CentraleSupélec

Projet de SDP : Organisation du travail chez CompuOpti

Seong Woo Ahn

Rodolphe Nonclercq

Tanguy Blervacque

Paris,

Janvier 2023

Contents

1 Introduction

Ce projet vise à mettre en place un outil de planification pour la société *CompuOpti* qui souhaite optimiser l'allocation des créneaux de travail pour ses ingénieurs développeurs. En effet chaque jour, chaque ingénieur, s'il n'est pas en vacances, peut utiliser une de ses qualifications pour travailler sur un projet.

Pour répondre à ce problème, nous avons développé un outil d'optimisation Gurobi se basant sur notre modélisation des différentes variables, contraintes et objectifs qui entrent en jeu. Étant donné que *CompuOpti* souhaite répondre à plusieurs critères d'évaluation, notre modélisation est multi-objectifs et mène non pas à une solution claire mais plutôt à un ensemble de solutions non-dominées. Ainsi, nous avons également misent en oeuvre plusieurs stratégies de décisions, se basant soit sur la priorisation d'objectifs, soit sur la comparaison des résultats candidats.

2 Modélisation

2.1 Les variables

La première phase de modélisation conciste en la définition des variables de notre problème. Par souci de clarté, nous avons séparé nos variables en deux catégories. d'un côté, les variables de décision représentent directement une décision de l'entreprise alors que de l'autre, les variables de modélisation sont celles qui découlent directement des variables de décision, et qui permettent d'évaluer les objectifs. Il faut garder en tête que cette distinction est purement syntaxique puisque Gurobi ne fait pas de différence entre ces deux types de variables.

Voici les différentes notations qui ont été utilisées:

Table 1: Notation des variables

Indexes	
c	Collaborateur
p	Projet
j	Jour
q	Qualification
Variables de décision	
$job\_is\_done_p$	Vecteur binaire pour savoir si le projet p est réalisé
$work_{cpjq}$	Matrice binaire pour savoir si un collaborateur c travaille sur le projet p le jour j sur la qualification q
$participate_{cp}$	Matrice binaire pour savoir à quels projets à participé chaque collaborateur
$job\_is\_active_{pj}$	Matrice binaire pour savoir si le projet p est entamé le jour j (si au moins un collaborateur travaille dessus)
Variables de modélisation	
$start\_date_p$	Vecteur pour connaître la date de début de chaque projet
$finish\_date_p$	Vecteur pour connaître la date de réalisation de chaque projet
$job\_has\_delay_p$	Vecteur binaire pour savoir si un job est terminé en retard
$job\_penalty_p$	Vecteur des pénalités par projets
job_profit_p	Vecteur des profits par projets
$max\_job\_per\_staff$	Nombre de projets du collaborateur qui en a le plus
$max\_len\_job\_staff$	Nombre de jours que prend le projet le plus long

Les instances de ces variables ont été initialisées via la fontion `model.addMVar()` de la librairie Gurobi.

Par exemple, voici l'instanciation des variables `job_is_done` et `work`:

```
#vecteur binaire pour savoir si un projet est realise
job_is_done = m.addMVar(
    (n_jobs), vtype=GRB.BINARY
```

```
)

#matrice binaire pour savoir si un collaborateur c travail sur le projet p le jour j sur la
    qualification s
work = m.addMVar(
    (n_staff,
     n_jobs,
     horizon,
     n_qualifications), vtype=GRB.BINARY
)
```

2.2 Les contraintes

Des contraintes ont également été modélisées pour éviter d'avoir des cas qui n'ont pas de sens (exemple: collaborateur 1 qui travaille sur 2 projets différents le même jour) et également pour prendre en compte les contraintes artificiellement imposées (congés). Il existe deux types de contraintes dans notre modélisation. Tout d'abord, les contraintes dites "de modélisations" sont celles qui permettent de contraindre les variables de décision pour qu'elles respectent certaines règles pré-établies dans notre sujet. Ensuite, les contraintes dites "de caractérisation" sont celles qui permettent de définir les variables de modélisation. Davantage de détails seront donnés ci-dessous.
Voici la liste des contraintes implémentées:

- Contrainte de non dépassement du taux horaire d'un projet (on ne travaille pas plus que nécessaire sur un projet).

```
for job_idx, job in enumerate(data['jobs']):
    for qualification, quantity in job['working_days_per_qualification'].items():
        if qualification in job['working_days_per_qualification'].keys():
            m.addConstr(work[:, job_idx, :, qualification_to_idx[qualification]].sum() <=
                job['working_days_per_qualification'][qualification])
        else:
            m.addConstr(work[:,job_idx,:,qualification_to_idx[qualification]].sum() <= 0)
```

- Contrainte de réalisation d'un projet selon la réalisation de toutes les tâches de ce projet.

```
for job_idx, job in enumerate(data['jobs']):
    for qualification, quantity in job['working_days_per_qualification'].items():
        #Contrainte dans les deux sens car le comportement depend de l'objectif fixe
```

```
            m.addConstr(work[:, job_idx, :, qualification_to_idx[qualification]].sum() >=
                quantity - 1 + epsilon - M * (1 - job_is_done[job_idx]))
            m.addConstr(work[:, job_idx, :, qualification_to_idx[qualification]].sum() <=
                quantity - 1 + M * job_is_done[job_idx])
```

- Contrainte d'unicité de l'affectation quotidienne du personnel (1 seule tâche sur 1 seul projet par jour).

```
for staff_idx, staff in enumerate(data['staff']):
    for day in range(horizon):
        m.addConstr(work[staff_idx, :, day, :].sum() <= 1)
```

- Contrainte de qualification du personnel (un collaborateur doit posséder la qualification sur laquelle il travaille).

```
for staff_idx, staff in enumerate(data['staff']):
    for qualification in data['qualifications']:
        if qualification not in staff['qualifications']:
            m.addConstr(work[staff_idx, :, :, qualification_to_idx[qualification]].sum() <=
                0)
```

- Contrainte de vacances.

```
for staff_idx, staff in enumerate(data['staff']):
    for vacation_day in staff['vacations']:
        m.addConstr(work[staff_idx, :, vacation_day - 1, :].sum() <= 0)
```

- Contrainte de participation d'un collaborateur à un projet (p = 1 si le collaborateur a travaillé au moins 1 jour sur une compétence sur ce projet).

```
for staff_idx in range(n_staff):
    for job_idx in range(n_jobs):
        #Contrainte dans les deux sens car le comportement depend de l'objectif fixe
        m.addConstr(work[staff_idx, job_idx].sum() >= epsilon - M * (1 -
            participate[staff_idx, job_idx]) )
        m.addConstr(work[staff_idx, job_idx].sum() <= M * participate[staff_idx, job_idx])
```

- Contrainte de caractérisation de la variable `job_is_active`.

  L'idée des contraintes de caractérisation est de définir la valeur d'un variable par les contraintes qu'on lui impose. Ici par exemple, `job_is_active` vaut forcément 1 lorsqu'au moins un collaborateur travaille sur ce projet un le jour donné, et 0 sinon.

```
for job_idx, job in enumerate(data['jobs']):
    for day in range(horizon):
        #Contrainte dans les deux sens car le comportement depend de l'objectif fixe
        m.addConstr(work[:, job_idx, day, :].sum() >= epsilon - M * (1 -
            job_is_active[job_idx, day]))
        m.addConstr(work[:, job_idx, day, :].sum() <= + M * job_is_active[job_idx, day])
```

- Contrainte de caractérisation des variables `start_date` et `finish_date`.

```
for job_idx, job in enumerate(data['jobs']):
m.addConstr(start_date[job_idx] >= 0)
m.addConstr(finish_date[job_idx] <= horizon - 1)
for day in range(horizon):
    m.addConstr(start_date[job_idx] <= day * job_is_active[job_idx, day])
    m.addConstr(finish_date[job_idx] >= day * job_is_active[job_idx, day])
```

- Contrainte de caractérisation de la variable `job_has_delay`.

```
for job_idx, job in enumerate(data['jobs']):
    #Je mets les deux sens de contrainte car selon le cas il y a un interet a
        job_has_delay = 0 ou 1
    m.addConstr(finish_date[job_idx] >= job['due_date']-1 + epsilon - M * (1 -
        job_has_delay[job_idx]))
    m.addConstr(finish_date[job_idx] <= job['due_date']-1 + M * job_has_delay[job_idx])
```

- Contrainte de caractérisation de la variable `job_penalty`.

```
for job_idx, job in enumerate(data['jobs']):
    m.addConstr(job_penalty[job_idx] == job_has_delay[job_idx] * job['daily_penalty'] *
        (finish_date[job_idx] - start_date[job_idx]))
```

- Contrainte de caractérisation de `job_profit`.

```
for job_idx, job in enumerate(data['jobs']):
    m.addConstr(job_profit[job_idx] == job_is_done[job_idx] * (job['gain'] -
        job_penalty[job_idx]))
```

- Contrainte de caractérisation de `max_job_per_staff`.

```
for staff_idx in range(n_staff):
```

```
        m.addConstr(max_job_per_staff >= participate.sum())
```

- Contrainte de caractérisation de `max_len_job`.

```
    for job_idx, job in enumerate(data['jobs']):
        m.addConstr(max_len_job >= finish_date[job_idx] - start_date[job_idx])
```

2.3 Les objectifs

En terme d'objectifs, notre modélisation en compte 3:

1. Maximisation du profit. Ici le profit suit la formule suivante:

$$profit = \sum_p job\_profit_p$$

   avec :

$$job\_profit_p = job\_is\_done_p * (gain_p - job\_penalty_p)$$
$$job\_penalty_p = job\_has\_delay_p * daily\_penalty_p * (finish\_date_p - start\_date_p)$$

2. Minimisation du nombre de projet sur lequel un quelconque collaborateur est affecté.
   Pour cela, nous minimiserons le nombre de projet du collaborateur qui en a le plus, via la variable `max_job_per_staff`.

3. Minimisation du nombre de jours consécutifs passé sur un même projet.
   Pour cela, nous minimiserons la durée du projet le plus long, via la variable `max_len_job_staff`.

Pour garder la logique de minimisation sur tous les objectifs (ceci permet également de favoriser l'analyse des frontières de décision), on va chercher à minimiser `-profit`. Nos trois fonctions objectifs sont alors:

```
# Objectif 1
m.setObjectiveN(-job_profit.sum(), 0, 0)

# Objectif 2
m.setObjectiveN(max_job_per_staff, 1, 0)

# Objectif 3
m.setObjectiveN(max_len_job, 2, 0)
```

Notez que le troisième argument de `m.setObjectiveN` vaut 0 pour tous les objectifs, ce qui signifie qu'à ce stade ils ont tous la même importance.

En lançant l'optimisation avec la modélisation donnée précédemment, on va obtenir différentes solutions. Il est important de noter que ces solutions varient selon le poids, la priorité donnée aux différents objectifs. Ces résultats vont être exploitées dans la partie suivante.

3 Résultats de l'implémentation sur Gurobi

3.1 L'ensemble des solutions non dominées

Table 2: Solutions non-dominées

data	job profit	max job per staff	max len job
Toy dataset: solution 1	65	2	4
Toy dataset: solution 2	46	2	3
Medium dataset: solution 1	400	4	19
Medium dataset: solution 2	215	5	17
Large dataset: solution 1	800	6	34
Large dataset: solution 2	770	8	33
Large dataset: solution 3	685	6	31
Large dataset: solution 4	170	3	19
Large dataset: solution 5	150	2	19

Pour la suite, nous nous sommes focalisés sur le dataset "large". L'idée sera donc de mettre en place une stratégie de sélection de solution dans le but de converger vers une unique solution.

3.2 Visualisations

On a obtenu des visualisations pour illustrer les solutions proposées par Gurobi sur le large dataset. On a notamment réussi a obtenir une visualisation 3D interactive (fichier html retrouvable sur le répo Github), qui permet notamment de faire la distinction entre solutions dominées et non-dominées:

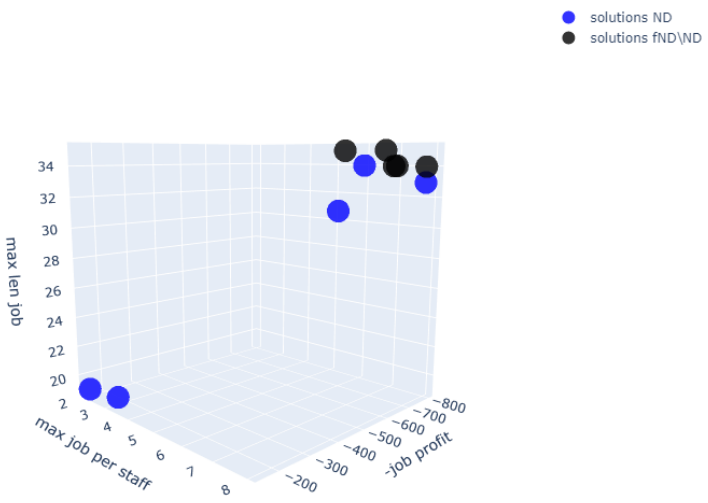

Figure 1: Visualisation 3D des solutions proposées

On a également obtenu les visualisations suivant 2 critères, ce qui permet de mieux voir le positionnement des solutions dominées parmi les solutions.

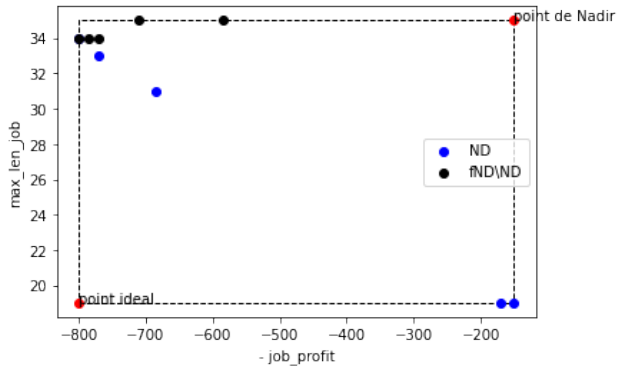

Figure 2: Visualisation 2D des solutions proposées 1

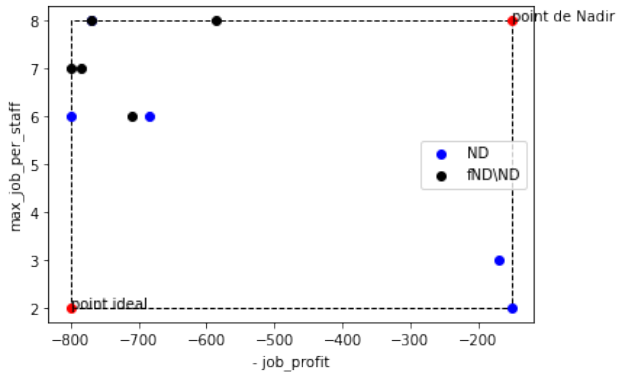

Figure 3: Visualisation 2D des solutions proposées 2

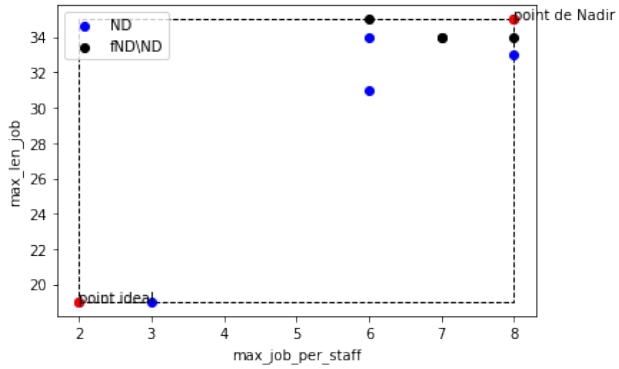

Figure 4: Visualisation 2D des solutions proposées 3

3.3 Algorithme de décision multi-objectif 1

Dans une première approche, nous considérons que le décideur a une préférence tranchée et explicite en terme de poids. Exemple : *"le bénéfice est la priorité absolue. Je privilégie le temps de réalisation d'un projet quitte à ce que les salariés travaillent sur différents projets."* Après avoir questionner le décideur on pourrait converger vers un vecteur de pondération pour donner un poids à chacun des critères.
On pose w_j la notation pour le coefficient d'importance associé à au critère j (on supposera que $\sum_j w_j = 1$). On note $S(a, a')$ l'ensemble de candidats correspondant à j : a est au moins aussi bon que a' sur le critère j. On pose également $\lambda \in [0.5, 1]$ un seuil majoritaire. On cherche finalement donc à visualiser les relations de surclassement entre les différents candidats en modélisant l'inéquation suivante:

$$\sum_{j \in S(a,a')} w_j \geq \lambda$$

On prend arbitrairement pour les coefficients d'importance:

- 0.6 pour le critère profit
- 0.3 pour le critère nombre de projets max. mené par un collaborateur
- 0.1 pour le nombre max. de jour consécutifs passés sur un projet

Cela mène alors à la matrice de pondération des poids où les relations de surclassement entre un candidat i et un candidat j sont modélisées à ligne i et la colonne j.

$$\begin{bmatrix} 1 & 0.9 & 0.9 & 0.6 & 0.6 \\ 0.1 & 1 & 0.6 & 0.6 & 0.6 \\ 0.4 & 0.4 & 1 & 0.6 & 0.6 \\ 0.4 & 0.4 & 0.4 & 1 & 0.7 \\ 0.4 & 0.4 & 0.4 & 0.4 & 1 \end{bmatrix}$$

En prenant $\lambda = 0.6$ pour le seuil majoritaire, on obtient la matrice de surclassement (suivant l'inéquation précédente) suivante:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

La première solution ($profit = 800$, $max\_job\_per\_staff = 6$, $max\_len\_job = 34$) est donc celle qui est retenue car ne présente que des 1 dans la matrice de surclassement.

3.4 Algorithme de décision multi-objectif 2

Dans une première approche, nous considérons que le décideur n'a pas une connaissance précise en terme de poids qu'il donne aux différents critères. En revanche, il est capable de donner un avis entre certaines paires de solutions. Considérons les préférences suivantes:

- *la solution 1 est meilleure que la solution 2* (1P2)
- *la solution 1 est meilleure que la solution 3* (1P3)
- *la solution 1 est meilleure que la solution 5* (1P5)
- *la solution 4 est meilleure que la solution 5* (4P2)

Cela mène alors à modéliser les contraintes suivantes sur les poids w_j:

- **1P2**: $(w_1 + w_2 \geq \lambda \ \& \ w_3 < \lambda)$
- **1P3**: $(w_1 + w_2 \geq \lambda \ \& \ w_2 + w_3 < \lambda)$
- **1P5**: $(w_1 \geq \lambda \ \& \ w_2 + w_3 < \lambda)$
- **4P5**: $(w_1 + w_3 \geq \lambda \ \& \ w_2 + w_3 < \lambda)$

On considère de plus que les coefficients d'importance valent au moins 0.1 (choix arbitraire du décideur). La modélisation de ces contraintes sur Gurobi en ajoutant une fonction objectif de minimisation de λ et la résolution par programmation linéaire donne:

- $w_1 = 0.8$
- $w_2 = 0.1$
- $w_3 = 0.1$
- $\lambda = 0.5$

On obtient ainsi les matrices de surclassement suivantes:

$$\begin{bmatrix} 1 & 0.9 & 0.9 & 0.8 & 0.8 \\ 0.1 & 1 & 0.8 & 0.8 & 0.8 \\ 0.2 & 0.2 & 1 & 0.8 & 0.8 \\ 0.2 & 0.2 & 0.2 & 1 & 0.9 \\ 0.2 & 0.2 & 0.2 & 0.2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

De nouveau, c'est la première solution ($profit = 800$, $max\_job\_per\_staff = 6$, $max\_len\_job = 34$) qui est retenue.

4 Conclusion

...