Programmation par contraintes

1 Introduction

1.1 Présentation générale

L'objectif de cette série d'exercice est de programmer un calculateur d'emploi du temps scolaire. Nous commencerons avec une version minimale, qui ne prend en charge qu'une seule classe et où tous les cours ont une charge horaire d'une heure.

Une fois que cette base sera en place, nous ajouterons peu à peu différentes fonctions : tout d'abord le fait de pouvoir avoir plusieurs quota horaires par cours. Ensuite nous rajouterons la gestion de plusieurs classes (ou plusieurs groupes), puis de différentes salles.

Essentiellement, le problème de calculer un emploi du temps est *un problème d'allocation* : il s'agit de trouver comment allouer certaines ressources (e.g. salles, créneaux horaires, etc.), tout en respectant certaines contraintes (e.g. une même classe ne doit pas avoir deux cours en même temps, le quota horaire des cours doit être respecté, etc.).

Remarque (Difficulté du problème) : Il y a plusieurs variantes de ce problème; mais plus ou moins toutes ces variantes sont *difficiles* (i.e. NP-complètes, voir par exemple `https://link.springer.com/chapter/10.1007/3-540-61794-9_66`). Il est à noter qu'il existe d'autres approches que celle que nous allons adopter pour résoudre ce problème (par exemple, des approches par *algorithmes génétiques*).

Ces difficultés nous obligent à simplifier, dans le cadre de cet exercice, le cas réel.

1.2 Fichiers fournis

Le code fourni contient deux fichiers : `EdT.py` et `affichage.py`. Le second ne vous servira qu'à la toute fin de la série d'exercices, si vous utilisez les variables de décisions tel que suggéré (c.f. 5); vous pouvez l'ignorer au début du TP.

Le fichier `EdT.py` contient quelques déclaration de bases. En particulier, il contient deux classes (`Cours` et `Classe`) qui pourront vous être utile dans les sections 3 et 4 (c.f. fichier `EdT.py`, lignes 5 à 25). Ces classes représentent respectivement un cours (avec un nombre d'heures, un nom, et un identifiant unique) et une classe (avec un identifiant unique et une liste de cours à suivre).

Quelques cours et classes sont déjà créés (c.f. fichier `EdT.py`, lignes 27 à 41), mais vous pouvez bien entendu en rajouter si besoin.

Par ailleurs, quelques variables supplémentaires sont définies aux lignes 47 à 56. Celles-ci définissent par exemple le nombre de créneaux de cours par jours, le nombre de salles disponibles, etc... Toutes ne sont pas utiles dès le début du TP, mais gardez les en tête pour quand vous en aurez besoin.

2 Emploi du temps minimaliste

Afin d'avoir une première version d'un calculateur d'emploi du temps, nous allons nous limiter à une seule classe, qui a une liste de cours à faire, et où chaque cours nécessite une heure.

2.1 Modélisation

Nous pouvons modéliser cette première instance du problème de la manière suivante : nous avons un tableau à deux dimensions : sur la première dimension, nous avons n lignes, où chacune représente un créneau

horaire disponible; sur la seconde dimension, nous avons m colonnes, où chacune représente un cours que la classe doit suivre.

Chaque case du tableau contient un booléen. Pour la case aux coordonnées (i, j), le booléen vaut `True` si la classe doit suivre le cours j au créneau i, `False` sinon.

	Cours 1	Cours 2	Cours 3
Créneau 1	`True`	`False`	`False`
Créneau 2	`False`	`False`	`True`
Créneau 3	`False`	`True`	`False`
Créneau 4	`False`	`False`	`False`

(a) Exemple d'une allocation correcte de trois cours sur trois créneaux : tous les cours sont alloués, et le sont sur des créneaux différents.

	Cours 1	Cours 2	Cours 3
Créneau 1	`True`	`False`	`False`
Créneau 2	`False`	`False`	`False`
Créneau 3	`False`	`True`	`True`
Créneau 4	`False`	`False`	`False`

(b) Exemple d'une allocation incorrecte de trois cours sur trois créneaux : les cours 2 et 3 sont alloués sur le même créneau.

À partir de ce modèle, nous pouvons exprimer les *contraintes* que doivent respecter les emplois du temps valides :

1. Il ne doit pas y avoir deux cours en même temps
2. Chaque cours doit avoir un unique créneau alloué

Par exemple (et sachant que Python converti `True` en `1` et `False` en 0), la première contrainte peut s'exprimer ainsi, où `allocation` est le tableau qui contient l'emploi du temps :

$$\forall 0 < \texttt{creneau} \leq n \cdot \left(\sum_{0 < \texttt{cours\_id} \leq m} \texttt{allocation}[\texttt{creneau}][\texttt{cours\_id}] \right) \leq 1$$

Question 1. *Formalisez la contrainte 2.*

Solution:

Question 2. *Dans le fichier python fourni, créez un ensemble de variables de décisions (le tableau de booléen), puis ajoutez les contraintes énoncées ci-dessus.*

Vous pouvez utiliser (et eventuellement ajuster) les variables fournies `max_creneau` et `max_cours` pour tester différentes configurations.

Nous allons maintenant ajouter l'infrastructure pour trouver une solution satisfaisante au problème.

Question 3. *À l'aide des méthodes `solve` (méthode de la classe `Model`) et `value` (méthode définie sur les différentes classes de variable de décision), écrivez un programme qui trouve et affiche (sommairement) une solution (i.e. qui trouve un emploi du temps satisfaisant les différentes conditions).*

3 Ajout du quota horaire

On souhaite maintenant pouvoir spécifier un quota horaire pour les cours. C'est-à-dire qu'un cours puisse avoir plus qu'une seule heure allouée.

Question 4. *Admettons qu'on souhaite que les cours aient deux heures. Quelle contrainte doit-on modifier ? Que devient-elle ?*

Solution:

Admettons qu'on souhaite que certains cours aient des durées différentes. Pour ça, on peut se donner un tableau de `Cours` qu'on appelle `cours`. Chaque `Cours` a un champs `nb_heures` qui indique le quota horaire du cours en question. Par exemple, si un cours `c` requiert 3h, alors `c.nb_heures` vaut `3`.

Question 5. *Adaptez la seconde contrainte pour prendre en compte les différents quotas horaires des cours (vous pouvez bien entendu utiliser le dictionnaire `cours`, où `cours[cours_id]` vous renvoie le `Cours` adéquat.).*

Solution:

Question 6. *Adaptez la réponse à vos questions 2 et 3, afin de prendre en compte les quotas horaires. Le code fourni défini une classe `Cours` qui contient un champs `nb_heures`. Un tableau de cours `cours` est alloué, que vous pouvez utiliser.*

4 Ajout des classes

On souhaite maintenant avoir plusieurs classes. Chaque classe a une liste de cours à suivre[1]. La différence par rapport à ce qui précède est que, maintenant, deux cours peuvent être donnés en même temps, si ces cours sont donnés à deux classes différentes.

Remarque : Le fichier fourni déclare un tableau de classes nommé `classes`. En l'occurrence, il contient deux classes. Vous pouvez utiliser ce tableau pour essayer votre programme.

Pour cela, nous allons devoir modifier légèrement notre modélisation. Il y a plusieurs manières de faire. Je vous propose d'opter pour l'approche qui consiste à rajouter une troisième dimension au tableau de booléens que nous avons vu plus haut (voir Figure 2).

Question 7. *Modifiez votre déclaration des variables afin de prendre en compte plusieurs classes (Précisez `shape=(nb_creneaux, nb_cours, nb_classes)`).*

[1] Si plusieurs classes ont le même cours, pour simplifier la modélisation, on considère que les cours ne sont pas mutualisés : ils n'ont pas nécessairement lieu au même moment.

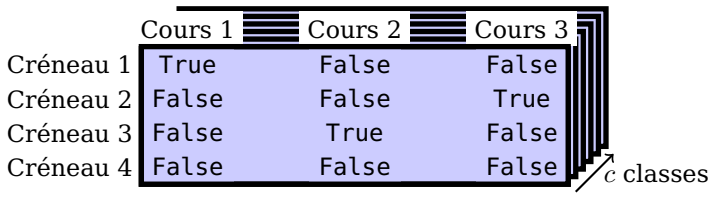

Figure 2: Modélisation avec différentes classes. Il est possible de préciser `shape=(nb_creneaux, nb_cours, nb_classes)` au moment de la déclaration des variables pour avoir un tableau de trois dimensions, incluant les classes.

Il faut maintenant adapter les contraintes que l'on spécifiées plus haut afin de gérer ces classes. Il y a plusieurs manières de faire :

- Itérer sur toutes les classes, et ajouter une contrainte par classe (e.g. en utilisant une boucle for).
- Ajouter une seule contrainte qui consiste en la conjonction de plusieurs sous-contraintes (une par classe), en utilisant `cp.all()`.

Question 8. *Modifiez les contraintes afin de prendre en charge plusieurs classes. (Vous pouvez, sur un brouillon, essayer les deux approches précisées ci-dessus.)*

5 Ajout des salles

Nous avons maintenant plusieurs classes qui peuvent avoir cours en même temps, mais nous ne prenons pas en compte la disponibilité de salles. La dernière étape de cette série d'exercice est de rajouter cette prise en compte.

Le fichier fourni déclare la variable `max_salles` qui contient le nombre maximum de salles disponibles.

Modifiez vos variable afin de rajouter une dimension au tableau de booléens, qui correspond aux différentes salles possibles (utilisez `shape=(max_creneau, max_salles, max_cours, max_classe)`).

Question 9 (Question non dirigée)**.** *Ajoutez la prise en charge d'un nombre de salle limité.*

Si vous avez déclaré vos variables de décision comme suggéré, vous pouvez décommenter la dernière ligne pour afficher votre emploi du temps.