Rapport de planification OR-Tools

BOYER Hugo, KRIKA Camila, KRIKA Jinane

24 janvier 2025

Table des matières

Chapitre 1

Introduction

Ce rapport décrit un problème de planification universitaire résolu à l'aide de la bibliothèque `OR-Tools` de Google. Le projet met en place une application complète grâce à une interface mobile développée sous `React Native`, et une API Python basée sur `Flask` pour la communication entre le front-end et le solveur d'optimisation.

L'objectif consiste à organiser, dans un même emploi du temps, deux types de cours :
— des **Cours Magistraux (CM)**, obligatoirement dispensés en ligne,
— des **Travaux Dirigés (TD)**, obligatoirement dispensés en présentiel.
L'application front-end (*React Native*) interroge une API Flask qui exécute l'algorithme de planification, modélisé comme un problème d'optimisation linéaire en nombres entiers (*Integer Linear Programming* ou ILP).

Chapitre 2

Présentation générale du problème

2.1 Ensembles et paramètres

Pour décrire la planification, on utilise les ensembles et paramètres suivants :

— **Groupes d'étudiants** G : par exemple, `DIA1, DIA2`, etc.
— **Enseignants** E : liste d'enseignants, chacun ayant des spécialités définies (matières qu'il peut enseigner).
— **Salles** R : salles physiques, chacune avec une capacité maximale.
— **Cours** C : chaque cours est de type CM ou TD.
— **Périodes** P : ensemble des créneaux horaires (par exemple, 4 créneaux par jour sur 5 jours).
— **Modalités** M : deux modalités possibles : *en ligne* ou *présentiel*.
— **Durée des cours** $d_c[c]$: nombre de créneaux qu'un cours c doit occuper.
— **Effectif des groupes** n_g : nombre d'étudiants dans chaque groupe g, utile pour vérifier la capacité des salles.

2.2 Objectif

L'objectif est de **minimiser le nombre total de créneaux** utilisés pour planifier l'ensemble des cours, de manière à concevoir un emploi du temps « compact ». Il s'agit donc d'une fonction de coût réduisant $\sum x[c, p, r, e]$, où x représente le fait de planifier un cours c au créneau p, dans la salle r, avec l'enseignant e.

Chapitre 3

Contraintes de planification

Le modèle prend notamment en compte les contraintes suivantes :

1. **Modalité imposée** (CM en ligne, TD en présentiel).
2. **Spécialité de l'enseignant** : un enseignant ne peut dispenser une matière que s'il y est spécialisé.
3. **Disponibilité des enseignants** : un enseignant ne peut pas enseigner deux cours au même créneau.
4. **Capacité de la salle** : pour un cours en présentiel, la salle choisie doit pouvoir accueillir le nombre total d'étudiants.
5. **Occupation des salles** : une salle ne peut pas héberger deux cours à la même période.
6. **Durée du cours** : chaque cours c doit occuper exactement $d_c[c]$ créneaux en tout.

Chapitre 4

Structure de l'application

4.1 Interface mobile (React Native)

Une application mobile développée en *React Native* permet de saisir dynamiquement les données suivantes :

— Les noms de groupes,
— Les salles disponibles (et leurs capacités),
— Les matières à planifier (avec indication CM ou TD),
— Les enseignants (avec leurs spécialités).

Une fois ces informations validées, l'application envoie une requête `POST` à l'API Flask, au format `JSON`. Les paramètres nécessaires sont passés au solveur, qui renvoie ensuite le résultat au format `JSON`. Enfin, l'interface affiche ce planning sous forme de calendrier interactif (jours vs. créneaux).

4.2 API Flask (Python)

Une API a été mise en place à l'aide de `Flask`. Son rôle est d'exposer un point d'entrée (e.g. `/schedule`) qui reçoit la requête du front-end et exécute :

— La fonction `determine_course_type` (si besoin) pour distinguer « CM » et « TD » d'après les noms des matières,
— La fonction de planification `university_scheduling`, qui construit et résout le modèle d'optimisation à l'aide d'OR-Tools.

Une fois le solveur terminé, l'API renvoie au client (*React Native*) soit les affectations de cours (créneaux, salles, enseignants, modalité) si le problème est faisable, soit un message d'erreur sinon.

Chapitre 5

Résultats et conclusions

5.1 Résultats obtenus

Après exécution, l'interface mobile reçoit un `JSON` contenant la planification calculée par OR-Tools. Chaque entrée du planning spécifie :

— La classe (groupe d'étudiants),
— La matière (ex. `Maths CM`),
— L'enseignant attribué,
— Le créneau horaire (ex. `Période 1`),
— La salle (ou « en ligne » pour un CM),
— La modalité (présentiel ou en ligne).

Le front-end reconstitue alors un **calendrier** hebdomadaire pour chaque groupe sélectionné, en affichant un bloc de couleur spécifique selon la modalité (présentiel ou en ligne). Si une solution optimale est trouvée, on observe en général une réduction du nombre total de créneaux occupés.

5.2 Perspectives d'extension

Ce modèle peut être enrichi par :

— La gestion plus fine des disponibilités des enseignants (indisponibilité le lundi matin, etc.).
— La possibilité de faire varier le nombre de créneaux par jour, ou le nombre de jours dans une semaine.
— L'ajout de contraintes plus complexes (pourcentage minimal de cours en ligne, continuité de certains cours, etc.).
— L'export des résultats dans des formats de calendrier (.ics).

Chapitre 6

Conclusion

La mise en place d'un solveur OR-Tools, couplé à une API Flask, et relié à une interface *React Native*, offre une solution modulaire pour la planification universitaire. Les utilisateurs peuvent configurer facilement les paramètres (groupes, salles, matières, enseignants), lancer le calcul et visualiser immédiatement l'emploi du temps généré.

Le modèle présenté illustre un cas d'étude « CM en ligne » vs. « TD en présentiel », mais peut être étendu et adapté à des scénarios beaucoup plus complexes.