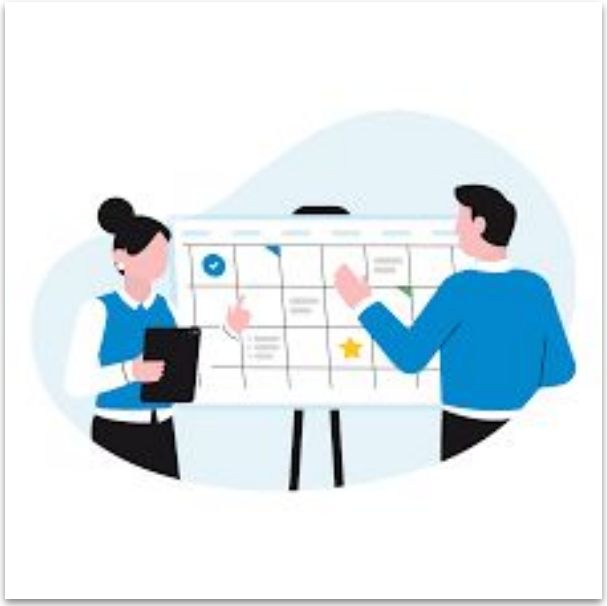
Système de Décision et Préférences

Nicolas Peruchot, Guilhem Prince, Thomas Vicaire

Contexte

- ➔ CompuOpti demande de l'aide dans le choix de ses **plannings** de personnel et d'affectation aux projets.
- ➔ Chaque projet nécessite un nombre de jour donné, par compétence
- ➔ Il faut répartir le personnel selon leurs compétences, de manière à **optimiser les bénéfices** de CompuOpti
- ➔ Des **contraintes** sont fournies sur le personnel, les projets, et les compétences.
- ➔ Modélisation grâce à **Gurobi** et proposition de discrimination des **solutions**.



Formalisme

- ★ Adéquation avec les notations du sujet
- ★ Matrice à 4 dimensions : $X_{i,j,k,t} \in \{0,1\}$ indiquant si la personne i réalise une qualification k pour le projet j, le jour t.
- ★ Reste les variables de profit, l'étendu maximale de jour travaillés sur un projet, le nombre maximum de projet par personne qui rentreront en compte dans l'optimisation
- ★ Mise en place de variable secondaire pour faciliter la mise en place des modèles (voir rapport)

Choix de modélisation

- Création des variables :

```
X = model.addVars(n_staff, horizon, n_qualifs, n_jobs, vtype=GRB.BINARY, name="assignements")
J = model.addVars(n_jobs, vtype=GRB.BINARY, name="completion")
E_D = model.addVars(n_jobs, lb=0, ub=horizon-1, vtype=GRB.INTEGER, name="end_dates")
L = model.addVars(n_jobs, lb=0, ub=horizon + 1, vtype=GRB.INTEGER, name="n_days_late")
```

- Une personne ne peut être affectée qu'à une seule tâche et une seule qualification par jour :

$$\sum_{j \in J, k \in Q} X_{i,j,k,t} \neq 1 \qquad \forall i \in S, \forall t \in H$$

```
model.addConstrs(quicksum(X[i,j,k,l] for l in range(n_jobs) for k in range (n_qualifs)) <= 1
                 for i in range(n_staff)
                 for j in range(horizon))
```

- Maximiser le gain tout en minimisant les pertes financières :

$$max\ profit = max \sum_{j \in J} (Y_j \times g_j - L_j \times c_j)$$

```
model.setObjective(
    quicksum( (J[index_job] * job["gain"] - job["daily_penalty"] * L[index_job]) for index_job, job in enumerate(jobs)),
    GRB.MAXIMIZE
```

Résultats

Fonction “mono-objectif”

$$max\ 10 * profit - max\_days - max\_jobs$$

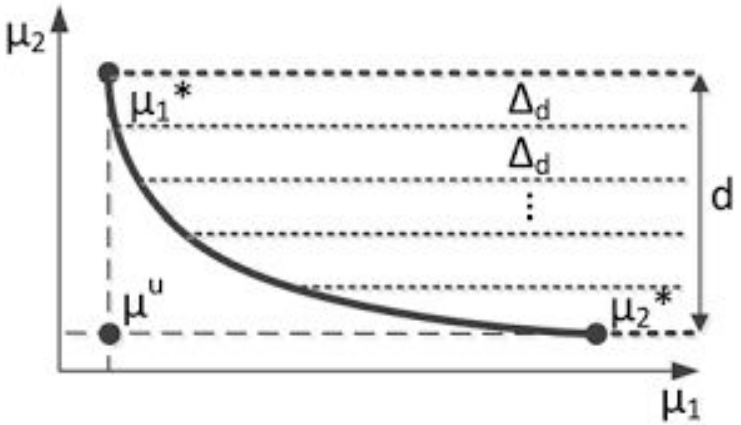
Exemple de planning pour l’instance moyenne :

FIGURE 1 – Instance moyenne

Méthode des Epsilon-contraintes

Pour trouver la surface des solutions non-dominées :

- Epsilon-contraintes, avec les critères max_days et max_jobs.
- Valeurs bornés, et entières : Epsilon = 1
- Maximiser le profit avec ces contraintes en plus : problème mono-objectif !



Résultats

Exemple de surface de solution non dominé :

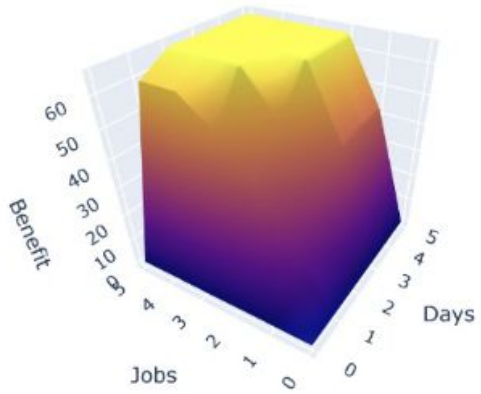


FIGURE 2 - Instance petite

Next steps

- Modèle de préférence , Implémentation en Gurobi :
  - création artificielle de préférences
  - modèle de préférences avec en variables les poids w1, w2, w3

- Utilisation de GPU pour les calculs de :
  - grands plannings
  - grandes surfaces

Des questions ?

Annexe 1 : générateur d’instance

On choisit le nombre de personne, de projet, d’horizon de temps maximum et de qualifications :

```
generate_instance(num_skills, num_people, num_jobs, max_due_date)
```

Annexe 2 : fonctions annexes

Des fonctions permettent d'ajouter plus simplement les contraintes et objectifs :

```
def in_vacation(i, j):
    data = {data["staff"][i]["name"]: data["staff"][i] for i in range(len(data["staff"]))}
    data = data[i]["vacations"]
    return j in data
```