TD LOG
À faire pour le 25/11
Groupe : Léo Wang, Alexandre Tranie, Matthieu Mayaud

Compte rendu du code : Calendrier ligue des champions

1) Générale :

Ce code a été réalisé pour effectuer le tirage de la ligue des champions dans son nouveau format et ainsi pouvoir proposer un calendrier avec 8 journées où toutes les équipes jouent. Deux méthodes sont proposées pour le faire :

- « Schedule First » : Un planning prédéfini de 8 jours est créé avec des emplacements réservés pour les équipes. Les équipes sont ensuite affectées à ces emplacements tout en respectant les contraintes, comme jouer contre des équipes de pots différents et alterner les matchs à domicile et à l'extérieur. Cette méthode implique de résoudre un problème de programmation linéaire en utilisant le solveur Gurobi pour optimiser le planning.

- « Matches First » : Les matchs sont attribués séquentiellement pour chaque équipe, en s'assurant que les matchs respectent les contraintes avant de planifier les journées de match. Le solveur Gurobi est utilisé pour optimiser la sélection des matchs, garantissant la faisabilité et le respect des contraintes de la compétition. Cette méthode offre une plus grande flexibilité pour des scénarios de planification réels, comme la disponibilité des stades ou les conditions météorologiques.

2) Schedule First :

Cette méthode est codé à l'aide de deux fichiers : *draw_build_schedule_first.jl* et *template_optimal_break.*

3.1) **draw_build_schedule_first.jl**:

Tout d'abord, ce code définit les constantes tel que le nombre d'équipes (36), de plots (4) et le nombre d'équipe par plots (9).

Ensuite, une liste opponent est créé pour définir les adversaires possibles de chaque équipes i → opponent[i] détermine les adversaires possibles pour l'équipe i. Au début, la liste opponent correspond au placeholder différents et se remplit des équipes au fur et à mesure.

De même, une liste teams de dictionnaire est créée avec les informations pour chaque équipe : club (le nom du club), nationalité (pays du club), classement ELO (indice de performance) et classement UEFA.

Une liste team_nationalities est également crée pour avoir directement le pays du club i. C'est un nombre entier qui indique le pays du club. Cela aide pour les contraintes de ne pas avoir des clubs d'un même pays qui s'affrontent.

La liste nationalities indique la liste des clubs appartenant à un pays et permet donc de savoir combien de clubs sont dans tel pays ou autre. Le nombre total de nationalité est aussi décrit.

Gurobi.env() est codé pour initialiser l'environnement du solveur Gurobi, qui sera utilisé pour résoudre les contraintes d'optimisation. Gurobi est un solveur de programmation mathématique utilisé pour gérer les contraintes complexes. C'est ici que toutes les variables et les contraintes seront traduites en un problème mathématique à résoudre.

Par la suite, une fois que les constantes et variables sont définis, des fonctions sont également définis. La première est la fonction is_solvable qui a pour rôle de vérifier si une équipe donnée (new_team) peut être placée dans un certain placeholder **(**new_placeholder**)** tout en respectant l'ensemble des contraintes imposées par le problème. Dans les paramètres de la fonction, on a already_filled qui est la liste des équipes déjà assignés à des placeholder spécifiques. La fonction renvoie true si la solution est possible et false sinon.

La deuxième fonction admissible_teams retourne la liste des équipes qui peuvent être placées dans un certain placeholder, tout en respectant les contraintes imposées par le tirage. Elle retourne une liste d'entiers (possible_teams), correspondant aux équipes qui peuvent être placées dans le placeholder donné, tout en respectant les contraintes.

Par la suite, on définit la fonction draw qui réalise le tirage complet des équipes en assignant chaque équipe à un placeholder, tout en respectant les contraintes définies (nationalités, pots, etc.). Elle utilise les équipes admissibles identifiées par la fonction admissible_teams. En sortie, elle nous donne une liste already_filled, où chaque index correspond à un placeholder, et la valeur à l'équipe qui y est assignée.

La dernière fonction de ce fichier est la fonction strength_opponents. Elle simule plusieurs tirages, calcule les forces des adversaires pour chaque équipe en termes d'ELO et de coefficient UEFA, et sauvegarde les résultats pour une analyse ultérieure dans des document .txt. Elle garantit que chaque simulation respecte les contraintes du tirage tout en offrant une évaluation précise de la difficulté relative pour chaque équipe.

Pour conclure, ce fichier permet de placer chaque équipe dans des placeholder afin de déterminer via la liste opponent les adversaires de chaque équipe pour respecter toutes les contraintes. Ce fichier détermine seulement cela mais ne donne pas un calendrier ou le détail des rencontres domiciles/extérieures.

3.2) **template_optimal_break:**

Ce code génère un calendrier optimisé de matchs, en respectant de nombreuses contraintes pour assurer l'équité et la faisabilité. Il produit des fichiers CSV permettant d'exploiter les résultats facilement. Tout d'abord, il y a la Création du modèle d'optimisation : on utilise JuMP avec le solver Gurobi et des variables x[i, j, t] qui sont égale à 1 si l'équipe i joue contre l'équipe j à la journée t et break_var[i, t] est égale à 1 si une équipe i subit un break à la journée t (joue deux fois de suite à domicile ou à l'extérieur). De plus, toutes les contraintes du modèle sont définies tel que le fait qu'une équipe ne peut jouer qu'une seule fois contre un adversaire, que chaque équipe joue contre deux équipes de chaque pot dont un match à domicile et un a l'extérieur. Suite à la création du model, on le résout avec « optimize!(model) ». On regarde s'il existe une solution et on fait tous les changements pour mettre en forme à savoir associer le nombre à un nom d'équipe, créer des fichiers csv, etc. Ce code semble indépendant de l'autre et réalise la création du calendrier en total indépendance et en respectant toutes les contraintes.

Explication de certains bouts de code :

- Une équipe ne peut pas jouer contre elle-même :
  ```
  @constraint(model, no_self_play[i in 1:N, t in 1:T], x[i, i, t] == 0)
  ```

- Une équipe joue au plus un match par journée :
  ```
  @constraint(model, one_game_per_day[t in 1:T, i in 1:N], sum(x[i, j, t] + x[j, i, t] for j in 1:N) == 1)
  ```

- Chaque équipe joue une fois contre chaque adversaire de son pot :
  ```
  for i in 1:4
     for t in 2:6
        @constraint(model, sum(x[(i-1)*9+1, j, t] + x[(i-1)*9+1, j, t+1] for j in 1:N) <= 1 + break_var[i, t])
     end
     @constraint(model, sum(break_var[i, t] for t in 2:6) <= 1)
  End
  ```

- Une fois le modèle résolu, les variables x[i, j, t] contiennent le calendrier des matchs. Bien que la sortie explicite des résultats ne soit pas affichée dans la partie solveur, elle peut être récupérée à l'aide de :
  ```
  for t in 1:T
     for i in 1:N
        for j in 1:N
           if value(x[i, j, t]) == 1
              println("Match : Équipe $i contre Équipe $j à la journée $t")
           end
        end
     end
  end
  ```