :

- Maximiser le résultat financier de l'entreprise
- Minimiser le nombre de projets sur lesquels un quelconque collaborateur est affecté
- Executer le projet le plus long en un minimum de jours.

Afin de pouvoir satisfaire l'ensemble des trois objectifs nous avons opté pour une approche de recherche exhaustive pour trouver une ou plusieurs solutions optimales pour un problème d'optimisation multi-objectif. Les deux variables "max_projet" et "max_durée" sont utilisées pour contrôler la complexité de la recherche.

La fonction model_optimise est appelée à chaque étape de la boucle pour résoudre le problème avec les valeurs actuelles de max_projet et max_durée. Le résultat est ajouté à la liste solution si la fonction s'exécute avec succès.

Lorsque les boucles sont terminées, toutes les solutions trouvées sont imprimées et retournées. Cependant, ce code n'implémente pas de mécanisme pour déterminer la ou les meilleures solutions parmi les solutions trouvées, ce qui est nécessaire pour un problème d'optimisation multi-objectif. Il est ainsi donc nécessaire d'ajouter une fonction supplémentaire pour déterminer la ou les meilleures solutions en utilisant une approche d'analyse de la surface de Pareto.

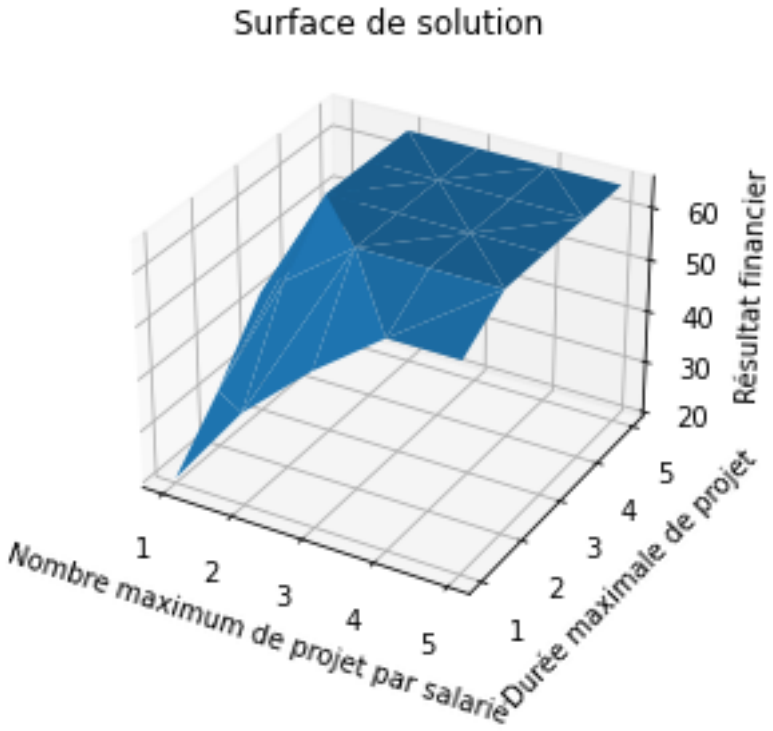
```
def get_solution(path):
    data = get_data(path)
    max_projet = len(data["jobs"])
    max_durée = data["horizon"]
    solution = []
    while max_projet > 0:
        while max_durée > 0:
            try:
                result = model_optimise(data, max_projet, max_durée)
            except:
                break
            solution.append([result, max_projet, max_durée])
            max_durée -= 1
        max_projet -=1
        max_durée = data["horizon"]
    print("Les solutions sont")
    print(np.array(solution))
    return solution
```

## Small instance

Les solutions pour le petit jeu de données sont :

```
[[65.  5.  5.]
 [65.  5.  4.]
 [65.  5.  3.]
 [65.  5.  2.]
 [59.  5.  1.]
 [65.  4.  5.]
 [65.  4.  4.]
 [65.  4.  3.]
 [65.  4.  2.]
 [59.  4.  1.]
 [65.  3.  5.]
 [65.  3.  4.]
 [65.  3.  3.]
 [65.  3.  2.]
 [49.  3.  1.]
 [65.  2.  5.]
 [65.  2.  4.]
 [65.  2.  3.]
 [55.  2.  2.]
 [37.  2.  1.]
 [42.  1.  5.]
 [42.  1.  4.]
 [42.  1.  3.]
 [30.  1.  2.]
 [20.  1.  1.]]
```

Et la visualisation de la surface est :



# IV Solutions non-dominées

## Génération des solutions non-dominées

La surface de Pareto est un concept en optimisation multi-objectif qui décrit l'ensemble des solutions optimales non dominées. Dans le domaine de l'optimisation multi-objectif, une solution peut être considérée comme dominée si une autre solution est plus avantageuse sur au moins un critère et aussi avantageuse ou tout aussi avantageuse sur les autres critères.

La fonction utilise une boucle for pour parcourir l'ensemble des solutions données en entrée. Pour chaque solution, elle utilise une autre boucle for pour vérifier si la solution actuelle est dominée par une autre solution dans l'ensemble.

Cependant, il est important de noter que cette approche peut être très coûteuse en termes de temps de calcul et de ressources, en particulier pour des problèmes de grande taille. Il peut être préférable d'utiliser des algorithmes plus efficaces tels que la méthode d'élagage de Pareto pour trouver la surface de Pareto.

```python
def get_non_dominated_solutions(solutions):
    non_dominated_solutions = []
    for i in range(len(solutions)):
        is_dominated = False
        for j in range(len(solutions)):
            if (solutions[j][0] >= solutions[i][0] and
                solutions[j][1] <= solutions[i][1] and
                solutions[j][2] <= solutions[i][2] and
                j != i):
                is_dominated = True
                break
        if not is_dominated:
            non_dominated_solutions.append(solutions[i])
    print("Les solutions non dominées")
    for solution in non_dominated_solutions:
        print(solution)
    return non_dominated_solutions
```

Les solutions non-dominées pour le petit jeu de données sont :

```
[59.0, 4, 1]
[65.0, 3, 2]
[49.0, 3, 1]
[65.0, 2, 3]
[55.0, 2, 2]
[37.0, 2, 1]
[42.0, 1, 3]
[30.0, 1, 2]
[20.0, 1, 1]
```

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

Il existe plusieurs techniques pour modéliser les préférences sur la surface de solutions non dominées. Voici quelques-unes des méthodes les plus courantes :

1. Méthode des points de référence : Cette méthode consiste à définir un point de référence qui correspond à la valeur maximale du premier objectif et aux valeurs minimales des deux autres objectifs. La distance entre les solutions et le point de référence est mesurée à l'aide de la formule de distance euclidienne. La solution la plus proche du point de référence est considérée comme la meilleure solution.

2. Analyse de la fonction d'utilité : Cette méthode implique la définition d'une fonction d'utilité qui traduit les préférences en termes quantitatifs. Cette fonction est utilisée pour évaluer les solutions sur la surface de Pareto et déterminer la meilleure solution.

3. Approches multi-critères d'aide à la décision : Il existe plusieurs approches multi-critères d'aide à la décision qui permettent de modéliser les préférences, telles que la méthode de l'analyse de la valeur (MAVT), la méthode d'analyse hiérarchique des critères (AHP) et la méthode des pondérations linéaires (LWP).

4. Apprentissage automatique : Il existe également des approches d'apprentissage automatique qui permettent de modéliser les préférences en utilisant des algorithmes d'apprentissage supervisé ou non supervisé pour prédire les préférences à partir de données historiques.

Le choix de la technique dépend des préférences de l'utilisateur, de la complexité du modèle, de la quantité de données disponibles et de la qualité des données.

## Méthode de points de référence

On va déterminer la meilleure solution parmi les solutions non dominées sur une surface de Pareto en utilisant une approche de comparaison de points de référence.

```python
def get_best_solution(solutions):
    reference_point = [max([s[0] for s in solutions]), min([s[1] for s in solutions]),
                       min([s[2] for s in solutions])]
    best_solution = solutions[0]
    best_distance = float("inf")
    for solution in solutions:
        distance = (solution[0] - reference_point[0])**2 + (solution[1] -
                reference_point[1])**2 + (solution[2] - reference_point[2])**2
        if distance < best_distance:
            best_distance = distance
            best_solution = solution
    print("La meilleure solution est: " ,best_solution)
    return best_solution
```

La meilleure solution pour le petit jeu de données est : [65.0, 3, 2]

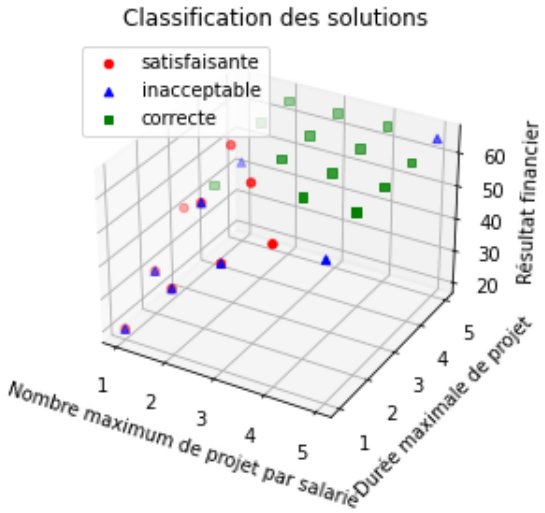
## Apprentissage automatique

Pour rappel il existe des approches qui permettent de modéliser les préférences en utilisant des algorithmes d'apprentissage supervisé pour prédire les préférences à partir de données historiques.Nous divisons les échantillons en 3 catégories basées sur les relations de dominance obtenues :

Les solutions satisfaisantes sont les solutions non dominées :
[59.0, 4, 1] [65.0, 3, 2] [49.0, 3, 1] [65.0, 2, 3] [55.0, 2, 2] [37.0, 2, 1] [42.0, 1, 3] [30.0, 1, 2] [20.0, 1, 1]

Les solutions inacceptables sont les solutions qui sont dominées pour les trois objectifs :
[65.0, 5, 5] [59.0, 5, 1] [49.0, 3, 1] [55.0, 2, 2] [37.0, 2, 1] [42.0, 1, 5] [30.0, 1, 2] [20.0, 1, 1]

Les solutions correctes sont le reste des solutions :
[65.0, 5, 4] [65.0, 5, 3] [65.0, 5, 2] [65.0, 4, 5] [65.0, 4, 4] [65.0, 4, 3] [65.0, 4, 2] [65.0, 3, 5] [65.0, 3, 4] [65.0, 3, 3] [65.0, 2, 5] [65.0, 2, 4] [42.0, 1, 4]




Nous définissons une fonction get_weight qui prend en entrée trois listes de solutions, appelées satisfaisante, inacceptable et correcte. Ces listes représentent les différentes classes de solutions qui ont été classifiées précédemment.

Ainsi, cette fonction implémente une régression logistique multi-classe pour classifier les solutions en trois classes en utilisant les relations de dominance obtenues. Les poids(résultat, maximum nombre projet, durée maximale) sont utilisés pour représenter la contribution de chaque variable à la détermination de la classe de chaque solution.
Weights : [ 0.02787445 -0.82786431 -0.83306955]

```python
def get_weight(satisfaisante, inacceptable, correcte):
    X1 = np.array(satisfaisante)
    y1 = np.full(X1.shape[0],1)
    X2 = np.array(inacceptable)
    y2 = np.full(X2.shape[0],2)
    X3 = np.array(correcte)
    y3 = np.full(X3.shape[0],3)
    X = np.concatenate((X1, X2, X3))
    y = np.concatenate((y1, y2, y3))
    clf = LogisticRegression(multi_class='auto', solver='lbfgs', random_state=0, max_iter=1000)
    clf.fit(X, y)
    weights = clf.coef_
    print("Weights:", weights[0])
```