Projet SDP

Lucie Clémot, Faustine Malatray, Côme Stephant

Décembre 2022 - Février 2023

1 Introduction

On s'intéresse à la société *CompuOpti* qui souhaite établir des planning d'affectation du personnel pour ses projets. Nous sommes mandatés pour réaliser une solution qui permettra à la PDG de *CompuOpti* de plannifier efficacement son personnel sur les projets.
Le problème étudié est donc 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 de CompuOpti possède certaines qualifications, parmi un ensemble donn é de qualifications (par exemple A,B,C,D,E), et des jours de congés prédéfinis intervenant durant l'horizon de temps.
Nous allons dans un premier temps reformuler mathématiquement ce problème d'optimisation et établir un modèle en introduisant ses paramètres, variables, contraintes et objectifs. Nous utiliserons ensuite le logiciel Gurobi pour calculer les solutions non-dominées du problème. Enfin, nous développerons un modièle de préférence permettant de discriminer entre les solutions de la surface des solutions non-dominées.

2 Reformulation du problème

2.1 Nomenclature

Ensemble des qualifications:

- $\mathcal{I} = \{1, ..., I\}, I = nombre\_qualifs \in N$
- $i \in I$ - les qualifications

Ensemble des projets:

- $\mathcal{J} = \{1, ..., J\}, J = nombre\_projets \in N$
- $j \in J$ - les projets

Ensemble des employés:

- $\mathcal{K} = \{1, ..., K\}, K = nombre\_employes \in N$
- $k \in K$ - les employés

Horizon de temps:

- $\mathcal{N} = \{1, ..., N\}, N = horizon \in N^*$
- $n \in N$ - les jours

2.2 Introduction des variables

Les variables suivantes sont définies dans ce modèle de programmation linéaire :

- x_{ijkn} : variable binaire, valant 1 si l'employé k travaille sur le projet j avec la qualification i au jour n.
- u_j : variable binaire, valant 1 si le projet j est réalisé totalement, 0 sinon.
- $dateDebut_j$: entier, date de début de réalisation du projet j.
- $dateFin_j$: entier, date de fin de réalisation du projet j.
- $affectation_{kj}$: variable binaire, valant 1 si l'employé k aura travaillé sur le projet j, 0 sinon.

2.3 Introduction des paramètres

Les données suivantes sont définies dans ce modèle de programmation linéaire :

- $conge_{kn}$: variable binaire sur l'employé k. 0 s'il est en congés au jour n, 1 sinon.
- $qualif_{ik}$: variable binaire sur l'employé k, valant 1 s'il a la qualification i et 0 sinon.
- d_j : entier, date de livraison négociée du projet j.
- p_j : entier, pénalité financière par jour de retard du projet j.
- b_j : entier, bénéfice financier pour le projet j.
- $besoin_{ij}$: entier, le nombre de jours de travail pour la qualification i sur le projet j.

2.4 Formulation des contraintes

Les contraintes sont les suivantes :

- Qualification du personnel :

  Pour chaque employé k et qualification j, quelque soit le projet j et le jour n:
  $x_{ijkn} \leq qualif_{ik}$.

- Contrainte de congé :

  Pour chaque employé k et jour n, quelque soit le projet j et la qualification i:
  $x_{ijkn} \leq conge_{kn}$.

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

  Pour tout projet j et pour toute qualification i :
  $\sum_k \sum_n (x_{ijkn}) \leq besoin_{ij}$

- Contrainte de couverture des qualifications du projet :

  Pour tout projet j :
  $\forall i \ (\sum_k \sum_n (x_{ijkn}) \geq besoin_{ij}) \Rightarrow u_j = 1$

  En transformant cette contrainte pour l'optimisation linéaire :
  Soit $Majorant = max(besoin_{ij})$ pour tout projet j et toute qualification i :

  $$Majorant * (1 - u_j) <= \sum_k \sum_n (x_{ijkn}) - besoin_{ij} < Majorant * u_j$$

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

  Pour toute personne k et tout jour n:
  $\sum_i \sum_j (x_{ijkn}) \leq 1$

- Date de début de réalisation :

  Pour chaque projet j:
  $\forall n, (dateDebut_j \leq job\_in\_process_{j,n} * n)$

- Date de fin de réalisation :

  Pour chaque projet j:
  $\forall i, k, n, \ (\ x_{ijkn} * n \leq end_j)$

2.5 Objectifs

- Bénéfice total :
  On souhaite maximiser le bénéfice total, comme ci-dessous :
  Notre date de livraison pour un projet j donné:

$$\begin{cases} delay_j = max(0, end_j - d_j) \\ Bénéfice\ total = \sum_j (b_j * u_j - delay_j * p_j) \end{cases}$$

On cherche donc :
$max_{xijkn}(Bénéfice\ total)$

- Nombre de projets maximum par personne :
  On veut donc minimiser le nombre de projets de l'employé qui en réalise le plus:
  $(\exists n, i$ tels que $x_{ijkn} = 1) \Rightarrow t_{k,j} = 1$
  Lorsqu'on reformule en programmation linéaire, si on introduit la variable projet_max qu'on va souhaiter minimiser:

  $\forall k, (proj_{max} \geq \sum_j t_{k,j})$
  Puis on cherchera $min(proj_{max})$

- Compacité :
  On veut également minimiser la compacité, c'est-à-dire le nombre de jours sur le projet le plus long, en jours consécutifs. Autrement dit, si on pose une variable longueur_max qu'on va vouloir minimiser:

  $longueur_{max} = max_j(end_j - begin_j)$
  Puis on cherchera $min(longueur_{max})$

  En programmation linéaire, cela revient à minimiser $longueur_{max}$ avec :
  $\forall j, (longueur_{max} \geq (end_j - begin_j))$

3 Implémentation sur Gurobi

3.1 Organisation du repo Github

Nous avons fonctionné en mode équipe sur notre repository Github disponible sous le lien suivant:
`https://github.com/osnapitzlu/project-sdp.git`

Notre code se trouve sur le branche master, avec l'architecture suivante:

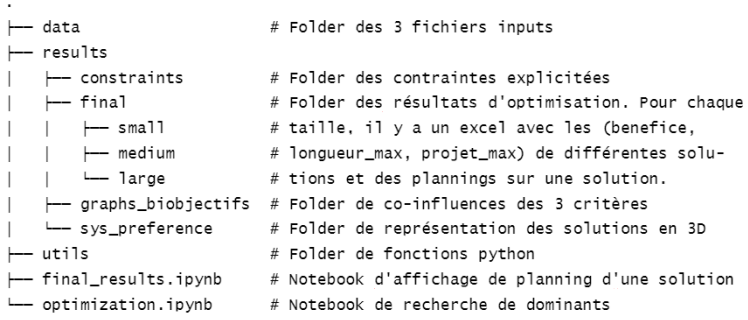

Figure 1: Architecture du repository Github

3.2 Démarche

Pour ce projet, nous avons adopté une démarche itérative pour rentrer avec toujours plus de précision dans le sujet.

En premier lieu, nous avons voulu obtenir une modélisation avec seul le bénéfice en fonction objectif - c'est-à-dire que nous n'avons pas implémenté tout de suite les objectifs de compacité et de nombre de projet maximum par personne.
Cette étape nous a permis de comprendre les ordres de grandeur pour chaque taille d'instance.

En second lieu, nous avons implémenté les deux autres fonctions objetif, en les faisant varier selon les paramètres de chaque problème (le nombre de projets maximum par personne n'excédera jamais le nombre total de projet par personne par exemple).
A la fin de cette étape, nous avions pour chaque taille d'instance les différents tuples (bénéfice, longueur_max, projet_max) correspondant à toutes les solutions du problème. Nous avons affiché les graphs 2D de chaque objectif en fonction d'un second pour déterminer quels systèmes de préférence nous devions construire ou préconiser.

Pour finir, nous avons donc modélisé un système de préférence en simulant des profils type de décideurs qui ont chacun un ordre de préférence de priorisation des trois objectifs. Pour déterminer ensuite le planning le plus adéquat par rapport aux préférences des décideurs, nous avons utilisé trois système de vote : Condorcet, Borda et Plurality with instant run-off. Ainsi, nous pouvons faire des recommendations selon les priorisations des critères.
A la fin de cette étape - et donc en cette fin de projet - nous avons construit un programme qui, à partir d'une instance d'entrée et de critères à remplir sur certaines contraintes, permet d'afficher un planning et de l'exporter en fichier excel.

3.3 Optimisation des 3 critères sans système de préférence

Dans cette section, détaillons la seconde itération de notre démarche. Pour rappel, nous avions déjà implémenté la fonction bénéfice et avions déjà une idée des ordres de grandeur pour chaque problème.

Nous avons donc implémenté avec Gurobi les deux autres fonctions objectif, soit : la compacité et le nombre maximum de projets par personne. Nous avions en tête qu'il nous faudrait des résultats en sortie de Gurobi nous permettant de construire des graphs de domination: pour chaque taille d'instance nous avons donc construit un programme d'optimisation qui nous sorte des tuples (bénéfice, longueur_max, projet_max) pour chaque solution.

Les résultats de ce programme nous a conduit à l'obtention des courbes suivantes:

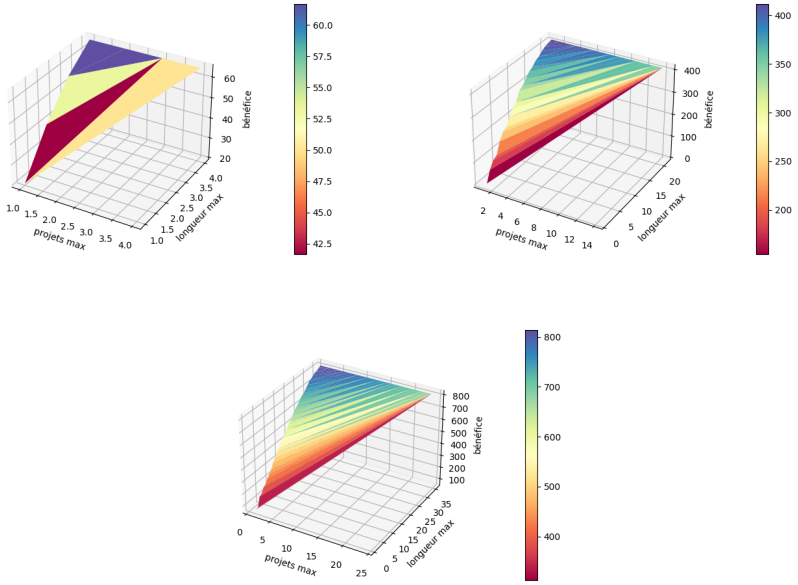

Figure 2: Graphs des triplets (bénéfice, longueur_max, projet_max) pour chaque solution de chaque instance

Ces courbes sont difficilement exploitables en tant que telles. Nous avons en effet une bonne idée des espaces de solutions, pourtant on ne peut pas facilement déterminer quelles sont les solutions dominées et non-dominées.

On peut intuiter cependant que la majorité des disparités d'informations se trouve dans le sous-plan (bénéfice, longueur_max). En effet, à projet_max fixé, il semble que les courbes passent par des valeurs très différentes de bénéfice en fonction de longueur_max. Cette observation ne se reproduit pas selon les autres critères. Nous avons cependant vérifié notre hypothèse en plottant les solutions

pour chaque instance sur seulement 2 critères à la fois:

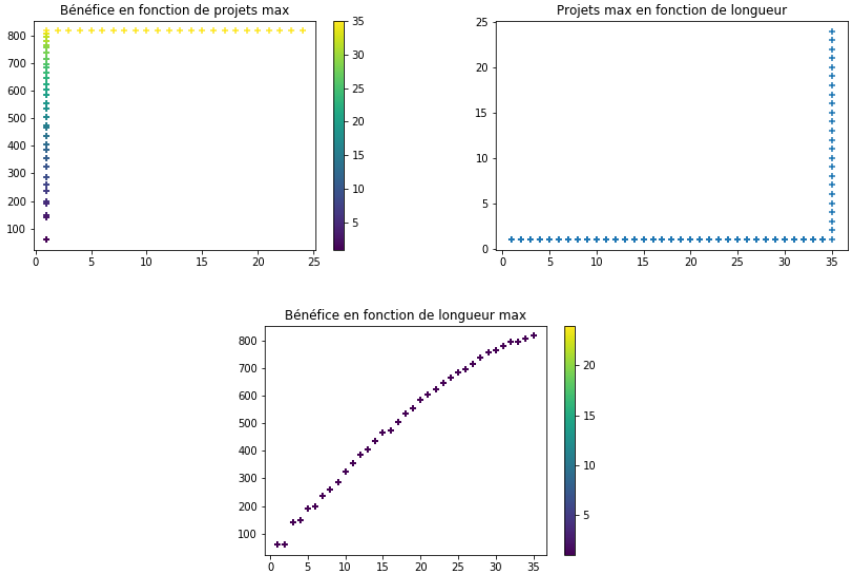

Figure 3: Graphs des solutions de l'instance large selon 2 des 3 critères à la fois

Les analyses de ces trois graphes - qui ont des allures très semblables sur les instances small et medium - nous apprennent plusieurs choses.

D'une part, sur les graphes "Bénéfice en fonction de projet_max" et "Projet_max en fonction de longueur_max", on voit bien qu'on a une solution dominée claire pour chacun. C'est-à-dire, si l'on éliminait longueur_max des objectifs dans le premier cas ou bénéfice dans le second, alors il y aurait une unique solution non-dominée à chaque fois, et donc aucun système de préférence à considérer.

Ce n'est d'une part pas très intéressant, d'autre part pas représentatif de notre problème. Analysons donc le troisième graphique à disposition: le bénéfice en fonction de la longueur_max d'un projet. Puisqu'on cherche à maximiser le bénéfice et minimiser la longueur_max, chaque point est donc un point non-dominé.

Ici, nous devons alors considérer différents systèmes de préférences et les modéliser pour annoncer un planning au client qui reflète la solution - objectivement et subjectivement - optimale.

4 Modèlisation des préférences

On se propose de modéliser le problème restant de décision comme suit: nous nous trouvons face au board de l'entreprise qui est constituée de 11 personnes. Ces 11 personnes ont un ordre personnel de hierarchiser notre tri-objectif.
Nous avons donc fait pluieurs itérations (11) du processus de décision pour ces 11 dirigeants, en considérant à chaque fois des classements des objectifs choisis au hasard.

A chaque instance, peu importe l'itération et peu importe le système de décision employé (nous avons implémenté les méthodes de Condorcet, de Borda et Run-off), nous avions la même solution qui ressortait.
Celles-ci étaient:

- Small : Bénéfice = 65 ; longueur_max = 4 ; projet_max = 1.
- Medium : Bénéfice = 413 ; longueur_max = 21 ; projet_max = 1.
- Large : Bénéfice = 817 ; longueur_max = 35 ; projet_max = 1.

5 Conclusion

Nous avons présenté notre démarche de résolution et avons déterminé des plannings optimaux selon nous pour un système de préférences donné.
Sur notre repo Github, se trouvent les différents pannings finaux que nous préconisons:

- Small : "results/final/small/4.xlsx"
- Medium : "results/final/medium/21.xlsx"
- Large : "results/final/large/35.xlsx"

Affichons un exemple avec notre planning préconisé pour l'instance small:

	Affectation				
Jour	**1**	**2**	**3**	**4**	**5**
Nom					
Emma	['Job4', 'C']	En congé	['Job1', 'C']	['Job3', 'C']	['Job5', 'C']
Liam	En congé	['Job4', 'B']	['Job1', 'A']	['Job3', 'A']	Non affecté.e
Olivia	['Job4', 'B']	['Job1', 'B']	Non affecté.e	['Job3', 'C']	['Job5', 'C']

Figure 4: Planning final pour l'instance small

Cependant, nous ne sommes pas à l'abri d'un changement de système de décision. Pour pallier à ce problème, nous avons donc conçu un notebook "final_results" qui premettra simplement d'afficher et d'enregistrer des plannings

suivant les nouveaux paradigmes sans avoir à coder - seulement entrer 3 entiers en paramètres.