TD LOG Projet

Étude des codes Julia du projet Tirage LDC

▼ Tirage Gurobi fast

1. Bibliothèques

```
using Gurobi, JuMP, CSV, DataFrames, Random
```

- `Gurobi` : Bibliothèque pour la résolution de problèmes d'optimisation linéaire et mixte.
- `JuMP` : Framework pour modéliser et résoudre des problèmes d'optimisation en Julia.
- `CSV` : Pour manipuler des fichiers CSV.
- `DataFrames` : Pour gérer des données tabulaires.
- `Random` : Pour des fonctionnalités liées au hasard (ex. mélanger des éléments).

---

2. Définition des structures

(a) Structure `Team`

```
struct Team
    club::String
    nationality::String
    elo::Int
    uefa::Float64
end
```

- Représente une équipe avec :
  - `club` : nom de l'équipe (ex. "Real").
  - `nationality` : pays d'origine.
  - `elo` : classement Elo du club estimée.
  - `uefa` : coefficient UEFA.

(b) Structure `TeamsContainer`

```
struct TeamsContainer
    pot1::NTuple{9, Team}
    pot2::NTuple{9, Team}
    pot3::NTuple{9, Team}
    pot4::NTuple{9, Team}
    index::Dict{String, Team}
end
```

- Organise les équipes en 4 chapeaux, chacun contenant 9 équipes.
- Inclut un dictionnaire (`index`) pour accéder rapidement aux équipes par leur nom.

(c) Structure `Constraint`

```
struct Constraint
    played_home::Set{String}
    played_ext::Set{String}
    nationalities::Dict{String, Int}
end
```

- Définit des contraintes pour chaque équipe :
  - `played_home` : équipes contre lesquelles cette équipe a joué à domicile.
  - `played_ext` : équipes contre lesquelles elle a joué à l'extérieur.
  - `nationalities` : suivi du nombre de matchs contre chaque nationalité.

Création et manipulation des containers

(d) `create_teams_container`

```
function create_teams_container(
    pot1::NTuple{9, Team},
    pot2::NTuple{9, Team},
    pot3::NTuple{9, Team},
    pot4::NTuple{9, Team}
)::TeamsContainer
    index = Dict(team.club => team for pot in (pot
1, pot2, pot3, pot4) for team in pot)
    return TeamsContainer(pot1, pot2, pot3, pot4, i
ndex)
end
```

- Prend 4 pots d'équipes et retourne une instance de `TeamsContainer`.
- Construit un dictionnaire `index` associant chaque nom de club à sa structure `Team`.

---

3. Initialisation des équipes

(a) Création des pots

```
pot1 = (
    Team("Real", "Spain", 1985, 136),
    ...
)
pot2 = ...
```

```
pot3 = ...
pot4 = ...
```

- Chaque pot contient 9 équipes avec leurs informations (`club`, `nationality`, `Elo`, `Coef UEFA`).

(b) Création d'un `TeamsContainer`

```
const teams = create_teams_container(pot1, pot2, po
t3, pot4)
```

- Appelle une fonction pour :
  1. Organiser les équipes en pots.
  2. Créer un dictionnaire `index` pour les recherches rapides.

---

4. Fonctions utilitaires

(a) `create_club_index`

```
function create_club_index(teams::TeamsContainer)::
Dict{String, Int}
    club_index = Dict{String, Int}()
    for (i, pot) in enumerate((teams.pot1, teams.po
t2, teams.pot3, teams.pot4))
        for (j, team) in enumerate(pot)
            club_index[team.club] = (i - 1) * 9 + j
        end
    end
    return club_index
end


const club_index = create_club_index(teams)
```

- Associe chaque club à un index unique basé sur son pot et sa position dans le pot.

(b) `get_li_nationalities`

```
function get_li_nationalities(teams::TeamsContaine
r)::Set{String}
    nationalities = Set{String}()
    for pot in (teams.pot1, teams.pot2, teams.pot3,
teams.pot4)
        for team in pot
            push!(nationalities, team.nationality)
        end
    end
    return nationalities
end

const all_nationalities = get_li_nationalities(team
s)
```

- Retourne un ensemble de toutes les nationalités représentées dans les équipes.

(c) `get_index_of_team`

```
function get_index_of_team(team_name::String)::Int
    return club_index[team_name]
end
```

- Renvoie l'index d'une équipe donnée à partir de son nom.

(d) `get_team_nationality`

```
function get_team_nationality(teams::TeamsContaine
r, index::Int)::String
    pot_index = div(index - 1, 9) + 1  # Détermine
le pot (1 à 4)
    team_index = (index - 1) % 9 + 1   # Détermine
l'index dans le pot (1 à 9)

    # Récupérer le bon pot en fonction de pot_index
```

```
    if pot_index == 1
        return teams.pot1[team_index].nationality
    elseif pot_index == 2
        return teams.pot2[team_index].nationality
    elseif pot_index == 3
        return teams.pot3[team_index].nationality
    elseif pot_index == 4
        return teams.pot4[team_index].nationality
    else
        error("Index out of bounds")
    end
end
```

- Retourne la nationalité d'une équipe à partir de son index.

(e) `get_team`

```
function get_team(team_name::String)::Team
    return teams.index[team_name]
end
```

- Renvoie les informations complètes (`Team`) pour une équipe donnée.

---

5. Initialisation et mise à jour des contraintes

(a) `initialize_constraints`

```
function initialize_constraints(teams::TeamsContain
er, all_nationalities::Set{String})::Dict{String, C
onstraint}
    constraints = Dict{String, Constraint}()
    for pot in (teams.pot1, teams.pot2, teams.pot3,
teams.pot4)
        for team in pot
            team_nationalities = Dict(nat => 0 for
nat in all_nationalities)
            team_nationalities[team.nationality] =
```

```
2
            constraints[team.club] = Constraint(Set
{String}(), Set{String}(), team_nationalities)
        end
    end
    return constraints
end
```

- Initialise les contraintes de chaque équipe.
- Chaque club commence avec une contrainte définissant :
  - Les nationalités des équipes adverses (initialisé à 0 sauf pour sa propre nationalité, fixée à 2).
  - Aucun match joué au début.

(b) `update_constraints`

```
function update_constraints(home::Team, away::Team,
constraints::Dict{String, Constraint})
    push!(constraints[home.club].played_home, away.
club)
    push!(constraints[away.club].played_ext, home.c
lub)
    constraints[home.club].nationalities[away.natio
nality] += 1
    constraints[away.club].nationalities[home.natio
nality] += 1
end
```

- Met à jour les contraintes :
  1. Ajoute les équipes jouées à domicile/extérieur.
  2. Incrémente les compteurs de matchs par nationalité.

Other `Silence_output`

```
function silence_output(f::Function)
    original_stdout = stdout
```

```
    original_stderr = stderr
    redirect_stdout(devnull)
    redirect_stderr(devnull)
    try
        return f()
    finally
        redirect_stdout(original_stdout)
        redirect_stderr(original_stderr)
    end
end
```

La fonction `silence_output` sert à **exécuter une autre fonction** sans afficher les messages ou les erreurs dans la console. Elle redirige les sorties (messages affichés) vers `devnull` pendant l'exécution de la fonction, puis les remet à la normale après.

Ça permet de cacher les messages d'exécution et éviter une surcharge de la console.

---

6. Fonction d'optimisation

(a) `solve_problem`

```
function solve_problem(selected_team::Team, constra
ints::Dict{String, Constraint}, new_match::NTuple
{2, Team})::Bool
    ...
end
```

- Définit et résout un problème d'optimisation :
  - Variables : `match_vars[i, j, t]` (binaire, vaut 1 si l'équipe `i` joue contre `j` à la journée `t`).
  - Contraintes :
    1. Une équipe ne peut jouer contre elle-même.
    2. Une paire d'équipes joue au plus une fois.
    3. Limites sur les nationalités dans un pot.

- Résolution : Retourne `true` si une solution existe.

---

7. Gestion des matchs admissibles

(a) `filter_team_already_played_home` et `filter_team_already_played_away`

```
function solve_problem(selected_team::Team, constra
ints::Dict{String, Constraint}, new_match::NTuple
{2, Team})::Bool
    model = Model(Gurobi.Optimizer; add_bridges=fal
se)
    set_optimizer_attribute(model, "Seed", rand(1:1
000000000)) # random solution
    set_optimizer_attribute(model, "OutputFlag", 0)
    T=8

    @variable(model, match_vars[1:36, 1:36, 1:8], B
in)

    # Objective function is trivial since we're not
maximizing or minimizing a specific goal
    @objective(model, Max, 0)

    # General constraints
    @constraint(model, [i=1:36], sum(match_vars[i,
i, t] for t in 1:8) == 0) # A team cannot play agai
nst itself

    @constraint(model, [i=1:36, j=1:36; i != j], su
m(match_vars[i, j, t] + match_vars[j, i, t] for t i
n 1:8) <= 1)  # Each pair of teams plays at most on
ce



    # Contraintes spécifiques pour chaque pot
    for pot_start in 1:9:28
        @constraint(model, [i=1:36], sum(match_vars
```

```
[i, j, t] for t in 1:8, j in pot_start:pot_start+8)
== 1)
        @constraint(model, [i=1:36], sum(match_vars
[j, i, t] for t in 1:8, j in pot_start:pot_start+8)
== 1)
    end


    # Constraint for the initially selected admissi
ble match
    home_idx, away_idx = get_index_of_team(new_matc
h[1].club), get_index_of_team(new_match[2].club)
    selected_idx = get_index_of_team(selected_team.
club)
    @constraint(model, sum(match_vars[selected_idx,
home_idx, t] for t in 1:T) == 1)
    @constraint(model, sum(match_vars[away_idx, sel
ected_idx, t] for t in 1:T) == 1)

    # Applying constraints based on previously play
ed matches and nationality constraints
    for (club, cons) in constraints
        club_idx = get_index_of_team(club)
        for home_club in cons.played_home
            home_idx = get_index_of_team(home_club)
            @constraint(model, sum(match_vars[club_
idx, home_idx, t] for t in 1:T) == 1)
        end
        for away_club in cons.played_ext
            away_idx = get_index_of_team(away_club)
            @constraint(model, sum(match_vars[away_
idx, club_idx, t] for t in 1:T) == 1)
        end
    end

    # Nationality constraints
    for (i, pot_i) in enumerate((teams.pot1, teams.
pot2, teams.pot3, teams.pot4))
```

```
        for (j, team_j) in enumerate(pot_i)
            team_idx = (i - 1) * 9 + j
            for (k, pot_k) in enumerate((teams.pot
1, teams.pot2, teams.pot3, teams.pot4))
                for (l, team_l) in enumerate(pot_k)
                    if team_j.nationality == team_
l.nationality && team_idx != ((k - 1) * 9 + l)
                        @constraint(model, sum(matc
h_vars[team_idx, (k - 1) * 9 + l, t] for t in 1:T)
== 0)
                    end
                end
            end
        end
    end

    for nationality in all_nationalities
        for i in 1:36
            @constraint(model, sum(
                match_vars[i, j, t] + match_vars[j,
i, t]
                for t in 1:8
                for j in 1:36
                if get_team_nationality(teams, j) =
= nationality
            ) <= 2)
        end
    end

    # Solve the problem
    optimize!(model)

    return termination_status(model) == MOI.OPTIMAL
end
```

- Modélise un problème d'optimisation pour vérifier si un match donné est faisable.

- `match_vars` : Variables binaires indiquant si deux équipes jouent un match à un moment donné.
- Ajoute des contraintes
  - Une équipe ne peut pas jouer contre elle-même.
  - Chaque paire d'équipes joue au plus une fois.
  - Respect des contraintes spécifiques (nationalités, matchs déjà joués).
- Vérifie si la solution est optimale.

(b) `filter_team_already_played_home`

```
function filter_team_already_played_home(selected_tea
    li_home_selected_team = constraints[selected_team
    for home_club_name in li_home_selected_team
        home_team = get_team(home_club_name)
        if home_team in opponent_group
            return home_team
        end
    end
    return nothing
end
```

- Trouver si l'équipe sélectionnée a déjà joué à domicile contre une équipe spécifique dans le groupe adverse.

(c) `filter_team_already_played_away`

```
function filter_team_already_played_away(selected_tea
    li_away_selected_team = constraints[selected_team
    for away_club_name in li_away_selected_team
        away_team = get_team(away_club_name)
        if away_team in opponent_group
            return away_team
        end
    end
```

```
    return nothing
end
```

- Même logique que `filter_team_already_played_home`, mais pour les matchs joués à l'extérieur.

(d) `true_admissible_matches`

```
function true_admissible_matches(selected_team::Tea
m, opponent_group::NTuple{9, Team}, constraints::Di
ct{String, Constraint})::Vector{Tuple{Team, Team}}
    true_matches = Vector{Tuple{Team, Team}}()
    #Si on a déjà tirer un adversaire pour l'équipe
sélectionné, on en s'embête pas à regarder tous les
couples (home,away) possible
    home_team = filter_team_already_played_home(sel
ected_team, opponent_group, constraints)
    away_team = filter_team_already_played_away(sel
ected_team, opponent_group, constraints)

    #On pourrait directement renvoyer (home_team, a
way_team) on fait le test par précaution
    if home_team != nothing && away_team != nothing
&& home_team != away_team
        match = (home_team, away_team)
        if home_team.nationality != selected_team.n
ationality && away_team.nationality != selected_tea
m.nationality
            if solve_problem(selected_team, constra
ints, match)
                push!(true_matches, match)
            end
        end
    end

    if home_team == nothing && away_team == nothing
        for home in opponent_group
            for away in opponent_group
```

```
                if home != away && home.nationality
!= selected_team.nationality && away.nationality !=
selected_team.nationality&&
                constraints[selected_team.club].nat
ionalities[home.nationality] <= 2 &&
                constraints[selected_team.club].nat
ionalities[away.nationality] <= 2 &&
                constraints[home.club].nationalitie
s[selected_team.nationality] <= 2 &&
                constraints[away.club].nationalitie
s[selected_team.nationality] <= 2 &&
                filter_team_already_played_away(hom
e, opponent_group, constraints) == nothing  &&#On v
érfie que home ne s'est pas déjà déplacé
                filter_team_already_played_home(awa
y, opponent_group, constraints) == nothing
                    match = (home, away)
                    if solve_problem(selected_team,
constraints, match)
                        push!(true_matches, match)
                    end
                end
            end
        end
    end

    if home_team == nothing && away_team != nothing
        for home in opponent_group
            if home != away_team &&
               home.nationality != selected_team.na
tionality &&
               away_team.nationality != selected_te
am.nationality &&
               constraints[selected_team.club].nati
onalities[home.nationality] <= 2 &&
               constraints[home.club].nationalities
[selected_team.nationality] <= 2 &&
               filter_team_already_played_away(hom
```

```
e, opponent_group, constraints) == nothing
                match = (home, away_team)
                if solve_problem(selected_team, con
straints, match)
                    push!(true_matches, match)
                end
            end
        end
    end

    if home_team != nothing && away_team == nothing
        for away in opponent_group
            if home_team != away &&
               home_team.nationality != selected_te
am.nationality &&
               away.nationality != selected_team.na
tionality &&
               constraints[selected_team.club].nati
onalities[away.nationality] <= 2 &&
               constraints[away.club].nationalities
[selected_team.nationality] <= 2 &&
               filter_team_already_played_home(awa
y, opponent_group, constraints) == nothing
                match = (home_team, away)
                if solve_problem(selected_team, con
straints, match)
                    push!(true_matches, match)
                end
            end
        end
    end
    return true_matches
end
```

- Retourne les matchs valides (home, away) pour une équipe sélectionnée, en respectant toutes les contraintes.
    - Si l'équipe sélectionnée a déjà joué contre deux équipes (domicile/extérieur), teste directement ces matchs.

- Sinon, parcourt toutes les combinaisons de matchs possibles (domicile/extérieur) dans le groupe adverse.
- Valide chaque combinaison avec `solve_problem`.
- Retourne les matchs admissibles.

---

8. Tirage au sort

(a) `tirage_au_sort`

```
function tirage_au_sort(constraints::Dict{String, C
onstraint}; sequential=false)
    println("Début de la fonction tirage_au_sort2")
    start_time = time()
    matches_list = []

    open("tirage_au_sort.txt", "w") do file
        for pot_index in 1:4
            println("Mélange des indices pour le po
t $pot_index")
            indices = shuffle!(collect(1:9))  # Mél
ange des indices

            # Accès au pot correspondant dans Teams
Container
            pot = if pot_index == 1
                teams.pot1
            elseif pot_index == 2
                teams.pot2
            elseif pot_index == 3
                teams.pot3
            elseif pot_index == 4
                teams.pot4
            else
                error("Index de pot invalide")
            end

            for i in indices
```

```
                selected_team = pot[i]
                li_opponents = [(selected_team.clu
b, "")]

                println(file, "Tirage pour l'équipe
: ", selected_team.club)

                for idx_opponent_pot in 1:4
                    opponent_pot = if idx_opponent_
pot == 1
                        teams.pot1
                    elseif idx_opponent_pot == 2
                        teams.pot2
                    elseif idx_opponent_pot == 3
                        teams.pot3
                    elseif idx_opponent_pot == 4
                        teams.pot4
                    else
                        error("Index de pot invalid
e")
                    end

                    matches_possible = true_admissi
ble_matches(selected_team, opponent_pot, constraint
s)

                    equipes_possibles = [(match[1].
club, match[2].club) for match in matches_possible]
                    selected_match = matches_possib
le[rand(1:end)]
                    home, away = selected_match

                    println(file, "Adversaires poss
ibles du pot $idx_opponent_pot : ", equipes_possibl
es)

                    if sequential
                        println("")
```

```
                        println("Equipe sélectionné
e: $(selected_team.club)")
                        println("Pot sélectionné:
$(idx_opponent_pot)")
                        println("")
                        println("Liste des couples
possibles")
                        println(equipes_possibles)
                        println("")
                        println("Match sélectionné
dans le pot $(idx_opponent_pot) : $(home.club) vs
$(away.club)")
                        println("Appuyez sur la bar
re d'espace suivi d'Entrée pour continuer...")

                        while true
                            input = readline()
                            if input == " "
                                break
                            else
                                println("Vous n'ave
z pas appuyé sur la barre d'espace suivi d'Entrée,
réessayez.")
                            end
                        end
                    end

                    update_constraints(selected_tea
m, home, constraints)
                    update_constraints(away, select
ed_team, constraints)
                    push!(li_opponents, (home.club,
away.club))
                end

                println(file, li_opponents)
                println(file, "\n---\n")  # Ajoute
une ligne de séparation après chaque équipe
```

```
                push!(matches_list, li_opponents)
            end
            println(file, "\n\n")  # Ajoute un espa
ce supplémentaire après chaque pot pour une meilleu
re visibilité
        end
    end

    println("Résultats du tirage au sort enregistré
s dans le fichier 'tirage_au_sort.txt'")
    total_time = time() - start_time
    println("Temps total d'exécution de tirage_au_s
ort : $(round(total_time, digits=2)) secondes")
end
```

- Effectue un tirage complet :
  1. Mélange les équipes dans chaque pot.
  2. Détermine les matchs valides pour chaque équipe.
  3. Sélectionne des matchs au hasard.
  4. Met à jour les contraintes après chaque tirage.
  5. Enregistre les résultats dans un fichier texte.

SUM UP du code

- Ce code gère un **tirage au sort de matchs** en tenant compte des contraintes de pot, de nationalité, et de matchs joués.
- Il combine des outils avancés (Gurobi, JuMP) pour optimiser la répartition des matchs tout en respectant des règles complexes.

▼ Tirage Gurobi Elo

Tirage Turbo Elo est une version plus avancée, conçue pour exécuter plusieurs tirages en parallèle et analyser les forces ELO/UEFA cumulées, tandis que **"Tirage Gurobi Fast"** est une version plus simple pour un tirage unique.

L’utilisation de multithreading permet une meilleure complexité temporelle et donc exécuter plusieurs tâches simultanément.

> 💡 Le **multithreading** est une technique en programmation qui permet à un programme d'exécuter plusieurs tâches **simultanément**, en utilisant plusieurs threads d'exécution.
>
> Un **thread** est une unité d'exécution indépendante au sein d'un programme. Cela signifie qu'un programme peut exécuter plusieurs threads en même temps, chacun effectuant une tâche différente. C'est particulièrement utile pour accélérer les programmes qui doivent exécuter des calculs longs ou traiter des données volumineuses.
>
> Dans **"Tirage Turbo Elo"**, le programme utilise `Base.Threads` pour :
>
> - Diviser les simulations (`nb_draw`) en plusieurs threads.
> - Chaque thread exécute une partie des simulations, ce qui accélère le processus total si ton CPU a plusieurs cœurs.
>
> Exemple : Si on a 4 threads et 100 simulations (`nb_draw = 100`), chaque thread exécutera environ 25 simulations. Cela prendra moins de temps qu'une exécution séquentielle (une simulation à la fois).

1. Dans Tirage Gurob Elo :

```
function tirage_au_sort(nb_draw::Int, constraints::
Dict{String, Constraint}; sequential=false)
```

Différence :

- Le paramètre `nb_draw` indique combien de simulations (ou tirages) doivent être effectuées.
- Cela permet de répéter le tirage plusieurs fois pour analyser les résultats.

Dans Tirage Gurobi Fast :

```
function tirage_au_sort(constraints::Dict{String, C
onstraint}; sequential=false)
```

- **Différence** :
  - Il n'y a pas de `nb_draw`. Le programme effectue **un seul tirage**.
  - Pas de boucle pour exécuter plusieurs tirages.

---

2. Calculs des forces cumulées (ELO et UEFA)

Dans Tirage Gurobi Elo :

```
elo_opponents = zeros(Float64, 36, nb_draw)
uefa_opponents = zeros(Float64, 36, nb_draw)
```

- **Différence** :
  - Deux matrices sont créées pour enregistrer les forces cumulées des adversaires (ELO et UEFA) pour chaque équipe sur toutes les simulations.
    - **36** : Correspond au nombre total d'équipes.
    - `nb_draw` : Nombre de simulations à exécuter.

Dans chaque tirage :

```
elo_opponents[get_index_of_team(selected_team.clu
b), s] += away.elo + home.elo
uefa_opponents[get_index_of_team(selected_team.clu
b), s] += away.uefa + home.uefa
```

- **Explication** :
  - Les forces ELO et UEFA des équipes adverses (domicile et extérieur) sont additionnées et enregistrées pour chaque simulation.

Dans Tirage Gurobi Fast :

- Aucune gestion des forces ELO/UEFA. Le programme effectue le tirage sans analyser ou enregistrer les forces des adversaires.

---

4. Enregistrement des résultats

Dans Tirage GurobiElo :

```
open("1elo_strength_opponents.txt", "a") do file
    for i in 1:nb_draw
        row = join(elo_opponents[:, i], " ")
        write(file, row * "\n")
    end
end

open("1uefa_strength_opponents.txt", "a") do file
    for i in 1:nb_draw
        row = join(uefa_opponents[:, i], " ")
        write(file, row * "\n")
    end
end
```

- **Différence** :
  - Les forces ELO et UEFA cumulées des adversaires sont enregistrées dans deux fichiers distincts :
    - `1elo_strength_opponents.txt`
    - `1uefa_strength_opponents.txt`

Dans Tirage Gurobi Fast :

```
open("tirage_au_sort.txt", "w") do file
    ...
end
```

- **Différence** :
  - Seul le tirage unique est enregistré dans un fichier texte `tirage_au_sort.txt`, sans données supplémentaires comme les forces des adversaires.

▼ Optimisations possibles

1. Améliorer la gestion des contraintes dans Gurobi

- **Réduire les contraintes superflues :**
  - Analyser si certaines contraintes peuvent être simplifiées ou regroupées : par exemple, au lieu d'ajouter une contrainte pour chaque équipe et chaque match, regrouper les contraintes similaires.
- **Utiliser des indexers intelligents dans Gurobi :**
  - Les opérations comme `sum(...)` sur des ensembles peuvent être optimisées avec des indexers spécifiques pour éviter les itérations complètes. Par exemple restreindre les index uniquement aux paires nécessaires pour éviter d'évaluer des équipes non pertinentes.

2. Réduire le nombre de combinaisons testées

Les boucles imbriquées (pour les contraintes de nationalité), créent une grande complexité temporelle en O(n^4):

```
for (i, pot_i) in enumerate((teams.pot1, teams.pot
2, teams.pot3, teams.pot4))
    for (j, team_j) in enumerate(pot_i)
        for (k, pot_k) in enumerate((teams.pot1, te
ams.pot2, teams.pot3, teams.pot4))
            for (l, team_l) in enumerate(pot_k)
                if team_j.nationality == team_l.nat
ionality && ...
```

- **Créer des structures pré-calculées pour éviter les boucles imbriquées :**
  - Construire une table de correspondance entre équipes et nationalités **avant** le tirage. Filtrer directement les équipes incompatibles avant d'ajouter des contraintes. Par exemple regrouper les équipes par nationalité pour éviter de vérifier chaque paire d'équipes.

```
# Pré-calcul des équipes par nationalité
function group_teams_by_nationality(teams::TeamsConta
    nationality_groups = Dict{String, Vector{Int}}()
    for (i, pot) in enumerate((teams.pot1, teams.pot2
        for (j, team) in enumerate(pot)
            team_idx = (i - 1) * 9 + j
            push!(get!(nationality_groups, team.natio
        end
    end
    return nationality_groups
end

const teams_by_nationality = group_teams_by_nationali

# Contraintes de nationalité optimisées
for (nat, indices) in teams_by_nationality
    if length(indices) > 1
        for i in indices
            for j in indices
                if i != j
                    @constraint(model, sum(match_vars
                end
            end
        end
    end
end
```

3. Optimisation des contraintes dans `solve_problem`

La fonction `solve_problem` ajoute un grand nombre de contraintes pour vérifier les matchs possibles. Cela peut ralentir considérablement Gurobi si trop de variables et de contraintes sont introduites.

Optimisation possible :

- **Limiter le nombre de variables :**
    - Réduire les dimensions inutiles de `match_vars`. Par exemple, si certaines combinaisons sont interdites à l'avance, exclure ces

variables dès leur déclaration.

Exemple d'optimisation :

```
@variable(model, match_vars[i, j, t=1:T] for i i
n 1:36, j in 1:36 if i != j, Bin)
```

- **Ajouter des contraintes générales avant l'appel à Gurobi :**
  - Filtrer les matchs non valides avant d'introduire les variables correspondantes.
