2025 International Mathematical Modelling Challenge (IM^2C)

Developing an Annual Scheduling model for a Global Sporting League

Summary

Major sports leagues have historically been restricted to regional tournaments that prevent the best countries from across the globe from competing against each other on a regular basis. The scheduling process consistently presents a major logistical challenge that is a huge obstacle for international sports leagues to become reality. In order to make such an event happen, a suitable model for the selection of teams and scheduling of matches must be made.

In the context of our current world, more considerations than just the effectiveness of the tournament ought to be made when selecting teams and generating schedules. The model we have been tasked with producing provides alternate methods to select teams for the tournament, and considers the environmental and economic impact of hosting games in different locations. Football has then been selected as the sport of choice for our complete schedule for the first Global Sports League tournament which would happen in 2027. Our model consists of three parts:

- First is a quantitative analysis of all countries that compete in a particular sport, weighing between their global standings, internal fan support, potential for growth, and environmental harm using an **analytical hierarchy process** to generate weights, culminating in an index that rates each national team on their priority to be entered in the tournament.
- Secondly, we modelled for the 'best' tournament format, considering each team's travel time, number of matches in total and by each team, and the accuracy of the final standings by the end of each tournament. This was simulated using a **Monte Carlo method**, where we used our selection of national teams and assumed an ideal scenario where we could accurately measure the win probability of each team. In doing so, we were able to measure the relative effectiveness of each tournament format in regards to how fairly it would produce a winner, as well as minimising its overall environmental impact. As many of these are grouped systems, we provided a model to generate the best groups considering these scheduling metrics using a **k-means clustering algorithm** so that our model may be generalized for any set of countries in a tournament.
- Lastly, we modelled for the ideal scheduling of games. Using **simulated annealing,** we ordered each game within the initial round robin groups to optimise for athlete performance and morale, then assigned each game to a country for play (either 'home' or 'away'). We then allocated each group to a month based on the optimal climate for play. Afterwards, using a **quantitative analysis** of people's preferred viewing times, we selected the ideal time in the day for each match to be held.

When adapting these models to our chosen sport of football, we looked at existing tournament formats and found information on countries and their national teams from existing datasets to generate our schedule and format. Our model found that the best format for our tournament would be a **Hybrid Grouped Single Round Robin progressing to knockouts,** with round robin matches occurring on weekends between the months of February to June. Subsequent rounds will occur in the next four months. These models created the most optimal tournament format and schedule for the Global Sports League. After knowing each initial matchup, the order of rounds, and the time of day that each round would be scheduled, the tournament organisers would be able to assign each game on a weekend based on stadium availability. After sensitivity testing our model through different weights for our AHP, we found that our selected format was a reliable choice, and that our schedule would be similar across all circumstances.

Letter to the Decision Makers

Dear Decision Makers of the IMMC,

In light of your mission to create a Global Sporting League, our team has arrived at a recommendation for a general scheduling process, as well as a complete schedule for a Football Global Sporting League for 20 countries beginning in February 2027. Importantly, what separates a Global Sporting League from other regular international tournaments like the *Volleyball World Cup* is the unique experience of playing at different stadiums, and the ability for all teams to get the opportunity to play in front of their home crowd that comes when the tournament is not entirely held in a centralised location. This key distinguishing characteristic is the cause of the difficult scheduling logistics that our team has solved. As you have read from our summary page, we provide comprehensive models for all parts of the tournament development process.

Our team selection process is adaptable to all different sports, though we believe that it is preferable in the context of national sports teams, rather than independent clubs. However, given the necessary data about sports clubs and their locations, our model will generate an accurate schedule for any tournament. In the specific context of a football tournament, our data for team rankings came from the 'World Football Elo Ratings', for which equivalents exist across many different sports. Crucially, the criteria we have selected are particularly sustainable for the tournament, as results from previous championships can be used for the future selection of teams. Using these criteria, our 20 selected teams for the first Football GSL are: Brazil, Spain, France, Argentina, Uruguay, Colombia, England, Paraguay, Germany, Ecuador, Portugal, Italy, Morocco, Egypt, South Korea, Japan, Mexico, Costa Rica, New Zealand and Australia.

As mentioned in the summary, our modelling process resulted in the selection of a Hybrid Grouped Single Round Robin system which progressed to knockout rounds as the best tournament format for the Global Sporting League. Given the Global Sporting League's need to balance between tournament accuracy, fair transit times, environmental sustainability, and catering to international fans, we believe that this tournament format will be the most suitable for the GSL regardless of what sport is being played (some sports innately have more unpredictability in their outcomes, to which our model is able to adapt to find a more suitable tournament format). However, as different countries and regions excel at different sports, and hence may produce more teams that have entry priority, the number and nature of groupings for the initial stages of the tournament may skew the balance of factors and make the tournament less reliable. To solve this, we have also developed a model using a k-means clustering algorithm to help determine the most optimal groups for the tournament, creating a matchmaking schedule that is reliable and highly suitable for the GSL.

Our scheduling algorithm to determine the location of matches as well as the time of day the matches are to be held was considerate of many factors. The athlete's performance due to the effects of jet lag as well as the environmental effect of aviation were taken into account when determining the order and location of rounds and we have provided a system to find this for any group of countries in the tournament. After weighing up the viewership in different countries, we have also scheduled the optimal start time for each game in the day. Lastly, considering the season in each country, we also provide a month for the group round to take place in.

Attached below is a one-page visual summary of our final schedule and the benefits of our modelling process. We hope it will provide a strong starting point for your future development of the GSL.

Yours sincerely,
Team 2025006

TEAM2025006

GSL PROPOSED SCHEDULING MODEL

FEWER CARBON EMISSIONS COMPARED TO TRADITIONAL DOUBLE ROUND ROBIN FORMATS!!!

TOURNAMENT FORMAT

HYBRID GROUPED ROUND ROBIN FORMAT

- 92 GAMES
- REGION GROUPED POOLS
- BEST OF SEVEN TESTED FORMATS FOR GSL

TEAM SELECTION GUIDE

MODEL SELECTS PREFERRED NATIONAL SPORTS TEAMS BASED ON IMMC PREFERENCES:

- HIGH SPORTS PERFORMANCE
- STRONG NATIONAL RELEVANCE IN SPORT
- ENVIRONMENTAL CONSIDERATIONS OF COUNTRIES

ROUND ALLOCATIONS

GIVEN OUR SELECTED TEAMS, OUR MODEL WILL ALLOCATE MATCHUPS FOR EACH ROUND IN THE TOURNAMENT CONSIDERING:

- ATHLETE PERFORMANCE DEFICITS DUE TO JETLAG AND CONSTANT TRAVELS
- OVERALL TRAVEL DISTANCES AND ENVIRONMENTAL SUSTAINABILITY
- FAIR DISTRIBUTION OF HOME/AWAY GAMES FOR EACH TEAM

PROPOSED TIMETABLE FOR GROUP STAGE

Central American February	Asian Pacific March	South American April	Mediterranean May	Northern Europe June
v. 7pm GMT-6 v. 7pm GMT-5	v. 8pm GMT+10 v. 6pm GMT+9	v. 7pm GMT-3 v. 7pm GMT-3	v. 7pm GMT+2 v. 6pm GMT+1	v. 7pm GMT+0 v. 7pm GMT+2
v. 7pm GMT-5 v. 7pm GMT-6	v. 6pm GMT+9 v. 7pm GMT+9	v. 7pm GMT-3 v. 7pm GMT-3	v. 7pm GMT+2 v. 6pm GMT+1	v. 7pm GMT+2 v. 7pm GMT+2
v. 7pm GMT-5 v. 8pm GMT-6	v. 7pm GMT+12 v. 7pm GMT+9	v. 7pm GMT-3 v. 7pm GMT-3	v. 7pm GMT+2 v. 7pm GMT+1	v. 7pm GMT+2 v. 7pm GMT+2

Table of Contents

Introduction

Since ancient times (Crowther, 2007) humans have used sporting events to unify the people of a region, and showcase the pinnacle of human physicality. From the Ancient Olympics to modern sporting leagues, sports has consistently played a large role in the cultural and national identity of nations and communities (Council of Europe, n.d.). However, aside from the very few instances of the Olympics, FIFA World Cup and Volleyball Nations League among some others (many of which have long periods of inactivity between competitions eg. four years between FIFA World Cups), international sporting competitions are often restricted to regional and national tournaments such as the Champions League in Europe and the NBA in the United States. This is unfortunate as it prevents teams from all around the world from regularly competing against each other, restricting cultural exchange and hindering these leagues from truly defining the greats in the sport.

One of the largest obstacles to more major annual international tournaments is the massive logistical challenge that arises when teams across different time zones and seasons need to collectivise for regular matches throughout a sports season. Our task from The International Multi-Continental Matchmaking Committee has been to develop a scheduling model that can be adapted to different sports and different tournament sizes that will allow such sports leagues to be run across a wider array of sports.

1. Key Considerations in the problem

The IMMC has provided us with necessary considerations in the development of the schedule:

- Schedule should be fair and effective
- Schedule needs to adapt to changes in the league
- Fair distribution of games across teams
- Equitable travel time and distance
- Balanced competitiveness of matchups

Alongside these metrics, our team also wanted to consider:

- Minimise environmental impact of the schedule
- Consideration of changes in time zones for athletes performance
- Ensuring that athletes have sufficient rest between games
- Minimising the number of games played in total
- Minimise home/away advantage for each teams
- Account for fans watching in different timezones

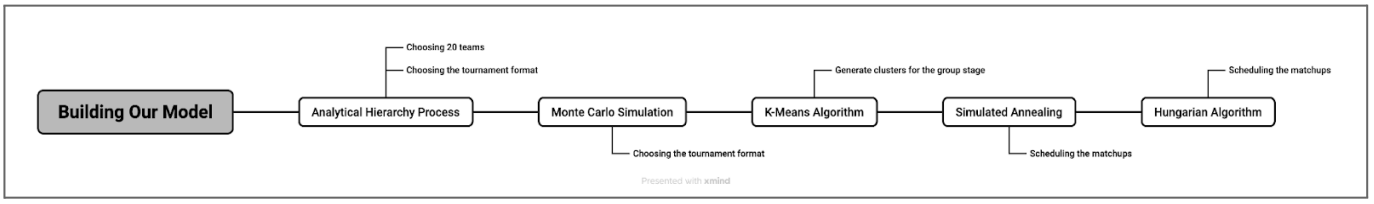
1.1 Definitions of key concepts

When considering and developing our model, we need to clearly define the considerations in order to have a clear direction and goal for our scheduling process:

- **What makes a scheduling system fair and balanced?**
    - As a competitive sporting league, the most important aspect of fairness is that the **team rankings at the end of the tournament align with their actual skill**. That means that the best teams should expect to be ranked very highly, and vice versa for the worst teams in the league.
    - **Equal travel times and number of games** is important for fairness. This is so that the cost of the tournament is relatively similar for all teams, and that each team gets roughly the same experience from the tournament in terms of play hours.
    - **Balanced competitive matchups** ensure that games are entertaining for fans and that teams are not destroyed in their games. However, this conflicts with the goal that the best teams are able to increase their ranking, as historically bad teams would not be able to escape their pools. As such, we would aim to ensure that teams do not **consecutively** face teams with large skill differences.

- **Aspects that affect athlete performance**
    - One of our primary scheduling goals is to ensure that the **difference in time zones** between consecutive games for a national team is small. Jet lag and its subsequent effect on a person's sleep has shown to have a meaningful impact on their cognitive performance, something necessary for good team cohesion (Lee & Galvez, 2012, 211-216) and team performance.
    - **Home game advantage** (Pinger, 2015) as a phenomenon has been shown to provide substantial benefits for the players on home turf. Our goal is to ensure that the number of home games each team has remains relatively consistent.
- **Factors that promote environmental sustainability**
    - Hosting games in countries with **green energy** heavily reduces the carbon cost of running these games. However, this may restrict some good countries who lack environmental regulations.
    - Reducing **travel times and the distance of flights** that teams need for the tournament will reduce one of the greatest causes of high carbon emissions in sports (Sharma & BOROS, 2023)
- **Fan engagement**
    - Ensuring that **all teams have at least one home game** will allow fans to engage directly with the sport. This is very important to improve the economic outcomes of the GSL, primarily for lower income countries.
    - In the instances where teams do not have home games, it is good to ensure that **across time zones, fans are able to watch at reasonable times** to support their teams.

Figure A: Modelling timeline



2. Selection of Sports and Teams

2.1 Defining and selecting a sport

2.1.1 Definition of sport

The problem defines the GSL as a sports league for **team sports**. Sports have often taken various definitions in the 21st century, with many considering games such as *chess* and computer games like *League of Legends* to be sports due to their cultural prevalence. However, in our selection of a sport for the GSL, our team only considered physical sports for a couple of reasons:

- Presentation of geographical diversity is most visible through physical sports on the field
- Multinational sports leagues are largely useful in their promotion of physical activity
- There is a more universal consensus on the cultural importance of physical sports.

2.1.2 Selection of sport

Our sport will be: **Association Football (Soccer)**. This has been selected because:

- It is by and large the largest and most popular sport in the world with around 3.5 billion fans. (Singh, 2025)
- It is a highly accessible sport being very popular in regions of developing countries. (Wikipedia Contributors, n.d.)

- This is important in demonstrating how the **GSL framework** might work on a global scale, making adaptation to other sports easier as development for more niche sports strengthens.

2.2 Team Selection

Due to the global nature of the GSL, the tournament will only allow **national teams** to compete as opposed to independent clubs for football. This is to ensure that individual countries are not overrepresented in the tournament, and allow for resulting rewards in the tournament to trickle down to a wider array of countries. By only allowing national teams, we also create greater national sentiment throughout the tournament.
The goal of our team selection process is to **promote the best teams in each regional block into the GSL tournament, while allowing adequate representation for countries with strong football cultures**. This is to ensure that the global prominence and rewards of the GSL tournament benefit countries that would be most susceptible to long term development, as well as maximising revenue to fund this expensive project.

2.2.1 Criteria

To fairly and adequately select sports teams for this tournament, our team created a weighted criteria index to rank national sports teams in selection priority.
Our team considered the following criteria when selecting teams:

1. Current **World Football Elo Ratings** (Eloratings, 2025) for each national football team.
   Justification: The Elo rating system provides an accurate measure of a team's skill. This measure is also particularly useful in predicting the expected outcome of each match up. Our expectation for future GSL tournaments is that each continent will operate a regional qualifying tournament for their guaranteed two slots, after which our weighted criteria index will select the remaining teams. The reason teams' skills are important is because having a tournament of very skilled teams will maximise revenue which is needed for an expensive global tournament like this.
2. Total **number and proportion of football fans** (Country Cassette, n.d.) in the country.
   Justification: This figure is a strong measure for the cultural effect that the sport has on each country looking to participate in the tournament. This is important to prioritise countries that would derive the most enjoyment and utility from this tournament. This is also needed to maximise the revenue generated from this tournament as a larger fanbase would drive ticket sales, merchandise sales, television viewership etc.
3. **Government sports funding** (Data 360, 1999-2023) as a percentage of GDP
   Justification: Countries with high amounts of sports funding are likely to have good infrastructure to host these games well, as well as having more youth development to promote the future growth of the program.
4. Proportion of **renewable energy production** (Ember and Energy Institute, 2024) of a country
   Justification: Countries involved in the tournament will likely expect to host games, as it brings an influx of capital from fans' spending as well as good reputation for the country. However, the operation of major stadiums and sports games leads to significant carbon emissions and environmental impact (Directorate-General for Climate Action, 2024) decreasing the sustainability of the GSL. Hence, we would look to countries with strong green energy programs, which would ensure that sports events hosted there have minimal impact on the environment.

> **Note:** For countries with **high nuclear energy proportions**, we have manually adjusted (World Nuclear Association, 2025) their green energy percentages as nuclear is not counted as 'renewable' but is clean and green.

> **Assumption 1**
> We assume that **green energy is used by all industries and stadiums uniformly for a single country**. This will help with the later generalisation of the model and is fair because much of the carbon cost of sports events are non-sports related anyways (eg. hotel electricity).

Each of these criteria are adaptable for each different sport that the GSL plays. For instance, to replace the **World Football Elo Rating** would be other credible rating systems to rank national teams for the particular sport. Proportion of sports fans for the specific sport would be found through their own datasets.

> **Assumption 2**
> We will assume that government sports funding is proportionally distributed across all major sports, so the **government funding index will remain the same when we generalise the model**. This is because data for individual sports funding doesn't exist, and is reasonable because we can assume that funding goes towards a country's most popular sports, which would correlate with their world ranking. Infrastructure like stadiums can also be used for a large variety of different sports.

2.2.2 Selection of 20

We evaluate each country using a weighted index of the aforementioned criteria, weighted as to align with the objectives of the **GSL**.

We form five separate uniformly distributed 0-1 scale indexes, two of them being separate indexes for the total number of fans and the proportion of people in a country who are fans of football. We feel that this is necessary in order to accurately scale our index for **fan support** to account for very small and large countries. For both the Skill Index and Population Proportion Index we normalize the data for elo rating and proportion of the country who are football fans using a **min-max normalization** to form our index. For the total number of football players and government sports spending, the obvious choice of min-max normalisation would be **inappropriate due to extreme outliers of very high population countries**, so instead we perform a **z-score normalisation**. We find a the corresponding z-score given by $z = \frac{x-\mu}{\sigma}$. We can then map this z-score to a cumulative frequency distribution to obtain the percentile the country belongs to, which ranges from 0-1. For our Green Index, we use the proportion of energy produced in a country which is green, which already has a nice uniform distribution of values between 0 and 1. To obtain suitable weights for our combined index we employ the **analytic hierarchy process (AHP)**. We look at every pair of criteria and assign relative importances comparing the two on a 10 point scale based on our research and the goals of the GSL. This forms a decision matrix (Saaty, 2001). (This decision matrix can be found in **Appendix A.1**)

Finally, we find the eigen-vector of this decision matrix with the greatest real eigen-value. This provides the most accurate weights based on the comparisons between pairs of criteria. The elements of this eigen-vector $(2.84,\ 0.78,\ 0.62,\ 0.33,\ 1)$ correspond to the weights chosen to be used to combine our five indexes into one final combined index: the Skill Index, Fan Proportion Index, Fan Count Index, Government Funding Index and Green Index.

$$I_{priority} = 2.84 \cdot I_{skill} + 0.78 \cdot I_{fan\ proportion} + 0.63 \cdot I_{fan\ count} + 0.333 \cdot I_{funding} + 1 \cdot I_{green}$$

This combined index scores each country on their **priority to be entered into the tournament**; we will call this a **priority index**.

In accordance with our goals for team selection, we select the two highest priority teams from each continent to fulfill geographical representation, and then the remaining 12 countries are selected based on their **priority index** (the data and priority index for all countries is found in **Appendix B**). The 20 chosen countries are: Brazil, Spain, France, Argentina, Uruguay, Colombia, England, Paraguay, Germany, Ecuador, Portugal, Italy, Morocco, Egypt, South Korea, Japan, Mexico, Costa Rica, New Zealand and Australia.

> **Note:** In this report, we define the continents as Asia (including the Middle East), Europe, Africa, North & Central America, South America and Oceania (Australian Pacific region).

3. Development of Initial Schedule

3.1 Variables table

Tournament Design	
T_g	Total number of games played in a season (lower = better)
T_a	Total air time of travels for all the teams within the season (lower = better)
$N_1, N_2, \cdots, N_{20}$	Number of games that each individual team play in the season
σ_N	Standard deviation of the dataset $(N_1, N_2, \cdots, N_{20})$ (lower = better)
$A_1, A_2, \cdots, A_{20}$	Total air time for each individual teams through the season
σ_A	Standard deviation of the dataset $(A_1, A_2, \cdots, A_{20})$ (lower = better)
P_A	$\frac{\sigma_A}{T_a}$ (lower = better)
L	Loss index: The size of difference between the expected ranking (following our criteria for fairness) and the outcomes caused by each format (lower = better)
I_F	Format Adequacy Index: how suitable, overall, a tournament format is for the selected twenty countries. (higher = better)
Mapping and Scheduling	
φ	Latitude
λ	Longitude

3.2 Choosing the GSL tournament format

To develop a suitable schedule, it is necessary to first decide on a tournament format that will be deemed as "best". We will then be building the schedule around this decided format. The conditions for which a format will be considered "best" will depend on several factors, namely: T_g, T_a, σ_N, P_A, L as outlined above in the variables table.

To see why these variables are relevant, notice that: T_g and T_a are measures of **environmental sustainability** as they correlate to travel times as well as travel distances. σ_N is the measure of the equitable distribution of games across teams, P_A is the measure of equitable travel times, and L relates to the **fairness** of our tournament format.

3.2.1 Weighing between factors

The Format Adequacy index I_F will be calculated from the five factors named above. Each factor will be normalised to a range between 0 to 1 with a **min-max normalisation** with the data obtained from our **simulation of the tournaments**. The weights of these factors will be once again determined via an **analytical hierarchy process (AHP)** (The decision matrix used here can be found in **Appendix A.2**). We found the most appropriate weights to be $(0.21, 0.34, 0.07, 0.10, 0.28)$, through finding the eigen-vector with the greatest real eigen-value.

3.2.2 Simulating tournament formats

For each tournament format, we will run a simulation of the tournament 1000 times. (The code for this simulation can be found in **Appendix C.1**). We will then gather data in each format for:

- Number of games played in each simulation
- Average combined airtime across all simulations
- Average ranking for each team across all simulations
- Average number of games played for each team
- Average standard deviation for the total flight times across each team
- Average **loss** (see below for loss calculations)

Several functions are used for this simulation, including: win rate of countries, distance between two countries, evaluation for the loss index (L), and simulating one match.

- **Winrate**: to calculate the win rate of one particular country over another, we deploy the **Elo** system (Wikipedia Contributors, n.d.):

$$P(A\ beats\ B) = \frac{1}{1+10^{(R_B-R_A)/400}}$$

Where R_A, R_B are the elo ratings of teams A and B respectively.

> **Assumption 3**
> When testing the validity of formats, we assume that a team's chance of winning is consistent **in keeping with their Elo rating**. This means we can not model for major "upsets", as our model is made to ensure fairness and not to predict sporadic changes in a team's performance.

- **Distance between two countries**: As geometrically the earth is a sphere, the shortest distance between two points on earth P and Q would be the minor arc of the **great circle** passing through the two points, that is the minor arc of the unique circle on the surface of earth, whose center is also the center of earth. Details of the calculation can be found in **Appendix E.1**.

Assumption 4
We assume plane flights do not have layovers and **are all direct flights**. Furthermore, we will assume that **all teams fly to their destinations even if they are on the same continent**. This does not change our results by large margins, as many national teams travel by private jet (Airmacs Aviation, n.d.).

- **Evaluation for the loss index**: For every simulated tournament, to evaluate how accurate the results generated were, we developed a **loss score** that is based on the number of **inversions** in the ranking: that is the number of ordered pairs of teams A and B where A ended up ranking higher than B but B is actually better than A. Additionally we also weigh the significance of each inversion to account for the following factors:
    - It is generally less surprising when the ranking of teams who are close in Elo are inverted, compared to large differences in Elo because we would expect teams of similar Elo to be of similar skill and occasionally beat one another. However, when there is a large difference between Elo and hence skill, it is extremely unlikely that the less skilled team beats the more skilled one.
    - It is generally less significant when there are disparities between expected ranking and actual ranking when a team isn't ranked highly in both. This is because fans and viewers are usually most engaged with teams at the top of the ladder, and also because there is generally less to lose for a team who is ranked lower than they should have been if their original ranking was low to start.

  To account for the following factors, for each inversion we calculate the amount of **bias** in favour of one team winning compared to random chance. The more likely a team was to rank higher than another team, the more surprising it is if they rank lower. This is given by
  $$Bias = |P(A\ beats\ B) - 50\%|$$
  We also calculate a weight representing how significant the team is due to their high ranking. The significance of a particular ranking is inversely proportional to the rank itself. We calculate a combined weight score using
  $$Weight = \frac{1}{Pr(Team\ 1)} + \frac{1}{Ar(Team\ 1)} + \frac{1}{Pr(Team\ 2)} + \frac{1}{Ar(Team\ 2)}$$
  for a single inversion where Pr and Ar stand for a team's predicted rank and actual rank. We form this weight score using the principle that highly ranked team have more "weight" compared to lowly ranked and that the difference between high ranks such a #1 and #2 are more significant than the differences between #8 and #9. The reciprocal relationship is given by Zipf's Law which relates to the linguistic and psychological prioritisation for highly ranked items (Zipf, 2012).
  The **loss score** of a single inversion is then given by taking the product of the **bias** and the **weight** of the inversion. The total **loss score** for each simulation is found by summing up the **loss score** of all inversions in the simulation.
- **Simulating one particular match**: To determine the rankings, we will follow the three points for a win standard (Keep the Score, 2025) , where we award the winning team with 3 points, and the losing team with 0 points. A random real value between 0 and 1 is generated, and if this value is less than the win rate of team A (given by the winrate function), we declare team A the winner, and conversely if it's larger, team B will be the winner.

Assumption 5
We assume that **two teams can not draw** for a single match. This is aligned with many football leagues, and is possible for most other sports through systems like **penalty shoot-outs**. We have done this to

> simplify the process of generalisation, as some sports like volleyball do not accept draws.

3.2.3 Calculating variables

- T_g: We will count the number of times that the "simulating one particular match" function runs.
- T_a: The airtime is calculated from the total distance travelled by all the teams, divided by the speed of the plane they travel in, which we take to be 900km/h (Epic Flight Academy, n.d.). For the sake of our calculations, we will have each team flying back home after their away games before going to their next away game. This will be consistent across all formats, so it will have no effect on the comparison between the models. However, we will optimise this later during scheduling.

> **Assumption 6**
> The carbon emissions of a plane due to takeoff/taxi/landing as well as transport within a country is **insignificant compared to the carbon emissions from flight time**.

- σ_N: This is calculated from the standard deviation formula on $(N_1, N_2, \cdots, N_{20})$.
- σ_A: This is calculated from the standard deviation formula on $(A_1, A_2, \cdots, A_{20})$. We will randomly generate the home and away games for each matchup at this stage to attribute the travel distances to one particular team out of the two. This will be balanced later on as we will ensure that teams have similar numbers of home and away games.
- P_A: As the total flight times across all teams increase, naturally the standard deviation for the individual teams will be greater. To make this a measurable metric across our models, we will use $P_A = \frac{\sigma_A}{T_a}$.
- L: This will be the output of the "evaluation of loss index" function.
- I_F: After normalising the factors $(T_g, T_a, \sigma_N, P_A, L)$ with min-max normalisation, we will take the weighted average as follows (using the weights determined from the AHP):

$$I_F = 0.21 \cdot T_{g\,(norm)} + 0.34 \cdot T_{a\,(norm)} + 0.07 \cdot \sigma_{N\,(norm)} + 0.10 \cdot P_{A\,(norm)} + 0.28 \cdot L_{(norm)}$$

When calculating I_F to compare the formats, each of these variables are **min-max normalised** compared to all other simulated formats. This makes the I_F value **relative to each simulation pool**. While not an absolute measure of a systems suitability, it is accurate in comparing different groups and determining the best outcome.

3.2.4 Tournament format comparisons

See **Appendix D** for more detailed explanations of each format:

- **#1 Full Single Round Robin**
  - Each team plays every other team once, and are ranked based on their accumulated points.

T_g	T_a	σ_N	σ_A	L	I_F
190 Games	3783 Hours	0.00	65.9 Hours	1.48	0.704

- **#2 Full Double Round Robin**
  - Each team plays every other team twice, and are ranked based on their accumulated points.

T_g	T_a	σ_N	σ_A	L	I_F
380 Games	7567 Hours	0.00	110.2 Hours	0.81	0.449

- **#3 Hybrid Grouped Single Round Robin + Knockouts (by region)**
  - Teams are allocated groups based on their region and play each other once. The best teams progress to a knockout stage.

T_g	T_a	σ_N	σ_A	L	I_F
109 Games	1524 Hours	2.49	45.2 Hours	5.58	0.634

- **#4 Hybrid Grouped Double Round Robin + Knockouts (by region)**
  - Teams are allocated groups based on their region and play each other twice. The best teams progress to a knockout stage.

T_g	T_a	σ_N	σ_A	L	I_F
211 Games	2920 Hours	5.39	82.0 Hours	4.41	0.527

- **#5 Hybrid Grouped Single Round Robin + Knockouts - (by region + size)**
  - Teams are allocated groups based on their region and play each other once. Larger regions are split up so all groups are of similar size. The best teams progress to a knockout stage.

T_g	T_a	σ_N	σ_A	L	I_F
92 Games	1464 Hours	0.932	51.0 Hours	3.87	0.721

**81% fewer flight hours compared to Double Round Robin claim on visual graphic*

- **#6 Hybrid Grouped Double Round Robin + Knockouts - (by region + size)**
  - Teams are allocated groups based on their region and play each other twice. Larger regions are split up so all groups are of similar size. The best teams progress to a knockout stage.

T_g	T_a	σ_N	σ_A	L	I_F
177 Games	2798 Hours	2.73	94.9 Hours	3.02	0.620

- **#7 Random Knockout**
  - A knockout bracket is generated and teams play until they are knocked out.

T_g	T_a	σ_N	σ_A	L	I_F
19 Games	354 Hours	0.52	19.3 Hours	8.03	0.617

Therefore we can see that, comparing the value of I_F across all different formats, we should choose **#5 Hybrid Grouped Single Round Robin + Knockouts - (by region + size)** as our tournament format. This has been highlighted above.

To address the question about the **number of games that each team must play to reasonably determine a winner**, we claim that the least possible number of games that a team needs to play to be crowned champion is 6, and the most number of games that they'll need to play is 15. To see this, a team can advance to the knockout stage by qualifying at the top of their regional group, which will require them to play 3 games. An additional 3 games is required for them to go through the knockout stage and win as the champion. The upper bound of 15 games is seen from the case where the 2nd and 3rd countries in the regional round goes into the qualifier round robin, advances to the knockouts by qualifying as the top 3 in that, and winning the knockouts. This gives 3 + 9 + 3 = 15 games. However on average, the number of games that each team will play in a season will be

$$Average\ number\ of\ games\ = \frac{2 \cdot Total\ number\ of\ games\ played}{Number\ of\ countries\ in\ the\ tournament}$$

Where the factor of 2 comes from the fact that there are two teams playing in each game. This gives us an **average of 9.2 games for each team** to play over the season, which corresponds to roughly 1 game per month in our 8-9 months season. The minimum number of games that a team might play in the tournament is 4, which happens if they rank at the top of their group stage but are knocked out of the first round of the quarterfinals.

3.2.5 Fleshing out the tournament format

As part of our hybrid grouped system, countries are each assigned a pool for the group stages first. These pools are based on region, so will be designed to minimise travel distance, and made to create groups of similar size, as that gave the lowest average **loss** in **3.2.4.** The way we determine these pools is through a **k-means clustering algorithm** (Piech & Ng, 2012). Suppose we need to group the 20 countries into k clusters. The steps are as follows (Code for this algorithm can be found in **Appendix C.2**):

1. Initialisation (this has been done with the Forgy Method (Wikipedia Contributors, n.d.)): randomly choose k countries out of the 20 as the starting centroids.
2. Cluster each of the countries to their nearest centroid, ensuring that each cluster does not exceed the maximum number of countries which is $\frac{20}{k}$.
3. Calculate the centroid of each of our defined clusters as $P = \left(\frac{\varphi_1 + \varphi_2 + \cdots + \varphi_N}{N}, \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_N}{N}\right)$, where P is the new centroid, (φ_i, λ_i) represents the latitude and longitude of the ith country in the cluster, and N is the total number of countries in that cluster.
4. Update the centroids list to our newly calculated centroids.
5. Repeat steps 2, 3 and 4 until the centroids from the previous iteration are exactly the same as the newly calculated centroids.

As an example, below is the algorithm process for grouping the 20 countries into 3 clusters of 7, 7, and 6 countries. The solid circles represent the centroid of the clusters of that corresponding colour.

Figure B.1:
3 countries are randomly selected as starting centroids. In this case, the selected countries are: Egypt, South Korea, and New Zealand. The algorithm started with the blue group at South Korea, clustering the 6 countries closest to it. Then moved to the red and finally the green.

Figure B.2:
Using the previously established groups, new centroids are generated (these are the coloured dots on the map). The algorithm then searches for the nearest countries near that centroid, and groups them into one cluster.

Figure B.3:
The same process as figure B.2 is repeated. Now, after the calculation of the new centroids, we find that the clusters do not change when we apply the algorithm again. Thus, we end the algorithm here.
(Earth is a sphere so groups can span the Pacific and still be optimal)

To find the optimal number of clusters, we will only consider the number of clusters from 2 to 8. This is because by our tournament design, the top ranked country within each group will advance to the quarterfinal knockouts. However if the number of clusters is larger than 8, then some countries at the top of their group will not make the knockouts. This gives us our upper bound of 8, and the lower bound of 2 follows from the fact that if the initial number of clusters was 1, then our tournament format will not be different from a full single round robin, which we have found to be worse than our format.

For each of these clusters, we have used the k-means algorithm to generate our pool, then simulated the tournament 1000 times with that pool. By using the same process as before, we have found the I_F values for each of these simulations.

No. of Clusters	2	3	4	5	6	7	8
I_F	0.436	0.445	0.637	**0.727**	0.650	0.452	0.501

As seen from the table, the number of clusters that gave the highest I_F value is 5. Thus for our designated schedule of 20 teams, we will have 5 clusters of 4 countries each, as seen below.

Figure C: Clusters of countries in our group round robin.

3.3 Creating the Schedule

3.3.1 Matchmaking and game locations

In a single round robin tournament with n team each team has to play a total of $n - 1$ separate games. We first create a schedule S where $S[r][t]$ represents the location that each team t has to play at in round r. The time that each match is scheduled to be played is to be determined later. The objective is to find a schedule which minimizes the total distance that each team will have to travel by plane; this is also going to be the most environmentally friendly schedule as a large portion of carbon emissions will be eliminated when there is less time spent flying teams across countries. However whatever schedule we come up with has to be **fair and balanced** for all the teams involved, to ensure that this is the case we apply the following **constraints** to our schedule S:

- We restrict the maximum number of consecutive home/away games to two, this is because teams get significant advantages from playing on home turf (Pinger, 2015) and it is important to distribute this advantage across the league season.
- Teams should not have three consecutive matches where there is a significant difference in time zone between two matches. We have deemed this number to be 5 hours as that is the time difference when people begin to feel major jet lag symptoms (Heid & Wu, 2023). It is suggested that east-to-west trips take 1.5 times longer to fully recover compared to west-to-east trips, we account for this in our model by scaling the time zone difference of all east-to-west trips by 1.5 (Brain Institute OHSU, n.d.).
- Teams should not have more than two consecutive matches where they are against teams which are much more skilled, teams should also not have more than three consecutive matches where they are against teams that are much less skilled. We have decided that a team is considered significantly more skilled than another team if one team's odds of winning are 3:1 or more.

$$0.75 < P(A\ beats\ B) = \frac{1}{1+10^{\frac{1}{400}(R_B - R_A)}}$$

$$\Rightarrow R_B - R_A < 400 \log_{10}\left(\frac{1}{0.75} - 1\right) = 190 \quad .$$

Hence we consider games to be heavily one sided if the difference in elo is greater than 190.

As the tournament scheduling is an **NP hard** problem (Bhattacharyya, 2016) it is computationally difficult to find the optimal tournament schedule when the number of teams becomes large as in the case of the ten team knockout qualifier round robin tournament. In order to accommodate for this limitation and the possibility of expansion and generalization, we have decided to use **simulated annealing**: a **metaheuristic** algorithm which is able to efficiently find approximate optimal solutions for optimization problem, theses approximate solutions are good enough as the difference from the theoretically most optimal solution is marginal.

The algorithm explores the total space of all possible schedules by making small adjustments to the schedule over time. Initially the algorithm is given more freedom to make any adjustment randomly in order to incentivise it to explore the total state space so that it finds a solution close to theoretically the most optimal solution. Then over time the algorithm is constrained to only making adjustments that are more optimal so it settles towards the global maximum.

We do not want to prevent our algorithm from exploring schedules that violate our constraints as those can lead to more optimal solutions when modified slightly; only those that violate many constraints and cannot be fixed without drastic changes to the schedule. The cost function we use for our **simulated annealing** algorithm is given by:

$$cost(S) = \sqrt{[total\ distance(S)]^2 + (\omega \cdot f[number\ of\ constraint\ violations(S)])^2}$$

(Anagnostopoulos et al., 2006)

Where ω is a parameter of the **simulated annealing** algorithm and $f(n) = 1 + \sqrt{v} \cdot \ln(\sqrt{v})$.

Our algorithm for determining an optimal schedule works roughly as such: (Code for the algorithm found in **Appendix C.3**)

1. Creating an initial schedule which is not optimal
2. Choose a random adjustment to make to the schedule
3. We decide whether or not to make the the adjustment (performed some number of times):
    - If the cost of the schedule after making the adjustment is less, we make the adjustment
    - Otherwise we choose to make the adjustment with probability $e^{(-\Delta cost\ /\ T)}$ where T is the current temperature, which is a variable that determines how much freedom our algorithm is to make suboptimal adjustments.
4. We decrease the temperature T by a factor of β: $T \leftarrow T \cdot \beta$ where β is another parameter of our algorithm
5. Repeat steps 2, 3, and 4 some number of times.
6. Choose a schedule with zero violations and with the least cost out of all the schedules we have explored

Our matchmaking pairs and rounds are found in the complete schedule in **3.4 Final Round Robin Schedule**.

3.3.2 Match times in year

There are seasonal factors that impact whether or not it is suitable to play games in a particular time of year. The summer and winter months are particularly bad because of extreme temperatures and weather. Because of this we will have each group play their own grouped rounds in an assigned month that most closely aligns with the seasonality of the sport and temperature of play. We assume that the ideal temperature for play is a daily average

of 20 degrees celsius. We measure how suitable a particular month by how small the difference $\left|T_{average} - T_{ideal}\right|$ between the average temperature of a region in a particular month (the average of all countries in a region) and the ideal temperature for play, which we call the **cost**. To find an optimal assignment of regions to months, we evaluate how suitable an assignment is by adding up the differences between average and ideal temperatures for each region and we use the assignment which minimizes this **sum**.

In order to minimize this sum we use the well known "**hungarian algorithm"** (CP Algorithms, 2023) which can be used to solve the problem of **optimal assignment (**Appendix C.5). We form a cost matrix M where $M_{i,j}$ represents the cost of team i playing in the jth month.

	FEB	MAR	APR	MAY	JUN
ASIAN PACIFIC	| 12.50 - 20 |	| 14.00 - 20 |	| 15.50 - 20 |	| 16.75 - 20 |	| 17.75 - 20 |
CENTRAL AMERICAN	| 17.25 - 20 |	|17.75 - 20 |	| 18.75 - 20 |	| 18.25 - 20 |	| 17.75 - 20 |
SOUTH AMERICAN	| 24.75 - 20 |	| 23.25 - 20 |	| 21.50 - 20 |	| 17.75 - 20 |	| 16.00 - 20 |
NORTHERN EUROPEAN	| 05.00 - 20 |	| 07.75 - 20 |	| 11.00 - 20 |	| 14.25 - 20 |	| 18.75 - 20 |
MEDITERRANEAN	| 12.50 - 20 |	| 14.50 - 20 |	|17.00 - 20 |	| 20.25 - 20 |	| 23.00 - 20 |

(Time and Date, 2025)

The optimal assignment of groups to months is as shown:

ASIAN PACIFIC	CENTRAL AMERICAN	SOUTH AMERICAN	NORTHERN EUROPEAN	MEDITERRANEAN
MARCH	FEBRUARY	APRIL	JUN	MAY

3.3.3 Time of day

We will be scheduling each match on the weekend. This will maximise global viewership, enhance fan attendance, and ensure logistical feasibility, capturing the largest possible audience and increasing revenue and affordability. To find the time of day for the match to be scheduled, we will be considering several factors.

- Local time in the home country
- Local time in the away country
- Fan engagement in the home and away countries (refer to **section 2.2.2**).
    - Fan proportion index
    - Fan population index
- Home country priority (see explanation below).

Viewership in the home country will be prioritised, due to the fact that it is likely that the home crowd support for a team will attract greater viewership within the country. For our match scheduling model, we will weigh the home country support with a weight of 2, over the away country. The model ensures that matches involving the home team are scheduled at peak times while also capturing away country audiences through slot allocations and streaming options.

In addition, the local times and fan engagement of the two countries will impact game viewership and thus the revenue which the league generates. This is essential to consider, as for the league to run sustainably in the future, there must be revenue from the games to go into development of the league in the future years. We will consider the local times in the two countries with the distribution of TV viewership throughout the time of the day (Krantz, 2018).

> **Assumption 7 & 8**
> The distribution of TV viewership at different times of the day across the world will be the same as the distribution in the United States.
> We also assume that people's preferred sports consumption aligns with the time they watch TV as it indicates when people have the most free time and are in the mood for media consumption.

We generate a combined fan index for each of the two countries playing with this formula:

$$Fan\ Index\ Home = \frac{0.78}{0.63 + 0.78} \cdot I_{fan\ proportion} + \frac{0.63}{0.63 + 0.78} \cdot I_{fan\ population}$$

We will calculate $Fan\ Index\ Away$ with the same formula. Note that the values of $I_{fan\ proportion}$, $I_{fan\ population}$, and $0.78,\ 0.63$ were taken and generated in **section 2.2.2**, and the indexes of fan proportion and fan population will align with data of that particular country (see **Appendix B**).

For a particular time slot of one hour, we will weigh the TV view percentage of the two countries at that time slot with their respective fan index, in addition to the country weight.

$$Total\ Viewership = \frac{2}{3} \cdot \frac{Fan\ Index\ Home}{Fan\ Index\ Home + Fan\ Index\ Away} \cdot V_H + \frac{1}{3} \cdot \frac{Fan\ Index\ Away}{Fan\ Index\ Away + Fan\ Index\ Home} \cdot V_A$$

Where we see that V_H is the percentage of the population that watches TV at this particular time slot in the home country, and V_A is the TV viewership percentage of the away country at this time. Weights of $\frac{2}{3}$, $\frac{1}{3}$ are due to the home country's priority. We will repeat this process 24 times for each of the 24 one hour periods during the day. This generates us the total viewership of the two countries across all the time slots. Assuming that a football event takes 3 hours (VOOR, n.d.), we will consider each of the 24 possible 3 hour time slots (characterised by their start times), and take the sum of the total viewership in these three hours. Thus, the 3 hour time window with the highest total viewership sum will be our designated time to schedule the event (the code to this can be found in **Appendix C.4**).

Figure D.1

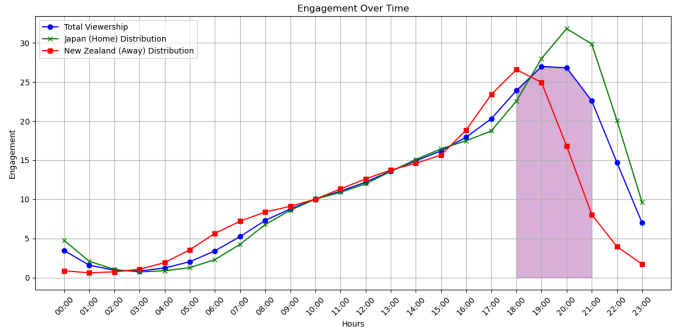
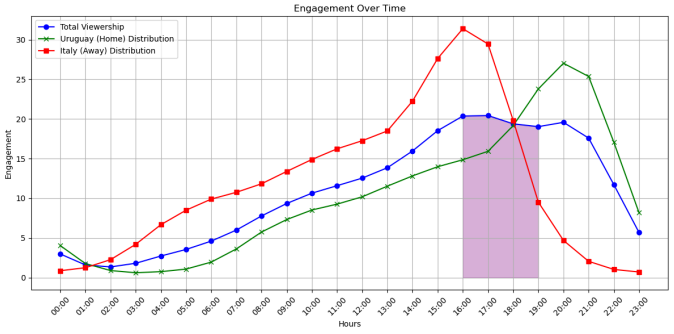


Figure D.2



The two figures above showcase the distribution of the TV viewership between the two countries. **The time on the x-axis is based on the time in the home country.**
Figure D.1 shows a particular match between Japan (home) and New Zealand (away). Note that these two countries are in the same group, and by our clustering design, the time zone difference between the two is not significant. Most of the TV viewership appears near the "prime time" of around 7pm, so our model chose 6-9pm Japan time as the scheduled time to balance viewership in Japan and New Zealand.

Figure D.2 is a hypothetical matchup between Uruguay (Home) and Italy (Away), which could happen in our qualifier round robin. The time zone difference between these two is more extreme, as shown by the two different peaks (red & green) in the figure. This results in a plateau of the total viewership from 4pm to 8pm as all plausible starting times, however our model chose 4pm Uruguayan time due to the higher fan index in Italy. (4pm in Uruguay is 9pm in Italy).

3.4 Final Schedule

3.4.1 Final round robin schedule

Group	Asian Pacific	Central American	South American	Northern Europe	Mediterranean
Round 1	South Korea v. **Australia** *8pm GMT+10* **Japan** v. New Zealand *6pm GMT+9*	**Mexico** v. Ecuador *7pm GMT-6* Costa Rica v. **Colombia** *7pm GMT-5*	**Brazil** v. Argentina *7pm GMT-3* Paraguay v. **Uruguay** *7pm GMT-3*	**England** v. Germany *7pm GMT+0* **France** v. Spain *7pm GMT+2*	**Italy** v. Portugal *7pm GMT+2* **Morocco** v. Egypt *6pm GMT+1*
Round 2	**South Korea** v. New Zealand *6pm GMT+9* **Japan** v. Australia *7pm GMT+9*	Mexico v. **Colombia** *7pm GMT-5* **Costa Rica** v. Ecuador *7pm GMT-6*	Brazil v. **Uruguay** *7pm GMT-3* **Argentina** v. Paraguay *7pm GMT-3*	England v. **France** *7pm GMT+2* **Germany** v. Spain *7pm GMT+2*	**Italy** v. Morocco *7pm GMT+2* **Portugal** v. Egypt *6pm GMT+1*
Round 3	**New Zealand** v. Australia *7pm GMT+12* **South Korea** v. Japan *7pm GMT+9*	**Ecuador** v. Colombia *7pm GMT-5* **Costa Rica** v. Mexico *8pm GMT-6*	Uruguay v. **Argentina** *7pm GMT-3* **Paraguay** v. Brazil *7pm GMT-3*	France v. **Germany** *7pm GMT+2* **Spain** v. England *7pm GMT+2*	Italy v. **Egypt** *7pm GMT+2* **Portugal** v. Morocco *7pm GMT+1*

**Bolded country is the home country where the game will be hosted, time is when the match is scheduled to begin in the home country.*

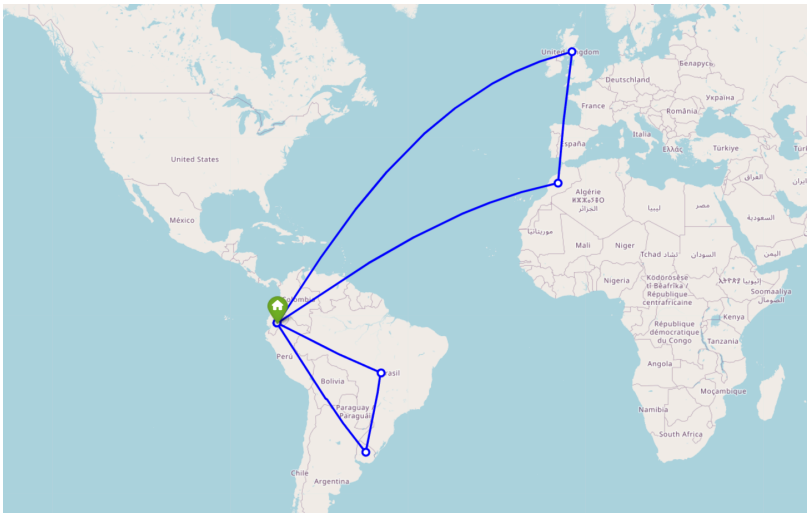
3.4.2 Simulating Future Scheduling

After the grouped round robin is completed, a qualifier round robin (2nd and 3rd from the groups) will take place, which we explore in the next paragraph. A placement round robin is also held for the 4th country from their respective groups and will follow a similar format as the round robins before. In terms of the knockout round for the top 8, we will be scheduling them in a centralised location after the round robins are finished. This is because in doing so, we raise excitement and engagement for the best 8 teams in the league, as fans across the world travel to the centralised location to support. To determine this location, the participating countries will follow a bidding system much like the Olympics.

Due to the nature of our format, the qualifier and placement round robins are unable to be scheduled beforehand. Instead, we provide a theoretical pool of 10 teams for the knockout qualifier round, and show how our scheduling algorithm is able to produce accurate results for all possible future rounds, regardless of the result. Using our **simulated annealing** algorithm, we have created a schedule shown in **Appendix G**. The matches of Ecuador and its team's flight path is shown below.

Ecuador [1911]

vs	Italy	[1914]	home
vs	Morocco	[1807]	away
vs	England	[2012]	away
vs	France	[2031]	home
vs	Mexico	[1817]	home
vs	Uruguay	[1922]	away
vs	Brazil	[1994]	away
vs	South Korea	[1745]	home
vs	Australia	[1736]	home



4. Expansion of schedule

4.1 Addition of teams

In accordance with our goals for team selection and the index we developed to fulfill them, the next four teams will be the four following Italy on our **priority index**, which are Turkey, Switzerland, Norway and the Netherlands.

> **Note:** Unfortunately, all four of the new teams are European, which seems to conflict with our goals for geographical representation. However, as the model weighs between many different factors that the GSL cares about, it is unlikely that the long term harm of this is prevalent. Countries like Iran and Senegal possess very strong national teams, but due to their lack of green energy, prevents them from being considered as strongly. Their deprioritization incentivises them to develop more environmentally friendly sporting facilities.

4.2 Effect of added teams on the model

4.2.1 Determining new groups

To determine the clusters for our 24 countries, we will run the same simulation as in **3.2.5**. The number of clusters against the I_F value is presented below:

No. of Clusters	2	3	4	5	6	7	8
I_F	0.391	0.453	0.564	0.652	**0.762**	0.640	0.398

From this data we can see that the most optimal number of clusters to choose in this case is 6 clusters. The groups in this case can be seen on this map:

Figure E: Clusters of countries in the hypothetical 24 countries format

4.2.2 Details of the model after the addition of teams

Our tournament format will be adjusted for the addition of the new teams. Specifically, after the top team in each group stage qualify for the knockouts, the 2nd and 3rd team in each group (this will be $2 \times 6 = 12$) teams will be in the qualifier round robin, and only the top two in this round robin will advance to the knockouts. See **Appendix D** for a more detailed explanation. The data for this model after 1000 simulations is presented in the following table, in alignment with the naming conventions of the variables in **section 3.1**.

T_g	T_a	σ_N	σ_A	L	I_F
124 Games	1901 Hours	0.80	57.8 Hours	5.36	0.712

To look at the number of games that each team plays, we get that each team will play an average of 10.3 games each. The minimum number of games that a team will need to play in order to be crowned champion stays the same at 6, while the upper bound has increased to 17, due to the need for the team to play 2 extra games in the knockout qualifier, as we have 12 teams in it instead of the 10 originally.

4.3 Sensitivity Analysis

We conduct sensitivity analysis to two parts in our model, where we used AHP to create the weights for our indexes. Using the **One at A Time (OAT)** method, we perturb each weight by a **fixed percentage of 10%**. We will then analyse the effects that these changes had on our priority index and format adequacy index.

4.3.1 Sensitivity Analysis on the selection of the 20

We will look at the effects of increasing/decreasing the weights used to determine our **selection of 20 teams** for our league and how that changes the teams that we choose. We increased/decreased each of the weights for the indexes I_{skill}, $I_{fan\ proportion}$, $I_{fan\ count}$, $I_{funding}$ and I_{green} one by one. We found that no changes were made to our selection of 20 teams seven times out of ten. The three times a team was swapped out, we found that Egypt was replaced with another African country, that being Senegal or Keyna. However, we can see from **Appendix B** that these countries have very similar index scores, making it not very surprising. There was only one instance where two teams were swapped out, that being Egypt with Senegal and Italy with Switzerland. Again, from **Appendix B** we can see that Switzerland was highly ranked to begin with and Italy was the lowest ranked out of the 12 teams chosen after picking 2 teams from each continent. This means that the model we used to select our initial 20 teams is very accurate and our selection of weights is appropriate.

4.3.2 Sensitivity Analysis for the decision of the tournament formats

We will look at the effects of increasing/decreasing the **weights** on each of the variables T_g, T_a, σ_N, P_A, L by 10% on the format adequacy index I_F. Through our calculations using the data from the 1000 simulations before, the full table of the I_F against the changes in weights for the respective variables can be found in **Appendix F.1**. It is worth noting that regardless of the change on the weights, the preferred format stays the same as #5 Hybrid Grouped Single Round Robin + Knockouts - (by region + size). However, there were particular changes to the weights that affected the dominance of the #5 format, and these changes had been extracted from the appendix and presented below:

	#1	#2	#3	#4	#5	#6	#7
T_a: Decreased	0.721	0.483	0.636	0.531	**0.725**	0.632	0.616
L: Decreased	0.708	0.449	0.649	0.533	**0.732**	0.629	0.645

First, note that the closest contender to format #5 is format #1, full single round robin. When T_a's weighting was decreased, there was a much closer gap between the two, and when the weighting for L was decreased, the dominance of the #5 format was much more apparent. This makes intuitive sense, as when each teams play every other team as in #1, the loss L is minimised, but this comes at a cost of greater T_a. So when the weight for L is decreased, the advantage of #1 in the L category is diminished, thus #5 becomes the clear winning format. Similar argument can be made for the change in T_a.

5. Generalisation of the model

5.1 Considerations for Generalisation

When generating a schedule for other team sports, special considerations might be taken for how a sport works in a tournament setting.

- How are points typically allocated in different sports?
    - In volleyball, points are often assigned based on the number of 'sets' a team wins. Our model still runs effectively, as the round robin system can be ranked based on the rules of the sport.
- How long do matches last?
    - For a majority of team sports, any typical game length (1-5 hours) will not affect our model, as we could change the **time scheduling model** to take the greatest total viewership over that corresponding time length.
    - However, some sports like Test Cricket can be played for days, which a generalised model should consider in terms of timetabling and required rest.
        - Our model can be adapted for situations like these, where weights can be adjusted to prioritise lower numbers of games, which would naturally allow for more time between matches and allow athletes to rest.
- Different ranking systems in other sports
    - When generating a schedule for other team sports, elo ratings can be used similarly to football to rank teams and predict match outcomes. These team sports have their own ranking systems which we assume can accurately represent a national team's competitive strength just like the

World Football Elo Ratings. For sports like basketball, Elo ratings can be adapted (eg. FIBA rankings for international teams), but sports without elo systems, like volleyball, may require alternative metrics such as world rankings or recent tournament performance.

- Viewership Patterns
    - Generalising our model to accommodate for other team sports would need data on specific viewership statistics for the particular team sport. We would implement the data into our model and predict peak viewership times and optimal patterns.
    - For different sports with different ideal playing times, our model is able to adapt when deciding the time in the day to host games.
- Scheduling
    - The model should adjust for sport-specific seasons, weather constraints, and cultural events, while optimising time slots for peak viewership based on the sport's audience patterns.

Conclusion and Discussion

Assessment of our model

Our model is capable of providing an accurate schedule for any given sport and event. However, it is not perfect. Where our model is flawed is its lack of generalisation for all sports as a scheduling model, as the weights in the AHP need to be manually adjusted for the specific goals of the tournament. However, given to any scheduling body with sufficient data, the model is able to effectively select teams and generate a schedule.

Strengths

- Our format selection algorithm is comprehensive and accurate, with generated weights able to reconcile a broad range of comparative importances. Here, it is easily adjustable to the specific needs of any tournament organiser.
- Our models for format selection and matchmaking are highly considerate of athletes and different goals of the tournament organisers and is able to maximise the fans' experience when watching our tournament from all around the world.
- Our models are highly resistant in our sensitivity tests, making them far more reliable.

Weaknesses

- The team selection model requires specific data for each national sports team and nation that is trying to enter the tournament. In practicality, if a central database was not found for the sport, our proposed team selection process could become time consuming.
- Our matchmaking algorithm to allocate team matchups on each round is not designed to create the perfect or ideal matchup, but runs until an approximate solution is found that is reasonably close to the ideal. This algorithm can be time consuming and require constant monitoring to be effective.

Conclusion

The task provided to us in this paper was to create a quantitative and generalised model for the scheduling and matchmaking of a global sports league. What we have done is create a set of adaptable algorithms that can be used to model for any sport league, selecting optimal teams, choosing ideal tournament formats, and timetabling them to appropriate times. The objectives outlined in the task were to ensure equitable numbers of games, travel distances, and matchups. We have completed that, as well as ensuring that a plethora of other necessary considerations were made to create a highly suitable scheduling model.

References

Crowther, N. B. (2007). *Sport in ancient times*. Bloomsbury Academic.

Council of Europe (n.d.). *Culture and Sport - Manual for Human Rights Education with Young people*. The Council of Europe. Retrieved April 18, 2025, from https://www.coe.int/en/web/compass/culture-and-sport

Lee, A., & Galvez, J. C. (2012). *Jet Lag in Athletes*. Sports Health, 4(3), 211-216. 10.1177/1941738112442340

Pinger, N. (2015, June 15). *Home Field Advantage: The Facts and the Fiction*. Chicago Booth. Retrieved April 18, 2025, from https://www.chicagobooth.edu/review/home-field-advantage-facts-and-fiction

Sharma, S., & BOROS, B. (2023, February). *What is the Carbon Footprint of Sport?* The Carbon Literacy Project. https://carbonliteracy.com/what-is-the-carbon-footprint-of-sport/

Singh, A. (2025, February 20). *The Most Popular Sports In The World - WorldAtlas*. World Atlas. https://www.worldatlas.com/articles/what-are-the-most-popular-sports-in-the-world.html

Wikipedia Contributors. (n.d.). *Sport in Africa*. Wikipedia. Retrieved April 18, 2025, from https://en.wikipedia.org/wiki/Sport_in_Africa

Wikipedia Contributors. (n.d.). *Sports in Asia*. Wikipedia. Retrieved April 18, 2025, from https://en.wikipedia.org/wiki/Sports_in_Asia

Eloratings. (2025, April 14). *World Football Elo Ratings*. World Football Elo Ratings. Retrieved April 18, 2025, from https://www.eloratings.net/

Country Cassette. (n.d.). *Football Fans by Country*. Country Cassette. Retrieved April 18, 2025, from https://countrycassette.com/rankings-sports-football-fans-by-country/

Data 360. (1999-2023). Government expenditure on recreational & sporting services. World Bank.

https://data360.worldbank.org/en/indicator/IMF_COFOG_GERS_GF0801?view=trend

Ember and Energy Institute. (2024, June 20). *Share of electricity production from renewables*. Our World in Data. Retrieved April 18, 2025, from https://ourworldindata.org/grapher/share-electricity-renewables

World Nuclear Association. (2025, April 1). *Nuclear Power in the World Today.* World Nuclear Association. Retrieved April 18, 2025, from https://world-nuclear.org/information-library/current-and-future-generation/nuclear-power-in-the-world-today

Directorate-General for Climate Action. (2024, July 26). *Sport – a key player in climate action?* European Climate Pact. https://climate-pact.europa.eu/articles-and-events/pact-articles/sport-key-player-climate-action-2024-07-26_en

Saaty, T.L. (2001). *Fundamentals of the Analytic Hierarchy Process.* In: Schmoldt, D.L., Kangas, J., Mendoza, G.A., Pesonen, M. (eds) *The Analytic Hierarchy Process in Natural Resource and Environmental Decision Making.* Managing Forest Ecosystems, vol 3. Springer, Dordrecht. https://doi.org/10.1007/978-94-015-9799-9_2

Wikipedia Contributors. (n.d.). *Elo rating system*. Wikipedia. Retrieved April 18, 2025, from https://en.wikipedia.org/wiki/Elo_rating_system

Keep the Score. (2025, February 28). *How does soccer scoring work? A beginner's guide*. KeepTheScore. https://keepthescore.com/blog/posts/soccer-scoring/

Epic Flight Academy. (n.d.). *How fast do commercial planes fly?* Epic Flight Academy. https://epicflightacademy.com/flight-school-faq/how-fast-do-commercial-planes-fly/

Airmacs Aviation. (n.d.). *Flying High in the Transfer Market: The Use of Private Jets in Football.*

https://airmacs.com/flying-high-in-the-transfer-market-the-use-of-private-jets-in-football

Piech, C., & Ng, A. (2012). *K Means*. CS221. thttps://stanford.edu/~cpiech/cs221/handouts/kmeans.html

Wikipedia Contributors. (n.d.). *k-means clustering.* Wikipedia. Retrieved April 19, 2025, from https://en.wikipedia.org/wiki/K-means_clustering

Zipf, G. K. (2012). *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Martino Fine Books

Heid, M., & Wu, C. (2023, May 19). *Jet Lag: Causes, Symptoms, Treatment, and Management.* Everyday Health. https://www.everydayhealth.com/sleep/jet-lag/guide/

Anagnostopoulos, A., Michel, L., Hentenryck, P. V., & Vergados, Y. (2006, April). *A Simulated Annealing Approach to the Traveling Tournament Problem*. Journal of Scheduling, 9. A Simulated Annealing Approach to the Traveling Tournament Problem

Krantz, R. (2018, September 28). *Television, capturing America's attention at prime time and beyond*. Bureau of Labor Statistics. https://www.bls.gov/opub/btn/volume-7/television-capturing-americas-attention.htm

VOOR. (n.d.). *How Long Does a Football Match Last*? https://voor.sport/en/mag/how-long-does-a-football-match-last

Brain Institute OHSU. (n.d.). *Jet Lag*. OHSU. Retrieved April 20, 2025, from https://www.ohsu.edu/brain-institute/jet-lag

Bhattacharyya, R. (2016, July). *Complexity of the Unconstrained Traveling Tournament Problem.* Operations Research Letters, 44. 10.1016/j.orl.2016.07.011

Time and Date. (2025, April 14). timeanddate.com. Retrieved April 21, 2025, from https://www.timeanddate.com/

CP Algorithms. (2023, December). *Hungarian algorithm for solving the assignment problem*. Retrieved April 21, 2025, from https://cp-algorithms.com/graph/hungarian-algorithm.html

Report on Use of AI

AI tools were used in two ways in the preparation of this paper.

1. Github(2025) *Copilot.* (April 20 2025 version, GPT-4o)
   'Autocomplete' feature was used in the coding process. This works in the same way as Google Doc's prediction feature, and the body of the code was written independently by ourselves.
2. Provided technical support for the formatting of the paper

OpenAI (2025) *ChatGPT.* (April 20 2025 version, ChatGPT-4o)
Query: <how to mimic tcolourbox in google docs>
Output: <Google Docs doesn't have a built-in equivalent to LaTeX's `tcolorbox`, but you *can* mimic its look and feel using tables or drawings. Here are a few easy ways to get that styled-box effect in your Google Doc:

Option 1: Use a 1x1 Table (the easiest way)

1. **Insert → Table → 1x1**
   This creates a single-cell table.

2. **Type your content** inside the cell.

3. **Customize it**:

   - Right-click the table → **Table properties**

   - Under **Table border**, adjust the **color**, **thickness**, and **style**

- Add **cell padding** for internal spacing
- Change the **background color** of the cell for a highlight effect

4. (Optional) Remove the border entirely or set it to a color that matches your theme.

🟦 *This gives you a clean bordered box with optional background fill—very similar to* `tcolorbox` *in LaTeX.*

Option 2: Use Google Drawings

1. **Insert → Drawing → + New**
2. Create a **shape** (like a rectangle with rounded corners)
3. Add **text inside** the shape
4. Customize:
   - Border thickness & color
   - Fill color
   - Font style & alignment
5. Click **Save and Close**

🖼️ *This lets you create more graphic-style boxes, great for callouts or color-coded blocks.*

Option 3: Use Paragraph Borders (less flexible)

1. Highlight a paragraph
2. Go to **Format → Paragraph styles → Borders and shading**
3. Add a border and background color

☐ *Good for inline notes or simple alert boxes, but not as customizable.*

Tips for a `tcolorbox`-like feel:

- Use **soft background colors** for gentle highlight boxes (`#f0f8ff`, `#ffffe0`, etc.)
- Use **bold or italic** to mimic `title` styling
- Add a small **icon** if you want to make it look like an alert or tip box (🔔 💡 !)

Want me to make a specific tcolorbox style from LaTeX in Google Docs style? Just let me know what you have in mind.>

Appendix

Appendix A: Analytical Hierarchy Process

1	3	5	7	9	2,4,6,8
Equal Importance	Slightly More Important	Significantly More Important	Strongly More Important	Extremely More Important	Values In Between

Appendix A.1: Analytical Hierarchy Process to generate weights

	SKILL	FAN PROP	FAN COUNT	FUNDING	RENEWABLE
SKILL	1	4	4	8	6
FAN PROP	1/4	1	2	5	1/3
FAN COUNT	1/4	1/2	1	5	1/3
FUNDING	1/8	1/5	1/5	1	1
RENEWABLE	1/6	3	3	1	1

Appendix A.2: Analytical Hierarchy Process for the Format Adequacy Index

	T_g	T_a	σ_N	P_A	L
T_g	1	1/2	5	2	1/2
T_a	2	1	5	5	1/2
σ_N	1/5	1/5	1	1	1/3
P_A	1/2	1/5	1	1	1/2
L	2	2	3	2	1

Appendix B: Table of countries and data sorted by priority for GSL Football

Country	Elo	Continent	Football fans (millions)	Football fans proportion	Government funding (% of GDP)	% Green energy production	Priority Index
Brazil	1994	South America	171.36	0.8	0.03	90	4.82
Spain	2150	Europe	37.84	0.8	0	63	4.68
France	2031	Europe	42.445	0.65	0.03	89	4.61
Argentina	2140	South America	38.93	0.85	0	35	4.43
Uruguay	1922	South America	3.15	0.9	0.115	91	4.40
Colombia	1953	South America	38.325	0.75	0.115	72	4.38
England	2012	Europe	46.97	0.7	0.115	54	4.31
Paraguay	1799	South America	5.55	0.75	0.115	100	4.15
Germany	1988	Europe	50.28	0.6	0.24	46	4.11
Ecuador	1911	South America	11.635	0.65	0.115	81	4.10
Portugal	1988	Europe	7.725	0.75	0.03	60	4.09
Italy	1914	Europe	45.3	0.75	0.21	36	4.00
Turkey	1837	Europe	63.9	0.75	0.115	42	3.99
Switzerland	1812	Europe	5.655	0.65	0.03	91	3.98
Norway	1828	Europe	2.97	0.55	0.11	99	3.97
Netherlands	1967	Europe	12.25	0.7	0.01	44	3.87
Sweden	1737	Europe	6.24	0.6	0.1	97	3.87
Denmark	1862	Europe	3.19	0.55	0.04	81	3.86
Mexico	1817	North America	95.119	0.73	0.115	29	3.84
Costa Rica	1653	North America	3.57	0.7	0.115	100	3.83
Austria	1837	Europe	4.95	0.55	0.28	78	3.81
Venezuela	1743	South America	17.34	0.6	0.115	78	3.78
Ukraine	1809	Europe	22.935	0.55	0.06	67	3.77
Belgium	1844	Europe	6.96	0.6	0	67	3.75

Peru	1725	South America	23.59	0.7	0.115	61	3.73
Croatia	1912	Europe	2	0.5	0.01	64	3.71
Chile	1722	South America	15.28	0.8	0.09	53	3.67
Panama	1724	North America	2.64	0.6	0.14	78	3.63
Russia	1790	Europe	72.95	0.5	0.115	36	3.62
South Korea	1745	Asia	34.371	0.67	0.115	40	3.60
Hungary	1716	Europe	5.184	0.54	0.97	70	3.59
Japan	1875	Asia	50.52	0.4	0.13	28	3.53
Slovakia	1685	Europe	2.75	0.5	0.115	83	3.52
New Zealand	1596	Oceania	3.06	0.6	0.115	87	3.50
Iceland	1531	Europe	0.2	0.5	0.04	100	3.39
Canada	1779	North America	5.805	0.15	0.32	82	3.37
Romania	1685	Europe	9.65	0.5	0.09	62	3.36
Slovenia	1744	Europe	0.84	0.4	0.04	67	3.34
United States	1723	North America	89.418	0.2698189499	0.115	41	3.34
Finland	1540	Europe	2.475	0.45	0.07	95	3.33
Georgia	1716	Europe	1.11	0.3	0.15	76	3.30
Morocco	1807	Africa	21.285	0.55	0.115	18	3.28
Egypt	1668	Africa	62.58	0.6	0.115	12	3.24
Poland	1705	Europe	24.57	0.65	0.05	21	3.24
Senegal	1759	Africa	11.34	0.6	0.115	26	3.24
Cameroon	1591	Africa	13.6	0.5	0.115	62	3.23
Bolivia	1652	South America	8.26	0.7	0.115	35	3.21
Kenya	1381	Africa	24.3	0.45	0.115	92	3.20
Honduras	1561	North America	5.83	0.55	0.115	63	3.17
Angola	1536	Africa	15.12	0.4	0.02	75	3.17
South Africa	1601	Africa	42	0.7	0.01	14	3.16
Luxembourg	1458	Europe	0.3	0.5	0.07	86	3.12
Uganda	1392	Africa	16.835	0.35	0.01	99	3.12

Ireland	1635	Europe	3	0.6	0.08	39	3.08
El Salvador	1391	North America	3.25	0.5	0.06	91	3.08
Guatemala	1495	North America	9.05	0.5	0.1	67	3.08
Ethiopia	1282	Africa	38.07	0.3	0.02	100	3.05
Zambia	1407	Africa	7.8	0.4	0.01	89	3.02
Nigeria	1561	Africa	82.44	0.4	0.115	22	2.99
Mozambique	1466	Africa	9.39	0.3	0.05	82	2.98
Vietnam	1357	Asia	51.15	0.5	0.115	50	2.95
Guinea	1453	Africa	6.03	0.45	0.115	66	2.92
Mali	1614	Africa	8.76	0.4	0.115	40	2.91
Ivory Coast	1593	Africa	5.72	0.55	0.115	31	2.91
Algeria	1711	Africa	22.35	0.5	0.115	1	2.89
Iran	1829	Asia	21.25	0.25	0.115	4	2.88
Bulgaria	1460	Europe	2.76	0.4	0.55	59	2.83
Australia	1736	Oceania	5.18	0.2	0.02	33	2.83
Ghana	1465	Africa	17.105	0.55	0.115	34	2.80
Indonesia	1339	Asia	165.48	0.5986975398	0.115	20	2.78
Jamaica	1560	North America	1.885	0.65	0.115	14	2.74
Jordan	1632	Asia	4.36	0.4	0.08	23	2.73
Zimbabwe	1397	Africa	5.215	0.35	0.115	67	2.73
Kyrgyzstan	1319	Asia	2.01	0.3	0.115	86	2.71
Suriname	1402	South America	0.33	0.55	0.115	48	2.71
Malawi	1282	Africa	5.35	0.25	0.115	94	2.70
Sudan	1396	Africa	13.47	0.3	0.115	62	2.69
United Arab Emirates	1540	Asia	4.95	0.5	0.115	24	2.69
Israel	1596	Asia	5.17	0.55	0.02	10	2.69
Lesotho	1240	Africa	0.55	0.25	0.115	100	2.65
Belarus	1491	Europe	4.23	0.45	0.2	32	2.64
Uzbekistan	1696	Asia	10.71	0.3	0	7	2.63

Eswatini	1226	Africa	0.36	0.3	0.115	93	2.60
China	1399	Asia	200	0.1403311816	0.115	35	2.60
Nicaragua	1322	North America	2.345	0.35	0.115	69	2.59
Latvia	1290	Europe	0.665	0.35	0.07	76	2.59
Tajikistan	1288	Asia	1.92	0.2	0.115	89	2.59
Lithuania	1291	Europe	0.98	0.35	0.07	75	2.59
Armenia	1387	Europe	1.085	0.35	0.04	58	2.58
Burkina Faso	1501	Africa	8.225	0.35	0.115	31	2.57
Saudi Arabia	1561	Asia	14.45	0.5	0.115	0	2.56
Tunisia	1613	Africa	5.355	0.45	0.115	3	2.56
Belize	1086	North America	0.22	0.55	0.115	81	2.48
Trinidad and Tobago	1431	North America	1.05	0.75	0.11	0	2.47
North Korea	1380	Asia	5.16	0.2	0.115	58	2.46
Malaysia	1258	Asia	22.035	0.65	0.115	19	2.43
Liberia	1265	Africa	1.56	0.3	0.115	67	2.42
Rwanda	1327	Africa	3.99	0.3	0.115	54	2.42
Burundi	1283	Africa	2.9	0.25	0.115	67	2.41
Haiti	1554	North America	3.36	0.3	0.115	13	2.40
Iraq	1538	Asia	14.42	0.35	0.115	2	2.40
Tanzania	1406	Africa	15.375	0.25	0.115	33	2.38
Cabo Verde	1506	Africa	0.21	0.35	0.12	16	2.37
Fiji	1229	Oceania	0.36	0.4	0.115	60	2.37
Qatar	1499	Asia	1.26	0.45	0.115	0	2.30
Estonia	1392	Europe	0.39	0.3	0.12	32	2.28
Thailand	1381	Asia	21.03	0.3	0.04	16	2.26
Bahrain	1498	Asia	0.68	0.4	0.115	0	2.25
Oman	1536	Asia	1.53	0.3	0.115	1	2.23
Congo	1226	Africa	35.63	0.35	0.115	23	2.21
Congo	1226	Africa	35.63	0.35	0.115	23	2.21

Nepal	942	Asia	7.275	0.25	0.25	100	2.19
Madagascar	1303	Africa	7.675	0.25	0.05	37	2.18
Benin	1413	Africa	4.84	0.4	0.115	2	2.15
Syria	1437	Asia	5.49	0.3	0.115	5	2.13
Kazakhstan	1425	Asia	4.75	0.25	0.1	12	2.12
Dominican Republic	1292	North America	4.86	0.45	0.05	14	2.10
Azerbaijan	1369	Asia	3.06	0.3	0.74	6	2.07
Philippines	1128	Asia	40.985	0.35	0	22	2.06
Afghanistan	1056	Asia	3.98	0.1	0.03	84	2.04
Lebanon	1342	Asia	2.38	0.35	0.115	9	2.03
India	1161	Asia	136	0.09456264775	0.115	24	2.02
Cyprus	1298	Asia	0.42	0.35	0.13	17	2.02
Malta	1256	Europe	0.225	0.45	0.09	13	2.00
Guyana	1269	South America	0.4	0.5	0.115	2	1.96
Niger	1324	Africa	7.26	0.3	0.115	6	1.96
Kuwait	1325	Asia	1.72	0.4	0.115	0	1.95
Botswana	1349	Africa	0.84	0.35	0.115	0	1.94
Moldova	1323	Europe	0.78	0.3	0.17	9	1.94
Comoros	1415	Africa	0.18	0.2	0.115	0	1.90
Cuba	1248	North America	4.52	0.4	0.115	5	1.89
Papua New Guinea	1104	Oceania	3.03	0.3	0.115	25	1.73
Myanmar	938	Asia	13.5	0.25	0.03	49	1.71
Mauritius	1044	Africa	0.52	0.4	0.115	19	1.65
Chad	1152	Africa	5.37	0.3	0.115	6	1.64
Singapore	1057	Asia	3.1	0.5	0.115	4	1.63
Cambodia	836	Asia	6.125	0.35	0.115	51	1.59
Turkmenistan	1214	Asia	1.28	0.2	0.115	0	1.56
Yemen	1107	Asia	4.575	0.15	0.115	20	1.55

Solomon Islands	1128	Oceania	0.245	0.35	0.09	0	1.55
Bhutan	635	Asia	0.16	0.2	0.115	100	1.54
Andorra	1066	Europe	0.04	0.4	0.115	0	1.49
South Sudan	1134	Africa	2.22	0.2	0.115	3	1.46
Barbados	950	North America	0.15	0.5	0.115	7	1.46
Vanuatu	1046	Oceania	0.075	0.25	0.115	14	1.45
Grenada	1073	North America	0.03	0.3	0.115	4	1.44
Pakistan	822	Asia	28.9	0.1196687371	0.115	42	1.43
Laos	684	Asia	1.975	0.25	0.115	73	1.42
Antigua and Barbuda	919	North America	0.045	0.45	0.115	6	1.34
Bangladesh	920	Asia	33.26	0.2	0.115	1	1.31
Seychelles	851	Africa	0.03	0.3	0.46	15	1.19
Somalia	915	Africa	3.54	0.2	0.115	11	1.16
Maldives	874	Asia	0.15	0.3	0.115	7	1.13
Samoa	730	Oceania	0.06	0.3	0.115	32	1.12
Liechtenstein	907	Europe	0.012	0.3	0.115	0	1.11
Djibouti	932	Africa	0.25	0.25	0.115	0	1.11
Bahamas	794	North America	0.18	0.45	0.115	0	1.06
Sri Lanka	732	Asia	2.7	0.12	0.115	37	1.02
San Marino	846	Europe	0.008	0.2666666667	0.115	0	0.97
Mongolia	729	Asia	1.02	0.3	0.07	11	0.92
Timor-Leste	600	Asia	0.35	0.25	0.115	0	0.53
Tonga	521	Oceania	0.025	0.25	0.115	13	0.52

Appendix C: Code used

Appendix C.1: Code used for simulation of tournament formats

Python

```python
# %% [markdown]
# # INITIALISATION OF COUNTRIES, ELO, LOCATION.

# %%
import random
import pandas as pd
from itertools import combinations

import matplotlib.pyplot as plt
import numpy as np


numtrials = 1000

# %%
countries = [
    "Brazil",
    "Spain",
    "France",
    "Argentina",
    "Uruguay",
    "Colombia",
    "United Kingdom",
    "Paraguay",
    "Germany",
    "Ecuador",
    "Portugal",
    "Italy",
    "Morocco",
    "Egypt",
    "South Korea",
    "Japan",
    "Mexico",
    "Costa Rica",
    "New Zealand",
    "Australia",
]

poolsunbalanced = {
    1: ['Mexico', 'Costa Rica', 'Ecuador', 'Colombia'],
    2: ['Brazil', 'Paraguay', 'Uruguay', "Argentina"],
    3: ['South Korea', 'Japan', 'New Zealand', 'Australia'],
     4: ['United Kingdom', 'Germany', 'France', 'Italy', 'Spain', 'Portugal', 'Morocco',
'Egypt']
}

pools = {
    1: ['Mexico', 'Costa Rica', 'Ecuador', 'Colombia'],
    2: ['Brazil', 'Paraguay', 'Uruguay', "Argentina"],
    3: ['South Korea', 'Japan', 'New Zealand', 'Australia'],
    4: ['United Kingdom', 'Germany', 'France', 'Italy'],
    5: ['Spain', 'Portugal', 'Morocco', 'Egypt']
}


elo_ratings = [
    1994,
    2150,
```

```
    2031,
    2140,
    1922,
    1953,
    2012,
    1799,
    1988,
    1911,
    1988,
    1914,
    1807,
    1668,
    1745,
    1875,
    1817,
    1653,
    1596,
    1736,
]

def get_elo(name):
    return countriesratings[name]

countriesratings = {country: elo_ratings[countries.index(country)] for country in
countries}
countries_ranked = sorted(countries, key=get_elo, reverse=True)

# %%
import folium
from IPython.display import display

locations = [
    (-14.2350, -51.9253),  # Brazil
    (40.4637, -3.7492),    # Spain
    (46.6034, 1.8883),     # France
    (-38.4161, -63.6167),  # Argentina
    (-32.5228, -55.7659),  # Uruguay
    (4.5709, -74.2973),    # Colombia
    (55.3781, -3.4360),    # United Kingdom
    (-23.4420, -58.4438),  # Paraguay
    (51.1657, 10.4515),    # Germany
    (-1.8312, -78.1834),   # Ecuador
    (39.3999, -8.2245),    # Portugal
    (41.8719, 12.5674),    # Italy
    (31.7915, -7.0926),    # Morocco
    (26.8206, 30.8025),    # Egypt
    (35.9078, 127.7669),   # South Korea
    (36.2048, 138.2529),   # Japan
    (23.6345, -102.5528),  # Mexico
    (9.7489, -83.7534),    # Costa Rica
    (-40.9006, 174.8860),  # New Zealand
    (-25.2744, 133.7751),   # Australia
]

locationdict = {country: locations[countries.index(country)] for country in countries}

games_played_country = {country: 0 for country in countries}
airtime_by_country = {country: 0 for country in countries}
```

```
# initial plotting of cuontries
m = folium.Map(location=[20, 20], zoom_start=2)
for name, rating, location in zip(countries, countriesratings.values(), locations):
    folium.Marker(
        location=location,
        icon=folium.Icon(color="red"),
        tooltip=f"{name} [{rating}]",
    ).add_to(m)
display(m)

# %% [markdown]
# # CALCULATING DISTANCE

# %%
from math import radians, sin, cos, sqrt, atan2

# DISTANCE CALCULATIONS
def haversine(pos1, pos2):
    lat1, lon1 = pos1
    lat2, lon2 = pos2
    # Convert degrees to radians
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    # Haversine formula
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    R = 6371.0 # Radius of Earth in kilometers
    return R * c # Distance in kilometers


def estimate_flight_time(pos1, pos2, speed_kmh=900):
    distance = haversine(pos1, pos2)
    time_hours = distance / speed_kmh
    return time_hours

# %% [markdown]
# # ELO SYSTEM + EVALUATION

# %%
# Elo system
def winrate(country1, country2):
    return (1/(1 + 10**((countriesratings[country2]-countriesratings[country1])/400)))

# Evaluation
def evaluate_ranking(ranking, true_ranking):
    """Determines how good a ranking is."""

    loss = 0

    # For each pair of countries in the ranking
    for i, A in enumerate(countries):
        for j, B in enumerate(countries):
            if i >= j:
                continue
            rank_A = ranking.index(A) + 1
```

```
            rank_B = ranking.index(B) + 1
            true_A = true_ranking.index(A) + 1
            true_B = true_ranking.index(B) + 1
            if (rank_A < rank_B) == (true_A < true_B):
                continue

            W = (
                1 / true_A
                + 1 / true_B
                + 1 / rank_A
                + 1 / rank_B
            )
            loss += W * (abs(0.5 - winrate(A, B)))
    return loss

# %% [markdown]
# # MATCH SIMULATION

# %%
def simulate_match(country1, country2):

    global totaltime
    totaltime += 2*estimate_flight_time(locationdict[country1], locationdict[country2])

    global total_games_played
    total_games_played += 1

    games_played_country[country1] += 1
    games_played_country[country2] += 1

    result_country = random.choice([country1, country2])
                                        airtime_by_country[result_country]          +=
2*estimate_flight_time(locationdict[country1], locationdict[country2])

    percent1 = winrate(country1, country2)

    if random.random() < percent1:
        result = 'win'
    else:
        result = 'loss'

    if result == 'win':
        return country1, 3, country2, 0
    elif result == 'loss':
        return country1, 0, country2, 3
    else:
        return country1, 1, country2, 1

# %%
def init_standings(teams):
    return {team: {'wins': 0, 'losses': 0, 'draws': 0, 'points': 0} for team in teams}

def sort_standings(standings):
    df = pd.DataFrame(standings).T
    df = df.sort_values(by=['points', 'wins'], ascending=False)
    return df

# %% [markdown]
```

```
# # TOURNAMENT SIMULATIONS

# %%
def round_robin(teams):
    standings = init_standings(teams)
    for team1, team2 in combinations(teams, 2):
        t1, p1, t2, p2 = simulate_match(team1, team2)
        standings[t1]['points'] += p1
        standings[t2]['points'] += p2
        if p1 == 3:
            standings[t1]['wins'] += 1
            standings[t2]['losses'] += 1
        elif p2 == 3:
            standings[t2]['wins'] += 1
            standings[t1]['losses'] += 1
        else:
            standings[t1]['draws'] += 1
            standings[t2]['draws'] += 1
    return standings

# %%
def simulate_knockout(teams):
    seeded = []

    for country in teams:
        seeded.append((country, countriesratings[country]))

    seeded.sort(key=lambda x: x[1], reverse=True)
       quarter_finalists  =  [seeded[0][0],  seeded[7][0],  seeded[3][0],  seeded[4][0],
seeded[1][0], seeded[6][0], seeded[2][0], seeded[5][0]]
    semi_finalists = []
    bottom4 = []
    thirdfourth = []


    # Quarterfinals
    for i in range(0, 8, 2):
        team1, team2 = quarter_finalists[i], quarter_finalists[i+1]
           winner = simulate_match(team1, team2)[0 if random.random() < winrate(team1,
team2) else 2]
        semi_finalists.append(winner)
        if winner != team1:
            bottom4.append(team1)
        else:
            bottom4.append(team2)

    # Sort bottom4 based on the order they appear in the list seeded
    bottom4.sort(key=lambda x: next(i for i, v in enumerate(seeded) if v[0] == x))

    # Semifinals
    finalists = []
    for i in range(0, 4, 2):
        team1, team2 = semi_finalists[i], semi_finalists[i+1]
           winner = simulate_match(team1, team2)[0 if random.random() < winrate(team1,
team2) else 2]
        finalists.append(winner)
        if winner != team1:
            thirdfourth.append(team1)
```

```
        else:
            thirdfourth.append(team2)
    thirdfourth.sort(key=lambda x: next(i for i, v in enumerate(seeded) if v[0] == x))


    # Final
      final_winner = simulate_match(finalists[0], finalists[1])[0 if random.random() <
winrate(team1, team2) else 2]
    if final_winner != finalists[0]:
        second = finalists[0]
    else:
        second = finalists[1]

    return {
        bottom4[0]: 5,
        bottom4[1]: 6,
        bottom4[2]: 7,
        bottom4[3]: 8,
        thirdfourth[0]: 3,
        thirdfourth[1]: 4,
        second: 2,
        final_winner: 1
    }

# %% [markdown]
# # PLOTTING

# %%
# ---------------------------- PLOTTING ---------------------------------
# Calculate rankings based on countriesratings
def plotgraph(avgstandings, title):
                ratings_ranking    =    {country:    rank    for    rank,    country    in
enumerate(sorted(countriesratings, key=countriesratings.get, reverse=True), start=1)}


    # Prepare data for the bar graph
    avg_standings_values = [avgstandings[country] for country in countries]

    # Perform max-min normalization for ratings
    min_avg_standing = min(avg_standings_values)
    max_avg_standing = max(avg_standings_values)

    normalized_ratings = {
              country:  max_avg_standing  -  ((max_avg_standing  -  min_avg_standing)  *
(countriesratings[country] - min(countriesratings.values())) /
                                                    (max(countriesratings.values())  -
min(countriesratings.values())))
        for country in countries
    }

    ratings_ranking_values = [normalized_ratings[country] for country in countries]



    x = np.arange(len(countries))
    width = 0.35

    fig, ax = plt.subplots(figsize=(12, 6))
```

```
    rects1 = ax.bar(x - width/2, avg_standings_values, width, label='Average Standing')
        rects2 = ax.bar(x + width/2, ratings_ranking_values, width, label='Ratings
Ranking')

    # Text
    ax.set_xlabel('Countries')
    ax.set_ylabel('Rankings')
    ax.set_title(title)
    ax.set_xticks(x)
    ax.set_xticklabels(countries, rotation=45, ha='right')
    ax.legend()
    fig.tight_layout()

    ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))
    plt.show()

# %% [markdown]
# # SPECIFIC MODELS

# %% [markdown]
# ## Full Single Round Robin

# %%
import statistics
avgstandings = {country: 0 for country in countries}

fits = {}

loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for i in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    standings = round_robin(countries)
     sorted_countries = sorted(standings.keys(), key=lambda x: standings[x]['points'],
reverse=True)

    for j in range(len(sorted_countries)):
        avgstandings[sorted_countries[j]] += (j+1)

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)
    total_std += statistics.stdev(airtime_by_country.values())

for country, ranking in avgstandings.items():
    avgstandings[country] = ranking/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Full Single Round Robin')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
```

```
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")

std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['fullsinglerobin']     =     {'games':     total_games_played/numtrials,     'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}

# %% [markdown]
# ## Full Double Round Robin

# %%
avgstandings = {country: 0 for country in countries}

loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for i in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    standings = round_robin(countries)
    standings2 = round_robin(countries)

    for country in standings:
        standings[country]['points'] += standings2[country]['points']
        standings[country]['wins'] += standings2[country]['wins']
        standings[country]['losses'] += standings2[country]['losses']
        standings[country]['draws'] += standings2[country]['draws']


     sorted_countries = sorted(standings.keys(), key=lambda x: standings[x]['points'],
reverse=True)

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for j in range(len(sorted_countries)):
        avgstandings[sorted_countries[j]] += (j+1)

    total_std += statistics.stdev(airtime_by_country.values())


for country, ranking in avgstandings.items():
    avgstandings[country] = ranking/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Full Double Round Robin')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
```

```
print(f"Average Loss: {loss_total/numtrials}")

std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['fulldoublerobin']     =     {'games':     total_games_played/numtrials,     'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}

# %% [markdown]
# ## Single Round Robin: Unbalanced Groups

# %%
avgstandings = {country: 0 for country in countries}

loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for _ in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    pool_results = {}
    direct_to_quarters = []
    qualifier_candidates = []
    bottom8 = []
    ninthtwelve = []

    # ---------- Phase 1: Group Roundrobin ----------
    for pool_name, teams in poolsunbalanced.items():

        standings = round_robin(teams)
        df = sort_standings(standings)
        pool_results[pool_name] = df
        direct_to_quarters.append(df.index[0])            # Top 1 to quarters
        qualifier_candidates.extend(df.index[1:3])

        if len(teams) == 4:
            bottom8.append(df.index[3])
        else:
            bottom8.extend(df.index[3:8])

    # ---------- Phase 2: Qualifiers: Next 8 teams run in qualifiers ----------
    qualifier_results = round_robin(qualifier_candidates)
    qualifier_df = sort_standings(qualifier_results)
    qualifier_top4 = list(qualifier_df.index[:4])
    ninthtwelve.extend(qualifier_df.index[4:])

    # ---------- Phase 3: Knockout ----------
    knockout_teams = direct_to_quarters + qualifier_top4
    knockout_results = simulate_knockout(knockout_teams)

    # ---------- Phase 4: Placement Round ----------
    placement_results = round_robin(bottom8)
    placement_df = sort_standings(placement_results)
```

```
            sorted_placement = sorted(placement_results.keys(), key=lambda x:
placement_results[x]['points'], reverse=True)

    standings = init_standings(countries)

    rank = 13
    for country in sorted_placement:
        standings[country] = rank
        rank += 1

    sorted_knockout = sorted(knockout_results.items(), key=lambda x: x[1])
    for country, position in sorted_knockout:
        standings[country] = position

    rank = 9
    for country in ninthtwelve:
        standings[country] = rank
        rank += 1

    sorted_countries = sorted(standings.keys(), key=lambda x: standings[x])

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for country, position in standings.items():
        avgstandings[country] += position

    total_std += statistics.stdev(airtime_by_country.values())

for country, position in avgstandings.items():
    avgstandings[country] = position/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Single Round Robin with Unbalanced Groups')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")
std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['singlerobinunbalanced'] = {'games': total_games_played/numtrials, 'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}


# %% [markdown]
# ## Double Round Robin: Unbalanced Groups

# %%
avgstandings = {country: 0 for country in countries}

loss_total = 0
totaltime = 0
```

```
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for _ in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    pool_results = {}
    direct_to_quarters = []
    qualifier_candidates = []
    bottom8 = []
    ninthtwelve = []

    # ---------- Phase 1: Group Roundrobin ----------
    for pool_name, teams in poolsunbalanced.items():

        standings = round_robin(teams)
        standings2 = round_robin(teams)

        for country in standings:
            standings[country]['points'] += standings2[country]['points']
            standings[country]['wins'] += standings2[country]['wins']
            standings[country]['losses'] += standings2[country]['losses']
            standings[country]['draws'] += standings2[country]['draws']

        df = sort_standings(standings)
        pool_results[pool_name] = df
        direct_to_quarters.append(df.index[0])      # Top 1 to quarters
        qualifier_candidates.extend(df.index[1:3])

        if len(teams) == 4:
            bottom8.append(df.index[3])
        else:
            bottom8.extend(df.index[3:8])


    # ---------- Phase 2: Qualifiers: Next 8 teams run in qualifiers ----------
    qualifier_results = round_robin(qualifier_candidates)
    qualifer2 = round_robin(qualifier_candidates)
    for country in qualifier_results:
        qualifier_results[country]['points'] += qualifer2[country]['points']
        qualifier_results[country]['wins'] += qualifer2[country]['wins']
        qualifier_results[country]['losses'] += qualifer2[country]['losses']
        qualifier_results[country]['draws'] += qualifer2[country]['draws']
    qualifier_df = sort_standings(qualifier_results)
    qualifier_top4 = list(qualifier_df.index[:4])
    ninthtwelve.extend(qualifier_df.index[4:])

    # ---------- Phase 3: Knockout ----------
    knockout_teams = direct_to_quarters + qualifier_top4
    knockout_results = simulate_knockout(knockout_teams)

    # ---------- Phase 4: Placement Round ----------
    placement_results = round_robin(bottom8)
    placement2 = round_robin(bottom8)
    for country in placement_results:
        placement_results[country]['points'] += placement2[country]['points']
        placement_results[country]['wins'] += placement2[country]['wins']
        placement_results[country]['losses'] += placement2[country]['losses']
```

```
        placement_results[country]['draws'] += placement2[country]['draws']
    placement_df = sort_standings(placement_results)


            sorted_placement = sorted(placement_results.keys(), key=lambda x:
placement_results[x]['points'], reverse=True)
    standings = init_standings(countries)

    rank = 13
    for country in sorted_placement:
        standings[country] = rank
        rank += 1

    sorted_knockout = sorted(knockout_results.items(), key=lambda x: x[1])
    for country, position in sorted_knockout:
        standings[country] = position

    rank = 9
    for country in ninthtwelve:
        standings[country] = rank
        rank += 1

    sorted_countries = sorted(standings.keys(), key=lambda x: standings[x])

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for country, position in standings.items():
        avgstandings[country] += position

    total_std += statistics.stdev(airtime_by_country.values())

for country, position in avgstandings.items():
    avgstandings[country] = position/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Double Round Robin with Unbalanced Groups')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")
std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['doublerobinunbalanced'] = {'games': total_games_played/numtrials, 'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}

# %% [markdown]
# ## Single Round Robin, 5 groups of 4

# %%
avgstandings = {country: 0 for country in countries}
```

```
loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for _ in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    pool_results = {}
    direct_to_quarters = []
    qualifier_candidates = []
    bottom4 = []
    ninth16 = []

    # ---------- Phase 1: Group Roundrobin ----------
    for pool_name, teams in pools.items():

        standings = round_robin(teams)
        df = sort_standings(standings)
        pool_results[pool_name] = df
        direct_to_quarters.append(df.index[0])      # Top 1 to quarters
        qualifier_candidates.extend(df.index[1:3])

        bottom4.append(df.index[3])


    # ---------- Phase 2: Qualifiers: Next 10 teams run in qualifiers ----------
    qualifier_results = round_robin(qualifier_candidates)
    qualifier_df = sort_standings(qualifier_results)
    qualifier_top4 = list(qualifier_df.index[:3])
    ninth16.extend(qualifier_df.index[3:])

    # ---------- Phase 3: Knockout ----------
    knockout_teams = direct_to_quarters + qualifier_top4
    knockout_results = simulate_knockout(knockout_teams)

    # ---------- Phase 4: Placement Round ----------
    placement_results = round_robin(bottom4)
    placement_df = sort_standings(placement_results)


            sorted_placement    =    sorted(placement_results.keys(),    key=lambda    x:
placement_results[x]['points'], reverse=True)
    standings = init_standings(countries)

    rank = 17
    for country in sorted_placement:
        standings[country] = rank
        rank += 1

    sorted_knockout = sorted(knockout_results.items(), key=lambda x: x[1])
    for country, position in sorted_knockout:
        standings[country] = position

    rank = 9
    for country in ninth16:
        standings[country] = rank
        rank += 1
```

```
    sorted_countries = sorted(standings.keys(), key=lambda x: standings[x])

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for country, position in standings.items():
        avgstandings[country] += position

    # Calculate standard deviation of flight times
    total_std += statistics.stdev(airtime_by_country.values())

for country, position in avgstandings.items():
    avgstandings[country] = position/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Single Round Robin with Balanced Groups')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")
std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['singlerobinbalanced']    =    {'games':    total_games_played/numtrials,    'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}

# %% [markdown]
# ## Double Round Robin: 5 Groups of 4

# %%
avgstandings = {country: 0 for country in countries}

loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for _ in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    sorted_countries = []

    pool_results = {}
    direct_to_quarters = []
    qualifier_candidates = []
    bottom4 = []
    ninth16 = []

    # ---------- Phase 1: Group Roundrobin ----------
    for pool_name, teams in pools.items():

        standings = round_robin(teams)
```

```
        standings2 = round_robin(teams)

        for country in standings:
            standings[country]['points'] += standings2[country]['points']
            standings[country]['wins'] += standings2[country]['wins']
            standings[country]['losses'] += standings2[country]['losses']
            standings[country]['draws'] += standings2[country]['draws']

        df = sort_standings(standings)
        pool_results[pool_name] = df
        direct_to_quarters.append(df.index[0])      # Top 1 to quarters
        qualifier_candidates.extend(df.index[1:3])

        bottom4.append(df.index[3])


    # ---------- Phase 2: Qualifiers: Next 10 teams run in qualifiers ----------
    qualifier_results = round_robin(qualifier_candidates)
    qualifer2 = round_robin(qualifier_candidates)
    for country in qualifier_results:
        qualifier_results[country]['points'] += qualifer2[country]['points']
        qualifier_results[country]['wins'] += qualifer2[country]['wins']
        qualifier_results[country]['losses'] += qualifer2[country]['losses']
        qualifier_results[country]['draws'] += qualifer2[country]['draws']
    qualifier_df = sort_standings(qualifier_results)
    qualifier_top4 = list(qualifier_df.index[:3])
    ninth16.extend(qualifier_df.index[3:])

    # ---------- Phase 3: Knockout ----------
    knockout_teams = direct_to_quarters + qualifier_top4
    knockout_results = simulate_knockout(knockout_teams)

    # ---------- Phase 4: Placement Round ----------
    placement_results = round_robin(bottom4)
    placement2 = round_robin(bottom4)
    for country in placement_results:
        placement_results[country]['points'] += placement2[country]['points']
        placement_results[country]['wins'] += placement2[country]['wins']
        placement_results[country]['losses'] += placement2[country]['losses']
        placement_results[country]['draws'] += placement2[country]['draws']
    placement_df = sort_standings(placement_results)


            sorted_placement    =    sorted(placement_results.keys(),    key=lambda    x:
placement_results[x]['points'], reverse=True)
    standings = init_standings(countries)

    rank = 17
    for country in sorted_placement:
        standings[country] = rank
        rank += 1

    sorted_knockout = sorted(knockout_results.items(), key=lambda x: x[1])
    for country, position in sorted_knockout:
        standings[country] = position

    rank = 9
    for country in ninth16:
```

```
        standings[country] = rank
        rank += 1

    sorted_countries = sorted(standings.keys(), key=lambda x: standings[x])

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for country, position in standings.items():
        avgstandings[country] += position

    total_std += statistics.stdev(airtime_by_country.values())

for country, position in avgstandings.items():
    avgstandings[country] = position/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Double Round Robin with Balanced Groups')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")

for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")
std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['doublerobinbalanced']    =    {'games':    total_games_played/numtrials,    'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}

# %% [markdown]
# ## Random Elimination

# %%
import random

def simulate_random_elimination(countries):
    # Initialize standings
    standings = {country: 0 for country in countries}
    eliminated = []

    # Random elimination rounds
    round_number = 1
    while len(countries) > 1:
        random.shuffle(countries)
        next_round = []

        for i in range(0, len(countries), 2):
            if i + 1 < len(countries):
                country1 = countries[i]
                country2 = countries[i + 1]
                t1, p1, t2, p2 = simulate_match(country1, country2)

                if p1 == 3:
                    # country1 won
```

```
                    winner = t1
                else:
                    winner = t2

                loser = country1 if winner == country2 else country2
                next_round.append(winner)
                eliminated.append(loser)
            else:
                # If odd number of countries, the last one automatically advances
                next_round.append(countries[i])

        countries = next_round
        round_number += 1

    # Final winner
    winner = countries[0]
    standings[winner] = 1

    # Assign rankings to eliminated countries
    rank = 2
    for country in reversed(eliminated):
        standings[country] = rank
        rank += 1

    return standings

# Simulate the random elimination system
avgstandings = {country: 0 for country in countries}
loss_total = 0
totaltime = 0
total_games_played = 0
games_played_country = {country: 0 for country in countries}
total_std = 0

for _ in range(numtrials):
    airtime_by_country = {country: 0 for country in countries}
    random_elimination_standings = simulate_random_elimination(countries)

        sorted_countries = sorted(random_elimination_standings.keys(), key=lambda x:
random_elimination_standings[x])

    loss_total += evaluate_ranking(sorted_countries, countries_ranked)

    for country, position in random_elimination_standings.items():
        avgstandings[country] += position

    total_std += statistics.stdev(airtime_by_country.values())

for country, position in avgstandings.items():
    avgstandings[country] = position/numtrials

airtime_std = total_std/numtrials

# Plotting + Showing relevant data
plotgraph(avgstandings, 'Random Elimination')
print(f"Total games played: {total_games_played/numtrials}")
print(f"Total time: {(totaltime/numtrials)} hours")
```

```
for country, num in games_played_country.items():
    games_played_country[country] = num/numtrials
print(games_played_country)
print(f"Average Loss: {loss_total/numtrials}")
std_dev = statistics.stdev(games_played_country.values())
print(f"Standard Deviation of number of games played: {std_dev}")
print(f"Standard Deviation of airtime of countries: {airtime_std}")
fits['elimination']     =     {'games':     total_games_played/numtrials,     'time':
totaltime/numtrials, 'deviation': std_dev, 'loss': loss_total/numtrials, 'airtimedev':
airtime_std/(totaltime/numtrials)}



# %% [markdown]
# # ANALYSIS

# %%

normalized_fits = {}
metrics = ['games', 'time', 'deviation', 'loss', 'airtimedev']



# Extract min and max
min_max = {metric: (min(fit[metric] for fit in fits.values()), max(fit[metric] for fit
in fits.values())) for metric in metrics}

# Normalize
for key, values in fits.items():
    normalized_fits[key] = {
            metric: (values[metric] - min_max[metric][0]) / (min_max[metric][1] -
min_max[metric][0])
        for metric in metrics
    }

# weights = [0.1968708483, 0.2797103175, 0.0807229401, 0.442695894]
# weights = [0.2214610339, 0.3101115125, 0.07330141242,      0.3951260413]

weights = [0.2104155797,   0.3404226382, 0.07273615894,      0.2821103089, 0.0943153142]
perturbance = 0.1

# Sensitivity analysis
sensitivity_results = {}

for i, original_weight in enumerate(weights):
    perturbed_weights = weights.copy()

    # Increase
    perturbed_weights[i] = original_weight * (1+perturbance)
    increased_index = {}
    for key, values in normalized_fits.items():
          increased_index[key] = 1 - sum(values[metric] * weight for metric, weight in
zip(metrics, perturbed_weights))

    # Decrease
    perturbed_weights[i] = original_weight * (1-perturbance)
    decreased_index = {}
    for key, values in normalized_fits.items():
```

```
            decreased_index[key] = 1 - sum(values[metric] * weight for metric, weight in
zip(metrics, perturbed_weights))

    sensitivity_results[metrics[i]] = {
        "increased": increased_index,
        "decreased": decreased_index
    }

for metric, result in sensitivity_results.items():
    print(f"Sensitivity analysis for {metric}:")
    print("Increased weight:")
    for mode, index in result["increased"].items():
        print(f"  {mode}: {index:.3f}")
    print("Decreased weight:")
    for mode, index in result["decreased"].items():
        print(f"  {mode}: {index:.3f}")
    print()

# Overall index
overall_index = {}
for key, values in normalized_fits.items():
       overall_index[key] = 1 - sum(values[metric] * weight for metric, weight in
zip(metrics, weights))

for mode, index in overall_index.items():
    print(mode, index)


# %%
```

Appendix C.2: Code for the k-mean cluster algorithm

Python

```
# %%
import folium
from IPython.display import display

countries = [
    "Brazil",
    "Spain",
    "France",
    "Argentina",
    "Uruguay",
    "Colombia",
    "United Kingdom",
    "Paraguay",
    "Germany",
    "Ecuador",
    "Portugal",
```

```
    "Italy",
    "Morocco",
    "Egypt",
    "South Korea",
    "Japan",
    "Mexico",
    "Costa Rica",
    "New Zealand",
    "Australia",
    "Turkey",
    "Switzerland",
    "Norway",
    "Netherlands"
]

elo_ratings = [
    1994,
    2150,
    2031,
    2140,
    1922,
    1953,
    2012,
    1799,
    1988,
    1911,
    1988,
    1914,
    1807,
    1668,
    1745,
    1875,
    1817,
    1653,
    1596,
    1736,
    1837,
    1812,
    1828,
    1967
]

locations = [
    (-14.2350, -51.9253),  # Brazil
    (40.4637, -3.7492),    # Spain
    (46.6034, 1.8883),     # France
    (-38.4161, -63.6167),  # Argentina
    (-32.5228, -55.7659),  # Uruguay
    (4.5709, -74.2973),    # Colombia
    (55.3781, -3.4360),    # United Kingdom
    (-23.4420, -58.4438),  # Paraguay
    (51.1657, 10.4515),    # Germany
    (-1.8312, -78.1834),   # Ecuador
    (39.3999, -8.2245),    # Portugal
    (41.8719, 12.5674),    # Italy
    (31.7915, -7.0926),    # Morocco
    (26.8206, 30.8025),    # Egypt
    (35.9078, 127.7669),   # South Korea
```

```
    (36.2048, 138.2529),   # Japan
    (23.6345, -102.5528),  # Mexico
    (9.7489, -83.7534),    # Costa Rica
    (-40.9006, 174.8860),  # New Zealand
    (-25.2744, 133.7751),  # Australia
    (38.9637, 35.2433),    # Turkey
    (46.8182, 8.2275),     # Switzerland
    (60.4720, 8.4689),     # Norway
    (52.1326, 5.2913)      # Netherlands
]

locationdict = {country: locations[countries.index(country)] for country in countries}

# %%
import numpy as np
from math import radians, sin, cos, sqrt, atan2

# DISTANCE CALCULATIONS
def haversine(pos1, pos2):
    lat1, lon1 = pos1
    lat2, lon2 = pos2
    # Convert degrees to radians
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    # Haversine formula
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    R = 6371.0 # Radius of Earth in kilometers
    return R * c # Distance in kilometers

# Prepare the data
data    =    np.array([[location[0],    location[1]]    for    country,    location    in
locationdict.items()])
data = [tuple(location) for location in data]
print(data)

# %%


def calculate_centroid(cluster):
    if not cluster:
        return (0, 0)
    latitudes = [point[0] for point in cluster]
    longitudes = [point[1] for point in cluster]
    return (sum(latitudes) / len(latitudes), sum(longitudes) / len(longitudes))

def k_cluster(k, data):
    # Points of latitude and longitude
    # Forgy Method
    random_indices = list(np.random.choice(len(data), size=k, replace=False))
    centroids = [data[indx] for indx in random_indices]
    clusters = {i: [] for i in range(k)}


    converged = False
    while not converged:
        clusters = {i: [] for i in range(k)}
```

```
        for point in np.random.permutation(data):
                    distance_to_centroids = [haversine(point, centroid) for centroid in
centroids]
                    idx = distance_to_centroids.index(min(distance_to_centroids))
                    count = 1
                    while len(clusters[idx]) >= len(data)/k:
                           idx = sorted(range(len(distance_to_centroids)), key=lambda x:
distance_to_centroids[x])[count]
                        count += 1

                    clusters[idx].append(point)
        new_centroids = [calculate_centroid(cluster) for cluster in clusters.values()]


        # Displaying the map
        combined_map = folium.Map(location=[20, 20], zoom_start=2)

        # Add centroids to the map
        for idx, centroid in enumerate(centroids):
            folium.CircleMarker(
                location=centroid,
                radius=10,
                color=colors[idx % len(colors)],
                fill=True,
                fill_color=colors[idx % len(colors)],
                fill_opacity=1,
                tooltip=f"Centroid {idx}"
            ).add_to(combined_map)
        # Add countries to the map
        for cluster_id, points in clusters.items():
            for point in points:
                folium.Marker(
                    location=point,
                    icon=folium.Icon(color=colors[cluster_id % len(colors)]),
                    tooltip=f"Cluster {cluster_id}"
                ).add_to(combined_map)

        display(combined_map)

             max_shift = max(haversine(new, old) for new, old in zip(new_centroids,
centroids))
        # endloop criteria
        converged = max_shift < 1.0
        # converged = (new_centroids == centroids)
        centroids = new_centroids
        if converged:
            return clusters

num_clusters = 6
colors = ['red', 'blue', 'green', 'purple', 'orange', 'pink'] * (num_clusters //
len(['red', 'blue', 'green', 'purple', 'orange', 'pink']) + 1)
colors = colors[:num_clusters]

clusters = k_cluster(num_clusters, data)
```

Appendix C.3: Code schedule generation based on Simulated Annealing

The code below was run with pypy 3.10.14 to improve performance.

Python

```
from colorama import init
from termcolor import colored

import random
import time
from copy import deepcopy
from math import radians, sin, cos, sqrt, atan2, log, exp

init()

names = [
    "Brazil",
    "Spain",
    "France",
    "Argentina",
    "Uruguay",
    "Colombia",
    "United Kingdom",
    "Paraguay",
    "Germany",
    "Ecuador",
    "Portugal",
    "Italy",
    "Morocco",
    "Egypt",
    "South Korea",
    "Japan",
    "Mexico",
    "Costa Rica",
    "New Zealand",
    "Australia",
]

elo_ratings = [
    1994,
    2150,
    2031,
    2140,
    1922,
    1953,
    2012,
    1799,
    1988,
    1911,
    1988,
    1914,
    1807,
    1668,
    1745,
    1875,
    1817,
    1653,
    1596,
```

```
    1736,
]

locations = [
    (-14.2350, -51.9253),
    (40.4637, -3.7492),
    (46.6034, 1.8883),
    (-38.4161, -63.6167),
    (-32.5228, -55.7659),
    (4.5709, -74.2973),
    (55.3781, -3.4360),
    (-23.4420, -58.4438),
    (51.1657, 10.4515),
    (-1.8312, -78.1834),
    (39.3999, -8.2245),
    (41.8719, 12.5674),
    (31.7915, -7.0926),
    (26.8206, 30.8025),
    (35.9078, 127.7669),
    (36.2048, 138.2529),
    (23.6345, -102.5528),
    (9.7489, -83.7534),
    (-40.9006, 174.8860),
    (-25.2744, 133.7751),
]


def haversine(pos1, pos2):
    """https://en.wikipedia.org/wiki/Haversine_formula"""
    lat1, lon1 = pos1
    lat2, lon2 = pos2
    # Convert degrees to radians
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    d_lat = lat2 - lat1
    d_lon = lon2 - lon1
    # Haversine formula
    a = sin(d_lat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(d_lon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    R = 6371.0  # Radius of Earth in kilometres
    return R * c  # Distance in kilometres


def eft(pos1, pos2, speed_kmh=900):
    """estimated flight time between two locations"""
    distance = haversine(pos1, pos2)
    time_hours = distance / speed_kmh
    return time_hours


number_of_teams = 10

# Brazil
# France
# Uruguay
# England
# Ecuador
# Italy
# Mexico
```

```python
# South Korea
# Morocco
# Australia

team_ids = [0, 2, 4, 6, 9, 11, 16, 14, 12, 19]
# team_ids = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# team_ids = [11, 10, 12, 13]

for id in team_ids:
    print(f"{names[id]:20} [{elo_ratings[id]}]")
print()

number_of_matches = number_of_teams * (number_of_teams - 1)
number_of_rounds = number_of_teams - 1

distance_matrix = [[0 for _ in range(number_of_teams)] for _ in range(number_of_teams)]
for k in range(number_of_teams):
    for j in range(number_of_teams):
        distance_matrix[k][j] = haversine(
            locations[team_ids[k]], locations[team_ids[j]]
        )

#
https://www.google.com/search?hl=en&q=calculate%20timezone%20difference%20between%20two
%20locations%20on%20earth%20formula
# how much worse going east is than going west
MULTIPLIER = 1.5
# https://www.timeanddate.com/time/dst/2023.html
time_zone_matrix = [[0 for _ in range(number_of_teams)] for _ in
range(number_of_teams)]
for k in range(number_of_teams):
    for j in range(number_of_teams):
        time_zone_matrix[k][j] = (
            min(
                MULTIPLIER
                * ((locations[team_ids[j]][1] - locations[team_ids[k]][1]) % 360),
                ((locations[team_ids[k]][1] - locations[team_ids[j]][1]) % 360),
            )
            / 15
        )


def generate_schedule(number_of_teams):
    """Create a round_robin schedule for a given number of teams."""
    teams = list(range(number_of_teams))
    schedule = []
    if len(teams) % 2 == 1:
        teams = teams + [None]

    n = len(teams)
    map = list(range(n))
    mid = n // 2
    for i in range(n - 1):
        l1 = map[:mid]
        l2 = map[mid:]
        l2.reverse()
        round = []
        for j in range(mid):
```

```
            t1 = teams[l1[j]]
            t2 = teams[l2[j]]
            if j == 0 and i % 2 == 1:  # Alternates between home and away games
                round.append((t2, t1))
            else:
                round.append((t1, t2))
        schedule.append(round)
        map = map[mid:-1] + map[:mid] + map[-1:]
    return schedule


def get_initial_solution(number_of_teams):
    schedule = generate_schedule(number_of_teams)
    initial_solution = [
        [0 for _ in range(number_of_teams)] for _ in range(number_of_rounds)
    ]
    for r, round in enumerate(schedule):
        for home, away in round:
            initial_solution[r][away] = home
            initial_solution[r][home] = home

    return initial_solution


initial_solution = get_initial_solution(number_of_teams)
start = time.time()


# EVERYTHING BEFORE THIS POINT IS PRE-PROCESSING


def find_opponent(S, round, team):
    for other in range(number_of_teams):
        if other == team:
            continue
        if S[round][other] == S[round][team]:
            return other
    assert False, f"No opponent found {team, round, S[round]}"


def get_cost(S):
    """Calculates the cost of the current solution"""
    number_of_violations = 0
    # violation for 3 consecutive home and away break
    for i in range(number_of_teams):
        home_count = 0
        away_count = 0
        for k in range(number_of_rounds):
            # counts the number of consecutive home matches
            if S[k][i] == i:
                home_count += 1
                away_count = 0
            else:
                away_count += 1
                home_count = 0

            if home_count > 2:
                number_of_violations += 1
```

```
            if away_count > 2:
                number_of_violations += 1

    # penalty for a combined of 7 or more hours in 2 consecutive rounds
    HOUR_MAX = 5
    for i in range(number_of_teams):
        for k in range(number_of_rounds - 2):
            opp = find_opponent(S, k, i)
            # tz = (
            #     time_zone_matrix[S[k][i]][S[k + 1][i]]
            #     + time_zone_matrix[S[k + 1][i]][S[k + 2][i]]
            # )
        if (
            time_zone_matrix[S[k][i]][opp] > HOUR_MAX
            and time_zone_matrix[opp][S[k + 1][i]] > HOUR_MAX
        ):
            number_of_violations += 1

      # violation for consecutive high elo diff games no 2 consecutive high elo diff
games
    HIGH_DIFFERENCE_IN_ELO = 190
    for i in range(number_of_teams):
        best_count = 0
        worst_count = 0
        for k in range(0, number_of_rounds):
            opp = find_opponent(S, k, i)

            # determine if we are a lot better than our opponent
            if (
                elo_ratings[team_ids[i]] - elo_ratings[team_ids[opp]]
                > HIGH_DIFFERENCE_IN_ELO
            ):
                best_count += 1
            else:
                best_count = 0
            if best_count > 2:
                number_of_violations += 1
            # determine if we are a lot worse than our opponent
            if (
                elo_ratings[team_ids[opp]] - elo_ratings[team_ids[i]]
                > HIGH_DIFFERENCE_IN_ELO
            ):
                worst_count += 1
            else:
                worst_count = 0
            if worst_count > 2:
                number_of_violations += 1

    violations_S = number_of_violations

    total_distance = 0
    for i in range(number_of_teams):
        # from home to the first location
        total_distance += distance_matrix[i][S[0][i]]
        for k in range(1, number_of_rounds):
            # from location to location
            total_distance += distance_matrix[S[k - 1][i]][S[k][i]]
        # from last location to home
```

```
        total_distance += distance_matrix[S[k][i]][i]

    cost_S = total_distance
    return violations_S, cost_S


def calculate_penalty(violations):
    """Calculates the penalty cost based on the number of violations"""
    return 0 if violations == 0 else 1 + sqrt(violations) * log(violations / 2)


def weighted_cost(cost, penalty, weight):
    return sqrt(cost**2 + (weight * penalty) ** 2)


# http://aris.me/pubs/ttp.pdf


def make_move(S, alpha):
    """Make a random move in the schedule."""
    move = random.randint(0, 4)
    new_S = deepcopy(S)

    if move == 0:
        for _ in range(random.randint(1, 20)):
            # Pick a random pair of teams and swap their home/away games
            A, B = random.sample(range(number_of_teams), k=2)

            for k in range(number_of_rounds):
                if S[k][A] == B and S[k][B] == B:
                    new_S[k][A] = A
                    new_S[k][B] = A
                    break
                elif S[k][A] == A and S[k][B] == A:
                    new_S[k][A] = B
                    new_S[k][B] = B
                    break

    elif move == 1:  # Swaps the order of two Rounds
        A, B = random.sample(range(number_of_rounds), k=2)
        new_S[A], new_S[B] = new_S[B], new_S[A]

    elif move == 2:
        # Pick a random pair of teams and swap them in all their rounds
        # unless they are playing each other, then we don't care
        A, B = random.sample(range(number_of_teams), k=2)

        for k in range(number_of_rounds):
            if S[k][A] != B and S[k][B] != A:
                # swap the opponents of A and B
                new_S[k][A], new_S[k][B] = S[k][B], S[k][A]

                # if it was a home game, update to the new home
                if new_S[k][A] == B:
                    new_S[k][A] = A
                    for C in range(number_of_teams):
                        if S[k][C] == B and C != B:
                            new_S[k][C] = A
```

```
                        break

                if new_S[k][B] == A:
                    new_S[k][B] = B
                    for C in range(number_of_teams):
                        if S[k][C] == A and C != A:
                            new_S[k][C] = B
                            break

    elif move == 3:
        team = random.randint(0, number_of_teams - 1)
        r1, r2 = random.sample(range(number_of_rounds), k=2)
        queue = [team]
        added = set()

        while len(queue) > 0:
            current_team = queue.pop()
            # current_team, current_r1, current_r2 = queue.pop()
            opp1 = find_opponent(S, r1, current_team)
            opp2 = find_opponent(S, r2, current_team)

            # swaps the location of the current team in the two rounds
            (
                new_S[r1][current_team],
                new_S[r2][current_team],
            ) = (
                new_S[r1][current_team],
                new_S[r2][current_team],
            )

            # Fix the opponents to reflect the new match
            new_S[r1][opp2] = (
                current_team if new_S[r1][current_team] == current_team else opp2
            )
            new_S[r2][opp1] = (
                current_team if new_S[r2][current_team] == current_team else opp1
            )

            if opp1 not in added:
                added.add(opp1)
                queue.append(opp1)
            if opp2 not in added:
                added.add(opp2)
                queue.append(opp2)

    elif move == 4:
         # we pick a team and fix their schedule so that there are no three consecutive
home or away games
        for team in random.sample(range(number_of_teams), k=number_of_teams):
            home_count = 0
            away_count = 0
            for k in range(number_of_rounds):
                if S[k][team] == team:
                    home_count += 1
                    away_count = 0
                else:
                    away_count += 1
                    home_count = 0
```

```
                if home_count > 2:
                    opp = find_opponent(S, k, team)
                    new_S[k][team] = opp
                    new_S[k][opp] = opp
                    home_count = 0

                if away_count > 2:
                    opp = new_S[k][team]
                    new_S[k][team] = team
                    new_S[k][opp] = team
                    away_count = 0

    return new_S


def display_schedule(S):
    for i in range(number_of_teams):
        print(f"{names[team_ids[i]]} [{colored(elo_ratings[team_ids[i]], 'yellow')}]")
        for j in range(number_of_rounds):
            color = "grey"
            if (
                                    elo_ratings[team_ids[find_opponent(S,  j,  i)]]  -
elo_ratings[team_ids[i]]
                > 190
            ):
                color = "red"
            if (
                    elo_ratings[team_ids[i]] - elo_ratings[team_ids[find_opponent(S, j,
i)]]
                > 190
            ):
                color = "green"
            print(
                            f"   vs      {names[team_ids[find_opponent(S,  j,  i)]]:16}
{colored(f'{elo_ratings[team_ids[find_opponent(S,j,i)]]}',   color)}   {colored('home',
'green') if S[j][i] == i else colored('away', 'red')}",
            )
        print()


def simulated_annealing(maxR, maxP, maxC, T, beta, weight, theta, alpha):
    """https://en.wikipedia.org/wiki/Simulated_annealing"""
    # T0 = T

    # the current solution
    S = get_initial_solution(number_of_teams)
    violations_S, cost_S = get_cost(S)

    # If the solution is infeasible, we need to penalize it
    penality_cost = calculate_penalty(violations_S)
    cost_S = weighted_cost(cost_S, penality_cost, weight)

    # we will save the best feasible solution we have found so far
    best_solution = S
    best_feasible = 1e32
    nbf = 1e32
    best_infeasible = 1e32
```

```
    nbi = 1e32
    best_temp = T

    reheat = 0  # tracks the number of reheats without improvement
    counter = 0

    iterations = 0

    while reheat <= maxR:
        phase = 0
        while phase <= maxP:
            counter = 0
            while counter <= maxC:
                iterations += 1
                if iterations % 100_000 == 0:
                    if iterations % 1_000_000 == 0:
                        display_schedule(S)

                    print(
                         f"{colored('Iteration', 'yellow')} {iterations} at {time.time()
- start:.1f} seconds"
                    )
                # sillyness
                new_S = make_move(S, 0.5)

                # finished performing action, now calculate the cost
                violations_new_S, cost_new_S = get_cost(new_S)
                penality_cost = calculate_penalty(violations_S)
                cost_S = weighted_cost(cost_S, penality_cost, weight)

                # decide if we accept the altered solution
                accept = False
                if (
                    cost_new_S < cost_S
                    or (violations_new_S == 0 and cost_new_S < best_feasible)
                    or (violations_new_S > 0 and cost_new_S < best_infeasible)
                ):
                    accept = True
                else:
                    delta = float(cost_new_S - cost_S)
                    probability = exp(-delta / T)
                    chance = random.random()
                    if chance < probability:
                        accept = True

                if accept:  # we accepted the new solution
                    S = new_S
                    violations_S = violations_new_S
                    cost_S = cost_new_S

                    if violations_S == 0:
                        nbf = min(best_feasible, cost_S)
                    else:
                        nbi = min(best_infeasible, cost_S)

                    if nbf < best_feasible or nbi < best_infeasible:
                        # the solution is better than the best solution so far
                        best_temp = T
```

```
                        best_feasible = nbf
                        best_infeasible = nbi

                        if violations_S == 0:
                            # the solution is feasible
                            counter = 0  # keep cooking if good progress is made
                            phase = 0
                            best_solution = S

                            display_schedule(S)
                            print()
                            print(
                                                    f"{colored('New Best', 'magenta')}
[{best_feasible:.0f}, {best_infeasible:.0f}] at {time.time() - start:.1f} seconds"
                            )
                            print()

                            # write to file
                            with open("solution.txt", "w") as f:
                                f.write("\t".join(map(str, team_ids)) + "\n")
                                for round in best_solution:
                                    f.write("\t".join(map(str, round)) + "\n")

                        # Strategic Oscillation
                            weight = weight / theta if violations_S == 0 else weight *
theta
                else:  # we did not accept the new solution
                    counter += 1
            phase += 1
            T = T * beta  # decay the temperature

            if phase <= maxP and phase % 100 == 0:
                print()
                print(
                    f"{colored('Cooling', 'blue')} [{phase}/{maxP}] to {T:.1f} degrees"
                )
                print()

        reheat += 1
        T = best_temp * 2

        # pretty printing
        if reheat <= maxR:
            print()
                   print(f"{colored('Reheating', 'red')} [{reheat}/{maxR}] to {T:.1f}
degrees")
            print("-------------------------------------------------")
            print()
    return best_solution


# 0.
# maxC = 3000
# maxP = 710
# maxR = 1
# T = 1100
# beta = 0.999
# weight = 18000
```

```
# theta = 1.03

# 1.
# maxR = 10
# maxP = 5000
# maxC = 500
# T = 400
# beta = 0.99975
# weight = 4000
# theta = 1.04
# alpha = 0.0

# 2.
# maxR = 2
# maxP = 5000
# maxC = 10
# T = 700
# beta = 0.99975
# weight = 30000
# theta = 1
# # theta = 1.05
# alpha = 0.0

# 3. given for 10 teams
maxR = 10
maxP = 7100
maxC = 5000
T = 400
beta = 0.9999
weight = 6000
theta = 1.0
alpha = 0.0


# fast cooling:
# maxR = 1
# maxP = 70
# maxC = 5000
# T = 700
# beta = 0.98
# weight = 60000.0
# theta = 1.05
# alpha = 0.0


# quick search
# maxR = 3
# maxP = 7
# maxC = 100
# T = 700
# beta = 0.98
# weight = 60000
# theta = 1.05
# alpha = 0.0

# maxR = 3
# maxP = 7100
# maxC = 0
```

```
# T = 500
# beta = 0.9999
# weight = 10
# theta = 1.05
# alpha = 0.0


# note that beta ^ maxP is roughly 1/2

S = simulated_annealing(maxR, maxP, maxC, T, beta, weight, theta, alpha)
print("Initial Solution:")
for round in initial_solution:
    print(*round, sep="\t")
print()
print("Final Solution:")
for round in S:
    print(*round, sep="\t")

print()
print()

display_schedule(S)

print(get_cost(S))
```

Appendix C.4: Code for scheduling matches on the time of the day

Python

```
# %% [markdown]
# # BRUH

# %%
with open('timedata.txt', 'r') as file:
    lines = file.readlines()

third_column = [line.split()[3] for line in lines if line.strip()]

time_percentage_dict = {line.split()[0]: percentage for line, percentage in zip(lines,
third_column)}
time_percentage_dict = {key: float(value) for key, value in
time_percentage_dict.items()}

print(time_percentage_dict)

# %%
countries = [
    "Brazil",
    "Spain",
    "France",
    "Argentina",
    "Uruguay",
    "Colombia",
    "United Kingdom",
    "Paraguay",
```

```
    "Germany",
    "Ecuador",
    "Portugal",
    "Italy",
    "Morocco",
    "Egypt",
    "South Korea",
    "Japan",
    "Mexico",
    "Costa Rica",
    "New Zealand",
    "Australia",
]

countryfanindex = {
    # country: (fan proportion index, fan population index)
    "Brazil": (0.88, 1),
    "Spain": (0.88, 0.76),
    "France": (0.69, 0.81),
    "Argentina": (0.94, 0.77),
    "Uruguay": (1, 0.34),
    "Colombia": (0.81, 0.77),
    "United Kingdom": (0.75, 0.85),
    "Paraguay": (0.81, 0.37),
    "Germany": (0.63, 0.87),
    "Ecuador": (0.69, 0.44),
    "Portugal": (0.81, 0.39),
    "Italy": (0.81, 0.83),
    "Morocco": (0.57, 0.57),
    "Egypt": (0.63, 0.94),
    "South Korea": (0.71, 0.73),
    "Japan": (0.38, 0.87),
    "Mexico": (0.79, 1),
    "Costa Rica": (0.75, 0.34),
    "New Zealand": (0.63, 0.34),
    "Australia":(0.13, 0.36)
}

locations = [
    (-14.2350, -51.9253),  # Brazil
    (40.4637, -3.7492),    # Spain
    (46.6034, 1.8883),     # France
    (-38.4161, -63.6167),  # Argentina
    (-32.5228, -55.7659),  # Uruguay
    (4.5709, -74.2973),    # Colombia
    (55.3781, -3.4360),    # United Kingdom
    (-23.4420, -58.4438),  # Paraguay
    (51.1657, 10.4515),    # Germany
    (-1.8312, -78.1834),   # Ecuador
    (39.3999, -8.2245),    # Portugal
    (41.8719, 12.5674),    # Italy
    (31.7915, -7.0926),    # Morocco
    (26.8206, 30.8025),    # Egypt
    (35.9078, 127.7669),   # South Korea
    (36.2048, 138.2529),   # Japan
    (23.6345, -102.5528),  # Mexico
    (9.7489, -83.7534),    # Costa Rica
    (-40.9006, 174.8860),  # New Zealand
```

```
    (-25.2744, 133.7751),  # Australia
    (38.9637, 35.2433),    # Turkey
    (46.8182, 8.2275),     # Switzerland
    (60.4720, 8.4689),     # Norway
    (52.1326, 5.2913)      # Netherlands
]

timezones = [
    "America/Sao_Paulo",  # Brazil
    "Europe/Madrid",      # Spain
    "Europe/Paris",       # France
    "America/Argentina/Buenos_Aires",  # Argentina
    "America/Montevideo", # Uruguay
    "America/Bogota",     # Colombia
    "Europe/London",      # United Kingdom
    "America/Asuncion",   # Paraguay
    "Europe/Berlin",      # Germany
    "America/Guayaquil",  # Ecuador
    "Europe/Lisbon",      # Portugal
    "Europe/Rome",        # Italy
    "Africa/Casablanca",  # Morocco
    "Africa/Cairo",       # Egypt
    "Asia/Seoul",         # South Korea
    "Asia/Tokyo",         # Japan
    "America/Mexico_City",# Mexico
    "America/Costa_Rica", # Costa Rica
    "Pacific/Auckland",   # New Zealand
    "Australia/Sydney"    # Australia
]

locationdict = {country: locations[countries.index(country)] for country in countries}
timezonedict = {country: timezones[countries.index(country)] for country in countries}

tv_viewership_db = {country: time_percentage_dict for country in countries}

for data, item in tv_viewership_db.items():
    print(data, item)

# %% [markdown]
# https://www.bls.gov/opub/btn/volume-7/television-capturing-americas-attention.html

# %%
import pytz
from datetime import datetime
from datetime import timedelta

def converttime(time, country1, country2):
    start_time, end_time = time.split('-')

    # Calculate the time difference based on longitudes
    longitude_diff = locationdict[country2][1] - locationdict[country1][1]
    time_offset = longitude_diff // 15  # 15 degrees longitude equals 1 hour

    # Convert start_time
    start_time_obj = datetime.strptime(start_time, '%H:%M')
    adjusted_time_obj = start_time_obj + timedelta(hours=time_offset)
    start_time_converted = adjusted_time_obj.strftime('%H:%M')
```

```
    # Convert end_time
    end_time_obj = datetime.strptime(end_time, '%H:%M')
    adjusted_end_time_obj = end_time_obj + timedelta(hours=time_offset)
    end_time_converted = adjusted_end_time_obj.strftime('%H:%M')

    converted_time = f"{start_time_converted}-{end_time_converted}"
    return converted_time

def combinedist(home, away):
    # WEIGHTS.
            fan_index_home   =   countryfanindex[home][0]   *   (0.78/(0.63+0.78))   +
countryfanindex[home][1] * (0.63/(0.63+0.78))
            fan_index_away   =   countryfanindex[away][0]   *   (0.78/(0.63+0.78))   +
countryfanindex[away][1] * (0.63/(0.63+0.78))

    # HOME COUNTRY WEIGHT
    homecountryweight = 2
    s = fan_index_home + fan_index_away
    fanweighthome = (fan_index_home)/s
    fanweightaway = 1-fanweighthome

    combined = {}
    homedist = {}
    awaydist = {}
    for time in tv_viewership_db[home]:
        time2 = converttime(time, home, away)


         combined[time] = (homecountryweight/(1 + homecountryweight)) * fanweighthome *
tv_viewership_db[home][time]   +   (1/(1   +   homecountryweight))   *   fanweightaway   *
tv_viewership_db[away][time2]
        homedist[time] = fanweighthome * tv_viewership_db[home][time]
        awaydist[time] = fanweightaway * tv_viewership_db[away][time2]
    return combined, homedist, awaydist


homecountry = 'Uruguay'
awaycountry = 'Italy'
engagement, homedist, awaydist = combinedist(home=homecountry, away=awaycountry)
# for thing in engagement.items():
#     print(thing)

hours = list(engagement.keys())

import matplotlib.pyplot as plt

# Extract starting hours
starting_hours = [int(hour.split(':')[0]) for hour in hours]
starting_hours = [f"{hour:02}:00" for hour in starting_hours]

# Plot engagement against starting hours
plt.figure(figsize=(12, 6))
plt.plot(starting_hours, [engagement[hour] for hour in hours], marker='o', color='b',
label='Total Viewership')
plt.plot(starting_hours, [homedist[hour] for hour in hours], marker='x', color='g',
label=f'{homecountry} (Home) Distribution')
plt.plot(starting_hours, [awaydist[hour] for hour in hours], marker='s', color='r',
label=f'{awaycountry} (Away) Distribution')
```

```
plt.title('Engagement Over Time')
plt.xlabel('Hours')
plt.ylabel('Engagement')
plt.xticks(rotation=45)
plt.xticks(starting_hours)
plt.grid(True)
plt.tight_layout()
plt.legend(loc='upper left')

best_viewship = 0
best_index = 0
for i in range(len(hours)):
            viewership   =   engagement[hours[i%24]]   +   engagement[hours[(i+1)%24]]   +
engagement[hours[(i+2)%24]]

    if viewership > best_viewship:
        best_viewship = viewership
        best_index = i


print(hours[best_index%24])
print(hours[(best_index+1)%24])
print(hours[(best_index+2)%24])

plt.fill_between(
    starting_hours,
    [engagement[hour] for hour in hours],
     where=[best_index % 24 <= i <= (best_index + 3) % 24 if (best_index + 3) % 24 >=
best_index % 24 else (i >= best_index % 24 or i <= (best_index + 3) % 24) for i in
range(len(hours))],
    color='purple',
    alpha=0.3,
    label='Best Viewership Hours'
)

plt.show()
```

Appendix C.5: Code used to assign groups to months

C/C++

```
// implementation taken from https://cp-algorithms.com/graph/hungarian-algorithm.html

#include <bits/stdc++.h>
using namespace std;

vector<float> hungarian(vector<vector<float>>& A) {
    float n = A.size(), m = A[0].size();
    const float INF = 1000;
    vector<float> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
    for (float i = 1; i <= n; ++i) {
        p[0] = i;
        float j0 = 0;
        vector<float> minv(m + 1, INF);
```

```
        vector<bool> used(m + 1, false);
        do {
            used[j0] = true;
            float i0 = p[j0], delta = INF, j1 = 0;
            for (float j = 1; j <= m; ++j)
                if (!used[j]) {
                    float cur = A[i0 - 1][j - 1] - u[i0] - v[j];
                    if (cur < minv[j])
                        minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta)
                        delta = minv[j], j1 = j;
                }
            for (float j = 0; j <= m; ++j)
                if (used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            j0 = j1;
        } while (p[j0] != 0);
        do {
            float j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    vector<float> ans(n + 1);
    for (float j = 1; j <= m; ++j)
        ans[p[j]] = j;
    return ans;
}



signed main() {
    vector<vector<float>> temps{
        {24.75, 23.25, 21.5, 17.75, 16},
        {5, 7.75, 11,14.25,18.75},
        {17.25,17.75,18.75,18.25, 17.75},
        {12.5,14.5,17,20.25,23},
        {12.5,14,15.5,16.75,17.75},
    };

    vector<vector<float>> costs(temps.size(), vector<float>(temps[0].size()));
    for (int i = 0; i < temps.size(); ++i) {
        for (int j = 0; j < temps[0].size(); ++j) {
            costs[i][j] = fabs(temps[i][j] - 20);
        }
    }

    vector<float> result = hungarian(costs);
    cout << "The optimal assignment is:" << endl;
    for (float i = 1; i < result.size(); ++i) {
        cout << "region " << i << " is assigned to month " << result[i] + 1 << endl;
    }
    return 0;
}
```

Appendix D: Glossary of terms

- **Full Single Round Robin**
    - All teams are placed in a combined pool, playing every other team exactly once. The teams are then ranked according to the '3 points for a win' scoring system, and that becomes the result of the tournament.
- **Full Double Round Robin**
    - All teams are placed in a combined pool, playing every other team twice, one home and one away. The teams are then ranked according to the '3 points for a win' scoring system, and that becomes the result of the tournament.
- **Hybrid Grouped Single Round Robin + Knockouts (by region)**
    - Teams are separated into 3 groups of 4 teams and 1 group of 8 teams based on their geographical location and proximity to other national teams. A single round robin is then played within each group. Based on the '3 points for a win' scoring system, 1st place in each of the 4 groups will advance to the knockout stage. 2nd and 3rd in each group will advance to the qualifier single round robin group to determine the top 4 teams to join the knockout stage. Teams below 3rd in each group will be placed into the placement single round robin group, and their final rank will be determined by points accumulated in the placement single round robin. The knockout bracket will be paired in a way so that the first seed (based on elo) versus the eighth seed, the second seed versus the seventh seed and so on. The final standings will be as follows: knockout winner, knockout runner up, semifinalists ranked by seed, quarterfinalists ranked by seed, qualifier single round robin teams ranked by points, and placement single round robin teams ranked by points.
- **Hybrid Grouped Double Round Robin + Knockouts - (by region)**
    - Teams are separated into 3 groups of 4 teams and 1 group of 8 teams based on their geographical location and proximity to other national teams. A double round robin is then played within each group so that each national team plays each other once at home and once away. Based on the '3 points for a win' scoring system, 1st place in each of the 4 groups will advance to the knockout stage. 2nd and 3rd in each group will advance to the qualifier double round robin group to determine the top 4 teams to join the knockout stage. Teams below 3rd in their group stage will be placed into the placement double round robin group, and their final rank will be determined by points accumulated in the placement double round robin. The knockout bracket will be paired in a way so that the first seed (based on elo) versus the eighth seed, the second seed versus the seventh seed and so on. The final standings will be as follows: knockout winner, knockout runner up, semifinalists ranked by seed, quarterfinalists ranked by seed, qualifier double round robin teams ranked by points, and placement double round robin teams ranked by points.
- **Hybrid Grouped Single Round Robin + Knockouts - (by region + size)**
    - Teams are separated into 5 groups of 4 teams based on their geographical location and proximity to other national teams. A single round robin is then played within each group. Based on the '3 points for a win' scoring system, 1st place in each of the 5 groups will advance to the knockout stage. 2nd and 3rd in each group will advance to the qualifier single round robin group to determine the top 3 teams to join the knockout stage. Teams below 3rd in each group will be placed into the placement single round robin group, and their final rank will be determined by points accumulated in the placement single round robin. The knockout bracket will be paired in

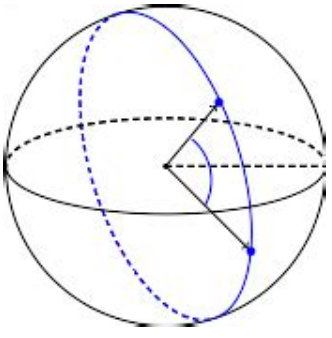
a way so that the first seed (based on elo) versus the eighth seed, the second seed versus the seventh seed and so on. The final standings will be as follows: knockout winner, knockout runner up, semifinalists ranked by seed, quarterfinalists ranked by seed, qualifier single round robin teams ranked by points, and placement single round robin teams ranked by points.

- **Hybrid Grouped Double Round Robin + Knockouts - (by region + size)**
    - Teams are separated into 5 groups of 4 teams based on their geographical location and proximity to other national teams. A double round robin is then played within each group so that each national team plays each other once at home and once away. Based on the '3 points for a win' scoring system, 1st place in each of the 5 groups will advance to the knockout stage. 2nd and 3rd in each group will advance to the qualifier double round robin group to determine the top 3 teams to join the knockout stage. Teams below 3rd in their group will be placed into the placement double round robin group, and their final rank will be determined by points accumulated in the placement double round robin. The knockout bracket will be paired in a way so that the first seed (based on elo) versus the eighth seed, the second seed versus the seventh seed and so on. The final standings will be as follows: knockout winner, knockout runner up, semifinalists ranked by seed, quarterfinalists ranked by seed, qualifier double round robin teams ranked by points, and placement double round robin teams ranked by points.
- **Random Knockout**
    - Teams are placed into a knockout bracket by random, and if they lose a match, they are eliminated from the tournament (due to the fact we have 20 teams which isn't a power of 2, there will be teams with BYEs). The final rankings will be as follows: knockout winner, runner-up, and then the remaining teams based on highest round reached (eg. semifinals) and then team seed.
- **Expanded Hybrid Grouped Single Round Robin + Knockouts - (by region + size)**
    - Teams are separated into 6 groups of 4 based on their geographical location and proximity to other national teams. A single round robin is then played within each group. Based on the '3 points for a win' scoring system, 1st place in each of the 6 groups will advance to the knockout stage. 2nd and 3rd in each group will advance to the qualifier single round robin group to determine the top 2 teams to join the knockout stage. Teams below 3rd will be placed into the placement double round robin group, and their final rank will be determined by points accumulated in the placement single round robin. The knockout bracket will be paired in a way so that the first seed (based on elo) versus the eighth seed, the second seed versus the seventh seed and so on. The final standings will be as follows: knockout winner, knockout runner up, semifinalists ranked by seed, quarterfinalists ranked by seed, qualifier single round robin teams ranked by points, and placement single round robin teams ranked by points.

Appendix E: Supporting Equations

Appendix E.1: Great Circle calculations

- The shortest distance between two points P and Q is then given by $D = \frac{\theta R}{2}$ where θ is the angle $\angle POQ$ (where O is the centre of the earth) measured in radians and $R = 6,371\ km$ is the radius of the earth. Although the airplane does not sit on the surface of the earth, the maximum elevation a commercial of an airplane above ground $11km$ is insignificant and hence ignored in the calculation. Also the variation in the radius of the earth is not important. The **haversine formula** which states that



$\sin^2\left(\frac{\theta}{2}\right) = a = \sin^2\left(\frac{\varphi_P - \varphi_Q}{2}\right) + \cos(\varphi_P)\cos(\varphi_Q)\sin^2\left(\frac{\lambda_P - \lambda_Q}{2}\right)$ where φ and λ correspond to the latitude and longitude of the points P and Q can then be used to find θ. Keep in mind that latitude and longitude are both measured in degrees and should be first converted into radians. We first calculate a then rearrange to give $\tan^2\left(\frac{\theta}{2}\right) = \frac{\sin^2\left(\frac{\theta}{2}\right)}{\cos^2\left(\frac{\theta}{2}\right)}$ $\left(\frac{\theta}{2}\right) = \frac{a}{1-a}$. Then to ensure that our value θ is the angle of the smaller arc we must keep the value of $\frac{\theta}{2}$ in the first quadrant of the cartesian plane, we use $\arctan_2$ where both the inputted x and y coordinates are positive.

$$\theta = 2 \cdot \arctan_2\left(\sqrt{a}, \sqrt{1-a}\right)$$

Appendix E.2: Time zone difference calculation

The earth rotates along its axis of rotation (perpendicular to longitude) at about 360 degrees every 24 hours, this means that in 1 hour the earth rotates 15 degrees. Using this we can determine that the difference in timezones between two locations P and Q in earth is about 1 hour every 15 degrees difference in longtitude. But note that logitude is a measure of angle and so we have to consider both the difference in longitude going from P to Q clockwise and anticlockwise

$$\frac{\angle POQ}{15} = min(\lambda_P - \lambda_Q, \lambda_Q - \lambda_P)$$

Appendix F: Sensitivity analysis tables

Appendix F.1: I_F Values compared to each perturbed weight

Each of the 7 columns corresponds to the 7 different tournament formats as outlined in 3.2. The entries in the table is the I_F value.

	#1	#2	#3	#4	#5	#6	#7
T_g: Increased	0.695	0.428	0.625	0.507	**0.715**	0.611	0.616
T_g: Decreased	0.715	0.470	0.635	0.530	**0.724**	0.629	0.616
T_a: Increased	0.689	0.415	0.624	0.507	**0.715**	0.609	0.616
T_a: Decreased	0.721	0.483	0.636	0.531	**0.725**	0.632	0.616
σ_N: Increased	0.705	0.449	0.627	0.511	**0.719**	0.616	0.616
σ_N: Decreased	0.705	0.449	0.633	0.526	**0.721**	0.624	0.617
P_A: Increased	0.704	0.449	0.626	0.515	**0.715**	0.616	0.607
P_A: Decreased	0.705	0.449	0.634	0.522	**0.725**	0.625	0.626
L: Increased	0.702	0.449	0.611	0.504	**0.707**	0.611	0.588

L: Decreased	0.708	0.449	0.649	0.533	**0.732**	0.629	0.645

Appendix G: Schedule for a simulated Knockout-Qualifier

Brazil [1994]
- vs Mexico 1817 home
- vs Italy 1914 home
- vs South Korea 1745 away
- vs Morocco 1807 away
- vs Australia 1736 home
- vs France 2031 away
- vs Ecuador 1911 home
- vs England 2012 home
- vs Uruguay 1922 away

France [2031]
- vs Uruguay 1922 home
- vs England 2012 home
- vs Mexico 1817 away
- vs Ecuador 1911 away
- vs South Korea 1745 home
- vs Brazil 1994 home
- vs Italy 1914 away
- vs Australia 1736 home
- vs Morocco 1807 home

Uruguay [1922]
- vs France 2031 away
- vs Australia 1736 home
- vs Morocco 1807 home
- vs Mexico 1817 away
- vs England 2012 home
- vs Ecuador 1911 home
- vs South Korea 1745 away
- vs Italy 1914 away
- vs Brazil 1994 home

England [2012]
- vs Morocco 1807 home
- vs France 2031 away
- vs Ecuador 1911 home
- vs Australia 1736 away
- vs Uruguay 1922 away
- vs Italy 1914 home
- vs Mexico 1817 away
- vs Brazil 1994 away
- vs South Korea 1745 home

Ecuador [1911]
- vs Italy 1914 home
- vs Morocco 1807 away
- vs England 2012 away
- vs France 2031 home
- vs Mexico 1817 home
- vs Uruguay 1922 away
- vs Brazil 1994 away
- vs South Korea 1745 home
- vs Australia 1736 home

Italy [1914]
- vs Ecuador 1911 away
- vs Brazil 1994 away
- vs Australia 1736 home
- vs South Korea 1745 home
- vs Morocco 1807 away
- vs England 2012 away
- vs France 2031 home
- vs Uruguay 1922 home
- vs Mexico 1817 away

Mexico [1817]
- vs Brazil 1994 away
- vs South Korea 1745 away
- vs France 2031 home
- vs Uruguay 1922 home
- vs Ecuador 1911 away
- vs Australia 1736 home
- vs England 2012 home
- vs Morocco 1807 away
- vs Italy 1914 home

South Korea [1745]
- vs Australia 1736 away
- vs Mexico 1817 home
- vs Brazil 1994 home
- vs Italy 1914 away
- vs France 2031 away
- vs Morocco 1807 home
- vs Uruguay 1922 home
- vs Ecuador 1911 away
- vs England 2012 away

Morocco [1807]
- vs England 2012 away
- vs Ecuador 1911 home
- vs Uruguay 1922 away
- vs Brazil 1994 home
- vs Italy 1914 home
- vs South Korea 1745 away
- vs Australia 1736 away
- vs Mexico 1817 home
- vs France 2031 away

Australia [1736]
- vs South Korea 1745 home
- vs Uruguay 1922 away
- vs Italy 1914 away
- vs England 2012 home
- vs Brazil 1994 away
- vs Mexico 1817 away
- vs Morocco 1807 home
- vs France 2031 away
- vs Ecuador 1911 away