# TP2: Problème d'emploi du temps



Ce TP à rendre et contient plusieurs questions à rédiger. Il est donc conseillé d'utiliser un environnement de programmation de type 'jupyter notebook' contenant à la fois du code exécutable et des commentaires formattés (cellules 'markdown'). Vous utiliserez ces cellules formattées pour répondre à certaines questions.

## Problème no 1

On a K créneaux horaires, m enseignants et n classes d'élèves.

```
• Créneaux : (Lu8,Lu10,...,Ve14,Ve16) - ici: K = 20.
```

- Professeurs : (Dupont, Durand, Duval, ...) par ex: m = 32.
- Classes : (6A,6B,...,3D) par ex: n = 16.

Chaque classe est suivie par 6 professeurs. Chaque professeur est affecté à 3 classes et réalise 3 séances pour chaque classe. Ainsi, chaque professeur réalise un total de 9 séances par semaine, et chaque classe suit 3x6 = 18 séances de cours (à répartir sur 20 créneaux disponibles).

**remarque**: on vérifie 3m = 6n soit m = 2n

## Affectation des professeurs

On suppose pour simplifier que les professeurs sont désignés par un numéro unique (de 0 à m-1) ainsi que les classes (classes 0 à n-1) et les créneaux (de 0 à K-1). On suppose qu'est donné au départ le tableau d'affectation (donnant les classes affectées à chaque professeur). Ainsi, pour tout i,  $A(i) = \{A(i,0),...,A(i,5)\}$  est la liste des 6 professeurs affectés à la classe i.

Indication: on veut définir la matrice des affectations A constituée de 6 lignes et 16 colonnes. Ces affectations seront effectuées de manière aléatoires. Les colonnes correspondent aux classes. Le code suivant est souvent incorrect (car un professeur paut etre affecté 2 fois à la même classe):



```
profs = list(range(32))*3
A={}
for i in range(16):
    A[i] = []
    for j in range(6):
        prof = np.random.choice(profs)
        A[i].append(prof)
        profs.remove(prof)
```

**Question** Testez ces deux codes. Réfléchissez à une méthode permettant d'obtenir une affectation valide:

- en effectuant un grand nombre de tirages (méthode de monte carlo)
- à partir d'une affectation initiale incorrecte, par permutation (méthode glouton)

remarque : Il est également possible de contourner le problème en retirant un nouveau professeur chaque fois qu'il y a collision. Le code suivant fonctionne la plupart du temps:

```
profs = list(range(32))*3
A={}
for i in range(16):
    A[i] = []
    for j in range(6):
        prof = np.random.choice(profs)
        if prof not in A[i]:
            break
    if trial<9:
        A[i].append(prof)
        profs.remove(prof)
    else:
        print("ECHEC: ESSAYEZ UNE NOUVELLE FOIS")
        break</pre>
```

#### **Emploi du temps**

Etant données une affectation A, une solution au problème d'emploi du temps consiste à définir la semaine de chaque classe sous la forme d'une séquence de K valeurs. Pour tout i, s(i) = (s(i,0), (s(i,1), s(i,2),...,s(i,19))) qui est une permutation de la liste (A(i,0), A(i,0), A(i,0), A(i,1), A(i,1), A(i,1), ..., A(i,5), A(i,5), A(i,5), -1, -1) où les valeurs -1 correspondent aux 2 séances d'"étude". Une solution prend donc la forme d'une matrice de n lignes et K colonnes.

A nouveau pour vous aider, voici les lignes de code permettant de définir un emploi du temps aléatoire:



```
s = []
for i in range(16):
        s.append([])
        for j in range(6):
            s[i].extend([A[i][j], A[i][j], A[i][j]])
        s[i].extend([-1,-1])
        s[i] = np.random.permutation(s[i])
s = np.array(s)
```

### **Questions:**

- Combien ce problème accepte-t-il de solutions?
- Comment définiriez-vous le voisin d'une solution s donnée?
- En fonction de la définition précédente, combien une solution s possède-t-elle de voisins?

### Problème:

On appelle collision le fait qu'un même professeur apparaisse 2 fois ou plus dans le même créneau (autrement dit une collision est un triplet (i,j,k) tel que (i \$\neq\$ j), (s(i,k)=s(j,k)) et s(i,k) \$\neq\$ -1). On cherche à proposer un emploi du temps dans lequel il n'y a pas de "collision".

- En vous inspirant du TP1, définissez les fonctions affiche\_edt, randomVoisin, tousLesVoisins, J, argmin\_J adaptées aux problèmes d'emploi du temps. En particulier, la fonction de coût sera égale au nombre total de collisions dans la matrice s.
- Il existe une manière de calculer le nombre de collisions en O(K\*N). Décrivez le principe de cet algorithme, puis écrivez-le.
- Reprenez certains algorithmes d'optimisation du TP1 (Monte Carlo, Glouton, Tabou, ...) qui, partant d'une solution s prise au hasard, effectuent une recherche par voisinage pour trouver un emploi du temps sans collision. Quels sont les algorithmes les plus efficaces?

# Problème no 2

Reprenez le problème précédent en considérant les contraintes suivantes:

- Chaque professeur suit 5 classes et assure 2 séances par classe.
- Chaque professeur effectue 10 séances par semaine
- 10 professeurs sont affectés à chaque classe
- Il y a 20 classes
- Il y a 40 professeurs

L'algorithme trouve-t-il toujours une solution?

From:

https://wiki.centrale-med.fr/informatique/ - WiKi informatique

Permanent link:

https://wiki.centrale-med.fr/informatique/restricted:opti-c-tp2

Last update: 2023/09/19 22:36

