BACHELOR'S DEGREE THESIS

# Degree in Mathematics

**Title: The Referee Assignment Problem**

**Author: Farners Vallespí i Soro**

**Advisor: Robert Nieuwenhuis**

**Department: Computer Science**

**Academic year: 2018 / 2019**



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Degree in Mathematics

Bachelor's Degree Thesis

# The Referee Assignment Problem

**Farners Vallespí i Soro**

Supervised by Robert Nieuwenhuis

June 2019

I would like to thank Robert Nieuwenhuis, for helping me throughout the project, sharing his expertise and giving me the chance to work in this project.

I would also like to acknowledge Oriol Porta and Pol Vallespí for always being there whenever I needed a second opinion and supporting me throughout the whole process.

## Abstract

In collaboration between a UPC spinoff, Barcelogic, and the Dutch Football Federation (KNVB), we define, study, implement and evaluate different approaches for solving the so-called Referee Assignment Problem (RAP). In this NP-complete constraint solving problem, numerous conditions must be met, such as the balance in the number of matches each referee must officiate, the frequency of each referee being assigned to a given team, the distance each referee must travel over the course of a season, etc.

### Keywords

Referee Assignment, Sports Scheduling, Optimization, Integer Linear Programming, NP-Complete

# Contents

# 1. Introduction

Professional sports gather a lot of interest all around the world and move masses of people and huge quantities of money. This is one of the main reasons optimization in sports has become a field that is gathering more importance as time goes by. Football is one of the most popular and economically significant sports followed by basketball and tennis, and with the money that it moves, it has gone past being considered not only a sport but an industry, moving researchers in order to find ways to make more money.

Among the most important fields in sports operations research we find leagues and tournaments scheduling, traveling tournament scheduling, playoff elimination and referee assignments. Other important areas of study include deciding the best tactics and strategy and forecasting, although this is usually done by individual competitors, and usually only those that have more to gain and move more money. In this project we will be working with the Referee Assignment Problem (RAP), which will be explained in more detail in section 1.1. Independently of its applications in practice, the RAP is also considered an interesting and challenging problem by itself, due to its hard combinatorial nature. Further detail about the other fields is given in subsection 1.3.

The RAP, once formulated, becomes a combinatorial optimization problem that can be solved with different methodologies among which are using complete solvers or local search algorithms, which are normally used in constraint satisfaction problems, where the goal is to find an assignment that fulfills all the constraints formulated, or optimization problems, where the goal is to find an assignment such that minimizes or maximizes a function. They are both methodologies that search virtual spaces, which are the domain of the function to be optimized. With local search methods, these virtual spaces are created with candidate solutions and are explored in order to find one that optimizes the objective function of the problem.

The main difference between both sets of methods is that complete algorithms are exhaustive, meaning they explore all possible solutions if needed and if there is a solution they find it if given enough time, meanwhile local search methods may become stagnated in the search and never find an optimum solution or even a solution at all. This is due to the fact that local search algorithms don't explore the whole search space, they follow an heuristic that tells them which states to visit and when they find a local optimum, they stop, restart or use some other method to escape from it, but they have no way of knowing whether this state is only a local optimum or a global one without exploring the whole space.

Another important difference is that, if no solution is found for a problem, complete solvers are able to prove the unsatisfiability of the problem and pinpoint the constraints that make the problem unable to be solved. It is also important to notice that local search algorithms, although they may not find the optimum solution, are usually faster to give a solution since they don't have to explore the whole search space.

Another procedure we have considered is breaking the problem into smaller sub-problems that are easier to solve since they have less variables and take into account less constraints. This way of working can produce good results with many problems, however, with optimization problems, it does not guarantee finding the optimum since the results obtained for the main problem are conditioned by the results from each sub-problem. This procedure, in fact, when working with constraint satisfaction problems, does not even assure finding a solution for the problem, since solving the problem one sub-problem at a time may run the problem towards a situation for which there is no assignment of values for the variables of the following sub-problem that fulfills all the constraints, leaving the next sub-problem to be faced without solution.

Given the experience and know-how of Barcelogic in the formulation of sports planning problems and the use of complete solvers, our aim in this project was to try to do this as well for the RAP and compare efficiency with local search techniques.

In this project we address two versions of the Referee Assignment Problem, which are presented and defined in section 2. The first version we work with is the basic one, which is introduced in subsection 2.1 and considers the general constraints taken into account when looking for refereeing assignments, and the second one we work with, which is presented in subsection 2.2, is the version with the specific requirements demanded by the Dutch Football Association (KNVB). For both of these versions of the RAP we present a description of the problem in full detail and then expose their mathematical formulation.

Section 3 defines and proves the complexity of the decision version of the referee assignment problem, section 4 presents the strategies used to solve the basic version of the problem using local search algorithms and sections 5 and 6 present the methodology used to use complete solvers to solve respectively the basic problem and the KNVB version of the problem. To use the complete solvers to face the RAP we have formulated each problem as an integer linear programming problem looking for the most appropriate formulation in terms of correctness and efficiency, since given the amount of data managed with this problem, not all formulations were viable.

Lastly, the results from implementing the different methodologies are presented and compared in section 7 and the concluding remarks and further work extensions are included in section 8.

## 1.1 Referee Assignment Problem

The Referee Assignment Problem is a common problem that is faced in sports management whenever a tournament or league is scheduled since all sports have at least one official that makes sure everything goes according to the rules. It basically consists on finding an assignment of referees to the refereeing slots that are generated for the games that are to be disputed. The calendar must be already scheduled before facing this problem since it conditions the assignments, for example, a referee cannot be in two places at the same time, so we have to know beforehand when will the games be disputed.

The RAP deals with a set of referees with different qualities, the calendar of the matches and information about these matches, a set of constraints that need to be fulfilled for the assignment to make sense and a set of preferences that are desired. Given all this information, the RAP looks for a feasible assignment that respects all the constraints and fulfills as many preferences as possible.

Each sport, nation and competition has its own set of rules, meaning each case has to be taken as a different problem since many things may vary, such as the number of refereeing slots that are generated for each game and the frequency of the matches. For example, an American football match needs 7 referees, a basketball game needs 3, a football match from La Liga, the Spanish top division league, needs 6 referees, and a match from the English Premier League, the first division league in professional football in England, needs 4 referees. This means each league will have its own set of constraints and preferences.

This makes the generalization of the problem quite difficult, so we will focus on the refereeing assignment in football leagues, since most leagues follow similar rules making it easier to formulate the basic problem.

## 1.2 Related work on the RAP

The Referee Assignment Problem is considered to have been defined by Duarte et al. in 2006 [1], who presented a general version of the problem and mentioned that each sport, nation and league would have its own particularities. Duarte is nowadays considered one of the most important contributors to this field of work.

The problem, however, had been studied before by Evans in 1984 [2] and 1988 [3], when the problem was applied to the assignment of the umpires in the American Baseball League. In this version of the problem the resting time of the referees took a lot of importance and several rules or constraints were imposed to take care of this, such as adding resting days between matches depending on the travelling distances between the stadiums where two consecutive matches took place. To solve the problem Evans used a support system to make decisions and optimization techniques, heuristic rules and the human judgment.

Since then, the techniques used to solve the problem have evolved and more promising ways to solve the RAP have been presented. In 1991, an heuristic algorithm was proposed by Wright [4] to solve the RAP applied to a cricket league from England, and in 2015 he himself published another article [5] adapting his solution to the assignment of the referees to several cricket leagues with different relevance.

Duarte et al., in their first article in 2006 [1], used an integral model for the RAP which was solved using a 3 phases heuristic-based algorithm. In the first phase a greedy heuristic looks for a feasible solution to the problem by assigning as many referees to the slots without violating constraints as possible, and then assigning the referees to the remaining refereeing slots until all of them are fulfilled. If the assignment obtained violates any constraints, the second phase is applied and an iterated local search is used to repair the solution by changing referee assignments one at a time for a given number of iterations. Finally an algorithm based on a meta-heuristic is used to search for a local optimum. This, obviously, does not guarantee nor an optimal solution nor a feasible one since the second phase may not find a feasible solution. In 2007 Duarte et al. published an extension of their previous work in which an hybrid iterated local search heuristic based on an integral mixed linear scheduling model was used in the third phase to look for the local optimum [6].

Duran et al. have also made several important contributions to this problem with the scheduling of the Chilean football league, however they have treated the problem mixed with the league scheduling. In 2005 [7] Duran et al. proposed the problem for the last matches of the first division games dividing the teams into 4 groups due to the play-off format of the competition and with the intention to minimize the travel time between consecutive matches. In 2010 [8] a new way to face the problem was proposed, which was applied to the second division league in Chile. In this new version Duran et al. looked for an assignment such that the number of times a team had two consecutive home or away matches was minimized. They were the first to mix scheduling with referee assignment.

Since then others have also presented the problem to be solved altogether with the league scheduling, the most important contribution being made by Atan and Hüseyinoğlu, who in 2015 proposed an integral mixed linear scheduling model for the Turkish football league using genetic algorithms [9].

In 2008 a model for the Turkish football league which was solved with local search algorithms was presented by Yavuz et al. [10] avoiding frequent assignments between referees and teams and in 2007 and in 2010 Ferland and Lamghari presented several versions of the RAP solved with tabu search and diversification strategies based on different neighborhoods [11], [12].

In 2013 Duran et al. proposed a new way to face the RAP [13], which was the most complete up to date, taking into account more things than any other model up until then, and was based on integer linear programming. Just like with his previous work, the model was applied to the First Division of the Chilean professional football league. With this approach the main goal was to balance the number of matches officiated per referee, the frequency of assignments per referee to a same team, the distance travelled and the difference between the skills of the referee and the importance of the match. Two formulations were given to solve this problem, one traditional and one based in pattern-based formulation, which got to reduce considerably the execution times.

Finally, in 2019, Linfati, Gatica and Escobar [14] presented a paper in which a non-linear binary program model was proposed. This model was intended to minimize the differences between the skills of the referees and the importance of the matches they are assigned to. The model is proved against real data from different sports such as football, volleyball and basketball and is solved using CPLEX.

Our model has been influenced by many of these works in different ways, mostly in order to decide how to model the problem and which constraints to implement. Several ideas about how to face the problem has also been gathered from these articles.

## 1.3 Other problems related to the RAP

As mentioned before, the other problems that are most relevant in sports operations research are the Playoff Elimination Problem, the League Scheduling Problem and the Travelling Tournament Problem.

The Playoff Elimination Problem arose from the eagerness of the fans, the press and the team employees to know whether the team is qualified for the playoffs of a competition or not and what does the team need to achieve in order to qualify, which usually requires acquiring a minimum position in the regular league. This problem takes into account the matches that have already been disputed and the possible outcomes from all the remaining matches to answer these questions.

This problem was first approached by Schwarts in 1966 [15], who applied a maximum-flow algorithm to solve it. In 1970 Hoffman and Rivlin [16] extended the problem adding the conditions necessary and sufficient to eliminate a team that is in a given position k or below this position in the league. In the year 1991 Robinson published another article [17] in which, using linear programming, he applied this problem to the baseball playoff eliminations and got results that eliminated the teams up to 5 days earlier than with the results from the league that took place in 1987.

The League Scheduling Problem consists on finding an assignment of teams to matches in a league alternating home and away games as much as possible and making each team play twice against the other teams. Initially the goal was characterizing the schedule so that it had as few breaks in the alternations as possible, however, as time has gone by and sports have become an industry this problem has evolved and now other goals are also contemplated, such as maximizing the profits obtained by the league and the clubs, which depend on the dates of the matches, their importance, the capacity of the venues, etc.

One of the versions of the League Scheduling Problem problem that has awakened more interest is the Travelling Tournament Problem, in which the main goal is to find the assignment that minimizes the travelling distance the teams must travel throughout the season, although other objectives are usually also considered. Besides the travelling distance other things are taken into account, just like in the League Scheduling Problem, such as logistic issues, different types of constraints that must be followed, stadiums availability, conflicting interests, etc.

The basic version of the League Scheduling Problem was first presented by Werra in 1981 [18], who also presented several theoretical models of the problem formulated with graphs in 1988 [19] and then faced a real case of this problem in 1990 and solved it using oriented factorization of complete graphs [20].

Throughout the years several approaches or techniques have been used to solve this problem, such as a mathematical programming approach, which has been used among others by Mcaloon, Tretkoff and Wetzel in 1997 [21], simulated annealing, which was used by Biajoli et al. in 2003 [22] and Van Hentenryck and Vegados in the year 2005 [23], and constraint-based programming approaches, which were used by Nemhauser and Trick in 2001 [24] and by Henz, Müller and Thiel in 2004 [25].

The Travelling Tournament Problem was first approached by Easton, Nemhauser and Trick in 2002 [26], who defined the problem making the tournament follow a double round robin schedule and solved it with a combined integer programming and constraint programming approach. In the year 2009, Uthus and Riddle [27] presented a conference paper in which they used an exact method to solve this problem, more precisely a depth first search, obtaining known optimal solutions in fewer computational time than past approaches. Later on an improved neighbourhood search was proposed by Langford in 2010 [28] and in the year 2012 Miyashiro, Matsui and Imahori presented a randomized approximation algorithm [29].

Hybrid methods have also been used to solve both versions of the problem and have showed better results. Some examples of hybrid method approaches are a combination of constraint and integer programming, which was presented in 2001 by Benoit et al. [30], and a mix of Tabu Search and agent based techniques, which has been used in an article by Adriane et al. presented this 2019 [31].

# 2. Definitions

In this chapter we will present both the basic or more general problem and the KNVB version of the problem adjusted to the needs of the Dutch Football Federation. We will describe both problems and expose their mathematical formulations.

## 2.1 Basic problem

The basic problem consists on assigning referees to all the matches of a football league. As this problem is being constructed in order to serve as a basis for as many different leagues as possible, no matter their specific needs, we will only consider constraints and preferences that are taken into account in most leagues.

While working the problem we will find hard constraints, meaning they are to never be broken if we want a consistent referee assignment, and soft constraints, which indicate preferences. The assignment we will be looking for will be such that fulfills all the hard constraints and violates as few soft constraints as possible.

### 2.1.1 Problem description

Given a league with n teams, the season is divided into 2n-2 rounds and each team plays a total of 2n-2 matches of the shape "team A against team B", one per round, where "team A" plays the role of the local team and "team B" exercises as the visiting team. At the end of the league every single team will have played twice against every other team in the league, one in the role of the local team and the other one as the visiting team.

We will part from a league in which the calendar of the matches is already decided, so we know who plays against who in each round, and we will assign the referees to each match. Usually every football match requires at least 3 referees: one main referee and two assistant referees or linesmen. As in some leagues the refereeing trios are previously defined and always go together, in this problem we will only assign the main refereeing role to the matches.

In order to assign the referees, we have to take into account the fact that referees must rest from time to time to avoid overloads, yet not too often in order to keep them busy and avoid uneasiness due to being assigned to too few matches. We must also try to avoid assigning a referee too often to the same team in order to avoid favoritism or troubles with the supporters. Moreover, we must consider the fact that not all matches have the same relevance nor are as easy to rule, so the referee assigned to the match has to be qualified or have the skill level required to face the match without any troubles.

Finally, we should consider that not all referees can be assigned to any match due to several circumstances, for example, a referee may not be available during a round due to international commitments, being on vacation, being ill, etc. Some referees are also banned from being assigned to certain teams due to past conflicts among other things. This is usually applied to avoid a referee being assigned to exercise on a match that is played nearby the place where he lives in order to avoid troublesome outcomes from a conflicting match.

### 2.1.2 Hard constraints

Now that we have presented the problem, the hard constraints that we can extract from the previous description and through using common sense and that must be fulfilled for the assignment to make sense are the following:

1. Every match has to have a referee assigned.
2. A referee cannot be assigned to more than one match per round.
3. The referee assigned to a match must have the required skill level.
4. Given an interval of rounds, every referee must have assigned more than a given minimum of matches and less than a given maximum.
5. Referees cannot have more than a given number of consecutive rounds with assigned matches.
6. A referee cannot be assigned twice to a same team before a certain number of rounds have passed.
7. A referee cannot be assigned twice to a match at the same stadium before a certain number of rounds have passed.
8. Given an incompatibility between a referee and a team, the referee cannot be assigned to matches with that team.
9. Given an incompatibility between a referee and a stadium, the referee cannot be assigned to matches played in that stadium.
10. Given an incompatibility between a referee and a round, the referee cannot be assigned to any match that takes place in that round.

### 2.1.3 Soft constraints

In order the get the assignment with the best quality we can get, making it as well balanced as possible, we consider the following soft constraints or preferences:

1. All referees must have the same number of assigned matches.
2. All referees must be assigned the same number of times to matches starring one team for all teams.

### 2.1.4 Model

In this section we will expose the mathematical formulation of the problem described above, which is formulated as an integer linear programming problem using boolean variables. Before starting, though, we introduce some definitions and notations.

For starters, we need to define the parameters that will be used to model the problem:

- $t_1 \dots t_N$ : teams taking part in the league.

- $a_1 \dots a_M$ : referees to assign.
- $r_1 \dots r_{2N-2}$ : rounds in the league.
- nRI : number of consecutive rounds in an interval of rounds.
- minM : minimum number of matches a referee must be assigned to given an interval of rounds.
- maxM : maximum number of matches a referee can be assigned to given an interval of rounds.
- maxCR : maximum number of consecutive rounds in which a referee can have a match assigned to.
- nRT : number of rounds that must pass before a referee can be assigned to a same team.
- nRS : number of rounds that must pass before a referee can be assigned to a match in the same stadium.
- $qa(a_i)$ : given constant value between 1 and 10 indicating the skill level of a referee $a_i$ with $i \in [1, M]$.
- $qp(t_i, t_j)$ : given constant value between 1 and 10 indicating the difficulty of the match between the teams $t_i$ and $t_j$ with $i, j \in [1, N]$.
- $p(t_i, t_j, r_k)$ : given constant that equals 1 if the match between the teams $t_i$ and $t_j$ is played in the round $r_k$, with $i, j \in [1, N]$ and $k \in [1, 2N-2]$. Otherwise it equals 0.
- $ie(a_i, t_j)$ : given constant that equals 1 if there is an incompatibility between the referee $a_i$ and the team $t_j$, with $i \in [1, M]$ and $j \in [1, N]$. Otherwise it equals 0.
- $is(a_i, t_j)$ : given constant that equals 1 if there is an incompatibility between the referee $a_i$ and the stadium in with team $t_j$ plays, with $i \in [1, M]$ and $j \in [1, N]$. Otherwise it equals 0.
- $ir(a_i, r_k)$ : given constant that equals 1 if there is an incompatibility between referee $a_i$ and the round $r_k$, with $i \in [1, M]$ and $k \in [1, 2N-2]$. Otherwise it equals 0.

We also need to define the variables that we are going to use to formulate the problem:

- $A(t_i, t_j, r_k, a_l)$ : boolean variable that will equal 1 if referee $a_l$ is assigned to the match between teams $t_i$ and $t_j$ played in the round $r_k$ with $t_i$ as the local team, where $i, j \in [1, N]$, $k \in [1, 2N-2]$ and $l \in [1, M]$, and will equal 0 otherwise.
- $WR(a_i, r_k)$ : boolean variable that will equal 1 if referee $a_i$ has a match assigned to in the round $r_k$, where $i \in [1, M]$ and $k \in [1, 2N-2]$, and will be equal to 0 otherwise.
- $DWR(a_i, a_j)$ : boolean variable that will equal 1 if referee $a_i$ if assigned to more matches than $a_j$, with $i, j \in [1, M]$, and will equal 0 otherwise.
- $DT(a_i, a_j, t_k)$ : boolean variable that will equal 1 if referee $a_i$ is assigned to matches in which team $t_k$ plays than $a_j$, with $i, j \in [1, M]$ and $k \in [1, N]$, and will equal 0 otherwise.

We can observe that, by defining variables *DWR* and *DT* this way, every time one of them equals 1 it means there is a soft constraint that is not being fulfilled: for every *DWR* variable that equals 1, there is a couple of referees for which the first one has more assignments than the second one, and for every *DT* variable that equals 1, there is a trio of two referees and a team for which the first referee is assigned to more matches starring the team than the second referee. As we want the assignment that minimizes the number of unfulfilled soft constraints, we will define the objective function as the addition of all these variables.

The integer linear programming model we propose in order to minimize the number of soft constraints that are broken is shown below. It consists on finding a set of values for the variables minimizing the objective function subjected to the constraints from (2) to (20). The objective function is described in (1), and, as it has been said before, tries to minimize the number of unfulfilled soft constraints.

$$min \left( \sum_{i=1}^{M} \sum_{j=1}^{M} DWR(a_i, a_j) + \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_{k=1}^{N} DT(a_i, a_j, t_k) \right) \tag{1}$$

The constraints used to limit the solutions are expressed below. Constraint (2) imposes that all existing matches have one and only one referee and that non-existing matches cannot have a referee assigned to them. Constraint (3) imposes that no referee can be assigned to more than one match per round, while constraint (4) ensures all referees assigned to a match have the skill level required to rule the match. With these constraints we make sure to fulfill the first three hard constraints exposed in section 2.1.2.

$$\sum_{l=1}^{M} A(t_i, t_j, r_k, a_l) = p(t_i, t_j, r_k) \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2] \tag{2}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} A(t_i, t_j, r_k, a_l) \leq 1 \quad \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{3}$$

$$A(t_i, t_j, r_k, a_l) \cdot (qa(a_l) - qp(t_i, t_j)) \geq 0 \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{4}$$

Constraints (5), (6) and (7) are used to ensure that the constraints about the number of matches that can be assigned to a referee for each interval of rounds are fulfilled, guaranteeing this way the 4th and 5th hard constraints described above. The first constraint asserts every referee has assigned at least *minM* matches per interval of rounds, while the second one makes certain that every referee has at most *maxM* assignments per interval of rounds, both of them considering intervals of rounds containing *nRI* consecutive rounds. The last of the three constraints mentioned in this paragraph ensures no referee has more consecutive matches assigned than is allowed.

$$\sum_{k=0}^{nRI-1} WR(a_i, r_{j+k}) \geq minM \quad \forall i \in [1, M], \forall j \in [1, 2N-1-nRI] \tag{5}$$

$$\sum_{k=0}^{nRI-1} WR(a_i, r_{j+k}) \leq maxM \quad \forall i \in [1, M], \forall j \in [1, 2N-1-nRI] \tag{6}$$

$$\sum_{k=0}^{maxCR} WR(a_i, r_{j+k}) \leq maxCR \quad \forall i \in [1, M], \forall j \in [1, 2N-2-maxCR] \tag{7}$$

Constraint (8) imposes that once a referee is assigned to a match, he or she cannot be assigned to a match repeating one of the teams before $nRT$ rounds have passed. This is checked by imposing that for every $(nRT+1)$ rounds, every referee can be assigned at most once to all the matches featuring one same team. Constraint (9), applying almost the same but only taking into account matches with the same local team, ensures referees don't have an assignment to the same stadium before $nRS$ rounds have passed. With these two constraints we guarantee the fulfillment of the 6th and 7th hard constraints mentioned in 2.1.2.

$$\begin{gathered}\sum_{t=0}^{nRT} \sum_{j=1}^{N} A(t_i, t_j, r_k + t, a_l) + A(t_j, t_i, r_k + t, a_l) \leq 1 \\ \forall i \in [1, N], \forall l \in [1, M], \forall k \in [1, 2N-1-nRT]\end{gathered} \tag{8}$$

$$\sum_{t=0}^{nRS} \sum_{j=1}^{N} A(t_i, t_j, r_k + t, a_l) \leq 1 \quad \forall i \in [1, N], \forall l \in [1, M], \forall k \in [1, 2N-1-nRS] \tag{9}$$

The remaining hard constraints that we have not mentioned yet are secured by the constraints between (10) and (13). Constraints (10) and (11) ensure incompatibilities between referees and teams are respected by first forbidding referees being assigned to home matches with a forbidden team in the role of the local team and then applying the same to the games in which the forbidden team is the visitor. Constraint (12) makes certain incompatibilities between referees and stadiums are respected by forbidding the assignment of referees to games in which the local team is the one that plays in the forbidden stadium, and constraint (13) ensures referees are not assigned to matches in rounds in which they are not available.

$$A(t_i, t_j, r_k, a_l) \leq 1 - ie(a_l, t_i) \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{10}$$

$$A(t_i, t_j, r_k, a_l) \leq 1 - ie(a_l, t_j) \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{11}$$

$$A(t_i, t_j, r_k, a_l) \leq 1 - is(a_l, t_i) \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{12}$$

$$A(t_i, t_j, r_k, a_l) \leq 1 - ir(a_l, r_k) \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{13}$$

Constraint (14) is used to define the variables WR which, for every referee and round, equal 1 if and only if the referee has a match assigned that round, meaning the variable is the sum of all the A variables for a given referee and round.

$$WR(a_l, r_k) = \sum_{i=1}^{N} \sum_{j=1}^{N} A(t_i, t_j, r_k, a_l) \quad \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{14}$$

Constraint (15) defines the variables DWR, which are to equal 1 if and only if, for a given pair of referees and a team, the first referee has more games assigned than the second referee. To do this, we calculate the difference between the number of working rounds for each referee and then impose that this value minus a huge quantity multiplied by the DWR variable must be less than or equal to 0. This will make the DWR variable 1 only if the difference value is positive. We have decided to use 1000 as this huge quantity since a normal league has about 400 games, so 1000 is a safe amount. To define the variables DT, which is done in constraint (16), we have used this same procedure but with the difference of matches with a same team between the two referees.

$$\sum_{k=1}^{2N-2} WR(a_i, r_k) - \sum_{k=1}^{2N-2} WR(a_j, r_k) - 1000 \cdot DWR(a_i, a_j) \leq 0 \quad \forall i, j \in [1, M] \tag{15}$$

$$\begin{gathered}\sum_{l=1}^{N} \sum_{m=1}^{M} (A(t_k, t_l, r_m, a_i) + A(t_l, t_k, r_m, a_i)) - \sum_{l=1}^{N} \sum_{m=1}^{M} (A(t_k, t_l, r_m, a_j) + A(t_l, t_k, r_m, a_j)) - \\ - 1000 \cdot DT(a_i, a_j, t_k) \leq 0 \quad \forall i, j \in [1, N], \forall k \in [1, N]\end{gathered} \tag{16}$$

Finally, constraints (17), (18), (19) and (20) impose that the variables are boolean, meaning they can only be equal to either 0 or 1.

$$A(t_i, t_j, r_k, a_l) \in \{0, 1\} \quad \forall i, j \in [1, N], \forall k \in [1, 2N-2], \forall l \in [1, M] \tag{17}$$

$$WR(a_i, r_k) \in \{0, 1\} \quad \forall i \in [1, M], \forall k \in [1, 2N-2] \tag{18}$$

$$DWR(a_i, a_j) \in \{0, 1\} \quad \forall i, j \in [1, M] \tag{19}$$

$$DT(a_i, a_j, t_k) \in \{0, 1\} \quad \forall i, j \in [1, M], \forall k \in [1, N] \tag{20}$$

## 2.2 KNVB problem

For this second version of the problem, we have focused on the needs of the Dutch Football Federation, attending all of their demands and making the model completely adapted to their league. Having modeled first the basic problem, we already had some work done, at the same time, though, many things are new or different in some way, so some things have been adapted, others have been removed and others have been incorporated and are new in relation to what we had before.

With this problem we will also find hard and soft constraints and we will treat them the same way we have done with the basic version of the problem, making sure the resulting assignment fulfills all the hard ones and as many of the soft ones as possible.

### 2.2.1 Problem description

The main goal of this second problem is, just like before, to assign referees to all refereeing positions that are available and look for the assignment that fulfills the most soft constraints. In this case, however, we are not dealing with just one league, but two, and we have more refereeing positions to assign.

The leagues to which we will have to assign the matches are the Eredivisie or first division league, which is the highest echelon in professional football in the Netherlands and has 18 teams, and Eerste Divisie or second division, which is the second highest and has 20 teams. Both leagues are intertwined, meaning they share referees and many rounds of both leagues are played at the same time.

For each Eredivisie match we will have 6 refereeing slots to fulfill: the main referee, the two assistant referees or linesmen, the forth referee, the video assistant referee or VAR, and the assistant video assistant referee or AVAR. For the Eerste Divisie matches however, we will only have to fulfill 4 refereeing slots since the VAR system is not applied to second division games, so we will only need the main referee, the two linesmen and the forth referee.

For this problem referees are split into 2 categories, one for the referees that can be assigned to the roles of main referee, forth referee and VAR, whom we will refer to simply as referees, and one for the referees that can develop the role of linesmen or assistant referees or AVAR, whom we will refer to as assistant referees. Moreover, referees and assistant referees are classified as senior, junior, masterclass or talententraject according to their experience and skills.

First division games must have a ratio of 7 senior referees and 14 senior assistant referees in the main roles and talententraject referees or assistant referees cannot be assigned to any game in any role. Also, between 3 and 6 of the VAR positions must be assigned to senior referees, at least 8 AVAR positions must be assigned to senior assistant referees and 7 of the referees in the role of the forth referee have to be a masterclass or a junior referee.

For second division games, there must be 8 masterclass referees as main referees and none of the other 2 can be talententraject referees, all of the forth refereeing roles, however, must be assigned to talententraject referees. For assistant refereeing roles, there have to be between 8 and 14 masterclass and at most 2 talententraject assignments.

Similarly to what we considered for the basic problem, referees can only have one assignment as main referee per round, and so do assistant referees, however they can be assigned to two games in the same round if the role they play in one of the games is one that doesn't require them to move much, meaning being the forth referee or the VAR in the case of a referee or the AVAR in the case of an assistant referee.

We also have to consider incompatibilities between referees and rounds since referees have international assignments such as UEFA Championship games, can become ill or have injuries or can take a leave, and incompatibilities with teams since, for example, the Dutch Football Federation forbids referees from being assigned to games in their hometown. This also applies to assistant referees.

Just like in the previous problem, we have a minimum of rounds that must pass before a referee or assistant referee can be assigned again to a match with the same team and we also have a maximum of rounds in an interval of rounds in which the referee is playing a main role, such as main referee or assistant referee. We will also want to try to avoid as much as possible a referee being assigned to 2 games in 4 days, assuring they get some rest between matches.

Matches and referees have a punctuation assigned to them, however they work differently than how we used them before. Matches from Eredivisie are given a qualification between 2 and 4, and matches from Eerste Divisie are between 0 and 1. Referees qualifications are also between 0 and 4. In this case though, we have a maximum of rounds a referee can go without being assigned to a match of a certain level. This qualifications also serve to reward referees who have better performances, meaning they have a better qualification, by giving them more important matches and more matches in general.

Finally, some refereeing trios, meaning the main referee and the 2 linesmen, which are the refereeing roles we refer to as main roles, must always go together in order to have practice for international appointments.

### 2.2.2 Hard constraints

From the previous definition of the problem we can extract the following hard constraints:

1. Every match has one referee, two assistant referees and a forth referee assigned.
2. Every Eredivisie match must have one VAR and one AVAR assigned.
3. Eerste Divisie matches do not have neither VAR nor AVAR assignments.
4. Referees and assistant referees cannot be assigned twice to the same match.
5. Every referee and assistant referee can have at most one main role per round.
6. Every referee and assistant referee can have at most two roles per round.
7. Given an incompatibility between a referee and a round, the referee cannot be assigned to any role in any match in the given round. The same goes for assistant referees.
8. Given an incompatibility between a referee and a team, the referee cannot be assigned to any role in any game in which the team is playing. The same is applied to assistant referees.
9. Designated refereeing trios must always go together when they are assigned to main refereeing roles.
10. For every interval of rounds, referees and assistant referees cannot be assigned to more main roles than a given maximum.

11. After being assigned to a main role in a match, referees and assistant roles cannot be assigned to a match with one of the teams playing the game before a given minimum of rounds have passed.
12. For every interval of rounds, senior referees and assistant referees must be assigned to at least one match with qualification 0 or 1, one with qualification 2 and one with qualification 3 or 4 in a main role.
13. Talententraject referees and assistant referees cannot be assigned to any roles in Eredivisie games.
14. For every Eredivisie round, there must be 7 senior referees assigned to the role of main referee.
15. For every Eredivisie round, there must be 14 senior assistant referees as linesmen.
16. For every Eredivisie round, there must be 2 senior referees assigned to the role of forth referee.
17. For every round, VAR positions must be filled by between 3 and 6 senior referees and AVAR positions must be filled by at least 8 senior assistant referees.
18. For every Eerste Divisie round, there must be 8 masterclass referees assigned to the role of main referee.
19. For every Eerste Divisie round, there must be between 8 and 14 masterclass assistant referees and at most 2 talententraject assistant referees assigned to the linesmen roles.
20. For every Eerste Divisie round, all the forth referee positions must be filled by talententraject referees.

### 2.2.3 Soft constraints

1. For every couple of referees or assistant referees, if one has a better skill punctuation than the other, he or she must be assigned to more main roles in games with higher punctuation than the other one.
2. For every couple of referees, if one has a better qualification than the other, he or she must have more assignments in main roles than the other referee. This also applies to assistant referees.
3. Referees and assistant referees cannot be assigned to main roles in 2 games in 4 consecutive days.

### 2.2.4 Model

In this section we will expose the mathematical formulation of this version of the problem. This problem will be formulated as an integer programming problem using boolean variables, and we will use the definitions and notations presented up next.

The parameters that we will need for the formulation of the model are the following:

- $t_1 \dots t_{n_t}$ : teams playing in any of the two leagues. Teams from $t_1$ to $t_{n_{t1}}$ play in Eredivisie and teams from $t_{n_{t1}+1}$ to $t_{n_t}$ play in Eerste Divisie.
- $r_1 \dots r_{n_r}$ : referees.
- $a_1 \dots a_{n_a}$ : assistant referees.

- $w_1 \ldots w_{n_w}$ : number of rounds played joining both leagues. As both calendars don't match, some rounds only have matches from one of the leagues.
- nRI : number of rounds in an interval of rounds.
- maxM : maximum number of main roles a referee or assistant referee can be assigned to in an interval of rounds.
- nRT : maximum number of rounds before a referee or assistant referee can be assigned to a main role in a match repeating one of the teams.
- maxRL : size of the interval of rounds in which a senior referee or assistant referee has to be assigned to one game with 0 or 1 qualification, one with a qualification of 2, and one with a qualification of 3 or 4.
- $rq(r_i)$ : given constant between 0 and 4 indicating the skills of the referee $r_i$, with $i \in [1, n_r]$.
- $aq(a_i)$ : given constant between 0 and 4 indicating the skills of the assistant referee $a_i$, with $i \in [1, n_a]$.
- $mq(t_i, t_j)$ : given constant between 0 and 4 indicating the qualification of the match between teams $t_i$ and $t_j$, with $i, j \in [1, n_t]$. If the teams belong to different divisions, meaning they will never face each other in a league match, the constant equals 0.
- $m(t_i, t_j, w_k)$ : given constant that equals 1 if the match between $t_i$ and $t_j$ is played in round $w_k$, with $i, j \in [1, n_t]$ and $k \in [1, n_w]$. Otherwise it equals 0.
- $irt(r_i, t_j)$ : given constant that equals 1 if there is an incompatibility between referee $r_i$ and team $t_j$, with $i \in [1, n_r]$ and $j \in [1, n_t]$. Otherwise it equals 0.
- $iat(a_i, t_j)$ : given constant that equals 1 if there is an incompatibility between assistant referee $a_i$ and team $t_j$, with $i \in [1, n_a]$ and $j \in [1, n_t]$. Otherwise it equals 0.
- $irw(r_i, w_j)$ : given constant that equals 1 if there is an incompatibility between referee $r_i$ and round $w_j$, with $i \in [1, n_r]$ and $j \in [1, n_w]$. Otherwise it equals 0.
- $iaw(a_i, w_j)$ : given constant that equals 1 if there is an incompatibility between assistant referee $a_i$ and round $w_j$, with $i \in [1, n_a]$ and $j \in [1, n_w]$. Otherwise it equals 0.
- $4d(t_{i1}, t_{j1}, t_{i2}, t_{j2}, w_k)$ : given constant that equals 1 if the match between $t_{i1}$ and $t_{j1}$ disputed in round $w_k$ is played less than 4 days before the match between $t_{i2}$ and $t_{j2}$ from the following round, with $i1, i2, j1, j2 \in [1, n_t]$ and $k \in [1, n_{w-1}]$. Otherwise it equals to 0.
- $trio(r_i, a_j, a_k)$ : given constant that equals 1 if $r_i$, $a_j$ and $a_k$ form a refereeing trio that must always go together, with $i \in [1, n_r]$ and $j, k \in [1, n_a]$. Otherwise it equals to 0.
- $sr(r_i)$ : given constant that equals 1 if $r_i$ is a senior referee, with $i \in [1, n_r]$. Otherwise it equals 0.
- $jr(r_i)$ : given constant that equals 1 if $r_i$ is a junior referee, with $i \in [1, n_r]$. Otherwise it equals 0.
- $mr(r_i)$ : given constant that equals 1 if $r_i$ is a masterclass referee, with $i \in [1, n_r]$. Otherwise it equals 0.

- $tr(r_i)$ : given constant that equals 1 if $r_i$ is a talententraject referee, with $i \in [1, n_r]$. Otherwise it equals 0.

- $sa(a_i)$ : given constant that equals 1 if $a_i$ is a senior assistant referee, with $i \in [1, n_a]$. Otherwise it equals 0.

- $ja(a_i)$ : given constant that equals 1 if $a_i$ is a junior assistant referee, with $i \in [1, n_a]$. Otherwise it equals 0.

- $ma(a_i)$ : given constant that equals 1 if $a_i$ is a masterclass assistant referee, with $i \in [1, n_a]$. Otherwise it equals 0.

- $ta(a_i)$ : given constant that equals 1 if $a_i$ is a talententraject assistant referee, with $i \in [1, n_a]$. Otherwise it equals 0.

Furthermore, we will need the following variables:

- $AR(t_i, t_j, w_k, r_l)$ : boolean variable that will equal 1 if referee $r_l$ is assigned to the role of main referee in the match between teams $t_i$ and $t_j$ played in the round $w_k$, where $i, j \in [1, n_t]$, $k \in [1, n_w]$ and $l \in [1, n_r]$, and will equal 0 otherwise.

- $AL(t_i, t_j, w_k, a_l)$ : boolean variable that will equal 1 if assistant referee $a_l$ is assigned to the role of linesman in the match between teams $t_i$ and $t_j$ played in the round $w_k$, where $i, j \in [1, n_t]$, $k \in [1, n_w]$ and $l \in [1, n_a]$, and will equal 0 otherwise.

- $A4(t_i, t_j, w_k, r_l)$ : boolean variable that will equal 1 if referee $r_l$ is assigned as the forth referee in the match between teams $t_i$ and $t_j$ played in the round $w_k$, where $i, j \in [1, n_t]$, $k \in [1, n_w]$ and $l \in [1, n_r]$, and will equal 0 otherwise.

- $AVAR(t_i, t_j, w_k, r_l)$ : boolean variable that will equal 1 if referee $r_l$ is assigned to the role of VAR in the match between teams $t_i$ and $t_j$ played in the round $w_k$, where $i, j \in [1, n_t]$, $k \in [1, n_w]$ and $l \in [1, n_r]$, and will equal 0 otherwise.

- $AAVAR(t_i, t_j, w_k, a_l)$ : boolean variable that will equal 1 if assistant referee $a_l$ is assigned to the role of AVAR in the match between teams $t_i$ and $t_j$ played in the round $w_k$, where $i, j \in [1, n_t]$, $k \in [1, n_w]$ and $l \in [1, n_a]$, and will equal 0 otherwise.

- $RWR(r_i, w_j)$ : boolean variable that will equal 1 if referee $r_i$ is assigned to any role in a match in round $w_j$, where $i \in [1, n_r]$, $j \in [1, n_w]$. Otherwise it will equal 0.

- $AWR(a_i, w_j)$ : boolean variable that will equal 1 if assistant referee $a_i$ is assigned to any role in a match in round $w_j$, where $i \in [1, n_a]$, $j \in [1, n_w]$. Otherwise it will equal 0.

- $MRWR(r_i, w_j)$ : boolean variable that will equal 1 if referee $r_i$ is assigned to a main role in a match in round $w_k$, where $i \in [1, n_r]$, $j \in [1, n_w]$. Otherwise it will equal 0.

- $MAWR(a_i, w_j)$ : boolean variable that will equal 1 if assistant referee $a_i$ is assigned to a main role in a match in round $w_j$, where $i \in [1, n_a]$, $j \in [1, n_w]$. Otherwise it will equal 0.

- $DMR(r_i, r_j)$ : boolean variable that will equal 1 if referee $r_i$ is better qualified than referee $r_j$ yet $r_i$ is assigned to fewer main roles, where $i, j \in [1, n_r]$. Otherwise it will equal 0.

- $DMA(a_i, a_j)$ : boolean variable that will equal 1 if assistant referee $a_i$ is better qualified than assistant referee $a_j$ yet $a_i$ is assigned to fewer main roles, where $i, j \in [1, n_a]$. Otherwise it will equal 0.

- $DPR(r_i, r_j)$ : boolean variable that will equal 1 if referee $r_i$ is better qualified than referee $r_j$ yet $r_i$ is assigned to less main roles in matches with higher punctuation, where $i, j \in [1, n_r]$. Otherwise it will equal 0.

- $DPA(a_i, a_j)$ : boolean variable that will equal 1 if assistant referee $a_i$ is better qualified than assistant referee $a_j$ yet $a_i$ is assigned to less main roles in matches with higher punctuation, where $i, j \in [1, n_a]$. Otherwise it will equal 0.

- $2G4DR(r_i, w_j)$ : boolean variable that will equal 1 if the match referee $r_i$ is assigned to a main role in round $w_j$ is played less than 4 days before the next game he or she is assigned to, with $i \in [1, n_r]$ and $j \in [1, n_w - 1]$. Otherwise it will be equal to 0.

- $2G4DA(a_i, w_j)$ : boolean variable that will equal 1 if the match assistant referee $a_i$ is assigned to a main role in round $w_j$ is played less than 4 days before the next game he or she is assigned to, with $i \in [1, n_a]$ and $j \in [1, n_w - 1]$. Otherwise it will be equal to 0.

The integer programming model we propose in order to minimize the number of soft constraints that are unfulfilled is described below and consists on minimizing the objective function subject to the constraints from (22) to (94). The objective function for this problem is described in (21) and, just like in the previous problem, uses the variables that only equal 1 if a soft constraint is broken and adds them up. In this case the variables that are taken into account are the last 6 described above, which are the variables *DMR* and *DMA*, that equal 1 if given two referees or assistants the one with bigger punctuation is assigned to fewer matches, the variables *DPR* and *DPA*, that equal 1 if for every pair of workers the one with better punctuation is assigned to less important matches, and the variables $2G4DR$ and $2G4DA$, that equal 1 if a referee or assistant referee is assigned to 2 games in 4 days, leaving them no time to rest.

$$\begin{aligned} min \Big( \sum_{i=1}^{n_r} \sum_{j=1}^{n_r} \Big( DMR(r_i, r_j) + DPR(r_i, r_j) \Big) + \sum_{i=1}^{n_r} \sum_{j=1}^{n_w} 2G4DR(r_i, w_j) + \\ + \sum_{i=1}^{n_a} \sum_{j=1}^{n_a} \Big( DMA(a_i, a_j) + DPA(a_i, a_j) \Big) + \sum_{i=1}^{n_a} \sum_{j=1}^{n_w} 2G4DA(a_i, w_j) \Big) \end{aligned} \tag{21}$$

The following 6 constraints define the number of referees that are to be assigned to every match per position. Constraints (22), (23) and (24) ensure each match has assigned exactly one main referee, two linesmen and one forth referee, constraints (25) and (26) make sure Eredivisie games have one referee in the role of VAR and one assistant referee in the role of AVAR, and constraint (27) asserts Eerste Divisie games have neither VAR nor AVAR assignments. With these constraints we fulfill the first three hard constraints described in 2.2.2.

$$\sum_{l=1}^{n_r} AR(t_i, t_j, w_k, r_l) = m(t_i, t_j, w_k) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w] \tag{22}$$

$$\sum_{l=1}^{n_a} AL(t_i, t_j, w_k, a_l) = 2 * m(t_i, t_j, w_k) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w] \tag{23}$$

$$\sum_{l=1}^{n_r} A4(t_i, t_j, w_k, r_l) = m(t_i, t_j, w_k) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w] \tag{24}$$

$$\sum_{l=1}^{n_r} AVAR(t_i, t_j, w_k, r_l) = m(t_i, t_j, w_k) \quad \forall i, j \in [1, n_{t1}], \forall k \in [1, n_w] \tag{25}$$

$$\sum_{l=1}^{n_a} AAVAR(t_i, t_j, w_k, a_l) = m(t_i, t_j, w_k) \quad \forall i, j \in [1, n_{t1}], \forall k \in [1, n_w] \tag{26}$$

$$\sum_{i=n_{t1}+1}^{n_t} \sum_{j=n_{t1}+1}^{n_t} \sum_{k=1}^{n_w} \Big( \sum_{l=1}^{n_r} AVAR(t_i, t_j, w_k, r_l) + \sum_{l=1}^{n_a} AAVAR(t_i, t_j, w_k, a_l) \Big) = 0 \tag{27}$$

With the next constraints we ensure the fulfillment of the forth, the fifth and the sixth hard constraints described above. Constraints (28) and (29) ensure nobody is assigned to two different roles in the same match and constraints (30) and (31) make sure no referee or assistant referee is assigned to two matches in a main role in the same round. Constraints (32) and (33) assert no referees or assistant referees are assigned to more than two games in the same round.

$$\begin{gathered} AR(t_i, t_j, w_k, r_l) + A4(t_i, t_j, w_k, r_l) + AVAR(t_i, t_j, w_k, r_l) <= 1 \\ \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \end{gathered} \tag{28}$$

$$AL(t_i, t_j, w_k, a_l) + AAVAR(t_i, t_j, w_k, a_l) <= 1 \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{29}$$

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} AR(t_i, t_j, w_k, r_l) <= 1 \quad \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{30}$$

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} AL(t_i, t_j, w_k, a_l) <= 1 \quad \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{31}$$

$$\begin{gathered} \sum_{i=1}^{n_t} \sum_{j=1}^{n_t} AR(t_i, t_j, w_k, r_l) + A4(t_i, t_j, w_k, r_l) + AVAR(t_i, t_j, w_k, r_l) <= 2 \\ \forall k \in [1, n_w], \forall l \in [1, n_r] \end{gathered} \tag{32}$$

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} AL(t_i, t_j, w_k, a_l) + AAVAR(t_i, t_j, w_k, a_l) <= 2 \quad \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{33}$$

Constraints from (34) to (45) are used to impose the incompatibilities between referees or assistant referees and teams or rounds, and the following 4 constraints, from (46) to (49), impose that defined refereeing trios must always go together in the main roles. This means that if the referee is assigned to the main refereeing role, the assistants have to be assigned as linesman, and if an assistant is assigned to a main role, the referee and the other assistant referee have to be assigned to the other main roles in the match.

$$RWR(r_i, w_k) <= 1 - irw(r_i, w_k) \quad \forall i \in [1, n_r], \forall k \in [1, n_w] \tag{34}$$

$$AWR(a_i, w_k) <= 1 - iaw(a_i, w_k) \quad \forall i \in [1, n_a], \forall k \in [1, n_w] \tag{35}$$

$$AR(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_i) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{36}$$

$$AR(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_j) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{37}$$

$$A4(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_i) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{38}$$

$$A4(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_j) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{39}$$

$$AVAR(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_i) \quad \forall i, j \in [1, n_{t_1}], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{40}$$

$$AVAR(t_i, t_j, w_k, r_l) <= 1 - irt(r_l, t_j) \quad \forall i, j \in [1, n_{t_1}], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{41}$$

$$AL(t_i, t_j, w_k, a_l) <= 1 - iat(a_l, t_i) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{42}$$

$$AL(t_i, t_j, w_k, a_l) <= 1 - iat(a_l, t_j) \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{43}$$

$$AAVAR(t_i, t_j, w_k, a_l) <= 1 - iat(a_l, t_i) \quad \forall i, j \in [1, n_{t_1}], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{44}$$

$$AAVAR(t_i, t_j, w_k, a_l) <= 1 - iat(a_l, t_j) \quad \forall i, j \in [1, n_{t_1}], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{45}$$

$$\begin{gathered} AR(t_i, t_j, w_k, r_l) + trio(r_l, a_m, a_n) - AL(t_i, t_j, w_k, a_m) <= 1 \\ \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r], \forall m, n \in [1, n_a] \end{gathered} \tag{46}$$

$$\begin{gathered} AR(t_i, t_j, w_k, r_l) + trio(r_l, a_m, a_n) - AL(t_i, t_j, w_k, a_n) <= 1 \\ \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r], \forall m, n \in [1, n_a] \end{gathered} \tag{47}$$

$$AL(t_i, t_j, w_k, a_m) + trio(r_l, a_m, a_n) - AR(t_i, t_j, w_k, r_l) <= 1 \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r], \forall m, n \in [1, n_a] \tag{48}$$

$$AL(t_i, t_j, w_k, a_n) + trio(r_l, a_m, a_n) - AR(t_i, t_j, w_k, r_l) <= 1 \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r], \forall m, n \in [1, n_a] \tag{49}$$

Constraints (50) and (51) ensure that for every interval of rounds referees and assistant referees are assigned to less that the given maximum of matches per interval of rounds, which is $maxM$, and constraints (52) and (53) impose that $nRT$ rounds must pass before somebody repeats an assignment in a main role with a team. The following 3 constraints, from (54) to (59), are used so that for every interval of $maxRL$ consecutive rounds all senior referees and assistant referees are assigned to at least one match with punctuation 0 or 1, one with punctuation 2 and another one with punctuation 3 or 4.

$$\sum_{k=0}^{nRI-1} MRWR(r_i, w_{j+k}) \leq maxM \quad \forall i \in [1, n_r], \forall j \in [1, n_w - nRI + 1] \tag{50}$$

$$\sum_{k=0}^{nRI-1} MAWR(a_i, w_{j+k}) \leq maxM \quad \forall i \in [1, n_a], \forall j \in [1, n_w - nRI + 1] \tag{51}$$

$$\sum_{t=0}^{nRT} \sum_{j=1}^{n_t} AR(t_i, t_j, w_{k+t}, r_l) + AR(t_j, t_i, w_{k+t}, r_l) \leq 1 \quad \forall i \in [1, n_t], \forall l \in [1, n_r], \forall k \in [1, n_w - nRT + 1] \tag{52}$$

$$\sum_{t=0}^{nRT} \sum_{j=1}^{n_t} AL(t_i, t_j, w_{k+t}, a_l) + AL(t_j, t_i, w_{k+t}, a_l) \leq 1 \quad \forall i \in [1, n_t], \forall l \in [1, n_a], \forall k \in [1, n_w - nRT + 1] \tag{53}$$

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} \sum_{k=0}^{maxRL-1} AR(t_i, t_j, w_{m+k}, r_l) \cdot (2 - mq(t_i, t_j)) \cdot (3 - mq(t_i, t_j)) \cdot (4 - mq(t_i, t_j)) \geq 1 \quad \forall l \in [1, n_r], \forall m \in [1, n_w - maxRL + 1] \tag{54}$$

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} \sum_{k=0}^{maxRL-1} AR(t_i, t_j, w_{m+k}, r_l) \cdot mq(t_i, t_j) \cdot (1 - mq(t_i, t_j)) \cdot (3 - mq(t_i, t_j)) \cdot \cdot (4 - mq(t_i, t_j)) \geq 1 \quad \forall l \in [1, n_r], \forall m \in [1, n_w - maxRL + 1] \tag{55}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=0}^{maxRL-1} AR(t_i,t_j,w_{m+k},r_l)\cdot mq(t_i,t_j)\cdot(1-mq(t_i,t_j))\cdot(2-mq(t_i,t_j))\geq 1 \quad \forall l\in[1,n_r],\forall m\in[1,n_w-maxRL+1] \tag{56}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=0}^{maxRL-1} AL(t_i,t_j,w_{m+k},a_l)\cdot(2-mq(t_i,t_j))\cdot(3-mq(t_i,t_j))\cdot(4-mq(t_i,t_j))\geq 1 \quad \forall l\in[1,n_a],\forall m\in[1,n_w-maxRL+1] \tag{57}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=0}^{maxRL-1} AL(t_i,t_j,w_{m+k},a_l)\cdot mq(t_i,t_j)\cdot(1-mq(t_i,t_j))\cdot(3-mq(t_i,t_j))\cdot \cdot(4-mq(t_i,t_j))\geq 1 \quad \forall l\in[1,n_a],\forall m\in[1,n_w-maxRL+1] \tag{58}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=0}^{maxRL-1} AL(t_i,t_j,w_{m+k},a_l)\cdot mq(t_i,t_j)\cdot(1-mq(t_i,t_j))\cdot(2-mq(t_i,t_j))\geq 1 \quad \forall l\in[1,n_a],\forall m\in[1,n_w-maxRL+1] \tag{59}$$

With all the constraints mentioned up until now, all hard constraints described in 2.2.2 up to the twelfth are fulfilled. To impose the remaining ones, that are the ones that impose the ratio of referees and assistant referees assigned to matches per role and classification, we use the constraints between (60) and (72). Constraints (60) and (61) ensure talententraject referees and assistant referees are never assigned to Eredivisie games and constraints from (62) to (67) make sure there will be 7 senior referees in the main role, 2 as forth referee and between 3 and 6 as VAR and 14 senior assistant referees as linesmen and at least 8 as AVAR per round in Eredivisie games. The following 5 constraints, from (68) to (72), impose the presence of 8 masterclass referees as main referees, between 8 and 14 masterclass assistant referees and at most 2 talententraject as linesmen, and ensures all forth referees are talententraject in Eerste Divisie rounds.

$$tr(r_l)*AR(t_i,t_j,w_k,r_l)=0 \quad \forall i,j\in[1,n_{t1}],\forall k\in[1,n_w],\forall l\in[1,n_r] \tag{60}$$

$$ta(a_l)*AL(t_i,t_j,w_k,a_l)=0 \quad \forall i,j\in[1,n_{t1}],\forall k\in[1,n_w],\forall l\in[1,n_a] \tag{61}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_r} AR(t_i,t_j,w_k,r_l)\cdot sr(r_l)=7 \quad \forall k\in[1,n_w] \tag{62}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_a} AL(t_i,t_j,w_k,a_l)\cdot sa(a_l)=14 \quad \forall k\in[1,n_w] \tag{63}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_r} A4(t_i,t_j,w_k,r_l)\cdot sr(r_l) = 2 \quad \forall k\in[1,n_w] \tag{64}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_r} AVAR(t_i,t_j,w_k,r_l)\cdot sr(r_l) \geq 3 \quad \forall k\in[1,n_w] \tag{65}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_r} AVAR(t_i,t_j,w_k,r_l)\cdot sr(r_l) \leq 6 \quad \forall k\in[1,n_w] \tag{66}$$

$$\sum_{i=1}^{n_{t1}}\sum_{j=1}^{n_{t1}}\sum_{l=1}^{n_a} AAVAR(t_i,t_j,w_k,a_l)\cdot sa(a_l) \geq 8 \quad \forall k\in[1,n_w] \tag{67}$$

$$\sum_{i=n_{t1}+1}^{n_t}\sum_{j=n_{t1}+1}^{n_t}\sum_{l=1}^{n_r} AR(t_i,t_j,w_k,r_l)\cdot mr(r_l) = 8 \quad \forall k\in[1,n_w] \tag{68}$$

$$\sum_{i=n_{t1}+1}^{n_t}\sum_{j=n_{t1}+1}^{n_t}\sum_{l=1}^{n_a} AL(t_i,t_j,w_k,a_l)\cdot ma(a_l) \geq 8 \quad \forall k\in[1,n_w] \tag{69}$$

$$\sum_{i=n_{t1}+1}^{n_t}\sum_{j=n_{t1}+1}^{n_t}\sum_{l=1}^{n_a} AL(t_i,t_j,w_k,a_l)\cdot ma(a_l) \leq 14 \quad \forall k\in[1,n_w] \tag{70}$$

$$\sum_{i=n_{t1}+1}^{n_t}\sum_{j=n_{t1}+1}^{n_t}\sum_{l=1}^{n_a} AL(t_i,t_j,w_k,a_l)\cdot ta(a_l) \leq 2 \quad \forall k\in[1,n_w] \tag{71}$$

$$\sum_{i=n_{t1}+1}^{n_t}\sum_{j=n_{t1}+1}^{n_t}\sum_{l=1}^{n_r} A4(t_i,t_j,w_k,r_l)\cdot tr(r_l) = 10 \quad \forall k\in[1,n_w] \tag{72}$$

The remaining constraints define all the variables. Constraints (73) and (74) define the variables RWR and AWR, constraints (75) and (76) define MRWR and MAWR, constraints (77) and (78) define the variables DMR and DMA, constraints (79) and (80) define DPR and DPA, and lastly, constraints (81) and (82) define the variables 2G4DR and 2G4DA.

$$\begin{gathered}\prod_{i=1}^{n_t}\prod_{j=1}^{n_t}(1-AR(t_i,t_j,w_k,r_l))\cdot(1-A4(t_i,t_j,w_k,r_l))\cdot(1-AVAR(t_i,t_j,w_k,r_l)) + RWR = 1 \\ \forall k\in[1,n_w],\forall l\in[1,n_r]\end{gathered} \tag{73}$$

$$\begin{gathered}\prod_{i=1}^{n_t}\prod_{j=1}^{n_t}(1-AL(t_i,t_j,w_k,a_l))\cdot(1-AAVAR(t_i,t_j,w_k,a_l)) + AWR = 1 \\ \forall k\in[1,n_w],\forall l\in[1,n_a]\end{gathered} \tag{74}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t} AR(t_i, t_j, w_k, r_l) = MRWR(r_l, w_k) \quad \forall l \in [1, n_r], \forall k \in [1, n_w] \tag{75}$$

$$\sum_{i=1}^{n_t}\sum_{j=1}^{n_t} AL(t_i, t_j, w_k, a_l) = MAWR(a_l, w_k) \quad \forall l \in [1, n_a], \forall k \in [1, n_w] \tag{76}$$

$$\begin{gathered}(rq(r_i) - rq(r_j)) \cdot \Big(\sum_{k=1}^{n_w} MRWR(r_i,w_k) - \sum_{k=1}^{n_w} MRWR(r_j, w_k)\Big) \cdot (1 - DMR(r_i, r_j)) >= 0 \\ \forall i, j \in [1, n_r], \forall k \in [1, n_w]\end{gathered} \tag{77}$$

$$\begin{gathered}(aq(a_i) - aq(a_j)) \cdot \Big(\sum_{k=1}^{n_w} MAWR(a_i,w_k) - \sum_{k=1}^{n_w} MAWR(a_j, w_k)\Big) \cdot (1 - DMA(a_i, a_j)) >= 0 \\ \forall i, j \in [1, n_a], \forall k \in [1, n_w]\end{gathered} \tag{78}$$

$$\begin{gathered}(rq(r_m) - rq(r_n)) \cdot \Bigg(\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=1}^{n_w} mq(t_i, t_j) \cdot \Big(AR(t_i, t_j, w_k, r_m) - AR(t_i, t_j, w_k, r_n)\Big)\Bigg) \cdot \\ \cdot (1 - DPR(r_m, r_n)) >= 0 \quad \forall m, n \in [1, n_r], \forall k \in [1, n_w]\end{gathered} \tag{79}$$

$$\begin{gathered}(aq(a_m) - aq(a_n)) \cdot \Bigg(\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\sum_{k=1}^{n_w} mq(t_i, t_j) \cdot \Big(AL(t_i, t_j, w_k, a_m) - AL(t_i, t_j, w_k, a_n)\Big)\Bigg) \cdot \\ \cdot (1 - DPA(a_m, a_n)) >= 0 \quad \forall m, n \in [1, n_a], \forall k \in [1, n_w]\end{gathered} \tag{80}$$

$$\begin{gathered}AR(t_{i_1}, t_{j_1}, w_k,r_l) \cdot AR(t_{i_2}, t_{j_2}, w_{k+1}, r_l) \cdot 4d(t_{i_1}, t_{j_1}, t_{i_2}, t_{j_2}, w_k) = 2G4DR(r_l, w_k) \\ \forall i_1, j_1, i_2, j_2 \in [1, n_t], \forall k \in [1, n_w - 1], \forall l \in [1, n_r]\end{gathered} \tag{81}$$

$$\begin{gathered}AL(t_{i_1}, t_{j_1}, w_k,a_l) \cdot AL(t_{i_2}, t_{j_2}, w_{k+1}, a_l) \cdot 4d(t_{i_1}, t_{j_1}, t_{i_2}, t_{j_2}, w_k) = 2G4DA(a_l, w_k) \\ \forall i_1, j_1, i_2, j_2 \in [1, n_t], \forall k \in [1, n_w - 1], \forall l \in [1, n_a]\end{gathered} \tag{82}$$

Finally, to impose the fact that all the variables are boolean, meaning they can either be 0 or 1, constraints from (83) to (97) are imposed.

$$AR(t_i, t_j, w_k, r_l) \in \{0, 1\} \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{83}$$

$$AL(t_i, t_j, w_k, a_l) \in \{0, 1\} \quad \forall i, j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{84}$$

$$A4(t_i, t_j, w_k, r_l) \in \{0,1\} \quad \forall i,j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{85}$$

$$AVAR(t_i, t_j, w_k, r_l) \in \{0,1\} \quad \forall i,j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_r] \tag{86}$$

$$AAVAR(t_i, t_j, w_k, a_l) \in \{0,1\} \quad \forall i,j \in [1, n_t], \forall k \in [1, n_w], \forall l \in [1, n_a] \tag{87}$$

$$RWR(r_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_r], \forall k \in [1, n_w] \tag{88}$$

$$AWR(a_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_a], \forall k \in [1, n_w] \tag{89}$$

$$MRWR(r_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_r], \forall k \in [1, n_w] \tag{90}$$

$$MAWR(a_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_a], \forall k \in [1, n_w] \tag{91}$$

$$DMR(r_i, r_j) \in \{0,1\} \quad \forall i,j \in [1, n_r] \tag{92}$$

$$DMA(a_i, a_j) \in \{0,1\} \quad \forall i,j \in [1, n_a] \tag{93}$$

$$DPR(r_i, r_j) \in \{0,1\} \quad \forall i,j \in [1, n_r] \tag{94}$$

$$DPA(a_i, a_j) \in \{0,1\} \quad \forall i,j \in [1, n_a] \tag{95}$$

$$2G4DR(r_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_r], \forall k \in [1, n_w] \tag{96}$$

$$2G4DA(a_i, w_k) \in \{0,1\} \quad \forall i \in [1, n_a], \forall k \in [1, n_w] \tag{97}$$

# 3. Complexity of the problem

In this section we are going to prove that the decision version of the Referee Assignment Problem, i.e., deciding whether the RAP has a solution or not, is NP-complete. To do so, we adapt a result published in 1987 by Esther M. Arkin and Ellen B. Silverberg [32] to a simplified version of the basic RAP we consider here. We first introduce the following problem:

**Problem 3.1.** *Job scheduling with fixed start and ending times*
*INPUT: A set $J = \{J_1, \dots, J_n\}$ of n jobs of equal value, the start and ending times $(s_i, t_i)$ of each job $J_i$ and a job-machine mapping between J and the set of k machines stating which machines can develop each job.*
*QUESTION: Is there an assignment of jobs to machines such that each machine is assigned to at most one job at a time and all jobs are processed?*

**Theorem 3.2.** *The decision version of the Referee Assignment Problem is NP-complete.*

*Proof.* To prove a problem is NP-complete, we have to see it is in NP, which is the set of decision problems that given a candidate solution can tell in polynomial time if it is indeed a solution, and that it is NP-hard, which means that it is at least as hard as the hardest problems in NP. To see a problem is NP-hard we have to prove it can be reduced in polynomial time to a problem that is NP-hard. Since if given an assignment of referees to matches, checking if this assignment is a feasible solution for the RAP can be done in polynomial time due to it being formulated as an integer lineal programming problem, it is obvious that the decision version of the RAP is in NP. To prove it is indeed NP-complete, we are going to use the article mentioned above and the simplified version of the basic RAP described up next.

For this simplified version of the basic RAP we will consider a refereeing assignment problem in which the minimum of matches per interval of rounds is 0 and the maximum is the number of rounds, referees can work in as many consecutive rounds as needed and can be assigned to two consecutive matches with a same team or stadium. To prove this version of the problem is NP-complete we are going to transform it into the job scheduling problem described in Problem 3.1, which is proved to be NP-complete in [32].

We will now see that, given a set $J = \{J_1, \dots, J_n\}$ of n jobs, the start and end times $(s_i, t_i)$ of each job $J_i$, $k$ machines with a set of jobs each can develop and an assignment of machines to the jobs so that each job is developed and each machine is assigned to at most one job at a time, we have a solution for the simplified version of the RAP mentioned above. To do so, we will consider $n$ to be the number of matches that take place in the league, being $J_i$ each of the matches, $k$ will be considered the number of referees we dispose of to do the assignments and $t_i$ and $s_i$ will be considered the round the matches are played and the next round respectively. Finally, we will consider the mapping between the jobs and the machines, a mapping between the referees and the matches indicating which matches can each referee officiate depending on the skills of the referee and the punctuation of the match and the incompatibilities.

Given the assignment of jobs to machines, as each machine can develop at most one job at a time, the referees will never be assigned to two matches at the same time, and as all jobs have a machine assigned to them, all matches will have a referee assigned. Moreover, since the machines are only assigned to the jobs they are mapped to, the referees will not be assigned to matches they are not able to officiate, fulfilling this way all the incompatibilities and the skill level requirements. Finally, as the minimum number of matches per interval of rounds considered for this problem is 0 and the maximum is the number of rounds and referees can be assigned to as many consecutive matches as needed and can repeat assignment to teams and stadiums in consecutive rounds, we can see that all the hard constraints imposed for this simplified version of the RAP are fulfilled and so we have a solution for the problem.

We will now see the opposite, that is that given an assignment of referees to the matches for this version of the RAP, we have a solution for the job scheduling problem. To do so we will consider one job for each of the matches with starting time the number of the round the match is played in and ending time the number of the following round and we will consider one machine for each of the referees, being $k$ the total number of referees. Finally, we will consider that the jobs can be assigned to the machines such that the match represented by the job can be assigned to the referee represented by the machine, which can be done if the referee does not have any incompatibility with any of the teams disputing the match, the stadium where the match is played in or the round it takes place in and the skill level of the referee is enough to officiate the match.

Given an assignment of referees to the matches, as each match has one referee, all the jobs would be assigned to one machine, and as each referee is assigned to one match per round and the jobs have starting and ending times that are identified with the rounds, each machine would be developing at most one job at a time. Finally, as the referees are never assigned to matches they cannot officiate, each job would be assigned to a machine that can develop it, giving us a solution for the job scheduling problem.

□

# 4. Local Search solution for the basic problem

To solve the basic problem with local search methods we have proposed the use of two of the most famous methods, Hill Climbing and Simulated Annealing, whose implementations are explained up next.

## 4.1 Local Search Algorithms

Local search algorithms are algorithms based on heuristic methods that are normally used to solve computationally hard optimization problems. To find the optimum solution for a problem, local search methods move through a space of candidate solutions from one solution to a neighbor applying one movement at a time until a local optimum is found or the time given to solve the problem is exhausted. The movements considered are mostly based on applying local changes to the last candidate solution contemplated or the best one found so far, and the method used to decide which movement to apply depends on the heuristic function and the algorithm that is being used. To apply local search algorithms we need an initial candidate solution or state from which the algorithm will start exploring the space of candidate solutions, a set of movements to be applied to move from one state to another neighborly one, and an heuristic function that will lead the algorithm towards the local optimum solution.

To solve the basic version of the RAP we have considered as a representation of the states a matrix $M$ with *nReferees* rows and *nMatches* columns, where *nReferees* is the number of referees available and *nMatches* is the number of matches that take place in the league. Each element in the matrix is a boolean value indicating the assignments, meaning a value equals 1 if and only if the referee represented by the row is assigned to the match represented by the column. The initial state is generated assigning to each match a random referee that has not been assigned to any other match played in the same round, this way we make sure our initial solution has exactly one referee assigned to each match and referees are assigned at most to one match per round. No further reasoning is applied behind the construction of the initial solution in order to avoid wasting computation time to get local optimums from the start, which would prevent most solvers from finding better solutions or moving to neighboring nodes to explore the space of candidate solutions, which is the reason the local search methods are used, making the solver return the initial solution no matter if it is a really bad one.

We have considered two different sets of movements: changing the referee assigned to a match and swapping the referees between two matches. The moves are only applied if the resulting matrix does indeed represent a correct state, meaning each match has exactly one referee assigned to them and each referee has at most one match per round. With the first movement we generate $\mathcal{O}(nReferee \cdot nMatches)$ neighbor states, and with the second one we generate $\mathcal{O}(nMatches^2)$, making a total of $\mathcal{O}(nReferee \cdot nMatches + nMatches^2)$ possible states the algorithm can go to from the current state. It is important to notice that with these movements we are able to explore the whole space of candidate solutions, so no other movements are needed.

Finally, as for the heuristic function, we have considered a function that adds up all the broken constraints by the candidate solution represented by the state, but weighting more the hard constraints and less the soft constraints in order to make sure the algorithm prefers states fulfilling all the hard constraints, since if not all hard constraints are fulfilled we cannot consider the result obtained a solution for our problem. The hard constraints that weight the most are the ones that ensure every match has one referee and referees are only assigned to one match per round, which are constraints that should never be broken taking into account the way we have defined the movements and how the initial state is generated: every time one of these constraints is broken, an additional cost of 1000000 is added to the heuristic value. For every incompatibility unfulfilled, every time a referee is assigned to more consecutive matches than is allowed or to more or less matches per interval of rounds than he should, the function adds a penalty of 30000 points, when a referee is assigned to a match demanding a bigger skill level than his, 20000 points are added to the function, and if a referee repeats assignment to a team or stadium sooner than is allowed, a penalty of 10000 points is added. Finally, broken soft constraints have a weight of 1 so that if a solution is found, the quality of the solution can be compared to the one obtained using complete solvers.

The reason why hard constraints have different weights in the heuristic function compared to each other is that, in case no solution is found, we want the resulting state to fulfill the most important constraints that give shape to the problem. An example explaining the reasoning applied to this would be that it is better to have a referee being assigned twice to the same team sooner than he should, which would not end up causing any troubles, than assigning him to a match he is not skilled enough to officiate, which can result in making the parties involved angry if bad decisions are taken by him due to inexperience or lack of good judgment, or assigning him to a match in a round in which he is not available, leaving the match without referee.

## 4.2 Hill Climbing

For the first method we have considered Hill Climbing, a search heuristic that uses a greedy approach and only moves to neighboring states that improve the value of the heuristic function with respect to the previous state. There are three basic Hill Climbing variants depending on how the next state is chosen: the first one selects the first neighboring state explored that has a better heuristic cost than the current state, the second one chooses a random neighboring state and then decides whether to go there or look for another state depending on the improvement gained with the value of the heuristic function, and the third one explores all the neighboring states and chooses the best.

Taking into account the size of the problem, which can be quite big, we have chosen to implement the first of the above-mentioned variants, which is usually referred to as Best First Hill Climbing. Another reasoning we have applied to choose this variant is that since the movements to be applied are generated randomly, with this variant we also dispose of the the random factor applied in the second method. Moreover, this first variant is faster than the third since it does not have to explore all the neighboring states before choosing one, and we do not have any guarantee that by exploring all the neighboring states and choosing the best the solution obtained at the end will be better.

The biggest problem with Hill Climbing algorithms is that, since they only go straight ahead towards solutions improving the results and never explore states with worse heuristic values, they get stuck in local optimums most of the time. To face this we have proposed a little variation in the algorithm so that once the algorithm gets stuck it can jump to a random state in the space of candidate solutions in order to explore a little bit more and see if a better solution can be found. Since we are not interested in exploring the whole space, which is the characteristic of the local search methods, the number of times the algorithm is able to jump to random states once it gets stuck is limited. The pseudo-code in Algorithm 1 shows the general scheme followed by the algorithm.

**Algorithm 1** Hill Climbing pseudo-code

```
state ← generateInitialState()
bestState ← state
for attempt ← 1 to maxAttempts do
    for iteration ← 1 to maxIterations do
        foundNextState ← false()
        for move ← 1 to maxMoves do
            nextState ← applyRandomMove(state)
            if heuristicValue(nextState) == 0 then return nextState
            else if heuristicValue(nextState) < heuristicValue(state) then
                state ← nextState
                foundNextState ← true
                break
            end if
        end for
        if heuristicValue(state) < heuristicValue(bestState) then
            bestState ← state
        else if not foundNextState then
            break
        end if
    end for
    state ← generateRandomState()
end for
return bestState
```

## 4.3 Simulated Annealing

Simulated Annealing is a search metaheuristic that uses probability techniques to approximate global optimization. The inspiration for this method comes from annealing in metallurgy, a technique based on heating and cooling materials in order to increase the crystals and reduce the defects of the materials.

To choose which state to move to, Simulated Annealing applies a random move to the current state and checks if the heuristic value in the resulting state is better than in the current state. If it is better, the new state is kept and the algorithm goes on from there, and if it is not, a random number between 0 and 1 and a number obtained from the acceptance probability function are compared and if the random number is smaller than the other one, the state is kept and the algorithm goes on from there. Otherwise the algorithm looks for another neighboring state. This acceptance probability function is what prevents the method from becoming stuck on local optimums as much as other methods such as Hill Climbing and depends on the heuristic values of both states and a time-varying parameter T referred to as the temperature. This function is usually chosen so that the probability of accepting a move decreases as the temperature decreases or the difference between the heuristic values from both states increases. As time goes by the temperature is decreased, making the acceptance probability slowly decrease and the method settle with a branch of the space of candidate solutions and stop jumping from state to state unless the solution is better.

Just like with the Hill Climbing method, we have implemented a variation that allows the algorithm to jump a limited amount of times to a random state in the space of candidate solutions once it gets stuck. In Simulated Annealing this can be considered a reheat since the temperate is increased again so that the method can start from scratch in another zone of the space of candidate solutions. The pseudo-code in Algorithm 2 shows the general scheme followed by the method implemented.

The results from applying both methods to solve the problem are explained and compared among them in Section 7, and the codes implementing these algorithms in C++ and are included in Appendix A.

**Algorithm 2** Simulated Annealing pseudo-code

```
state ← generateInitialState()
bestState ← state
cold ← false
T ← 0.5
beta ← 0.99
for reheat ← 1 to maxReheats do
    phase ← 1
    while not cold & phase < numPhasesPerReheat do
        move ← 1
        while not cold & move < numMovesPerPhase do
            nextState ← applyRandomMove(state)
            if heuristicValue(nextState) == 0 then return nextState
            else if heuristicValue(nextState) < heuristicValue(state) then
                state ← nextState
                if heuristicValue(state) < heuristicValue(bestState) then
                    bestState ← state
                end if
            else
                difCost ← heuristicValue(nextState) - heuristicValue(bestState)
                r ← randomNumberBetween(0,1)
                p ← exp(-diffCost/T)
                if r < p then
                    state ← nextState
                end if
            end if
            move ← move + 1
        end while
        phase ← phase + 1
        T ← T*beta
        if exp(−1/T) < 10e − 10 then
            cold ← true
        end if
    end while
    state ← generateRandomState()
    T ← 0.5
    cold ← false
end for
return bestState
```

# 5. ILP solution for the basic problem

To solve the basic version of the RAP through the use of complete solvers we have used the formulation as an integer linear programming mathematical problem. As mentioned before, complete solvers use methods that search the whole space of solutions, ensuring they find the optimum solution if there is one and they are given enough time. Moreover, some complete solvers can pinpoint the constraints that make the problem unsolvable facilitating the task of relaxing the constraints in some degree to find the most approximate solution.

In order to use the complete solvers we have developed a program in charge of generating the constraints in a format readable for the solvers given the data from the problem. This program has been developed using Prolog, a declarative programming language used in logic programming and mostly associated with artificial intelligence and computational linguistics that expresses the program logic in terms of relations, represented as facts and rules. In this section we will explain how the program works and will go over the format of the resulting file that is given to the complete solvers in order the face the problem. The complete code developed can be found in appendix B.

The basic idea of logic programming is to express data through relations, for example, $team(T, P)$ can express that team $T$ has a punctuation $P$. When writing in Prolog, this is used to create the programs and functions together with the fact that when a call fails or returns false, the program goes back to the last call made where he had different options to chose from and chooses another one if there are more or fails if there are no other options. To illustrate this we have the example in Listing 1, where we introduce three teams and their punctuations and want to get pairs of teams to form matches such that the sum of their punctuation is 8. When calling $match(X, Y)$, the values of $X$ and $Y$ are going to be $FCB$ and $ATH$, since the first pair of teams to match all the conditions demanded is going to be this one. The first pair of values $(X, Y)$ explored whenever this call is made is $(FCB, FCB)$, however this does not fulfill $X \backslash= Y$, so the last decision made, which is choosing team $Y$, is taken back and rethought and so $Y$ goes on to be $MAD$. This combination does not fulfill the last demand, which is that the punctuations must add up 8, so $Y$ is changed again, this time to $ATH$, creating a combination of teams that fulfills all the conditions. If the predicate from where $match(X, Y)$ is called were to fail, the following pair of teams that would be returned would be $(MAD, ATH)$, $(ATH, FCB)$ and $(ATH, MAD)$. As these are the only pairs of teams fulfilling the clause, if the call were to be made a fifth time, it would fail.

Listing 1: Prolog example

```
team(FCB,5).
team(MAD,5).
team(ATH,3).
match(X,Y):- team(X,PX), team(Y,PY), X \= Y, P is PX + PY, P = 8.
```

The main reason we have chosen to work with Prolog is that these pattern matching qualities, together with the fact that it works well with problems that involve objects and with rules or relations between them, makes this programming language specially well suited for constraint programming. Moreover, it is a language than can be easily read and understood and allows the user to declare the facts and rules that apply to those facts with great ease.

The program we have written uses the syntax shown in the example and works as follows: once executed, it reads the data for the problem from two files, one containing the calendar of the league with all the matches listed declaring the local team, the visitor and the round the match is played in, and the other containing the parameters of the problem, the referees and the teams participating in the league. Once this is done, the program writes the constraints and the objective function that define the problem into a file and sends it to the solvers, and finally, if the solvers find a solution, the solution is printed. The constraints are written through the use of 15 predicates that work similarly to the example above and are commented up next.

All the information needed for the problem is introduced as instances of predicates such as *match(teamA, teamB, round)* for the matches that take place in the league, *referee(refereeId, skill)* for the referees and *team(teamId, qualification)* for the teams (each parameter mentioned in Section 2.1.4 has a way of being introduced, which can be seen in Appendix B as mentioned before). Each of the above mentioned 15 predicates in charge of writing the constraints obtains the variables needed through the use of these instances and then writes the constraints for all the possible combinations of values using the syntax presented in the last example. Notice that the names of the variables and parameters used with the implementation of the constraints in Prolog do not match the names given in Section 2.1.4. This is because of the particularities of the language, that establish certain rules the names of the variables must follow, and because the syntax that has been used in each part of the project has been chosen to facilitate the development of the corresponding task.

In Listing 2 there is an example of how the predicates work with the predicate used to generate the constraints indicating that every match must have exactly 1 referee. Once *everyMatchHasAReferee* is called, for every match found, meaning each trio of values for the variables *S*, *T* and *R* that match with the call *match(S,T,R)*, the predicate calls the function *findall(assign(Ref,S,T,R), referee(Ref,_), Sum)*, that looks for all the variables *assign(Ref,S,T,R)*, that represent the assignment of the referee Ref to the match between *S* and *T* played in round *R*, and puts them in the list *Sum*. Up next a constraint imposing that the addition of all the variables in the list Sum must equal 1 is written and then the function fails and looks for the next match that has not been used yet. Line 4 is used to ensure the program does not fail, since after the last match is found the predicate fails. Whenever an underscore is used, it is expressing that we are not interested in the parameter in that position, for example, *referee(Ref,_)* gives us the identifier of the referee but does not care about the skill level assigned to that referee, and *match(_,_,1)* would match all the matches that take place in the first round.

Listing 2: Predicate ensuring one referee per match

```
1 everyMatchHasAReferee:-
2     match(S,T,R), findall(assign(Ref,S,T,R), referee(Ref,_), Sum),
3     writeConstraint(Sum = 1), fail.
4 everyMatchHasAReferee.
```

This predicate does exactly the same as equation 2 in the integer linear programming model proposed in Section 2.1.4, but only using the combination of variables representing teams and rounds that do really form a match, generating less constraints than have been needed for the model. The result from executing this line with the Spanish La Liga league data, for the match facing Athletic and Alavés played in the first round of the league would be the following constraint:

```
+ 1 assign(1,ath,leg,1) + 1 assign(2,ath,leg,1) + 1 assign(3,ath,leg,1)
```

```
+ 1 assign(4,ath,leg,1) + 1 assign(5,ath,leg,1) + 1 assign(6,ath,leg,1)
+ 1 assign(7,ath,leg,1) + 1 assign(8,ath,leg,1) + 1 assign(9,ath,leg,1)
+ 1 assign(10,ath,leg,1) + 1 assign(11,ath,leg,1) + 1 assign(12,ath,leg,1)
+ 1 assign(13,ath,leg,1) + 1 assign(14,ath,leg,1) + 1 assign(15,ath,leg,1)
+ 1 assign(16,ath,leg,1) + 1 assign(17,ath,leg,1) + 1 assign(18,ath,leg,1)
+ 1 assign(19,ath,leg,1) + 1 assign(20,ath,leg,1) =  1
```

The rest of the predicates work similarly to this one, so we will not explain them one by one. We will comment however, that some of the resulting constraints obtained with this program differ from the ones written for the model presented in Section 2.1.4, mostly due to them depending on other parameters, since when using Prolog we can filter first the variables in order to write only those constraints that are indispensable. For example, to impose referees cannot be assigned to games with greater difficulty that their skill level, using the predicate in Listing 3 the resulting constraint is simply an equality equation assigning 0 to the variable representing the assignment between the match and the referee, meanwhile in the model presented in section 2.1.4, the corresponding constraint, that is represented by equation (4), is more complex.

Listing 3: Predicate forbidding assignments of referees to matches with higher punctuation

```
refereeMinimumSkillLevelPerMatch:-
    referee(Ref,L), match(S,T,R), team(S,LS), team(T,LT), L < LS + LT,
    writeClause([-assign(Ref,S,T,R)],[]), fail.
refereeMinimumSkillLevelPerMatch.
```

Given the match between F.C.Barcelona and R.Madrid, that each have a punctuation of 5, and a referee with skill level 9 and id 7, the resulting constraint using the code shown above is the one that follows:

```
+ 1 assign(7,rma,bar,26) =  0
```

To solve this formulation of the problem with complete solvers we have used IBM ILOG CPLEX Optimization Studio, an optimization software package from IBM informally referred to simply as CPLEX. The results obtained are commented and compared altogether with the results from the local search methods in Section 7.

# 6. ILP solution for the KNVB problem

As the KNVB version of the problem has more than 200000 variables and a lot more constraints than the basic version of the problem, we have decided to just solve this version using complete solvers and not implement the whole structure needed to use local search methods, since it would take too long.

Formulating this version of the problem as an integer linear programming problem we have faced some challenges that have limited our options, the most important one being that, given the amount of data taken into account for this problem, several of the models we had first proposed made the computer run out of memory when trying to solve them. This is due to the fact that CPLEX needs an amount of memory space directly proportional to the number of variables declared in the problem, so the more variables used the less free space has the program to run. This has made it impossible for us to use those first models since the solver was not even able to start the execution and we have had to look for alternative formulations using less variables, which has meant using more complex constraints.

Similarly to how we have worked with the ILP solution for the basic problem, which is explained in Section 5, we have used a Prolog program to act as a bridge between the data for the problem and the solvers. Most of the program works just as before but facing a different problem: the data is read, the constraints are written and passed to the solvers, and if a solution is found, it is printed. With this problem, however, there are 32 predicates in charge of writing the constraints instead of 15 and there are intern variables created to facilitate the generation of the constraints for the problem. The Prolog code used to generate the constraints can be found in Appendix C.

The main difference in relation to the ILP solution for the basic problem, besides the differences grammatically-wise and the new constraints that are to be imposed, is that with this problem we offer the possibility of breaking the problem into smaller sub-problems to facilitate the task of finding solutions. The problem can be divided into smaller leagues with less rounds but with the same constraints by indicating the initial and ending rounds forming the interval of rounds taken into account for this smaller league. To take into account the results from the sub-problems and build the results towards the solution of the main problem we have written a code in C++ that takes the assignments from the previously solved sub-problems and writes them so they can be inserted together with the calendar and the data of the league as new data to Prolog execution of the next sub-problem that is faced. Doing it this way, we ensure that the global constraints can be taken into account, respecting this way all the constraints even if they affect intervals of rounds that are divided into different sub-problems.

The ideal usage of this feature would be solving the sub-problem for the first 10 rounds, for example, keeping the results, solving the problem for the following 10 rounds using the data from the previous assignments obtained before, and so on. When a solution is found using this method for a sub-problem having used data from previous sub-problems, the cost of the solution is the cost of the league composed by all the rounds included in any of the sub-problems previously solved and the newest sub-problem. This means that when the sub-problem containing the last rounds of a league is solved inserting the data from all the previous sub-problems, the cost obtained is the one corresponding to assignment of the referees to the matches from the whole league.

As mentioned in the introduction of this project, using this procedure makes solving the problem easier, however, when facing optimization problems, using this usually implies giving up finding the optimum solution since the solution of the whole problem is conditioned by the solutions of the sub-problems, which may be optimum for the sub-problems but that does not guarantee that the solution obtained by adding them up will be the optimum of the main problem. In fact, it does not even guarantee finding a feasible solution at all when using constraint satisfaction problems. The main reason to use this procedure to look for solutions is to reduce the computing time.

This feature can also be used whenever a referee is injured to recalculate the assignments taking into account that the referee will have new incompatibilities with the following rounds. This can also be used to change other parameters such as the skills of the referee if they are working better than expected or worse or the importance of a match. The assisting C++ program can also be used to impose assignments of referees to certain matches and positions beforehand, since this program reads data and transforms it into instances of predicates that are later on read and imposed.

The results from solving the problem with this method can be found in section 7 and the codes developed for this can be found in appendix C.

# 7. Experiments

In this section we present some results obtained applying the different resolution methods to different instances of the problem. The results shown below are obtained with a Toshiba Satellite P50 portable computer with an Intel CORE i7-4700MQ processor with 2.4GHz and 4 cores and 2 threads per core. The computation time results obtained may vary if the program is executed on a computer with different specs.

## 7.1 Basic Problem

In order to evaluate the different methods we have implemented to solve the basic problem, we have proposed different instances of the problem and compared the results obtained. The instances proposed have different sizes and demands in order to compare the results under different conditions. Since the cost of the solution increases as the number of soft constraints that are broken by the solution increases, the best results will be those with lower solution cost. To develop the experiments we have limited the jumps the local search methods are allowed to do to random states when they get stuck on a state to 5. In the proposed instances of the problem we use similar quantities of referees and teams, since we have observed this is a patron followed in several leagues, most likely to facilitate the assignments and the resting times of the referees.

When using CPLEX we have not let the program run indefinitely since we do not know how long it could take the solver to end the execution, so we have set a time limit for each execution that has deemed it possible to find a solution and lets the program explore the space of solutions. For the results obtained from using CPLEX we have taken into account the best solution that has been found in the computing time we have left the program run and the approximate time it has taken the solver to find this solution. We have also taken note of the gap between the cost of the best solution found and the best bound found so far, which is the cost of the best solution of the LP relaxation, ie., the problem without taking into account that the variables are integers, or boolean values in our case. When this gap is of 100% this means the best bound found so far is 0 and the solution found has a cost bigger than 0, and when the gap is of 0%, this means the best solution and the best bound have the same cost.

**Problem 1:**

For this instance of the problem, we consider a small league with 6 teams and 6 referees, meaning 30 matches will be disputed throughout the duration of the league separated into 10 rounds with 3 matches each. We want to look for an assignment such that for every 3 consecutive rounds, all referees are assigned to 1 or 2 matches, referees are never assigned to 3 consecutive matches and 2 rounds must pass before a referee repeats assignment with a team and 3 must pass before being assigned twice to the same stadium.

When applying Hill Climbing to this problem we get a solution breaking 40 soft constraints in just over 9 seconds and applying Simulated Annealing we get a solution breaking 38 soft constraints, but with a computation time much higher, of almost 125 seconds. When applying the complete solver to solve this problem, we find that CPLEX finds a solution breaking 39 soft constraints just after 10 seconds, however this solver is unable to find better solutions in under 1500 seconds.

**Problem 2:**

For this problem we consider a league with 10 teams, a few more than before, and 10 referees. This means the league is divided into 18 rounds with 5 matches each. Given this situation, we want to look for an assignment of referees to the matches such that for every 4 consecutive rounds, each referee is assigned to between 1 and 3 matches, no referee is assigned to matches in more than 3 consecutive rounds, and after being assigned to a game, referees must wait 2 rounds before repeating assignment with a team and 3 before being assigned to the same stadium.

Solving this problem with Hill Climbing we obtain a solution breaking 106 soft constraints with a computational time of almost 38 seconds, and with Simulated Annealing we get a solution that breaks 96 soft constraints, 10 less than the solution obtained with Hill Climbing, in over 404 seconds. With the complete solver, however, after letting CPLEX run for 1500 seconds, the best solution found breaks 164 soft constraints, which is a significant amount more than the broken by the solutions provided by the local search algorithms. The solution provided, however, has been found in under 35 seconds, which is faster than the computational time the local search solvers have needed.

**Problem 3:**

For this problem we use a real case: we consider the calendar of the Spanish La Liga league and their referees, meaning we dispose of 20 teams, 380 matches and 20 referees with different qualifications and incompatibilities. We want an assignment of referees such that referees are assigned to at most 5 consecutive matches and 3 and 5 rounds must go by before a referee repeats assignment with a same team or in a same stadium respectively. For this problem we also want the assignment to be so that for every set of 5 consecutive rounds, each referee is assigned to more than 2 matches and less than 4.

We have let this problem run for 7000 seconds with CPLEX and the best solution obtained, which is obtained in under 100 seconds, breaks 3079 soft constraints. Applying Hill Climbing we have gotten a result after more than 6000 seconds with an heuristic cost of 152784, and with Simulated Annealing it has taken a total computation time of almost 5250 seconds to get a result, and heuristic cost of the state provided as an answer is of 13128.

Taking into account that for this problem we have 20 teams and 20 referees, a total of 8400 soft constraints are considered, each with a cost of 1 in the heuristic function (see equation (1) in subsection 2.1.4). This means that if the total cost of the state is bigger than 8400, at least one hard constraint is broken and the state considered does not represent a solution, so neither of the local search methods that have been applied have been able to find a solution. This also means that the result provided by the Simulated Annealing method breaks one hard constraint and 3128 soft ones, meanwhile the result provided by Hill Climbing breaks between 5 and 15 hard constraints and 2784 soft ones.

**Problem 4:**

For this problem we also use the data from the La Liga league. In this problem, though, we want to see what happens when the constraints are relaxed a little bit. To do so we look for an assignment fulfilling the same demands as before except for the last one. For this problem we will consider intervals of 10 rounds instead of 5 and give a minimum of assignments per interval of rounds of 2, and a maximum of 8, which should be easier to fulfill.

Running this problem with CPLEX for 7000 seconds, we have found that, similarly to before, the solver finds a solution quite fast, in this case breaking 3116 soft constraints, and after finding this solution in unable to find better solutions. Using the local search methods we have found solutions with both methods. Hill Climbing has provided a solution breaking 2310 constraints in 2050 seconds and Simulated Annealing has taken over 4900 seconds to provide the solution, which has a cost of 2076.

**Problem 5:**

For this last problem we consider the same scenario as in Problem 3 but with referees being able to be assigned to any match. To do so we set all referee's skill levels to 10 and do not set any incompatibilities between referees and rounds, teams or stadiums.

Applying the local search methods we have found, once again, a solution with each of the solvers. With Hill Climbing the solver has taken 1085 seconds and has provided a solution with a cost of 588, using Simulated Annealing the solution is obtained after almost 2640 seconds and has a cost of 416. Applying the complete solver we have been met with a situation similar to the one found in the previous problems: the best solution found has a cost of 2975 and it has been found rather quickly, in under 200 seconds in this case, however the solver has not been able to find better solutions in the remaining time until the 4000 seconds the solver has as time limit have been spent.

The results obtained from applying the different solvers to these problems are documented in the tables 1 and 2 in the next page. The first table contains the results obtained with the local search methods documenting whether the result provided is a solution or not, the heuristic cost of the state returned and the computational time that has taken the solver to return an answer. The second table contains the information from the solutions provided by the complete solver. In this table we can find whether a solution has been found, the time we have let the solver run for, the cost of the solution found and how long it has taken the solver to find it and, finally, the best bound found by CPLEX once the time has run out and the gap between the best solution and the best bound.

With the data obtained through solving the instances of the problem with local search methods we can observe that the results obtained with Simulated Annealing are usually better than the ones obtained with Hill Climbing. We can also observe, though, that the computational time of the second method is quite higher in general, in fact Simulated Annealing normally takes at least twice as long to end and the only situation we have been able to observe in which Hill Climbing takes longer that Simulated Annealing is when no solution is found. We have not met any instance of the problem that has been solved with only one of the two proposed methods, for every instance of the problem we have tried we've either found a solution with both solvers or none at all. This brings us to the conclusion that if we are looking for better results, Simulated Annealing is the best of the two methods, and if we are looking for good results obtained quickly, we should go with Hill Climbing.

| | Hill Climbing | | | Simulated Annealing | | |
|---|---|---|---|---|---|---|
| **Problem** | **Has it found a solution?** | **Solution cost** | **Computation time (s)** | **Has it found a solution?** | **Solution cost** | **Computation time (s)** |
| **1** | yes | 40 | 9.02994 | yes | 38 | 124.984 |
| **2** | yes | 106 | 37.8191 | yes | 96 | 404.296 |
| **3** | no | 152784 | 6227.71 | no | 13128 | 5249.16 |
| **4** | yes | 2310 | 2050.17 | yes | 2076 | 4901.83 |
| **5** | yes | 588 | 1085.12 | yes | 416 | 2639.4 |

Table 1: Local Search methods results

Comparing these results to the ones obtained with the complete solver, we can see an example of the main difference between local search methods and complete solvers, which is that complete solvers always find the solution if there is one. This is exemplified with the third problem since the only solver that has been able to find a solution is CPLEX. We can also observe that CPLEX obtains good results rather quickly, but takes a lot longer to improve those. In fact, we have not been able to see improvements after the results shown in the table and have been met several times with a situation in which Hill Climbing is able to find better solutions in less time.

| Problem | Has it found a solution? | Solution cost | Time to find the solution (s) | Total computation time (s) | Best bound | Gap |
|---|---|---|---|---|---|---|
| **1** | yes | 39 | <13 | 1500 | 7.1791 | 81.59% |
| **2** | yes | 164 | <35 | 1500 | 6.8079 | 95.85% |
| **3** | yes | 3079 | <70 | 7000 | 254.7431 | 91.73% |
| **4** | yes | 3116 | <90 | 7000 | 255.9968 | 91.78% |
| **5** | yes | 2975 | <200 | 4000 | 0.0000 | 100% |

Table 2: Complete solver results

## 7.2 KNVB Problem

Since we do not have an implementation of any local search methods for this problem, we have only applied complete methods to solve it. We have, however, used the feature that allows us to divide the problem into smaller sub-problems that are easier to solve in order to compare the results obtained using different partitions to the result obtained without breaking the problem into sub-problems.

To compare the results obtained with the different partitions of the problem into sub-problems, since the whole league has a total of 41 rounds, we have proposed facing the whole league at once and partitioning the problem into 2, 4 and 8 sub-problems with similar distributions of rounds per instance of the sub-problem. The 4 methodologies are explained in detail up next together with the results obtained.

For each of the methods proposed we have studied two solutions, one obtained in a limited amount of time and the optimum one. For the first proposed solution we have considered the solutions that can be obtained with each of the methods using 8000 seconds to solve the whole problem. This will let us see the quality of the solutions that can be obtained through each method if the time we want to spend solving the problem is limited and compare them.

The data used for this problem has been provided by the Dutch Football Federation (KNVB) and corresponds to the season 2018 - 2019. A total of 47 referees and 71 assistant referees with different classifications and skill levels are taken into account and 41 rounds between matches of the two leagues are disputed. The assignment we have looked for is one such that referees are not assigned twice to matches with a same team in less than 3 rounds, for every 6 rounds referees are assigned to leading roles to at most 5 of them and they cannot develop main roles in matches for more than 4 consecutive rounds and all officers have to be assigned to leading roles in a match of each importance for every interval of 20 rounds.

**Method 1:**

For this first method we have broken the problem into 8 sub-problems, the seven firsts taking into account 5 consecutive rounds of the league each one and the last one considering the last 6. The sub-problems are faced in order and after solving each sub-problem the results obtained are included in the next sub-problem. Since we have a total amount of 8000 seconds to solve the whole problem to see the solution that can be obtained in this time we have let the solver run each sub-problem for 1000 seconds and taken note of the results obtained.

| Rounds considered | Cost | Time | Best Bound | Gap |
|---|---|---|---|---|
| **1 to 5** | 17 | 1000 | 0.0000 | 100% |
| **6 to 10** | 26 | 1000 | 0.0000 | 100% |
| **11 to 15** | 307 | 1000 | 0.0000 | 100% |
| **16 to 20** | 251 | 1000 | 12.7483 | 94.92% |
| **21 to 25** | 17 | 1000 | 7.8908 | 53.28% |
| **26 to 30** | 128 | 1000 | 10.1625 | 92.06% |
| **31 to 35** | 94 | 1000 | 52.6667 | 43.97% |
| **36 to 41** | 89 | 26.48 | 89.0000 | 0.00% |

Table 3: Results from breaking the KNVB problem into 8 sub-problems with limited time

The solution obtained for the whole problem after letting each sub-problem run for at most 1000 seconds has a cost of 89 and has been obtained after 7026.48 seconds of computing time. This means that at the end, once all the assignments have been made, the solution obtained for the whole problem breaks 89 soft constraints, which is surprisingly small number of constraints taking into account the size of the problem. With the results obtained from each sub-problem in the time limited version, which can be seen in Table 3, we can observe that with the first 3 sub-problems the solver is further away from finding the optimum solution or deciding the best one found so far is the optimum one since the gap is of 100%, however after the third sub-problem the gap starts to diminish, meaning the solver gets closer to finding the optimum solution. This is mainly due to the fact that after each sub-problem, the number of constraints limiting the space of solutions for the next sub-problem increases, and so the solver has less options to take into account. The most obvious expression of this is the last sub-problem, which is heavily conditioned by the results from the previous sub-problems and so the optimum is found in under 30 seconds, meanwhile the optimum has not been found for any of the other problems in the 1000 seconds each one has been left.

When looking for the optimum solution for each of the sub-problems we have run the problem into a situation that has no solution: after solving the first 6 sub-problems we have found that, taking into account the assignments from the previous rounds, there is no feasible assignment for the rounds from the 31st to the 35th. The results from executing the other sub-problems can be found in Table 4.

| Rounds considered | Cost | Time |
|---|---|---|
| **1 to 5** | 0 | 1240.90 |
| **6 to 10** | 9 | 3576.24 |
| **11 to 15** | 68 | 2257.10 |
| **16 to 20** | 112 | 8435.81 |
| **21 to 25** | 33 | 7851.03 |
| **26 to 30** | 39 | 9158.26 |
| **31 to 35** | - | - |
| **36 to 41** | - | - |

Table 4: Results from breaking the KNVB problem into 8 sub-problems looking for the optimum

With the results obtained looking for the optimum solutions we exemplify the fact commented in the introduction and once again in section 6, that is that breaking the problem into smaller sub-problems does neither assure finding the optimum solution nor finding a solution at all.

**Method 2:**

In this second method we have done as in method 1, but breaking the league into 3 smaller leagues of 10 rounds each and one last league with the last 11 rounds since the league has 41 rounds and one would be left alone otherwise. For the time limited solution we have let each sub-problem run for 2000 seconds and taken note of the results obtained.

| Rounds considered | Cost | Time | Best Bound | Gap |
|---|---|---|---|---|
| **1 to 10** | 3 | 2000 | 0.0000 | 100% |
| **11 to 20** | 509 | 2000 | 0.7333 | 99.86% |
| **21 to 30** | 893 | 2000 | 55.4036 | 93.80% |
| **31 to 41** | 157 | 2000 | 153.3599 | 2.32 % |

Table 5: Results from breaking the KNVB problem into 4 sub-problems with limited time

The results obtained after using CPLEX to solve the sub-problems can be seen in Table 5. In there we can see that the time limited version of the problem has been obtained in 8000 seconds, unlike the solution found using the first method that needed less time for the last sub-problem, and has a cost of 157, which is worse than the cost obtained with the first method in 8000 seconds.

Looking for the optimum solution for each of the sub-problems however, we have run into the same situation that we have met in the first method, that is that we have ended up with a sub-problem that cannot be solved when using the data from the previous assignments. This time this has happened with the third sub-problem, that looks for the assignment of referees to the matches from the 21st to the 30th rounds. In the table 6 we can observe that the cost of the solutions belonging to the first 2 sub-problems is better than the ones obtained limiting the computing time of the solver and that the number of seconds needed to find the optimum solutions is much higher than the 2000 seconds we had left for each sub-problem, meaning that we were quite far from finding the optimum solutions and that letting them run for longer periods of time, other solutions may have been found.

| Rounds considered | Cost | Time |
|---|---|---|
| **1 to 10** | 1 | 10527.76 |
| **11 to 20** | 98 | 15436.21 |
| **21 to 30** | - | - |
| **31 to 41** | - | - |

Table 6: Results from breaking the KNVB problem into 4 sub-problems looking for the optimum

**Method 3:**

For this third method we have partitioned the league into 2 smaller leagues of 20 and 21 rounds each and have let each of the sub-problems run for 4000 seconds in order to evaluate the solution obtained within 8000 seconds.

| Rounds considered | Cost | Time | Best Bound | Gap |
|---|---|---|---|---|
| **1 to 20** | 1559 | 4000 | 0.0000 | 100% |
| **21 to 41** | 1643 | 4000 | 122.1246 | 92.57% |

Table 7: Results from breaking the KNVB problem into 2 sub-problems with limited time

As we can see in the tables 7 and 8, which contain the data from the resolution of the sub-problems using CPLEX, limiting the solver so it works at most 4000 seconds per sub-problem we have obtained a solution with a cost of 1643, however without limiting the time the solver can work on each sub-problem we have obtained a solution with cost 57 after 41495.17 seconds, which is about 11 hours and a half. After seeing these results, it is obvious that letting the solver run for at most 4000 seconds with problems of this size is not enough since the solutions differ a lot, however, it is important to notice that we have gotten a solution in more than 5 times less computation time.

| Rounds considered | Cost | Time |
|---|---|---|
| **1 to 20** | 21 | 32579.34 |
| **21 to 41** | 57 | 8915.83 |

Table 8: Results from breaking the KNVB problem into 2 sub-problems looking for the optimum

**Method 4:**

In this last method we have considered the whole problem and looked for the optimum solution for the

KNVB problem. For the time limited solution, since there are no partitions of this problem into sub-problems, we have let the problem run for 8000 seconds.

| Rounds considered | Cost | Time | Best Bound | Gap |
|---|---|---|---|---|
| **1 to 41** | 5977 | 8000 | 0.0000 | 100% |

Table 9: Results from the KNVB problem with limited time

After letting the solver run for 8000 seconds, we have obtained a solution with a cost of 5977. However, when trying to solve the problem without limiting the time, we have not been able to find the optimum solution even after letting the solver run for 36 straight hours. After running CPLEX for 3 hours, the best solution found has a cost of 655, and after running it for 5, the best bound of the problem is set to 57, however the best cost and the best bound are not improved again. We can observe that the cost of the best bound found is the same as the cost of the solution obtained looking for the optimum solutions with method 3, which means that we have found the optimum solution of the problem with the 3rd method finding the optimum solution for each sub-problem.

After experimenting with the different methodologies we can corroborate that breaking the problem into smaller sub-problems and solving them individually does not guarantee finding the optimum solution or finding a solution at all since we have been met twice with a sub-problem that has no solution due to the solutions found for previous sub-problems.

We can also observe that, with this problem, the results obtained with a limited computing time and each of the different methods are better as the problem is divided into more sub-problems. This means that disposing of a limited time, the best of the 4 methods is the first one. In fact, the solution obtained with the first method in 8000s, which breaks only 89 soft constraints, is really close to the optimum solution, which breaks 57 soft constraints and we have only been able to find after running the solver for almost 41500 seconds, which is more than 5 times the time spent to find the first solution.

# 8. Conclusions and Further Work

In this project we have introduced the Referee Assignment Problem, a rather new problem in sports optimization that was presented in 2006, and done a short review of the most important problems in sport optimization. We have described and modeled two different versions of the RAP, a general one that can serve as a basis for an implementation with more detail for any football league, and another one with the specific details used by the Dutch Football Federation (KNVB) to assign their referees to the two highest professional football leagues in The Netherlands. We have also proved that the decision version of this problem is NP-complete.

We have implemented two local search methods to solve the basic version of the RAP with C++, the first one using Hill Climbing and the second one using Simulated Annealing, and seen with the results obtained through executing different problems with them that Hill Climbing is faster then Simulated Annealing but obtains worse results. We have also formulated the problem using integer linear programming with the help of a Prolog written program that automatically formulates the constraints given the problem's data and solved it with CPLEX, a complete solver created by IBM. Observing the results obtained solving the problem with the complete solver we have been able to see that local search solvers are faster, meaning they end the execution and find a local optimum solution faster, however complete solvers are always able to find the optimum solution if given enough time meanwhile local search algorithms sometimes are not even able to find a solution. We have also observed that with big problems with a huge amount of constraints and variables such as problems based on professional football leagues, CPLEX finds solutions with acceptable costs faster than the other solvers but then becomes stuck and takes a long time to find the optimum, prove the best solution found so far is the optimum or even find better solutions.

To face the KNVB version of the problem we have formulated it using integer linear programming with the help of another Prolog written program and solved it using CPLEX . We have compared the results obtained from breaking the problem into different quantities of smaller sub-problems and without breaking it and done so limiting the time given to the solver to find the optimum solution and without limiting it. Through the results obtained we have been able to see that looking for the optimum solution for the whole problem takes a really long time but breaking the problem into small sub-problems, if the problem is not directed towards a situation that has no solution once joining the results from previous sub-problems, can provide really good results in a short amount of time in comparison to what takes to solve the whole problem. Moreover, we have been able to observe that, for this instance of the problem, as the quantity of sub-problems the main problem is broken into increases, the results obtained in a given amount of time improve. We have also noticed that breaking the problem into sub-problems and looking for the optimum solution for each of them seems to elevate the chances of running the problem towards an unsolvable situation.

After seeing the results obtained from the experiments developed with the KNVB problem and seeing that solving the problem partitioning it into sub-problems may make the problem unsolvable, it is important to notice the fact that, if a referee gets injured and the assignment of referees for the league has to be recalculated for the following rounds during the medical leave, it is possible that no solution can be found. Given this situation, the only way to proceed to find a new assignment is to change the parameters of the problem that give shape to the hard constraints, altering the ones that make it impossible to find a feasible assignment.

One thing that is really important from the results obtained is that breaking the problem into two sub-problems we have been able to find the optimum solution for the whole problem in over 11 hours, meanwhile it takes days to find the optimum solution for the whole problem without partitioning. It could be interesting to see if this has been a coincidence or if changing the parameters of the problem the same would happen. It could also be interesting to see if there is a relation between the number of sub-problems the problem in broken into an the number of soft constraints broken by the solutions obtained from each one with different time limits and different instances of the problem.

Seeing as the local search methods have obtained rather good results in short amounts of time in comparison to the time needed for complete solvers to find the optimum solution of a problem with the basic version of the problem, it could be interesting to implement a local search solver for the KNVB version of the problem to compare the results, since using this type of solvers could shorten the computing time, although there is also the possibility that local search solvers would not be able to find a solution given the size of the problem.

It could also be interesting to find a way to run at the same time a complete search method and a local search algorithm coordinating them in order to let the local search algorithms know the best bound found so far. This could help a lot since as local search algorithms do not know if they are dealing with the global optimum, this could help them know if they have found it. Moreover, combining both methods and coordinating them, we could get a method that gets the solutions quite fast, which is a quality we have observed with the local search algorithms, and we would be able to know when we have found the global optimum or how far we are from it cost-wise, which is a property from the complete solvers.

As a last remark, we want to comment the fact that with another computer the results obtained could have been different in several ways. The first thing we want to pay notice to is that using a computer with better computing performance, the running times would have been smaller and we could have found the optimum solution for the whole KNVB problem in a reasonable amount of time. This would have let us get more data which we could have used to make more and better comparisons and more experiments could have been developed given the same amount of time. The second thing we want to comment is the fact that using a computer with more available memory space we could have used some of the previously formulated integer linear programming models for the KNVB version of the problem, which we have not been able to run in this computer since CPLEX runs out of memory when formulating them. These previous versions of the program used more internal variables in order to avoid recalculating things, which would have meant using less constraints and reusing already calculated variables or parameters, reducing the computing time.

# References


[1] A. Duarte, C. Ribeiro, S. Urrutia, and E. Haeusler, "Referee assignment in sports leagues," vol. 3867, pp. 158–173, 01 2006.

[2] J. R. Evans, "Play ball!–the scheduling of sports officials," *Perspectives in Computing: Applications in the Academic and Scientific Community*, 01 1984.

[3] J. R. Evans, "A microcomputer-based decision support system for scheduling umpires in the american baseball league," *Interfaces*, vol. 18, pp. 42–51, 12 1988.

[4] M. Wright, "Scheduling english cricket umpires," *Journal of The Operational Research Society - J OPER RES SOC*, vol. 42, pp. 447–452, 06 1991.

[5] M. B. Wright, *Scheduling English Cricket Umpires*, pp. 87–96. 01 2015.

[6] A. Duarte, C. Ribeiro, and S. Urrutia, "A hybrid ils heuristic to the referee assignment problem with an embedded mip strategy," vol. 4771, pp. 82–95, 10 2007.

[7] G. Durán and M. Guajardo, "Programación matemática aplicada al fixture de la primera división del fútbol chileno," *Revista Ingeniera de Sistemas*, vol. 19, p. 29 – 48, 2005.

[8] G. M. Durán, G. and R. Wolf-Yadlin, "Programación del fixture de la segunda división del fútbol de chile mediante investigación de operaciones," *Revista Ingeniera de Sistemas*, vol. 24, p. 27 – 46, 2010.

[9] T. Atan and O. Pelin Hüseyinoğlu, "Simultaneous scheduling of football games and referees using turkish league data," *International Transactions in Operational Research*, vol. 24, 09 2015.

[10] M. Yavuz, U. Inan, and A. Fığlalı, "Fair referee assignments for professional football leagues," *Computers OR*, vol. 35, pp. 2937–2951, 09 2008.

[11] A. Lamghari and J. A. Ferland, "Structured neighborhood tabu search for assigning judges to competitions," pp. 238–245, 05 2007.

[12] A. Lamghari and J. A. Ferland, "Metaheuristic methods based on tabu search for assigning judges to competitions," *Annals OR*, vol. 180, pp. 33–61, 11 2010.

[13] F. Alarcon, G. Durán, and M. Guajardo, "Referee assignment in the chilean football league using integer programming and patterns," *International Transactions in Operational Research*, vol. 21, 10 2013.

[14] R. Linfati, G. Gatica, and J. Escobar, "A flexible mathematical model for the planning and designing of a sporting fixture by considering the assignment of referees," *International Journal of Industrial Engineering Computations*, vol. 10, pp. 281–294, 04 2019.

[15] B. L. Schwartz, "Possible winners in partially completed tournaments," *Siam Review - SIAM REV*, vol. 8, pp. 302–308, 07 1966.

[16] A. Hoffman and J. Rivlin, "When is a team "mathematically" eliminated?," *Proceedings of the Princeton Symposium on Mathematical Programming*, pp. 391–401, 01 1970.

[17] L. W. Robinson, "Baseball playoff eliminations: An application of linear programming," *Operations Research Letters - ORL*, vol. 10, pp. 67–74, 10 1991.

[18] D. de Werra, "Scheduling in sports," *Annals of Discrete Mathematics*, vol. 11, 12 1981.

[19] D. de Werra, "Werra, d.: Some models of graphs for scheduling sports competitions. discrete appl. math. 21, 47-65," *Discrete Applied Mathematics*, vol. 21, pp. 47–65, 09 1988.

[20] D. de Werra, L. Jacot-Descombes, and P. Masson, "A constrained sports scheduling problem," *Discrete Applied Mathematics*, vol. 26, pp. 41–49, 01 1990.

[21] K. Mcaloon, C. Tretkoff, and G. Wetzel, "Sports league scheduling," 11 1997.

[22] F. Biajoli, A. Chaves, M. Souza, and F. Souza, "Escala de jogos de torneios esportivos: Uma abordagem via simulated annealing," 11 2003.

[23] P. Van Hentenryck and Y. Vergados, "Minimizing breaks in sport scheduling with local search.," pp. 22–29, 01 2005.

[24] G. Nemhauser and M. Trick, "Scheduling a major college basketball conference," *Operations Research*, vol. 46, 10 2001.

[25] M. Henz, T. Müller, and S. Thiel, "Global constraints for round robin tournament scheduling," *European Journal of Operational Research*, vol. 153, pp. 92–101, 02 2004.

[26] K. Easton, G. Nemhauser, and M. Trick, "Solving the travelling tournament problem: A combined integer programming and constraint programming approach," pp. 100–112, 08 2002.

[27] D. C. Uthus, P. Riddle, and H. Guesgen, "Dfs* and the traveling tournament problem," vol. 5547, pp. 279–293, 05 2009.

[28] G. Langford, "An improved neighbourhood for the traveling tournament problem," 07 2010.

[29] R. Miyashiro, T. Matsui, and S. Imahori, "An approximation algorithm for the traveling tournament problem," *Annals of Operations Research*, vol. 194, pp. 317–324, 01 2012.

[30] T. Benoist, F. Laburthe, and B. Rottembourg, "Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems," 01 2001.

[31] M. Adriaen, N. Custers, and G. Vanden Berghe, "An agent based metaheuristic for the travelling tournament problem," 06 2019.

[32] E. M. Arkin and E. B. Silverberg, "Scheduling with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, pp. 1–8, 09 1987.

# A. Local Search Code

In this section of the appendix is found the code used for the implementations of local search methods, which, as mentioned before, is written in C++. The code is structured in several files that work together using object oriented programming and needs as input two files with the data from the problem. Up next we will show each of the files and explain what they are used for.

The first files are the files containing the data of the problem we want to study. Two files are needed to enter the data: one containing the parameters of the problem and another one with the matches. The first has to include the number of referees and matches, the rounds contained in one interval of rounds, the minimum and maximum of matches per interval of rounds, the maximum number of consecutive rounds a referee can officiate a match in, the number of rounds before a referee is assigned twice to the same team or stadium and the teams and the referees and their qualifications. The second file simply has to list the matches identificating the teams playing against each other and in which round the game is played. The files must follow the syntax of the files from Listing 4 and Listing 5, which can be used as examples. Following the standards used by Barcelogic to create the calendars for the KNVB league, the code has been implemented in a way such that the names of the teams must be 3 characters long, which makes the job of extracting the data from the files easier.

The code is structured in 5 classes that are used by the main file to solve the problem. It is structured this way so that parts of the code that independent from one another can be modified without affecting other parts of the code. For example, one file contains the implementation of the state and another one implements the heuristic function, this way, if we have to modify the definition of the states, we can do it without having to touch the files containing the heuristic function and making sure this way that nothing that should not be touched is touched by accident. Each of these classes is broken into 2 files, one ended in .hh, which contains the definitions or calls of the functions and the class, and another one ended in .cc, which contains the implementations of the functions declared in the other file, but both with the same name besides the termination.

The first class that is used is the Reader class, which is used to read and process the data from the previously mentioned files and uses the library *fstream* to open the files and read from them. This class can be found in the Listings 6 and 7.

The most important things to define when using local search methods are the definition of the state and how the data is kept, the heuristic function and the method or algorithm that is used and the moves that can be generated. These things are all defined in the classes State, HeuristicFunction and Solver respectively.

The class State can be found in Listings 8 and 9 and contains the structure of the state, which is the above-mentioned matrix of size the number of referees times the number of matches, and is in charge of modifying the assignments of referees and matches. This class has the method that generates random initial states that ensures the state has one referee per match and referees have at most one match per round.

The heuristic function can be found implemented in the class HeuristicFunction. This class basically implements a function that, given a state, calculates the number of constraints that are broken, gives them a weight depending on the type of constraint and their importance for the integrity of the result, and returns the addition of these numbers, which is interpreted as the heuristic value of the states. This class is presented in the Listings 10 and 11.

The last important class left to present is the one that makes most of the work, which is the class Solver. This class contains the implementation of the methods used for the Hill Climbing and the Simulated Annealing and a procedure that generates random moves that can be applied to the current state. The implementation of the methods basically follows the pseudo-codes in the Algorithms 1 and 2 in Section 4. This class continually calls the classes State and HeuristicFunction in order to check which movements can be made and to evaluate the value of the heuristic function, and its implementation is presented in Listings 12 and 13.

Finally, to write the solutions obtained after applying the algorithms we have the class Writer, implemented in the Listings 14 and 15, that creates a file $solutionHC.txt$ or $solutionSA.txt$ depending on if it is writing the solution obtained with Hill Climbing or Simulated Annealing, and writes there the results in a format easily read. An example of how the results are written can be found in Listings 18 and 19, which contain the results from applying Hill Climbing and Simulated Annealing respectively with the files in Listings 4 and 5 as the data and calendar files.

All these clases are used from the main file, which can be found in Listing 16. Once the main file is executed, the Reader class is called to read the data from the files, and the data obtained is kept in variables for future uses. Then an instance of the class Solver, an instance of the class HeuristicFunction, and an instance from the class Writer are created with the data from before, and, with these instances, the solver is called in order to solve the problem with Hill Climbing and then with Simulated Annealing and the results are written in the corresponding files. The whole code can be compiled using the Makefile in Listing 17 with the call "make" in an Ubuntu terminal, and the program can be executed with the call "$./main.exe < calendarFile >< dataFile >$", where $< calendarFile >$ is the file with the calendar of matches, and $< dataFile >$ is the file with the specifics of the problem, the teams and the referees.

Listing 4: Example of file with the data for the problem

```
nReferees 6
nMatches 30
intervalRounds 2
minMatchesPerInterval 0
maxMatchesPerInterval 4
minNumRoundsBeforeRepeatingTeam 0
minNumRoundsBeforeRepeatingStadium 1
maxConsecutiveRounds 3

referee(1,7).
referee(2,8).
referee(3,8).
referee(4,9).
referee(5,7).
referee(6,8).

team(te1,3).
team(te2,2).
team(te3,4).
team(te4,3).
team(te5,2).
team(te6,1).
```

Listing 5: Example of file with the matches of the league

```
match(te1,te2,1).
match(te3,te4,1).
match(te5,te6,1).

match(te3,te1,2).
match(te2,te5,2).
match(te4,te6,2).

match(te1,te4,3).
match(te5,te3,3).
match(te6,te2,3).

match(te5,te1,4).
match(te4,te2,4).
match(te3,te6,4).

match(te1,te6,5).
match(te2,te3,5).
match(te4,te5,5).

match(te2,te1,6).
match(te4,te3,6).
match(te6,te5,6).

match(te1,te3,7).
match(te5,te2,7).
```

```
match(te6,te4,7).

match(te1,te4,8).
match(te3,te5,8).
match(te2,te6,8).

match(te1,te5,9).
match(te2,te4,9).
match(te6,te3,9).

match(te6,te1,10).
match(te3,te2,10).
match(te5,te4,10).
```

Listing 6: Reader.hh

```
#ifndef READER_HH
#define READER_HH

#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <vector>
#include <cmath>
using namespace std;

class Reader {

protected:

    ifstream inFile; // value indicating the last read char from a file
    string dataFile; // name of the file that has the data for the problem
    string calendarFile; // name of the file that has the calendar
    int nTeams; // number of teams
    int nRef; // number of referees
    int nInc; // number of incompatibilities

    /* Reads the data from the calendarFile and writes it in the map */
    void readCalendar(map<int,string>& listOfMatches);

    /* Reads the data about hte referees and inserts it into the map*/
    void readReferee(string input, int nReferees, map<int,int>& listOfReferees);

    /* Reads the data about the teams and writes in into the map*/
    void readTeam(string input, int nTeamsMax, map<string,int>& listOfTeams);

    /* Read the data about the incompatibilities and writes it in the vector*/
    void readIncompatibility(string input,
        vector<vector<string>>& listOfIncompatibilities);

    /* Reads the parameters needed for the problem and inserts it into a
    vector that is returned*/
    vector<int> readParticulars();

    /* Reads the data from the files and insert it into the corresponding
    maps and vectors*/
    vector<int> readData(map<string,int>& listOfTeams, map<int,int>&
        listOfReferees, vector<vector<string>>& listOfIncompatibilities);

public:

    // Default constructor
    Reader();

    // Constructor with the names of the files
    Reader(string calendarFile, string dataFile);
```

```
    // Destructor
    ~Reader();

    /* Given empty maps, reads the data from the calendarFile and the dataFile,
    inserts it into the corresponding maps and vectors, and returns a vector
    with the parameters of the problem*/
    vector<int> read(map<int,string>& listOfMatches, map<string,int>&
        ↪ listOfTeams,
        map<int,int>& listOfReferees, vector<vector<string>>&
        listOfIncompatibilities);

};

#endif
```

Listing 7: Reader.cc

```

#include "Reader.hh"

Reader::Reader() {
    this->dataFile = "";
    this->calendarFile = "";
}

Reader::Reader(string calendarFile, string dataFile) {
    this->dataFile = dataFile;
    this->calendarFile = calendarFile;
}

Reader::~Reader() {}


void error(string message) {
    cout << message << endl;
    exit(1);
}


void Reader::readCalendar(map<int,string>& listOfMatches) {
    ifstream inFile;
    inFile.open(this->calendarFile);
    if (!inFile) error("Unable to open the file " + this->calendarFile);
    string input, loc, vis, round, def;
    int i = 0;
    while (inFile >> input) {
        loc = input.substr(6,3);
        vis = input.substr(10,3);
        if (i < 100) round = input.substr(14,1);
        else round = input.substr(14,2);
        def = loc + " " + vis + " " + round;
        listOfMatches.insert(pair<int,string>(i,def));
```

```
            ++i;
        }
}

void Reader::readReferee(string input, int nReferees,
    map<int,int>& listOfReferees) {
    if (this->nRef >= nReferees)
        error("There are more referees than accounted for");
    string ref;
    string points;
    if (nRef < 9) {
        ref = input.substr(8,1);
        if (input.substr(11,1) == ")") points = input.substr(10,1);
        else points = input.substr(10,2);
    }
    else {
        ref = input.substr(8,2);
        if (input.substr(12,1) == ")") points = input.substr(11,1);
        else points = input.substr(11,2);
    }
    listOfReferees.insert(pair<int,int>(stoi(ref),stoi(points)));
    ++this->nRef;
}

void Reader::readTeam(string input, int nTeamsMax,
    map<string,int>& listOfTeams) {
    if (this->nTeams >= nTeamsMax)
        error("There are more teams than there should");
    string team = input.substr(5,3);
    string points = input.substr(9,1);
    listOfTeams.insert(pair<string,int>(team,stoi(points)));
    ++this->nTeams;
}

void Reader::readIncompatibility(string input,
    vector<vector<string>>& listOfIncompatibilities) {
    string sub = input.substr(0,7);
    if (sub == "incRefT") {
        string ref;
        string team;
        if (input.substr(12,1) == ",") {
            ref = input.substr(11,1);
            team = input.substr(13,3);
        }
        else {
            ref = input.substr(11,2);
            team = input.substr(14,3);
        }
        listOfIncompatibilities.push_back({"T",ref,team});
    }
    else if (sub == "incRefS") {
        string ref;
```

```
        string team;
        if (input.substr(12,1) == ",") {
            ref = input.substr(11,1);
            team = input.substr(13,3);
        }
        else {
            ref = input.substr(11,2);
            team = input.substr(14,3);
        }
        listOfIncompatibilities.push_back({"S",ref,team});
    }
    else if (sub == "incRefR") {
        string ref;
        string round;
        if (input.substr(13,1) == ",") {
            ref = input.substr(12,1);
            if (input.substr(15,1) == ")") round = input.substr(14,1);
            else round = input.substr(14,2);
        }
        else {
            ref = input.substr(12,2);
            if (input.substr(16,1) == ")") round = input.substr(15,1);
            else round = input.substr(15,2);
        }
        listOfIncompatibilities.push_back({"R",ref,round});
    }
    else error("Incompatibility not contemplated");
}

vector<int> Reader::readParticulars() {
    vector<int> v = vector<int>(8);
    string input;
    int value;
    this->inFile >> input >> value;
    if (input != "nReferees")
        error("nReferees missing from " + this->dataFile);
    v[0] = value;
    this->inFile >> input >> value;
    if (input != "nMatches")
        error("nMatches missing from " + this->dataFile);
    v[1] = value;
    this->inFile >> input >> value;
    if (input != "intervalRounds")
        error("intervalRounds missing from " + this->dataFile);
    v[2] = value;
    this->inFile >> input >> value;
    if (input != "minMatchesPerInterval")
        error("minMatchesPerInterval missing from " + this->dataFile);
    v[3] = value;
    this->inFile >> input >> value;
    if (input != "maxMatchesPerInterval")
        error("maxMatchesPerInterval missing from " + this->dataFile);
```

```
    v[4] = value;
    this->inFile >> input >> value;
    if (input != "minNumRoundsBeforeRepeatingTeam")
        error("minNumRoundsBeforeRepeatingTeam missing from " + this->dataFile);
    v[5] = value;
    this->inFile >> input >> value;
    if (input != "minNumRoundsBeforeRepeatingStadium")
        error("minNumRoundsBeforeRepeatingStadium missing from " +
            this->dataFile);
    v[6] = value;
    this->inFile >> input >> value;
    if (input != "maxConsecutiveRounds")
        error("maxConsecutiveRounds missing from " + this->dataFile);
    v[7] = value;
    return v;
}

vector<int> Reader::readData(map<string,int>& listOfTeams,
    map<int,int>& listOfReferees, vector<vector<string>>&
    listOfIncompatibilities) {
    this->inFile;
    inFile.open(this->dataFile);
    if(!inFile) error("Unable to open the file " + this->dataFile);

    vector<int> v = readParticulars();

    string input;
    this->nRef = 0;
    this->nTeams = 0;
    int nTeamsMax = (1+sqrt(1+4*v[1]))/2;
    while (inFile >> input) {
        string sub = input.substr(0,3);
        if (sub == "ref") readReferee(input, v[0], listOfReferees);
        else if (sub == "tea") readTeam(input, nTeamsMax, listOfTeams);
        else if (sub == "inc") readIncompatibility(input,
            listOfIncompatibilities);
        else error("File " + this->dataFile + " is not written in a readable" +
            " format for this program");
    }
    return v;
}


vector<int> Reader::read(map<int,string>& listOfMatches,
    map<string,int>& listOfTeams, map<int,int>& listOfReferees,
    vector<vector<string>>& listOfIncompatibilities) {
    this->readCalendar(listOfMatches);
    vector<int> v = this->readData(listOfTeams, listOfReferees,
        listOfIncompatibilities);
    return v;
}
```

Listing 8: State.hh

```
#ifndef STATE_HH
#define STATE_HH

#include <iostream>
#include <vector>
using namespace std;

class State {

protected:

    int nReferees; // number of referees
    int nMatches; // number of matches
    vector<vector<bool>> M; // boolean matrix containing the assignments

public:

    // Default constructor
    State();

    // Construcor with the number of referees and matches
    State(int nReferees, int nMatches);

    // Destructor
    ~State();

    // returns the number of referees
    int getNReferees();

    // returns the number of matches
    int getNMatches();

    // sets the number of referees
    void setNReferees(int nReferees);

    // sets the number of matches
    void setNMatches(int nMatches);

    /* given a referee and a match, returns true if the referee is assigned
    to the match, and false otherwise */
    bool isAssigned(int referee, int match);

    /* given a referee and a match, assignes to the corresponding place in
    the matrix, the boolen isAssigned*/
    void setAssignment(int referee, int match, bool isAssigned);

    /* given a referee and a match, returns true if the referee is already
    assigned to one match in the same round the match is played in */
    bool refereeHasMatchThisRound(int ref, int match);

    /* generates randomly an assignment of referees to the matches making
```

```
    sure each match has exactly one referee and referees are assogned to
    one match per round at most*/
    void generateInitialState();

    /* given a match, returns the referee assigned to it*/
    int getRefereeOfTheMatch(int match);

    /* prints the state*/
    void printState();

};

#endif
```

Listing 9: State.cc

```

#include "State.hh"

State::State() {
    nReferees = 0;
    nMatches = 0;
    M = vector<vector<bool> > (0, vector<bool> (0));
}

State::State(int r, int m) {
    nReferees = r;
    nMatches = m;
    M = vector<vector<bool> > (nReferees, vector<bool> (nMatches));
}

State::~State() {}

int State::getNReferees() {
    return nReferees;
}

int State::getNMatches() {
    return nMatches;
}

void State::setNReferees(int r) {
    nReferees = r;
}

void State::setNMatches(int m) {
    nMatches = m;
}

bool State::isAssigned(int referee, int match) {
    return M[referee][match];
}

```

```
void State::setAssignment(int referee, int match, bool isAssigned) {
    M[referee][match] = isAssigned;
}

bool State::refereeHasMatchThisRound(int ref, int match) {
    int nmr = this->nReferees/2;
    int round = (match - match % nmr);
    for(int i = 0; i < nmr; ++i) {
        if(getRefereeOfTheMatch(round + i) == ref) return true;
    }
    return false;
}

void State::generateInitialState() {
    int referee;
    for (int match = 0; match < nMatches; ++match) {
        referee = rand() % nReferees;
        while (refereeHasMatchThisRound(referee,match))
            referee = rand() % nReferees;
        for(int i = 0; i < nReferees; ++i) {
            if (i == referee) setAssignment(i,match,true);
            else setAssignment(i,match,false);
        }
    }
}


int State::getRefereeOfTheMatch(int match) {
    for(int i = 0; i < nReferees; ++i) {
        if (M[i][match]) return i;
    }
    return -1;
}

void State::printState() {
    for (int i = 0; i < nReferees; ++i) {
        for(int j = 0; j < nMatches; ++j) {
            if (j % 10 == 0) cout << " ";
            cout << M[i][j];
        }
        cout << endl;
    }
    cout << endl;
}
```

Listing 10: HeuristicFunction.hh

```
#ifndef HEURISTICFUNCTION_HH
#define HEURISTICFUNCTION_HH

#include <iostream>
#include <map>
#include <string>
#include <vector>
#include "State.hh"
using namespace std;

class HeuristicFunction {

protected:

    int intervalRounds;
    int minMatchesPerInterval;
    int maxMatchesPerInterval;
    int minNumRoundsBeforeRepeatingTeam;
    int minNumRoundsBeforeRepeatingStadium;
    int maxConsecutiveRounds;
    map<int,string> matches;
    map<int,int> referees;
    map<string,int> teams;
    vector<vector<string>> incompatibilities;

public:

    // Default constructor
    HeuristicFunction();

    // Constructor with the data required by the heuristic function
    HeuristicFunction(int intervalRounds, int minMatchesPerInterval,
        int maxMatchesPerInterval, int minNumRoundsBeforeRepeatingTeam,
        int minNumRoundsBeforeRepeatingStadium, int maxConsecutiveRounds,
        map<int,string>& matches, map<int,int> referees, map<string,int> teams,
        vector<vector<string>> incompatibilities);

    // Destructor
    ~HeuristicFunction();

    /* Given a state, returns the number of matches that don't have exactly one
    referee multiplied by 100000*/
    int everymatchHasAReferee(State s);

    /* Given a state, returns the number of times a referee is assigned to more
    than one match per round multiplied by 100000*/
    int atMostOneMatchPerRefereePerRound(State s);

    /* Given a state, return the number of times incompatibilities are not
    respected or referees are assigned to matches with a level higher than
    their skills, multiplied by 1000*/
```

```
    int SkillAndWorkingRoundsChecks(State s);

    /* Given a state, returns the number of constraints broken regarding
    the assignment of referees to matches such as being assigned too soon
    to a same stadium or team, being assigned to more consecutive matches
    than it is allowed and being assigned to more or less matches than
    allowed per interval of rounds, multiplied by 1000 */
    int refereeChecks(State s);

    /* Given a state, return the number of pairs of referees that can be
    made such that one referee has more matches assigned than the other */
    int differenceNumberOfMatchesPerReferee(State s);

    /* Given a state, returns the sum of the absolute difference of
    assignments for every couple of referees to the same teams */
    int teamDistributionVars(State s);

    /* Given a state, returns the value of the heuristic function applied
    to it*/
    int evaluateCost(State s);

};

#endif
```

Listing 11: HeuristicFunction.cc

```

#include "HeuristicFunction.hh"

HeuristicFunction::HeuristicFunction() {
    this->intervalRounds = 0;
    this->minMatchesPerInterval = 0;
    this->maxMatchesPerInterval = 0;
    this->minNumRoundsBeforeRepeatingTeam = 0;
    this->minNumRoundsBeforeRepeatingStadium = 0;
    this->maxConsecutiveRounds = 0;
}

HeuristicFunction::HeuristicFunction(int intervalRounds,
    int minMatchesPerInterval, int maxMatchesPerInterval,
     int minNumRoundsBeforeRepeatingTeam,
     int minNumRoundsBeforeRepeatingStadium, int maxConsecutiveRounds,
      map<int,string>& matches, map<int,int> referees, map<string,int> teams,
       vector<vector<string>> incompatibilities) {
    this->intervalRounds = intervalRounds;
    this->minMatchesPerInterval = minMatchesPerInterval;
    this->maxMatchesPerInterval = maxMatchesPerInterval;
    this->minNumRoundsBeforeRepeatingTeam = minNumRoundsBeforeRepeatingTeam;
    this->minNumRoundsBeforeRepeatingStadium =
    minNumRoundsBeforeRepeatingStadium;
    this->maxConsecutiveRounds = maxConsecutiveRounds;
    this->matches = matches;
```

```cpp
    this->referees = referees;
    this->teams = teams;
    this->incompatibilities = incompatibilities;
}

HeuristicFunction::~HeuristicFunction() {}

int HeuristicFunction::everymatchHasAReferee(State s) {
    int sum = 0;
    int nR = s.getNReferees();
    int nM = s.getNMatches();
    for(int match = 0; match < nM; ++match) {
        int aux = 0;
        for(int referee = 0; referee < nR; ++referee) {
            if(s.isAssigned(referee,match)) ++aux;
        }
        if (aux != 1) sum += aux;
    }
    return sum*1000000;
}

int HeuristicFunction::atMostOneMatchPerRefereePerRound(State s) {
    int sum = 0;
    int nR = s.getNReferees();
    int nM = s.getNMatches();
    for(int referee = 0; referee < nR; ++referee) {
        for(int round = 0; round < nM; round = round+10) {
            int aux = 0;
            for(int match = 0; match < 10; ++match) {
                if(s.isAssigned(referee,round+match)) ++aux;
            }
            if (aux != 1) sum += aux;
        }
    }
    return sum*1000000;
}

int HeuristicFunction::SkillAndWorkingRoundsChecks(State s) {
    int sum = 0;
    int nR = s.getNReferees();
    int nM = s.getNMatches();
    int n = incompatibilities.size();
    vector<string> v;
    for(int match = 0; match < nM; ++match) {
        // checks incompatibilities
        for(int i = 0; i < n; ++i) {
            v = incompatibilities[i];
            int referee = s.getRefereeOfTheMatch(match);
            if(v[0] == "S" and to_string(referee) == v[1] and
                matches[match].substr(0,3) == v[2]) sum = sum + 30000;
            else if (v[0] == "T" and to_string(referee) == v[1]
                and (matches[match].substr(0,3) == v[2] or
```

```
                        matches[match].substr(4,3) == v[2])) sum = sum + 30000;
            else if (v[0] == "R" and to_string(referee) == v[1]
                and matches[match].substr(8) == v[2]) sum = sum + 30000;
        }
        // referee skill level is greater than the demanded by the match
        int referee = s.getRefereeOfTheMatch(match);
        int valL = this->teams[matches[match].substr(0,3)];
        int valV = this->teams[matches[match].substr(4,3)];
        int valR = this->referees[referee+1];
        if (valL + valV > valR) sum += (valL + valV - valR)*20000;
    }
    return sum;
}

int HeuristicFunction::refereeChecks(State s) {
    int sum = 0;
    int nR = s.getNReferees();
    int nM = s.getNMatches();
    int nMR = nR/2;
    for(int referee = 0; referee < nR; ++referee) {
        int consecutiveRounds = 0;
        int lastRound = -1;
        for(int match = 0; match < nM; ++match) {
            if (s.isAssigned(referee,match)) {
                string str = this->matches[match];
                string loc = str.substr(0,3);
                string vis = str.substr(4,3);
                // at least x rounds before repeating team assignment
                int m1 = this->minNumRoundsBeforeRepeatingTeam*nMR;
                int m2 = this->minNumRoundsBeforeRepeatingStadium*nMR;
                int m3 = this->intervalRounds*nMR;
                int m = m1;
                if (m2 > m1) m = m2;
                if (m3 > m) m = m3;
                int aux = 0;
                bool complete = false;
                int firstMatch = (match - match % nMR) + nMR;
                for(int extraMatches = 0; (extraMatches < m and
                 firstMatch + extraMatches < nM); ++extraMatches) {
                    if (s.isAssigned(referee, firstMatch + extraMatches)) {
                        string str2 = this->matches[firstMatch + extraMatches];
                        string loc2 = str2.substr(0,3);
                        string vis2 = str2.substr(4,3);
                        // checks referee isn't repeating team too soon
                        if (extraMatches < m1 and (loc == loc2 or loc == vis2
                         or vis == loc2 or vis == vis2)) sum = sum + 10000;
                        // checks referee isn't repeating stadium too soon
                        if (extraMatches < m2 and loc == loc2) sum = sum +
                           ↪ 10000;
                        // checks referee plays the right amount of matches
                        // given an interval of rounds
                        if (extraMatches < m3) ++aux;
```

```
                    }
                    if (firstMatch + extraMatches ==
                        (match/nMR + this->intervalRounds)*nMR - 1)
                        complete = true;
                }
                if (complete and aux < this->minMatchesPerInterval)
                    sum = sum + 30000;
                if (aux > this->maxMatchesPerInterval) sum = sum + 30000;

                // checks referee is not assigned more consecutive
                // rounds than possible
                int round;
                string str2 = this->matches[match];
                if (match < 100) round = stoi(str2.substr(8,1));
                else round = stoi(str2.substr(8,2));
                // if no matches had been assigned yet
                if (lastRound == -1) {
                    lastRound = round;
                    ++consecutiveRounds;
                }
                // consecutive match
                else if (lastRound == round-1) ++consecutiveRounds;
                else { // there has been at least a resting day
                    lastRound = round;
                    consecutiveRounds = 1;
                }
                if (consecutiveRounds > this->maxConsecutiveRounds)
                    sum = sum + 30000;
            }
        }
    }
    return sum;
}

int HeuristicFunction::differenceNumberOfMatchesPerReferee(State s) {
    int sum = 0;
    int nR = s.getNReferees();
    int nM = s.getNMatches();
    vector<int> numMatchesPerReferee = vector<int>(20);
    for(int referee = 0; referee < nR; ++referee) {
        int count = 0;
        for(int match = 0; match < nM; ++match)
            if (s.isAssigned(referee, match)) ++count;
        numMatchesPerReferee[referee] = count;
    }
    for(int referee1 = 0; referee1 < nR; ++referee1) {
        for(int referee2 = referee1 + 1; referee2 < nR; ++referee2) {
            if (numMatchesPerReferee[referee1]>numMatchesPerReferee[referee2]){
                sum += numMatchesPerReferee[referee1];
                sum -= numMatchesPerReferee[referee2];
            }
            else {
```

```
                    sum += numMatchesPerReferee[referee2];
                    sum -= numMatchesPerReferee[referee1];
                }
            }
        }
        return sum;
}

int HeuristicFunction::teamDistributionVars(State s) {
        int n = s.getNReferees();
        int sum = 0;
        vector< vector<int> > v = vector< vector<int> > (n, vector<int> (n, 0));
        int ref;
        for(int match = 0; match < s.getNMatches(); ++match) {
            ref = s.getRefereeOfTheMatch(match);
            string str = this->matches[match];
            string loc = str.substr(0,3);
            string vis = str.substr(4,3);
            ++v[ref][teams[loc]];
            ++v[ref][teams[vis]];
        }
        for(int r1 = 0; r1 < n; ++r1) {
            for(int r2 = r1 + 1; r2 < n; ++r2) {
                for (int t = 0; t < n; ++t) {
                    sum += abs(v[r1][t] - v[r2][t]);
                }
            }
        }
        return sum;
}

int HeuristicFunction::evaluateCost(State s) {
        int sum = 0;
        //sum += everymatchHasAReferee(s) + atMostOneMatchPerRefereePerRound(s);
        sum += SkillAndWorkingRoundsChecks(s);
        sum += refereeChecks(s);
        sum += differenceNumberOfMatchesPerReferee(s);
        sum += teamDistributionVars(s);
        return sum;
}
```

Listing 12: Solver.hh

```
#ifndef SOLVER_HH
#define SOLVER_HH

#include <iostream>
#include <algorithm>
#include <cmath>
#include "HeuristicFunction.hh"
#include "State.hh"
using namespace std;

class Solver {

private:

    /* Copies the values in the orig State to the State dest */
    void copyState(State orig, State& dest);

    /* Applies a random move to the initialState and saves the state obtained
    in the nextState */
    void randomMove(State initialState, State& nextState, int nReferees,
    int nMatches);

public:

    // Default constructor
    Solver();

    // Destructor
    ~Solver();

    /* Given the heuristic function, the number of referees, the number of
    matches and a maximum of iterations, generates a random initial state and
    returns pair containing the resulting state from applying hill climbing
    and its cost*/
    pair<State,int> hillClimbing(HeuristicFunction* hf, int nReferees,
        int nMatches, int maxIterations);

    /* Given the heuristic function, the number of referees, the number of
    matches and a maximum of iterations, generates a random initial state and
    returns pair containing the resulting state from applying simulated
    annealing and its cost*/
    pair<State, int> simulatedAnnealing(HeuristicFunction* hf, int nReferees,
     int nMatches, int maxIterations);

};

#endif
```

Listing 13: Solver.cc

```

#include "Solver.hh"

Solver::Solver() {}

Solver::~Solver() {}

void Solver::copyState(State orig, State& dest) {
    for (int referee = 0; referee < dest.getNReferees(); ++referee)
        for(int match = 0; match < dest.getNMatches(); ++match)
            dest.setAssignment(referee, match, orig.isAssigned(referee,match));
}

void Solver::randomMove(State initialState, State& nextState, int nReferees,
    int nMatches) {
    copyState(initialState,nextState);
    int randMove = rand() % 10;
    if (randMove < 6) { // changeReferee
        // looks for random match and random new referee
        int randMatch = rand() % nMatches;
        int oldReferee = initialState.getRefereeOfTheMatch(randMatch);
        int newReferee = rand() % nReferees;
        // checks the referee is available this round
        int i = 1;
        while(i < 1000 and (newReferee == oldReferee or
            initialState.refereeHasMatchThisRound(newReferee,randMatch))) {
            newReferee = rand() % nReferees;
            ++i;
        }
        nextState.setAssignment(oldReferee, randMatch, false);
        nextState.setAssignment(newReferee, randMatch, true);
    }
    else { //swapMatches
        // looks for two random matches and their referees
        int randMatch1 = rand() % nMatches;
        int referee1 = initialState.getRefereeOfTheMatch(randMatch1);
        int randMatch2 = rand() % nMatches;
        int referee2 = initialState.getRefereeOfTheMatch(randMatch2);
        // checks the 2n match can be swaped with the first one
        int j = 0;
        while (j < 1000 and (randMatch1 == randMatch2 or referee1 == referee2
            or initialState.refereeHasMatchThisRound(referee1, randMatch2)
            or initialState.refereeHasMatchThisRound(referee2, randMatch1))) {
            randMatch2 = rand() % nMatches;
            referee2 = initialState.getRefereeOfTheMatch(randMatch2);
            ++j;
        }
        nextState.setAssignment(referee1, randMatch1, false);
        nextState.setAssignment(referee2, randMatch2, false);
        nextState.setAssignment(referee1, randMatch2, true);
        nextState.setAssignment(referee2, randMatch1, true);
```

```
    }
}

pair<State,int> Solver::hillClimbing(HeuristicFunction* hf, int nReferees,
    int nMatches, int maxIterations) {

    cout << endl << " ----- Applying Hill Climbing ----- " << endl << endl;

    // Best state so far per attempt
    State state = State(nReferees,nMatches);
    state.generateInitialState();
    // State generated by the movements
    State nextState = State(nReferees,nMatches);
    // State keeping the best so far in all the attempts
    State bestState = State(nReferees,nMatches);
    copyState(state,bestState);
    int cost = hf->evaluateCost(state);
    int bestCost = cost;
    cout << "Initial cost : " << cost << endl;
    for(int attempt = 0; attempt < 5; ++attempt) {
        for(int i = 0; i < maxIterations; ++i) {
            bool found = false;
            int attemptedMoves = 0;
            while(not found and attemptedMoves < 10000) {
                randomMove(state, nextState, nReferees, nMatches);
                int nextCost = hf->evaluateCost(nextState);
                if (nextCost < cost) {
                    copyState(nextState,state);
                    cost = nextCost;
                    found = true;
                }
                ++attemptedMoves;
                if (cost == 0) return pair<State,int> (state,cost);
            }
            if (cost < bestCost){
                bestCost = cost;
                copyState(state,bestState);
            }
            if (not found) break;
            cout << "Iteration " << i << " => " << cost << endl;
        }
        cout << endl << "---------- RANDOMIZING ---------" << endl << endl;
        for(int i = 0; i < 2000; ++i)
            randomMove(state, state, nReferees, nMatches);
        cost = hf->evaluateCost(state);
    }
    return pair<State,int> (bestState,bestCost);
}



pair<State, int> Solver::simulatedAnnealing(HeuristicFunction* hf,
```

```
    int nReferees, int nMatches, int maxIterations) {

    cout << endl << " ----- Applying Simulated Annealing -----" << endl;
    cout << endl;

    State state = State(nReferees,nMatches);
    state.generateInitialState();
    State nextState = State(nReferees,nMatches);
    State bestState = State(nReferees,nMatches);
    copyState(state,bestState);
    int cost = hf->evaluateCost(state);
    int bestCost = cost;
    cout << "Initial cost : " << bestCost << endl;

    bool cold = false;
    double T = 0.5;
    double beta = 0.99;

    int maxNumReheats = 5;
    int numReheats = 0;
    while (numReheats < maxNumReheats) {
        int numPhasesPerReheat = 150;
        int numPhase = 0;
        while (not cold and numPhase <= numPhasesPerReheat) {
            int numMoveAttemptsPerPhase = 1000;
            int moveAttempts = 0;
            int bestOfThePhase = 10e8;
            while (not cold and moveAttempts <= numMoveAttemptsPerPhase) {
                ++moveAttempts;
                randomMove(state, nextState, nReferees, nMatches);
                int nextCost = hf->evaluateCost(nextState);
                if (nextCost < cost) {
                    copyState(nextState,state);
                    cost = nextCost;
                    if (cost < bestCost) {
                        copyState(state, bestState);
                        bestCost = cost;
                        cout << "    Better cost found at the ";
                        cout << moveAttempts << "th attempt with cost ";
                        cout << bestCost << endl;
                        if (bestCost == 0)
                            return pair<State,int> (bestState, bestCost);
                    }
                }
                else {
                    int difCost = nextCost - bestCost;
                    double r = ((double)rand()) / RAND_MAX;
                    int aux = -((double)difCost)/T;
                    int p = exp(aux);
                    if (r < p) {
                        copyState(nextState,state);
                        cost = nextCost;
```

```
                    }
                }
                if (nextCost < bestOfThePhase)
                    bestOfThePhase = nextCost;
                if (bestCost == 0)
                    return pair<State, int> (bestState, bestCost);
            }
            numPhase++;
            T = T*beta;
            cout << "Reheat " << numReheats << " and phase " << numPhase <<
                " : " << bestCost << " - best of the phase : " <<
                bestOfThePhase << endl;
            if (exp(-1/T) < 10e-10) {
                cout << "Ice cold, annealing terminated." << endl;
                cout << endl << endl;
                cold = true;
            }
        }
        cout << endl << "---------- REHEATING ----------" << endl << endl;
        for(int i = 0; i < 2000; ++i)
            randomMove(state, state, nReferees, nMatches);
        numReheats++;
        cost = hf->evaluateCost(state);
        T = 0.5;
        cold = false;
    }
    return pair<State, int> (bestState, bestCost);
}
```

Listing 14: Writer.hh

```
#ifndef WRITER_HH
#define WRITER_HH

#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <vector>
#include <cmath>
#include "State.hh"
using namespace std;

class Writer {

protected:

    map<int,string> listOfMatches; // map with data from the matches
    map<string,int> listOfTeams; // map with data from the teams
    map<int,int> listOfReferees; // map with data from the referees

public:

    // Default constructor
    Writer();

    // Constructor with the data
    Writer(map<int,string>& listOfMatches,
            map<string,int>& listOfTeams,
            map<int,int>& listOfReferees);

    // Destructor
    ~Writer();

    /* Given a state, its cost and a string identifying the algorithm that
    has been used, writes the assignment of referees represented by the state
    in a file */
    void writeSolution(State state, int cost, string alg);

};

#endif
```

Listing 15: Writer.cc

```

#include "Writer.hh"

Writer::Writer() {}

Writer::Writer(map<int,string>& listOfMatches, map<string,int>& listOfTeams,
    map<int,int>& listOfReferees) {
```

```
    this->listOfMatches = listOfMatches;
    this->listOfTeams = listOfTeams;
    this->listOfReferees = listOfReferees;
}

Writer::~Writer() {}

void Writer::writeSolution(State state, int cost, string alg) {
    ofstream outFile;
    if (alg == "HC") {
        outFile.open("solutionHC.txt");
        outFile << endl;
        outFile << "Cost of the solution found with Hill Climbing: ";
        outFile << cost << endl << endl;
    }
    else {
        outFile.open("solutionSA.txt");
        outFile << endl;
        outFile << "Cost of the solution found with Simulated Annealing: ";
        outFile << cost << endl << endl;
    }
    int nR = state.getNReferees();
    int nM = state.getNMatches();
    int nRounds = (nR - 1) * 2;
    string str, loc, vis;
    for (int round = 0; round < nRounds; ++round) {
        outFile << "Round " << round << " : " << endl;
        for(int matchIndex = 0; matchIndex < (nR/2); ++matchIndex) {
            int match = round*(nR/2) + matchIndex;
            int ref = state.getRefereeOfTheMatch(match);
            str = listOfMatches[match];
            loc = str.substr(0,3);
            vis = str.substr(4,3);
            outFile << "  " << loc << " - " << vis << " => referee ";
            outFile <<  ref+1 << endl;
        }
    }
    cout << endl << endl;
}
```

Listing 16: main.cc

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <fstream>
#include <cmath>
#include "State.hh"
#include "HeuristicFunction.hh"
#include "Reader.hh"
#include "Writer.hh"
#include "Solver.hh"
using namespace std;


int main(int argc, char *argv[]) {

    /* Checks the call is made with the parameters needed*/
    if (argc != 3) {
        cout << "ERROR: The call should be \"./main.exe";
        cout << " calendarFile dataFile\"" << endl;
        exit(1);
    }

    // initalitates the structures needed to contain the data for the problem
    map<int,string> listOfMatches;
    map<string,int> listOfTeams;
    map<int,int> listOfReferees;
    vector<vector<string>> listOfIncompatibilities;

    // Reads the data from the files and inserts it into the variables
    Reader r(argv[1],argv[2]);
    vector<int> v = r.read(listOfMatches, listOfTeams,
        listOfReferees, listOfIncompatibilities);
    int nReferees = v[0];
    int nMatches = v[1];
    int intervalRounds = v[2];
    int minMatchesPerInterval = v[3];
    int maxMatchesPerInterval = v[4];
    int minNumRoundsBeforeRepeatingTeam = v[5];
    int minNumRoundsBeforeRepeatingStadium = v[6];
    int maxConsecutiveRounds = v[7];

    // Generates an instance of the solver and the heuristic function
    Solver s = Solver();
    int maxIterations = 10e4;
    HeuristicFunction* hf = new HeuristicFunction(intervalRounds,
        minMatchesPerInterval, maxMatchesPerInterval,
        minNumRoundsBeforeRepeatingTeam,
        minNumRoundsBeforeRepeatingStadium,
        maxConsecutiveRounds, listOfMatches, listOfReferees,
        listOfTeams, listOfIncompatibilities);
```

```
    /* Generates a writer, applies Hill Climbing, calculates the time needed
    to find the best solution and writes the solution to solutionHC.txt*/
    Writer w(listOfMatches, listOfTeams, listOfReferees);
    const clock_t beginTimeHC = clock();
    pair<State,int> pHC = s.hillClimbing(hf, nReferees,
        nMatches, maxIterations);
    float timeDiffHC = float(clock() - beginTimeHC);
    timeDiffHC /=  CLOCKS_PER_SEC;
    w.writeSolution(pHC.first, pHC.second, "HC");

    /* Applies Simulated Annealing, calculates the time needed to find the
    best solution and writes the solution to solutionSA.txt */
    const clock_t beginTimeSA = clock();
    pair<State,int> pSA = s.simulatedAnnealing(hf, nReferees,
        nMatches, maxIterations);
    float timeDiffSA = float(clock() - beginTimeSA);
    timeDiffSA /= CLOCKS_PER_SEC;
    w.writeSolution(pSA.first, pSA.second, "SA");

    // Prints the cost and time for the Hill Climbing
    cout << endl;
    cout << "Hill Climbing results " << endl;
    cout << " * Best cost found : " << pHC.second << endl;
    cout << " * Time : " << timeDiffHC << " seconds" << endl;
    cout << endl;

    // Prints the cost and time for the Simulated Annealing
    cout << endl;
    cout << "Simulated Annealing results" << endl;
    cout << " * Best cost found : " << pSA.second << endl;
    cout << " * Time : " << timeDiffSA << " seconds" << endl;
    cout << endl;

}
```

Listing 17: Makefile

```

main.exe: main.o State.o Solver.o Writer.o Reader.o HeuristicFunction.o
	g++ -o main.exe main.o State.o Solver.o Writer.o Reader.o HeuristicFunction.o

HeuristicFunction.o: HeuristicFunction.cc HeuristicFunction.hh
	g++ -c HeuristicFunction.cc

Reader.o: Reader.cc Reader.hh
	g++ -c Reader.cc

Writer.o: Writer.cc Writer.hh
	g++ -c Writer.cc

Solver.o: Solver.cc Solver.hh
	g++ -c Solver.cc

State.o: State.cc State.hh
	g++ -c State.cc

main.o: main.cc
	g++ -c main.cc
clean:
	rm *.o *.exe
```

Listing 18: Solution obtained with Hill Climbing

```

Cost of the solution found with Hill Climbing: 32

Round 0 :
  te1 - te2 => referee 6
  te3 - te4 => referee 3
  te5 - te6 => referee 1
Round 1 :
  te3 - te1 => referee 2
  te2 - te5 => referee 4
  te4 - te6 => referee 6
Round 2 :
  te1 - te4 => referee 6
  te5 - te3 => referee 5
  te6 - te2 => referee 2
Round 3 :
  te5 - te1 => referee 3
  te4 - te2 => referee 5
  te3 - te6 => referee 1
Round 4 :
  te1 - te6 => referee 5
  te2 - te3 => referee 1
  te4 - te5 => referee 3
Round 5 :
  te2 - te1 => referee 5
  te4 - te3 => referee 4
  te6 - te5 => referee 2
Round 6 :
  te1 - te3 => referee 2
  te5 - te2 => referee 3
  te6 - te4 => referee 4
Round 7 :
  te1 - te4 => referee 1
  te3 - te5 => referee 6
  te2 - te6 => referee 5
Round 8 :
  te1 - te5 => referee 2
  te2 - te4 => referee 4
  te6 - te3 => referee 3
Round 9 :
  te6 - te1 => referee 4
  te3 - te2 => referee 6
  te5 - te4 => referee 1
```

Listing 19: Solution obtained with Simulated Annealing

```

Cost of the solution found with Simulated Annealing: 32

Round 0 :
  te1 - te2 => referee 4
  te3 - te4 => referee 6
  te5 - te6 => referee 3
Round 1 :
  te3 - te1 => referee 5
  te2 - te5 => referee 1
  te4 - te6 => referee 3
Round 2 :
  te1 - te4 => referee 2
  te5 - te3 => referee 4
  te6 - te2 => referee 6
Round 3 :
  te5 - te1 => referee 1
  te4 - te2 => referee 5
  te3 - te6 => referee 2
Round 4 :
  te1 - te6 => referee 4
  te2 - te3 => referee 2
  te4 - te5 => referee 6
Round 5 :
  te2 - te1 => referee 4
  te4 - te3 => referee 1
  te6 - te5 => referee 5
Round 6 :
  te1 - te3 => referee 3
  te5 - te2 => referee 2
  te6 - te4 => referee 6
Round 7 :
  te1 - te4 => referee 1
  te3 - te5 => referee 5
  te2 - te6 => referee 4
Round 8 :
  te1 - te5 => referee 5
  te2 - te4 => referee 3
  te6 - te3 => referee 1
Round 9 :
  te6 - te1 => referee 2
  te3 - te2 => referee 3
  te5 - te4 => referee 6
```

# B. ILP Code for the Basic Problem

In this section of the appendix is found the Prolog code used to generate the constraints needed to solve the basic version of the problem. To execute properly this program other files are needed, so we also include the file *xml2simple.pl* and the *Makefile*, that facilitate the use of the program.

The Prolog code is found in Listing 20. The lines up until the 10th are used to define the computation time and read the data from the files, which in this case are the file data1819.pl and the file calendar1819.pl. The predicate defined between the lines from the 23rd to the 39th calls the predicates that generates the constraints that define the problem, which are defined in the lines below. The objective function is defined in the lines between the 140th and the 146th, the variables are defined as boolean in the lines between the 152 and the 160 and are imposed to have values between 0 and 1 in the lines from the 162nd to the 170th. The lines from the 181th to the 218th are used to display in a readable way the solution and the lines from the 222nd until the end are the main code of the program, that makes all the needed calls and calls the solver, which in this case is CPLEX.

While executing the program several files are created, the first one being *c.lp*, that is the file where the constraints are written. Once all the constraints are written, the program writes the file *fileForCplex*, which contains the information that must be passed to the solver such the name of the file where the constraints are written, the maximum computation time or time limit and the name of the file where to write the solution. When the solver is called the file *cplex.log* keeps the prints made by the solver, *sol.pl* contains the solution of the problem with the value given to each of the variables and *sol.txt* contains the solution printed in more readable way.

The other files needed for the execution can be found in the Listings 21 and 22. With these files and having installed Swipl and CPLEX, to execute the code from a Ubuntu terminal the only things that need to be done is to call the Makefile and then execute the executable file that will be generated and will be called *rap*.

Listing 20: rap.pl - Prolog code to solve the basic problem

```
1
2  % ================================ INPUT DATA ====================================
3
4  identifier('Referee Assignment Problem: basic version').
5
6  maxComputationTime(100).
7
8  :-include(data1819).
9
10 :-include(calendar1819).
11
12 % ============================= NO MORE INPUT DATA ===============================
13
14 symbolicOutput(0).
15
16 %% Variables:
17 % assign(Ref,S,T,R) to assign to a match(s,t,r) the referee referee
```

```
% workingRound(Ref,R) if the referee Ref has a match assigned in round r
%% Definitions:
round(R):- numRounds(N), between(1,N,R).
ref(R):- referee(R,_).

writeConstraints:-
    everyMatchHasAReferee,
    atMostOneMatchPerRefereePerRound,
    refereeMinimumSkillLevelPerMatch,
    defineWorkingRound,
    atLeastMinMatchesPerIntervalOfRounds,
    atMostMaxMatchesPerIntervalOfRounds,
    atLeastMinRoundsRepeatingTeam,
    atLeastMinRoundsRepeatingStadium,
    incompatibilityRefereeRound,
    incompatibilityRefereeTeam,
    incompatibilityRefereeStadium,
    maxConsecutiveRoundsPerReferee,
    maxDifferenceWorkedRounds,
    differenceVars,
    teamDistributionVars,
    !.


everyMatchHasAReferee:-
    match(S,T,R), findall(assign(Ref,S,T,R), referee(Ref,_), Sum),
    writeConstraint(Sum = 1), fail.
everyMatchHasAReferee.

atMostOneMatchPerRefereePerRound:-
    ref(Ref), round(R), findall(assign(Ref,S,T,R), match(S,T,R),Sum),
     writeConstraint(Sum =< 1),fail.
atMostOneMatchPerRefereePerRound.

refereeMinimumSkillLevelPerMatch:-
    referee(Ref,L), match(S,T,R), team(S,LS), team(T,LT), L < LS + LT,
    writeClause([-assign(Ref,S,T,R)],[]),fail.
refereeMinimumSkillLevelPerMatch.

defineWorkingRound:-
    ref(Ref), round(R), findall(assign(Ref,S,T,R),match(S,T,R),Lits),
    expressOr(workingRound(Ref,R),Lits),fail.
defineWorkingRound.

atLeastMinMatchesPerIntervalOfRounds:-
    intervalRounds(N), minMatchesPerInterval(Min),
    ref(Ref), round(R1), R2 is R1+N-1, round(R2),
    findall( workingRound(Ref,R),  between(R1,R2,R), Sum),
    writeConstraint(Sum >= Min), fail.
atLeastMinMatchesPerIntervalOfRounds.
```

```
atMostMaxMatchesPerIntervalOfRounds:-
     intervalRounds(N), maxMatchesPerInterval(Max),
     ref(Ref), round(R1), R2 is R1+N-1, round(R2),
     findall( workingRound(Ref,R),  between(R1,R2,R), Sum),
     writeConstraint(Sum =< Max), fail.
atMostMaxMatchesPerIntervalOfRounds.


atLeastMinRoundsRepeatingTeam:-
     minNumRoundsBeforeRepeatingTeam(MinR),
     ref(Ref), match(S1,T1,R1), match(S2,T2,R2), R2 > R1,
     MinR >= R2-R1, sort([S1,T1,S2,T2],L), L\=[_,_,_,_],
     writeClause([-assign(Ref,S1,T1,R1),-assign(Ref,S2,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingTeam.


atLeastMinRoundsRepeatingStadium:-
     minNumRoundsBeforeRepeatingStadium(MinR),
     ref(Ref), match( S,T1,R1), match( S,T2,R2), R2 > R1, MinR >= R2-R1,
     writeClause([-assign(Ref,S,T1,R1),-assign(Ref,S,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingStadium.


incompatibilityRefereeRound:-
     ref(Ref), incRefRound(Ref,R), writeClause([-workingRound(Ref,R)],[]), fail.
incompatibilityRefereeRound.

incompatibilityRefereeTeam:-
     ref(Ref), incRefTeam(Ref,T), match(T,S,R),
     writeClause([-assign(Ref,T,S,R)],[]), fail.
incompatibilityRefereeTeam:-
     ref(Ref), incRefTeam(Ref,T), match(S,T,R),
     writeClause([-assign(Ref,S,T,R)],[]), fail.
incompatibilityRefereeTeam.

incompatibilityRefereeStadium:-
     ref(Ref), incRefStad(Ref,S), match(S,T,R),
     writeClause([-assign(Ref,S,T,R)],[]), fail.
incompatibilityRefereeStadium.

maxConsecutiveRoundsPerReferee:-
     maxConsecutiveRounds(MaxR), ref(Ref), round(R1), R2 is R1+MaxR,
     round(R2), findall( -workingRound(Ref,R), between(R1,R2,R), Lits ),
     writeClause(Lits,[]), fail.
maxConsecutiveRoundsPerReferee.

maxDifferenceWorkedRounds:-
     ref(Ref1), ref(Ref2), Ref1 \= Ref2,
     findall(   workingRound(Ref1,R), round(R), Sum1),
     findall(-1*workingRound(Ref2,R), round(R), Sum2),
     append(Sum1,Sum2,Sum), writeConstraint(Sum =< 2), fail.
maxDifferenceWorkedRounds.
```

```
differenceVars:-
    ref(Ref1), ref(Ref2), Ref1 \= Ref2,
    findall(    workingRound(Ref1,R), round(R), Sum1),
    findall(-1*workingRound(Ref2,R), round(R), Sum2), append(Sum1,Sum2,Sum),
    writeConstraint( [-1000 * dVar(Ref1,Ref2) |Sum ] =< 0), fail.
differenceVars.

teamDistributionVars:-
    ref(Ref1), ref(Ref2), Ref1 \= Ref2, team(S,_),
    findall(    assign(Ref1,S,T,R), match(S,T,R), Sum11),
    findall(    assign(Ref1,T,S,R), match(T,S,R), Sum12),
    findall(-1*assign(Ref2,S,T,R), match(S,T,R), Sum21),
    findall(-1*assign(Ref2,T,S,R), match(T,S,R), Sum22),
    append(Sum11,Sum12,Sum1), append(Sum21,Sum22,Sum2), append(Sum1,Sum2,Sum),
    writeConstraint( [-1000 * tVar(Ref1,Ref2,S) |Sum ] =< 0), fail.
teamDistributionVars.

writeObjectiveFunction:-
    write('obj: '), ref(Ref1), ref(Ref2), Ref1 \= Ref2, write(' + '),
    write(dVar(Ref1,Ref2)), fail.
writeObjectiveFunction:-
    ref(Ref1), ref(Ref2), team(S,_), Ref1 \= Ref2, write(' + '),
    write(tVar(Ref1,Ref2,S)), fail.
writeObjectiveFunction:- nl.

writeCost(M):- assertz(cost(M)), writeMon( M ), nl,!.

writeIntegerVars.

writeBooleanVars:- ref(Ref), match(S,T,R),
    write( assign(Ref,S,T,R) ), nl, fail.
writeBooleanVars:- ref(Ref), round(R),
     write( workingRound(Ref,R) ), nl, fail.
writeBooleanVars:- ref(Ref1), ref(Ref2), Ref1 \= Ref2,
    write( dVar(Ref1,Ref2) ), nl, fail.
writeBooleanVars:- ref(Ref1), ref(Ref2), Ref1 \= Ref2, team(S,_),
    write( tVar(Ref1,Ref2,S) ), nl, fail.
writeBooleanVars.

writeBounds:- ref(Ref), match(S,T,R),
    write('0 <= '), write( assign(Ref,S,T,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), round(R), write('0 <= '),
    write( workingRound(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref1), ref(Ref2), Ref1 \= Ref2,
    write('0 <= '), write( dVar(Ref1,Ref2) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref1), ref(Ref2), Ref1 \= Ref2, team(S,_),
    write('0 <= '), write( tVar(Ref1,Ref2,S) ), write(' <= 1'), nl, fail.
writeBounds.

wl([]).
wl([X|L]):- write(X), write('  '), wl(L),!.
```

```prolog

expressOr( Var, Lits ):- member(Lit,Lits), writeClause([ -Lit ], [ Var ]), fail.
expressOr( Var, Lits ):-  writeClause([ -Var ], Lits ),!.

% ============= DisplaySol =============

displaySol(_):- retractall(sol(_,_)), fail.
displaySol(M):- member(X=V,M), assertz(sol(X,V)), fail.

displaySol(_):-
    round(R), nl,  write('Round '), write(R), write(':  '), team(S,_),
    team(T,_) ,match(S,T,R), sol(assign(Ref,S,T,R),1),  write(' '),
    writeAssignment(Ref,S,T), fail.
displaySol(_):- nl, nl, write('==================================='), nl,fail.

displaySol(_):- nl, nl, write('Referees :  1  2  3  4  5  6  7  8  9 10 11 '),
    write('12 13 14 15 16 17 18 19 20'), nl, fail.
displaySol(_):- round(R), nl,  write('Round '), writeASpaceIfLess10(R),
    write(R), write(':  '), ref(Ref), write('  '), sol(workingRound(Ref,R),S),
    write(S), fail.
displaySol(_):- nl,  write('Total:      '), ref(Ref),
    findall( R, sol(workingRound(Ref,R),1), L), length(L,N), write(N),
    write(' '), fail.
displaySol(_):- nl, nl, write('==================================='), nl,fail.

displaySol(_):- nl, nl, write('Referees :  1  2  3  4  5  6  7  8  9 10 11 '),
    write('12 13 14 15 16 17 18 19 20'), nl, fail.
displaySol(_):-
    team(S,_),nl, write('Team '), write(S), write(' : '), ref(Ref),
    findall(R, (match(S,T,R), sol(assign(Ref,S,T,R),1)),Sum1),
    findall(R, (match(T,S,R), sol(assign(Ref,T,S,R),1)),Sum2),
    length(Sum1,N1), length(Sum2,N2), N is N1 + N2, write('  '), write(N), fail.

displaySol(_):- nl, nl, write('==================================='), nl,fail.
displaySol(_).

writeASpaceIfLess10(R):- R<10, write(' '),!.
writeASpaceIfLess10(_).
writeAssignment(Ref,S,T):-
    nl, write('     '), write(S), write(' - '), write(T),
    write(' is officiated by referee '), write(Ref),
    write('. Skill level comparison game vs referee: '), referee(Ref,RS),
    team(S,SS), team(T,ST), Skill is SS+ST, write(Skill), write(' - '),
    write(RS), write('.').

% ========================== MAIN ==========================

main:- symbolicOutput(1), !, writeConstraints, nl, halt.
main:-
%   current_prolog_flag(argv,[_,Mes|_]),
    unix('rm -f solCplex.sol fileForCplex salCplex c.lp cplex.log'),
```

```
    write('generating constraints...'),nl,

    tell('c.lp'),
    write('Minimize'   ), nl,   writeObjectiveFunction,
    write('Subject To' ), nl,   writeConstraints,
    write('Bounds'     ), nl,   writeBounds,
    write('Generals'   ), nl,   writeIntegerVars,
    write('Binary'     ), nl,   writeBooleanVars,
    write('End'        ), nl,   told,
    write('constraints generated'),nl,nl,nl,nl,


    tell(fileForCplex), maxComputationTime(T),
    write('read c.lp'), nl,
    write('set timelimit '), write(T), write(' s'), nl,
    write('set mip tolerance mipgap 0.03.  '), nl,
    write('opt'), nl, write('write solCplex.sol'), nl, write('quit'), nl, told,
    unix(' cplex < fileForCplex > salCplex'),
%   unix('cplex < fileForCplex ;'),
%   unix('cat fileForCplex '),
    checkIfSolution, nl,nl,
    halt.
main:-      write('constraints generation failed'),nl, halt.



checkIfSolution:-
    exists_file('solCplex.sol'), !,
    unix('xml2simple.pl solCplex.sol > sol.pl'),
    see('sol.pl'), readModel([],M), seen,
    nl,nl,nl,write('Solution found. Press <enter> to see it'), nl,nl,nl,
    get_char(_), identifier(Id), tell('sol.txt'), write(Id), nl, nl,
    displaySol(M), told, displaySol(M),!.
checkIfSolution:-
    shell('grep "Integer infeasible" cplex.log > salgrep', 0), nl,nl,
%   grep returns 0
    write('Solver: No solution exists'),!.
checkIfSolution:- maxComputationTime(T), nl,nl,
    write('Solver: No solution found under the given time limit of '),
    write(T), write(' s.'),!.

unix(Command):-shell(Command),!.
unix(_).

writeConstraint(C):- C =.. [Op,Sum,K], writeSum(Sum), write('  '), writeOp(Op),
    write('  '), write(K), nl.
writeSum([]):-  !.
writeSum([M|L]):-  writeMon(M), nl, writeSum(L), !.
writeMon(A*X):- A>=0, !,              write(' + '), write(A ), write(' '),
     write(X), !.
writeMon(A*X):- A<0, !,  AB is -A,  write(' - '), write(AB), write(' '),
    write(X), !.
```

```
writeMon(X):- !, write(' + '), write(1 ), write(' '), write(X), !.

writeClause( Neg, _    ):-  member(Lit,Neg), Lit \= -_,
    write(error('negative lit')), nl, halt.
writeClause(   _, Pos ):-  member(Lit,Pos), Lit  = -_,
    write(error('positive lit')), nl, halt.
writeClause( Neg, Pos ):-  length(Neg,N), K is 1-N,
    findall( -1*Lit, member(-Lit,Neg), NegLits ),
    append(NegLits,Pos,Sum), writeConstraint( Sum >= K ),!.

readModel(L1,L2):- read(XV), addIfNeeded(XV,L1,L2),!.
addIfNeeded(end_of_file,L,L):-!.
addIfNeeded(XV,L1,L2):- readModel([XV|L1],L2),!.

writeOp(=<):-write('<='),!.
writeOp(Op):-write(Op),!.
```

Listing 21: xml2simple.pl

```
#!/usr/bin/perl
if(!open(LIST,$ARGV[0])){
    open(LIST,"cplex.log");
    while(<LIST>){
    if(/Time limit exceeded/){
        print("Result: UNKNOWN\n");
        exit;
    }
    if(/Current MIP best bound is infinite./){
        print("Result: UNSAT\n");
        exit;
    }
    }
    print("Result: ERROR\n");
    exit;
}

#print("Result: SAT\n\n");

$cost;
$isOptimal = 0;

while(<LIST>) {
    if(/objectiveValue="(\d*)"/){
    $cost = $1;
    }
    if(/solutionStatusValue="101"/){
    $isOptimal=1;
    }
    if(/solutionStatusValue="102"/){
    $isOptimal=1;
    }
    # if(/variable name="(w\d+s\d+)" index.*value="(\d*)\.*(\d*)"/) {
    #   print("$1 = $2.$3.\n");
```

```
    # }
    if(/variable name="(.*)" index.*value="-0"/) {
    print("$1 = 0.\n");
    }
    if(/variable name="(.*)" index.*value="(\d*)"/) {
    print("$1 = $2.\n");
    }
}
#print("\n");
#print("Cost: $cost\n");
#print("Optimal: $isOptimal\n");
```

Listing 22: Makefile

```
file = rap

$(file): $(file).pl
	swipl --quiet -O -g main --stand_alone=true -o $(file) -c $(file).pl


clean:
	rm -f rap clone* c.lp cplex.log fileForCplex solCplex.sol sol.pl sol.txt
```

# C. ILP Code for the KNVB Problem

In this section of the appendix is found the Prolog code used to generate the constraints needed for the KNVB version of the problem, the C++ program that prepares the data obtained from previous rounds to be incorporated in the following sub-problem and the file that is used if no data is incorporated from previous rounds. To execute this program the same *xml2simple.pl* and *Makefile* files presented in Listings 21 and 22 in Appendix B can be used.

The Prolog code can be found in Listing 23 and works exactly like the Prolog code from the basic problem but with more constraints and variables. In Listing 25 can be found the file that is to be used if no data from previous sub-problems wants to be used and the file to adapt the data from previous sub-problems can be found in Listing 24. The file containing the data from previous rounds (or the file in Listing 25 if no data is to be imported) needs to be named *previousRounds.pl*. If after solving a problem the data wants to be reused, the only thing that needs to be done is execute the compile code from the C++ program and this file will be automatically generated, otherwise the file in Listing 25 needs to be saved with this name, and always in the same folder as the Prolog code is executed.

Listing 23: rap.pl - Prolog code to solve the KNVB problem

```

% ======================================== INPUT DATA ========================================
% INPUT DATA

identifier('Referee Assignment Problem: Eeredivisie and Eerste Divisie').

maxComputationTime(3600).

:-include(calendar1819).
:-include(refereesData).
:-include(previousRounds).

intervalRounds(5).
minMatchesPerInterval(2).
maxMatchesPerInterval(4).
minNumRoundsBeforeRepeatingTeam(3).
minNumRoundsBeforeRepeatingStadium(3).
maxConsecutiveRounds(4).
maxRoundsWithoutACertainLevelMatch(20).
initialRound(1).
endingRound(41).

% ==================================== NO MORE INPUT DATA ====================================

symbolicOutput(0).

%% Definitions:
round(R):- initialRound(I), endingRound(E), between(I,E,R).
ref(R):- referee(R,_,_,_).
as(A):- assistant(A,_,_,_).

```

```
refP(R,P):-referee(R,_,_,P).
asP(A,P):- assistant(A,_,_,P).

max(A,B,A):- A>B,!.
max(_,B,B).

match(S,T,W):- erematch(W,_,S,T,_,_).
match2(S,T,W):- eerstematch(W,_,S,T,_,_).

partit(W,S,T,D):- erematch(W,_,S,T,D,_).
partit(W,S,T,D):- eerstematch(W,_,S,T,D,_).

equip(T):- team(T,_).

game(S,T,R):- match(S,T,R).
game(S,T,R):- match2(S,T,R).

writeConstraints:-
	previousRoundsData,
	everyMatchHas1Referee,
	everyMatchHas2Assistants,
	everyMatchHas1Var,
	matches2DontHaveVar,
	everyMatchHas1AVar,
	matches2DontHaveAVar,
	everyMatchHasA4thRef,
	atMostOneMainRefereeRolePerRound,
	atMostOneMainAssistingRolePerRound,
	atMostTwoRefereeRolePerRound,
	atMostTwoAssistantRolePerRound,
	oneRefereeRolePerMatch,
	oneAssistantRolePerMatch,
	defineMainRefereeWR,
	defineMainAssistantWR,
	defineRefereeWR,
	defineAssistantWR,
	incompatibilityRefereeRound,
	incompatibilityAssistantRound,
	incompatibilityRefereeTeam,
	incompatibilityAssistantTeam,
	trioMustWorkTogether,
	atLeastMinRoundsRepeatingTeam,
	atMostMaxMatchesPerIntervalOfRounds,
	refereesLevelRatioPerRound,
	assistantsLevelRatioPerRound,
	defineImportanceOfMatches,
	maxRoundsWithoutCertainLevelMatches,
	differentLevelVars,
	numberOfGamesVars,
	minimize2gamesIn4days,
	!.
```

```
previousRoundsData:- notassignR(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([-assignR(Ref,S,T,R)],[]), fail.
previousRoundsData:- notassign4(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([-assign4(Ref,S,T,R)],[]), fail.
previousRoundsData:- notassignVAR(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([-assignVAR(Ref,S,T,R)],[]), fail.
previousRoundsData:- notassignAR(A,S,T,R), as(A), game(S,T,R),
    writeClause([-assignAR(A,S,T,R)],[]), fail.
previousRoundsData:- notassignAVAR(A,S,T,R), as(A), game(S,T,R),
    writeClause([-assignAVAR(A,S,T,R)],[]), fail.
previousRoundsData:- notmainRefereeWR(Ref,R), ref(Ref), numRounds(N),
    between(1,N,R), writeClause([-mainRefereeWR(Ref,R)],[]), fail.
previousRoundsData:- notmainAssistantWR(A,R), as(A), numRounds(N),
    between(1,N,R), writeClause([-mainAssistantWR(A,R)],[]), fail.
previousRoundsData:- notrefereeWR(Ref,R), ref(Ref), numRounds(N),
    between(1,N,R), writeClause([-refereeWR(Ref,R)],[]), fail.
previousRoundsData:- notassistantWR(A,R), as(A), numRounds(N),
    between(1,N,R), writeClause([-assistantWR(A,R)],[]), fail.

previousRoundsData:- yesassignR(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([],[assignR(Ref,S,T,R)]), fail.
previousRoundsData:- yesassign4(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([],[assign4(Ref,S,T,R)]), fail.
previousRoundsData:- yesassignVAR(Ref,S,T,R), ref(Ref), game(S,T,R),
    writeClause([],[assignVAR(Ref,S,T,R)]), fail.
previousRoundsData:- yesassignAR(A,S,T,R), as(A), game(S,T,R),
    writeClause([],[assignAR(A,S,T,R)]), fail.
previousRoundsData:- yesassignAVAR(A,S,T,R), as(A), game(S,T,R),
    writeClause([],[assignAVAR(A,S,T,R)]), fail.
previousRoundsData:- yesmainRefereeWR(Ref,R), ref(Ref), numRounds(N),
    between(1,N,R), writeClause([],[mainRefereeWR(Ref,R)]), fail.
previousRoundsData:- yesmainAssistantWR(A,R), as(A), numRounds(N),
    between(1,N,R), writeClause([],[mainAssistantWR(A,R)]), fail.
previousRoundsData:- yesrefereeWR(Ref,R), ref(Ref), numRounds(N),
    between(1,N,R), writeClause([],[refereeWR(Ref,R)]), fail.
previousRoundsData:- yesassistantWR(A,R), as(A), numRounds(N),
    between(1,N,R), writeClause([],[assistantWR(A,R)]), fail.

previousRoundsData.

%-------------------------------------------------------------------

everyMatchHas1Referee:-
    round(R), match(S,T,R), equip(T), equip(S),
    findall(assignR(Ref,S,T,R), ref(Ref), Sum), writeConstraint(Sum = 1), fail.
everyMatchHas1Referee:-
    round(R), match2(S,T,R), equip(T), equip(S),
    findall(assignR(Ref,S,T,R),ref(Ref), Sum), writeConstraint(Sum = 1), fail.
everyMatchHas1Referee.

```

```
everyMatchHas2Assistants:-
    round(R), match(S,T,R), equip(T), equip(S),
    findall(assignAR(A,S,T,R), as(A), Sum), writeConstraint(Sum = 2), fail.
everyMatchHas2Assistants:-
    round(R), match2(S,T,R), equip(T), equip(S),
    findall(assignAR(A,S,T,R), as(A), Sum), writeConstraint(Sum = 2), fail.
everyMatchHas2Assistants.

everyMatchHas1Var:-
    round(R), match(S,T,R), equip(T), equip(S),
    findall(assignVAR(Ref,S,T,R), ref(Ref), Sum),
    writeConstraint(Sum = 1), fail.
everyMatchHas1Var.

matches2DontHaveVar:-
    findall(assignVAR(Ref,S,T,R),(ref(Ref), match2(S,T,R), equip(T), equip(S)), Sum),writeConstraint(Sum = 0).

everyMatchHas1AVar:-
    round(R), match(S,T,R), equip(T), equip(S),
    findall(assignAVAR(A,S,T,R),as(A),Sum), writeConstraint(Sum = 1), fail.
everyMatchHas1AVar.

matches2DontHaveAVar:-
    findall(assignAVAR(Ref,S,T,R),(ref(Ref), match2(S,T,R), equip(T), equip(S)), Sum), writeConstraint(Sum = 0).

everyMatchHasA4thRef:-
    round(R), match(S,T,R), equip(T), equip(S),
    findall(assign4(Ref,S,T,R), ref(Ref), Sum), writeConstraint(Sum = 1), fail.
everyMatchHasA4thRef:-
    round(R), match2(S,T,R),equip(T), equip(S),
    findall(assign4(Ref,S,T,R), ref(Ref), Sum), writeConstraint(Sum = 1), fail.
everyMatchHasA4thRef.

%

atMostOneMainRefereeRolePerRound:-
    ref(Ref), round(R),
    findall(assignR(Ref,S,T,R), (game(S,T,R), equip(T), equip(S)), Sum),
    writeConstraint(Sum =< 1), fail.
atMostOneMainRefereeRolePerRound.

atMostOneMainAssistingRolePerRound:-
    as(A), round(R),
    findall(assignAR(A,S,T,R), (game(S,T,R), equip(T), equip(S)),  Sum),
    writeConstraint(Sum =< 1), fail.
atMostOneMainAssistingRolePerRound.

atMostTwoRefereeRolePerRound:-
    ref(Ref), round(R),
    findall(assignR(Ref,S,T,R), (equip(T), equip(S), game(S,T,R)), Sum1),
```

```
    findall(assign4(Ref,S,T,R), (equip(T), equip(S), game(S,T,R)), Sum2),
    findall(assignVAR(Ref,S,T,R), (equip(T), equip(S), game(S,T,R)), Sum3),
    append(Sum1,Sum2,Sum12), append(Sum12,Sum3,Sum),
    writeConstraint(Sum =< 2), fail.
atMostTwoRefereeRolePerRound.

atMostTwoAssistantRolePerRound:-
    as(A), round(R),
    findall(assignAR(A,S,T,R), (equip(T), equip(S),  game(S,T,R)), Sum1),
    findall(assignAVAR(A,S,T,R), (equip(T), equip(S), game(S,T,R)), Sum2),
    append(Sum1, Sum2, Sum), writeConstraint(Sum =< 1), fail.
atMostTwoAssistantRolePerRound.

oneRefereeRolePerMatch:-
    ref(Ref), round(R), match(S,T,R), equip(T), equip(S),
    writeConstraint([assignR(Ref,S,T,R),assign4(Ref,S,T,R),
        assignVAR(Ref,S,T,R)] =< 1), fail.
oneRefereeRolePerMatch:-
    ref(Ref), round(R), match2(S,T,R), equip(T), equip(S),
    writeConstraint([assignR(Ref,S,T,R),assign4(Ref,S,T,R),
        assignVAR(Ref,S,T,R)] =< 1), fail.
oneRefereeRolePerMatch.

oneAssistantRolePerMatch:-
    as(A), round(R), match(S,T,R),equip(T), equip(S),
    writeClause([-assignAR(A,S,T,R),-assignAVAR(A,S,T,R)],[]), fail.
oneAssistantRolePerMatch:-
    as(A), round(R), match2(S,T,R),equip(T), equip(S),
    writeClause([-assignAR(A,S,T,R),-assignAVAR(A,S,T,R)],[]), fail.
oneAssistantRolePerMatch.

% ------------------------------------------------------------------------

defineMainRefereeWR:-
    ref(Ref), round(R), findall(assignR(Ref,S,T,R),(equip(T), equip(S),
    game(S,T,R)),Lits), expressOr(mainRefereeWR(Ref,R),Lits),fail.
defineMainRefereeWR.

defineMainAssistantWR:-
    as(A), round(R), findall(assignAR(A,S,T,R), (equip(T), equip(S),
    game(S,T,R)), Lits), expressOr(mainAssistantWR(A,R),Lits), fail.
defineMainAssistantWR.

defineRefereeWR:- ref(Ref), round(R),
    findall(assignR(Ref,S,T,R),(equip(T), equip(S), game(S,T,R)),Lits1),
    findall(assign4(Ref,S,T,R),(equip(T), equip(S), game(S,T,R)),Lits2),
    findall(assignVAR(Ref,S,T,R),(equip(T), equip(S), game(S,T,R)),Lits3),
    append(Lits1,Lits2,Lits12), append(Lits12,Lits3,Lits),
    expressOr(refereeWR(Ref,R),Lits),fail.
defineRefereeWR.

defineAssistantWR:- as(A), round(R),
```

```
    findall(assignAR(A,S,T,R),(equip(T), equip(S), game(S,T,R)),Lits1),
    findall(assignAVAR(A,S,T,R),(equip(T), equip(S), game(S,T,R)),Lits2),
    append(Lits1,Lits2,Lits), expressOr(assistantWR(A,R),Lits),fail.
defineAssistantWR.

% ------------------------------------------------------------------

incompatibilityRefereeRound:-
    incRefRound(Ref,R), ref(Ref), round(R),
    writeClause([-refereeWR(Ref,R)],[]),  fail.
incompatibilityRefereeRound.

incompatibilityAssistantRound:-
    as(A), incAsRound(A,R), round(R), writeClause([-assistantWR(A,R)],[]), fail.
incompatibilityAssistantRound.

incompatibilityRefereeTeam:-
    ref(Ref), incRefTeam(Ref,T), game(T,S,R), round(R),
    writeClause([-assignR(Ref,T,S,R)],[]),
    writeClause([-assignVAR(Ref,T,S,R)],[]),
    writeClause([-assign4(Ref,T,S,R)],[]), fail.
incompatibilityRefereeTeam:-
    ref(Ref), incRefTeam(Ref,T), game(S,T,R), round(R),
    writeClause([-assignR(Ref,S,T,R)],[]),
    writeClause([-assignVAR(Ref,S,T,R)],[]),
    writeClause([-assign4(Ref,S,T,R)],[]), fail.
incompatibilityRefereeTeam.

incompatibilityAssistantTeam:-
    as(A), incAsTeam(A,T), game(T,S,R), round(R),
    writeClause([-assignAR(A,T,S,R)],[]),
    writeClause([-assignAVAR(A,T,S,R)],[]), fail.
incompatibilityAssistantTeam:-
    as(A), incAsTeam(A,T), game(S,T,R), round(R),
    writeClause([-assignAR(A,S,T,R)],[]),
    writeClause([-assignAVAR(A,S,T,R)],[]), fail.
incompatibilityAssistantTeam.

trioMustWorkTogether:-
    trio(Ref,A1,A2), round(R), game(S,T,R),
    writeClause([-assignR(Ref,S,T,R)],[assignAR(A1,S,T,R)]),
    writeClause([-assignR(Ref,S,T,R)],[assignAR(A2,S,T,R)]),
    writeClause([-assignAR(A1,S,T,R)],[assignR(Ref,S,T,R)]),
    writeClause([-assignAR(A2,S,T,R)],[assignR(Ref,S,T,R)]), fail.
trioMustWorkTogether.

% ------------------------------------------------------------------

atLeastMinRoundsRepeatingTeam:-
    minNumRoundsBeforeRepeatingTeam(MinR),
```

```
    initialRound(I), endingRound(E),
    A is I-MinR+1, max(1,A,M), ref(Ref),
    match(S1,T1,R1), equip(T1), equip(S1), between(M,E,R1),
    match(S2,T2,R2), equip(T2), equip(S2), round(R2),
    R2 > R1, MinR >= R2-R1, sort([S1,T1,S2,T2],L), L\=[_,_,_,_],
    writeClause([-assignR(Ref,S1,T1,R1),-assignR(Ref,S2,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingTeam:-
    minNumRoundsBeforeRepeatingTeam(MinR),
    initialRound(I), endingRound(E),
    A is I-MinR+1, max(1,A,M), ref(Ref),
    match2(S1,T1,R1), equip(T1), equip(S1), between(M,E,R1),
    match2(S2,T2,R2), equip(T2), equip(S2), round(R2),
    R2 > R1, MinR >= R2-R1, sort([S1,T1,S2,T2],L), L\=[_,_,_,_],
    writeClause([-assignR(Ref,S1,T1,R1),-assignR(Ref,S2,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingTeam:-
    minNumRoundsBeforeRepeatingTeam(MinR),
    initialRound(I), endingRound(E),
    A is I-MinR+1, max(1,A,M), as(AR),
    match(S1,T1,R1), equip(T1), equip(S1), between(M,E,R1),
    match(S2,T2,R2), equip(T2), equip(S2), round(R2),
    R2 > R1, MinR >= R2-R1, sort([S1,T1,S2,T2],L), L\=[_,_,_,_],
    writeClause([-assignAR(AR,S1,T1,R1),-assignAR(AR,S2,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingTeam:-
    minNumRoundsBeforeRepeatingTeam(MinR),
    initialRound(I), endingRound(E),
    A is I-MinR+1, max(1,A,M), as(AR),
    match2(S1,T1,R1), equip(T1), equip(S1), between(M,E,R1),
    match2(S2,T2,R2), equip(T2), equip(S2), round(R2),
    R2 > R1, MinR >= R2-R1, sort([S1,T1,S2,T2],L), L\=[_,_,_,_],
    writeClause([-assignAR(AR,S1,T1,R1),-assignAR(AR,S2,T2,R2)],[]), fail.
atLeastMinRoundsRepeatingTeam.

atMostMaxMatchesPerIntervalOfRounds:-
    intervalRounds(N), maxMatchesPerInterval(Max), initialRound(I),
    endingRound(E), A is I-N+1, max(1,A,M), ref(Ref), between(M,E,R1),
    R2 is R1+N-1, round(R2),
    findall( mainRefereeWR(Ref,R), between(R1,R2,R), Sum),
    writeConstraint(Sum =< Max), fail.
atMostMaxMatchesPerIntervalOfRounds:-
    intervalRounds(N), maxMatchesPerInterval(Max), initialRound(I),
    endingRound(E), A is I-N+1, max(1,A,M), as(AR), between(M,E,R1),
    R2 is R1+N-1, round(R2),
    findall( mainAssistantWR(AR,R), between(R1,R2,R), Sum),
    writeConstraint(Sum =< Max), fail.
atMostMaxMatchesPerIntervalOfRounds.

%------------------------------------------------------------------

validRound1(R):- round(R), match(S,T,R), equip(T), equip(S).

validRound2(R):- round(R), match2(S,T,R), equip(T), equip(S).

```

```
refereesLevelRatioPerRound:-
% Proporci desitjada d' rbitres de 1a : 7 de classe S i 2 de classe J o M
    validRound1(R), findall(assignR(Ref,S,T,R), (match(S,T,R), equip(T),
    equip(S), referee(Ref,_,s,_)), Sum),writeConstraint(Sum = 7), fail.
refereesLevelRatioPerRound:-
% Els arbitres de 1a no poden ser de classe T
    validRound1(R), match(S,T,R), equip(T), equip(S), referee(Ref,_,t,_),
    writeClause([-assignR(Ref,S,T,R)],[]),
    writeClause([-assignVAR(Ref,S,T,R)],[]),
    writeClause([-assign4(Ref,S,T,R)],[]), fail.
refereesLevelRatioPerRound:-
% Proporci desitjada d' rbitres de 2a : 8 de classe M i 2 de classe J o S
    validRound2(R), findall(assignR(Ref,S,T,R), (match2(S,T,R),equip(T), equip(S), referee(Ref,_,m,_)), Sum),writeConstraint(Sum = 8), fail.
refereesLevelRatioPerRound:-
% Els arbitres principals de 2a no poden ser de classe T
    validRound2(R), match2(S,T,R), equip(T), equip(S), referee(Ref,_,t,_),
    writeClause([-assignR(Ref,S,T,R)],[]), fail.
refereesLevelRatioPerRound:-
% Proporci desitjada del 4t rbitre de 1a : 7 han de ser de classe J o M
    validRound1(R), findall(assign4(Ref,S,T,R), (match(S,T,R), equip(T), equip(S),referee(Ref,_,s,_)), Sum),writeConstraint(Sum = 2), fail.
refereesLevelRatioPerRound:-
% Proporci desitjada del 4t rbitre de 2a : els 10 han de ser de classe T
    validRound2(R), findall(assign4(Ref,S,T,R), (match2(S,T,R), referee(Ref,_,t,_)), Sum),writeConstraint(Sum = 10), fail.
refereesLevelRatioPerRound:-
% Proporci desitjada del VAR : 4 de classe S i la resta de classe M o J
    validRound1(R), findall(assignVAR(Ref,S,T,R), (match(S,T,R), equip(T), equip(S), referee(Ref,_,s,_)), Sum),writeConstraint(Sum >= 3),
    writeConstraint(Sum =< 6), fail.
refereesLevelRatioPerRound.


assistantsLevelRatioPerRound:-
% els assistents de 1a no poden ser de classe T
    validRound1(R), match(S,T,R), equip(T), equip(S), assistant(A,_,t,_),
    writeClause([-assignAR(A,S,T,R)],[]), fail.
assistantsLevelRatioPerRound:-
% Proporci desitjada d'assistents de 1a : 14 de classe S i 4 de classe M o J
    validRound1(R), findall(assignAR(A,S,T,R), (match(S,T,R), equip(T), equip(S), assistant(A,_,s,_)), Sum),writeConstraint(Sum = 14), fail.
assistantsLevelRatioPerRound:-
% Proporci desitjada d'assistents de 2a : 10 de classe M i 8 de classe J, S o T
    validRound2(R), findall(assignAR(A,S,T,R), (match2(S,T,R), assistant(A,_,m,_)), Sum),writeConstraint(Sum >= 8),
    writeConstraint(Sum =< 14), fail.
assistantsLevelRatioPerRound:-
% Proporci desitjada d'assistents de 2a : 10 de classe M i 8 de classe J, S o T
```

```
    validRound2(R), findall(assignAR(A,S,T,R), (match2(S,T,R), assistant(A,_,t,_)), Sum),writeConstraint(Sum =< 2), fail.
assistantsLevelRatioPerRound:-
% Proporci  desitjada del AVAR : m nim 8 de classe S
    validRound1(R), findall(assignAVAR(A,S,T,R), (match(S,T,R),equip(T), equip(S), assistant(A,_,s,_)), Sum),writeConstraint(Sum >= 8), fail.
assistantsLevelRatioPerRound.

% --------------------------------------------------------------------------


defineImportanceOfMatches:-
    ref(Ref), endingRound(E), between(1,E,R),
    findall(assignR(Ref,S,T,R), eerstematch(R,_,S,T,_,0), Lits),
    expressOr(punctuation0MR(Ref,R),Lits),fail.
defineImportanceOfMatches:-
    ref(Ref), endingRound(E), between(1,E,R),
    findall(assignR(Ref,S,T,R), eerstematch(R,_,S,T,_,1), Lits),
    expressOr(punctuation1MR(Ref,R),Lits),fail.
defineImportanceOfMatches:-
    ref(Ref), endingRound(E), between(1,E,R),
    findall(assignR(Ref,S,T,R), erematch(R,_,S,T,_,2), Lits),
    expressOr(punctuation2MR(Ref,R),Lits),fail.
defineImportanceOfMatches:-
    ref(Ref), endingRound(E), between(1,E,R),
    findall(assignR(Ref,S,T,R), erematch(R,_,S,T,_,3), Lits),
    expressOr(punctuation3MR(Ref,R),Lits),fail.
defineImportanceOfMatches:-
    ref(Ref), endingRound(E), between(1,E,R),
    findall(assignR(Ref,S,T,R), erematch(R,_,S,T,_,4), Lits),
    expressOr(punctuation4MR(Ref,R),Lits),fail.
defineImportanceOfMatches:-
    as(A), endingRound(E), between(1,E,R),
    findall(assignAR(A,S,T,R), eerstematch(R,_,S,T,_,0), Lits),
    expressOr(punctuation0MA(A,R),Lits),fail.
defineImportanceOfMatches:-
    as(A), endingRound(E), between(1,E,R),
    findall(assignAR(A,S,T,R), eerstematch(R,_,S,T,_,1), Lits),
    expressOr(punctuation1MA(A,R),Lits),fail.
defineImportanceOfMatches:-
    as(A), endingRound(E), between(1,E,R),
    findall(assignAR(A,S,T,R), erematch(R,_,S,T,_,2), Lits),
    expressOr(punctuation2MA(A,R),Lits),fail.
defineImportanceOfMatches:-
    as(A), endingRound(E), between(1,E,R),
    findall(assignAR(A,S,T,R), erematch(R,_,S,T,_,3), Lits),
    expressOr(punctuation3MA(A,R),Lits),fail.
defineImportanceOfMatches:-
    as(A), endingRound(E), between(1,E,R),
    findall(assignAR(A,S,T,R), erematch(R,_,S,T,_,4), Lits),
    expressOr(punctuation4MA(A,R),Lits),fail.
defineImportanceOfMatches.
```

```prolog

% It only applies to s level referees and assistants
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, referee(Ref,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation0MR(Ref,R1), between(R,R2,R1), Lits0),
    findall(punctuation1MR(Ref,R1), between(R,R2,R1), Lits1),
    append(Lits0,Lits1,Lits), writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, referee(Ref,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation2MR(Ref,R1), between(R,R2,R1), Lits),
    writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, referee(Ref,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation3MR(Ref,R1), between(R,R2,R1), Lits3),
    findall(punctuation4MR(Ref,R1), between(R,R2,R1), Lits4),
    append(Lits3,Lits4,Lits),
    writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, assistant(A,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation0MA(A,R1), between(R,R2,R1), Lits0),
    findall(punctuation1MA(A,R1), between(R,R2,R1), Lits1),
    append(Lits0,Lits1,Lits),
    writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, assistant(A,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation2MA(A,R1), between(R,R2,R1), Lits),
    writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches:-
    maxRoundsWithoutACertainLevelMatch(Max), initialRound(I), endingRound(E),
    RM is I-Max+1, max(1,RM,M), N is E-Max+1, assistant(A,_,s,_),
    between(M,N,R), R2 is R+Max-1,
    findall(punctuation3MA(A,R1), between(R,R2,R1), Lits3),
    findall(punctuation4MA(A,R1), between(R,R2,R1), Lits4),
    append(Lits3,Lits4,Lits),
    writeConstraint(Lits >= 1), fail.
maxRoundsWithoutCertainLevelMatches.

% ------------------------------------------------------------------------

differentLevelVars:-
    refP(Ref1,P1), refP(Ref2,P2), Ref1 > Ref2,
    definePointsVarsR(Ref1,P1,Ref2,P2), fail.
```

```
differentLevelVars:-
    asP(A1,P1), asP(A2,P2), A1 > A2, definePointsVarsA(A1,P1,A2,P2), fail.
differentLevelVars.

definePointsVarsR(Ref1,P1,Ref2,P2):-
    P1 > P2, !, endingRound(E),
    findall( 1*punctuation0MR(Ref1,R),between(1,E,R),R1S0),
    findall( 2*punctuation1MR(Ref1,R),between(1,E,R),R1S1),
    findall( 3*punctuation2MR(Ref1,R),between(1,E,R),R1S2),
    findall( 4*punctuation3MR(Ref1,R),between(1,E,R),R1S3),
    findall( 5*punctuation4MR(Ref1,R),between(1,E,R),R1S4),
    findall(-1*punctuation0MR(Ref2,R),between(1,E,R),R2S0),
    findall(-2*punctuation1MR(Ref2,R),between(1,E,R),R2S1),
    findall(-3*punctuation2MR(Ref2,R),between(1,E,R),R2S2),
    findall(-4*punctuation3MR(Ref2,R),between(1,E,R),R2S3),
    findall(-5*punctuation4MR(Ref2,R),between(1,E,R),R2S4),
    append(R1S0,R2S0,S0), append(R1S1,R2S1,S1), append(R1S2,R2S2,S2),
    append(R1S3,R2S3,S3), append(R1S4,R2S4,S4), append(S0,S1,S01),
    append(S2,S3,S23), append(S01,S23,S0123),append(S0123,S4,Sum),
    writeConstraint( [+1000 * dpVarR(Ref1,Ref2) | Sum ] >= 0).

definePointsVarsR(Ref1,_,Ref2,_):-
    endingRound(E),
    findall(-1*punctuation0MR(Ref1,R),between(1,E,R),R1S0),
    findall(-2*punctuation1MR(Ref1,R),between(1,E,R),R1S1),
    findall(-3*punctuation2MR(Ref1,R),between(1,E,R),R1S2),
    findall(-3*punctuation3MR(Ref1,R),between(1,E,R),R1S3),
    findall(-4*punctuation4MR(Ref1,R),between(1,E,R),R1S4),
    findall( 1*punctuation0MR(Ref2,R),between(1,E,R),R2S0),
    findall( 2*punctuation1MR(Ref2,R),between(1,E,R),R2S1),
    findall( 3*punctuation2MR(Ref2,R),between(1,E,R),R2S2),
    findall( 4*punctuation3MR(Ref2,R),between(1,E,R),R2S3),
    findall( 5*punctuation4MR(Ref2,R),between(1,E,R),R2S4),
    append(R1S0,R2S0,S0), append(R1S1,R2S1,S1), append(R1S2,R2S2,S2),
    append(R1S3,R2S3,S3), append(R1S4,R2S4,S4), append(S0,S1,S01),
    append(S2,S3,S23), append(S01,S23,S0123),append(S0123,S4,Sum),
    writeConstraint( [+1000 * dpVarR(Ref1,Ref2) | Sum ] >= 0).

definePointsVarsA(A1,P1,A2,P2):-
    P1 > P2, !, endingRound(E),
    findall( 1*punctuation0MA(A1,R),between(1,E,R),A1S0),
    findall( 2*punctuation1MA(A1,R),between(1,E,R),A1S1),
    findall( 3*punctuation2MA(A1,R),between(1,E,R),A1S2),
    findall( 4*punctuation3MA(A1,R),between(1,E,R),A1S3),
    findall( 5*punctuation4MA(A1,R),between(1,E,R),A1S4),
    findall(-1*punctuation0MA(A2,R),between(1,E,R),A2S0),
    findall(-2*punctuation1MA(A2,R),between(1,E,R),A2S1),
    findall(-3*punctuation2MA(A2,R),between(1,E,R),A2S2),
    findall(-4*punctuation3MA(A2,R),between(1,E,R),A2S3),
    findall(-5*punctuation4MA(A2,R),between(1,E,R),A2S4),
    append(A1S0,A2S0,S0), append(A1S1,A2S1,S1), append(A1S2,A2S2,S2),
    append(A1S3,A2S3,S3), append(A1S4,A2S4,S4), append(S0,S1,S01),
```

```
    append(S2,S3,S23), append(S01,S23,S0123),append(S0123,S4,Sum),
    writeConstraint( [+1000 * dpVarA(A1,A2) | Sum ] >= 0).


definePointsVarsA(A1,_,A2,_):-
    endingRound(E),
    findall(-1*punctuation0MA(A1,R),between(1,E,R),A1S0),
    findall(-2*punctuation1MA(A1,R),between(1,E,R),A1S1),
    findall(-3*punctuation2MA(A1,R),between(1,E,R),A1S2),
    findall(-4*punctuation3MA(A1,R),between(1,E,R),A1S3),
    findall(-5*punctuation4MA(A1,R),between(1,E,R),A1S4),
    findall( 1*punctuation0MA(A2,R),between(1,E,R),A2S0),
    findall( 2*punctuation1MA(A2,R),between(1,E,R),A2S1),
    findall( 3*punctuation2MA(A2,R),between(1,E,R),A2S2),
    findall( 4*punctuation3MA(A2,R),between(1,E,R),A2S3),
    findall( 5*punctuation4MA(A2,R),between(1,E,R),A2S4),
    append(A1S0,A2S0,S0), append(A1S1,A2S1,S1), append(A1S2,A2S2,S2),
    append(A1S3,A2S3,S3), append(A1S4,A2S4,S4), append(S0,S1,S01),
    append(S2,S3,S23), append(S01,S23,S0123),append(S0123,S4,Sum),
    writeConstraint( [+1000 * dpVarA(A1,A2) | Sum ] >= 0).

numberOfGamesVars:-
    refP(Ref1,P1), refP(Ref2,P2), Ref1>Ref2, defineVarsR(Ref1,P1,Ref2,P2), fail.
numberOfGamesVars:-
    asP(A1,P1), asP(A2,P2), A1 > A2, defineVarsA(A1,P1,A2,P2), fail.
numberOfGamesVars.

defineVarsR(Ref1,P1,Ref2,P2):-
    P1 > P2, !, endingRound(E),
    findall(   mainRefereeWR(Ref1,R), between(1,E,R), Sum1),
    findall(-1*mainRefereeWR(Ref2,R), between(1,E,R), Sum2),
    append(Sum1,Sum2,Sum),
    writeConstraint( [+1000 * dgVarR(Ref1,Ref2) | Sum ] >= 0).
defineVarsR(Ref1,_,Ref2,_):-
    endingRound(E),
    findall(-1*mainRefereeWR(Ref1,R), between(1,E,R), Sum1),
    findall(   mainRefereeWR(Ref2,R), between(1,E,R), Sum2),
    append(Sum1,Sum2,Sum),
    writeConstraint( [+1000 * dgVarR(Ref1,Ref2) | Sum ] >= 0).
defineVarsA(A1,P1,A2,P2):-
    P1 > P2, !, endingRound(E),
    findall(   mainAssistantWR(A1,R), between(1,E,R), Sum1),
    findall(-1*mainAssistantWR(A2,R), between(1,E,R), Sum2),
    append(Sum1,Sum2,Sum),
    writeConstraint( [+1000 * dgVarA(A1,A2) | Sum ] >= 0).
defineVarsA(A1,_,A2,_):-
    endingRound(E),
    findall(-1*mainAssistantWR(A1,R), between(1,E,R), Sum1),
    findall(   mainAssistantWR(A2,R), between(1,E,R), Sum2),
    append(Sum1,Sum2,Sum),
    writeConstraint( [+1000 * dgVarA(A1,A2) | Sum ] >= 0).
```

```
menysDe4diesDeDiferencia(mon,tue).
menysDe4diesDeDiferencia(mon,wed).
menysDe4diesDeDiferencia(mon,thu).
menysDe4diesDeDiferencia(sun,tue).
menysDe4diesDeDiferencia(sun,wed).
menysDe4diesDeDiferencia(sat,tue).
menysDe4diesDeDiferencia(tue,fri).
menysDe4diesDeDiferencia(wed,sat).
menysDe4diesDeDiferencia(wed,fri).
menysDe4diesDeDiferencia(thu,fri).
menysDe4diesDeDiferencia(thu,sat).
menysDe4diesDeDiferencia(thu,sun).


minimize2gamesIn4days:-
    endingRound(E), ref(Ref), between(1,E,R1), R1 < E, R2 is R1+1,
    partit(R1,S1,T1,D1), partit(R1,S2,T2,D2),
    menysDe4diesDeDiferencia(D1,D2),
    writeClause([-assignR(Ref,S1,T1,R1)],[pen2gi4dR(Ref,R1)]),
    writeClause([-assignR(Ref,S2,T2,R2)],[pen2gi4dR(Ref,R1)]), fail.
minimize2gamesIn4days:-
    endingRound(E), as(A), between(1,E,R1), R1 < E, R2 is R1+1,
    partit(R1,S1,T1,D1), partit(R1,S2,T2,D2),
    menysDe4diesDeDiferencia(D1,D2),
    writeClause([-assignAR(A,S1,T1,R1)],[pen2gi4dA(A,R1)]),
    writeClause([-assignAR(A,S2,T2,R2)],[pen2gi4dA(A,R1)]), fail.
minimize2gamesIn4days.


% ================================= Writting =================================

%writeObjectiveFunction:- write('obj: 0 x'), nl,!.
writeObjectiveFunction:- write('obj: '), ref(Ref1), ref(Ref2), Ref1 > Ref2,
    write(' + '), write( dgVarR(Ref1,Ref2) ), fail.
writeObjectiveFunction:- ref(Ref1), ref(Ref2), Ref1 > Ref2, write(' + '),
    write( dpVarR(Ref1,Ref2) ), fail.
writeObjectiveFunction:- as(A1), as(A2), A1 > A2, write(' + '),
    write( dgVarA(A1,A2) ), fail.
writeObjectiveFunction:- as(A1), as(A2), A1 > A2, write(' + '),
    write( dpVarA(A1,A2) ), fail.
writeObjectiveFunction:- endingRound(E), ref(Ref), between(1,E,R), R < E,
    write(' + '), write( pen2gi4dR(Ref,R) ), fail.
writeObjectiveFunction:- endingRound(E), as(A), between(1,E,R), R < E,
    write(' + '), write( pen2gi4dA(A,R) ),   fail.
writeObjectiveFunction:- nl.

writeCost(M):- assertz(cost(M)), writeMon( M ), nl,!.

writeIntegerVars.

writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R), game(S,T,R),
```

```
    equip(T), equip(S), write( assignR(Ref,S,T,R)   ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R), game(S,T,R),
    equip(T), equip(S), write( assignAR(A,S,T,R)    ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R), game(S,T,R),
    equip(T), equip(S), write( assign4(Ref,S,T,R)   ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    game(S,T,R),equip(T), equip(S), write( assignVAR(Ref,S,T,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    game(S,T,R),equip(T), equip(S), write( assignAVAR(A,S,T,R)  ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( mainRefereeWR(Ref,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( mainAssistantWR(A,R) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( refereeWR(Ref,R) ),     nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( assistantWR(A,R) ),     nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( punctuation4MR(Ref,R) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( punctuation3MR(Ref,R) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( punctuation2MR(Ref,R) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( punctuation1MR(Ref,R) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E), between(1,E,R),
    write( punctuation0MR(Ref,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( punctuation4MA(A,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( punctuation3MA(A,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( punctuation2MA(A,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( punctuation1MA(A,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E), between(1,E,R),
    write( punctuation0MA(A,R) ), nl, fail.
writeBooleanVars:- ref(Ref1), ref(Ref2), Ref1 > Ref2,
    write( dgVarR(Ref1,Ref2) ), nl, fail.
writeBooleanVars:- ref(Ref1), ref(Ref2), Ref1 > Ref2,
    write( dpVarR(Ref1,Ref2) ), nl, fail.
writeBooleanVars:- as(A1), as(A2), A1 > A2, write( dgVarA(A1,A2) ), nl, fail.
writeBooleanVars:- as(A1), as(A2), A1 > A2, write( dpVarA(A1,A2) ), nl, fail.
writeBooleanVars:- ref(Ref), endingRound(E1), E is E1-1, between(1,E,R),
    write( pen2gi4dR(Ref,R) ), nl, fail.
writeBooleanVars:- as(A),    endingRound(E1), E is E1-1, between(1,E,R),
    write( pen2gi4dA(A,R) ),   nl, fail.
writeBooleanVars.

writeBounds:- ref(Ref), endingRound(E), between(1,E,R), game(S,T,R),  equip(T),
    equip(S),  write('0 <= '), write( assignR(Ref,S,T,R)   ), write(' <= 1'),
    nl, fail.
```

```
writeBounds:- as(A),     endingRound(E), between(1,E,R), game(S,T,R),  equip(T),
    equip(S),  write('0 <= '), write( assignAR(A,S,T,R)    ), write(' <= 1'),
    nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), game(S,T,R),  equip(T),
    equip(S),  write('0 <= '), write( assign4(Ref,S,T,R)   ), write(' <= 1'),
    nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), game(S,T,R), equip(T),
    equip(S),  write('0 <= '), write( assignVAR(Ref,S,T,R) ), write(' <= 1'),
    nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), game(S,T,R), equip(T),
    equip(S),  write('0 <= '), write( assignAVAR(A,S,T,R)  ), write(' <= 1'),
    nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( mainRefereeWR(Ref,R)  ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( mainAssistantWR(A,R)  ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( refereeWR(Ref,R)      ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( assistantWR(A,R)      ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation4MR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation3MR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation2MR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation1MR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation0MR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation4MA(A,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation3MA(A,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation2MA(A,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation1MA(A,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E), between(1,E,R), write('0 <= '),
    write( punctuation0MA(A,R) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref1), ref(Ref2), Ref1 > Ref2, write('0 <= '),
    write( dgVarR(Ref1,Ref2) ), write(' <= 1'), nl, fail.
writeBounds:- ref(Ref1), ref(Ref2), Ref1 > Ref2, write('0 <= '),
    write( dpVarR(Ref1,Ref2) ), write(' <= 1'), nl, fail.
writeBounds:- as(A1), as(A2), A1 > A2, write('0 <= '), write( dgVarA(A1,A2) ),
    write(' <= 1'), nl, fail.
writeBounds:- as(A1), as(A2), A1 > A2, write('0 <= '), write( dpVarA(A1,A2) ),
    write(' <= 1'), nl, fail.
writeBounds:- ref(Ref), endingRound(E1), E is E1-1, between(1,E,R),
    write('0 <= '), write( pen2gi4dR(Ref,R) ), write(' <= 1'), nl, fail.
writeBounds:- as(A),     endingRound(E1), E is E1-1, between(1,E,R),
    write('0 <= '), write( pen2gi4dA(A,R)   ), write(' <= 1'), nl, fail.
```

```prolog
writeBounds.

wl([]).
wl([X|L]):- write(X), write('  '), wl(L),!.


expressOr( Var, Lits ):- member(Lit,Lits), writeClause([ -Lit ], [ Var ]), fail.
expressOr( Var, Lits ):-  writeClause([ -Var ], Lits ),!.

% ========================== DisplaySol ==========================

displaySol(_):- retractall(sol(_,_)), fail.
displaySol(M):- member(X=V,M), assertz(sol(X,V)), fail.

%% Displays for each round all the matches in both divisions with all the assignments
displaySol(_):-
    endingRound(E), between(1,E,W), nl,  write('Week '), write(W), write(':  '),
    nl, displaySolEreMatches(W), displaySolEersteMatches(W), fail.

displaySol(_):- nl, nl, write('======================================'), nl, fail.

%% Dispays the rounds each referee has a match in and the importance of the match
displaySol(_):- write('             '), referee(Ref,_,s,_), writeSpace3(Ref),
    fail.
displaySol(_):- endingRound(E), between(1,E,R), nl, write('Week '), writeSpace3(R), referee(Ref,_,s,_), writeMatchLevel(Ref,R), fail.

displaySol(_):- nl, nl, write('======================================'), nl,
    fail.

%% Displays the punctuation of the referee and the number of matches he has assigned (for each match punctuation)
displaySol(_):-
    referee(Ref,_,s,P), nl, write('Referee '), write( Ref),
        write(' with punctuation '), write(P), write(' - '), endingRound(E),
    findall(punctuation0MR(Ref,R),(sol( punctuation0MR(Ref,R), 1 ),      between(1,E,R)), Lits0), length(Lits0,L0), write(' P0 : '), write(L0),
    findall(punctuation1MR(Ref,R),(sol( punctuation1MR(Ref,R), 1 ), between(1,E,R)), Lits1), length(Lits1,L1), write(' P1 : '), write(L1),
    findall(punctuation2MR(Ref,R),(sol( punctuation2MR(Ref,R), 1 ), between(1,E,R)), Lits2), length(Lits2,L2), write(' P2 : '), write(L2),
    findall(punctuation3MR(Ref,R),(sol( punctuation3MR(Ref,R), 1 ), between(1,E,R)), Lits3), length(Lits3,L3), write(' P3 : '), write(L3),
    findall(punctuation4MR(Ref,R),(sol( punctuation4MR(Ref,R), 1 ), between(1,E,R)), Lits4), length(Lits4,L4), write(' P4 : '), write(L4),
    findall(mainRefereeWR(Ref,R), (sol( mainRefereeWR(Ref,R),1), between(1,E,R)),Lits5), length(Lits5,L5), write(' => total : '), write(L5), fail.

displaySol(_):- nl, nl, write('======================================'), nl,
```

```
    fail.

displaySol(_).

displaySolEreMatches(W):-
    eredivisieRoundWeekEquivalence(W,R1), !,
    nl, write('  Eredivisie (Round '), write(R1), write(') :'), nl,
    displayGames(W).
displaySolEreMatches(_):- nl, write('  There are no Eredivisie matches this week
    .'), nl.

displaySolEersteMatches(W):-
    eersteDivisieRoundWeekEquivalence(W,R2), !,
    nl, write('  Eerste divisie (Round '), write(R2), write(') :'), nl,
    displayGames2(W).
displaySolEersteMatches(_):- nl, write('  There are no Eerste Divisie matches
    this week.'), nl.

eredivisieRoundWeekEquivalence(W,R1):- erematch(W,R1,_,_,_,_),!.
eersteDivisieRoundWeekEquivalence(W,R2):- eerstematch(W,R2,_,_,_,_),!.

displayGames(W):-
    erematch(W,_,S,T,D,_),
    equip(S), equip(T),
    sol( assignR(Ref,S,T,W), 1 ),
    sol( assignAR(A1,S,T,W), 1 ),
    sol( assignAR(A2,S,T,W), 1 ),  A1 < A2,
    sol( assign4(Ref4,S,T,W), 1 ),
    sol( assignVAR(VAR,S,T,W), 1 ),
    sol( assignAVAR(AVAR,S,T,W), 1 ),
    write('     '), writeAssignment(Ref,A1,A2,Ref4,VAR,AVAR,S,T,D), fail.
displayGames(_).

displayGames2(W):-
    eerstematch(W,_,S,T,D,_),
    equip(S), equip(T),
    sol( assignR(Ref,S,T,W), 1 ),
    sol( assignAR(A1,S,T,W), 1 ),
    sol( assignAR(A2,S,T,W), 1 ),  A1 < A2,
    sol( assign4(Ref4,S,T,W), 1 ),
    write('     '), writeAssignment2(Ref,A1,A2,Ref4,S,T,D), fail.
displayGames2(_).

writeAssignment(Ref,A1,A2,Ref4,VAR,AVAR,S,T,D):-
    write(S), write(' - '), write(T),
    %write(' : '),
    write(' ('), write(D), write(') : '),
    write('  R: '), referee(Ref,_,X,_), writeSpace(Ref,X),
    write('A1: '), assistant(A1,_,Y1,_), writeSpace(A1,Y1),
    write('A2: '), assistant(A2,_,Y2,_), writeSpace(A2,Y2),
    write('R4: '), referee(Ref4,_,X4,_), writeSpace(Ref4,X4),
    write('VAR: '), referee(VAR,_,XV,_), writeSpace(VAR,XV),
```

```
    write('AVAR: '), assistant(AVAR,_,YV,_), writeSpace(AVAR,YV), nl.

writeAssignment2(Ref,A1,A2,Ref4,S,T,D):-
    write(S), write(' - '), write(T),
    %write(' : '),
    write(' ('), write(D), write(') : '),
    write('  R: '), referee(Ref,_,X,_), writeSpace(Ref,X),
    write('A1: '), assistant(A1,_,Y1,_), writeSpace(A1,Y1),
    write('A2: '), assistant(A2,_,Y2,_), writeSpace(A2,Y2),
    write('R4: '), referee(Ref4,_,X4,_), writeSpace(Ref4,X4), nl.

writeMatchLevel(Ref,R):- sol( punctuation0MR(Ref,R), 1 ), !, write('  0  ').
writeMatchLevel(Ref,R):- sol( punctuation1MR(Ref,R), 1 ), !, write('  1  ').
writeMatchLevel(Ref,R):- sol( punctuation2MR(Ref,R), 1 ), !, write('  2  ').
writeMatchLevel(Ref,R):- sol( punctuation3MR(Ref,R), 1 ), !, write('  3  ').
writeMatchLevel(Ref,R):- sol( punctuation4MR(Ref,R), 1 ), !, write('  4  ').
writeMatchLevel(_,_):- write('     ').

writeSpace(N,X):- N > 9, !, write(N), write('('), write(X), write(')'),
    write('   ').
writeSpace(N,X):- write(N), write('('), write(X), write(')'), write('    ').

writeSpace2(N):- N > 9, !, write(' '), write(N).
writeSpace2(N):- write('  '), write(N).

writeSpace3(N):- N > 9, !, write(' '), write(N), write('  ').
writeSpace3(N):- write('  '), write(N), write('  ').

% ========================= No need to change the following:
    ↪ ==========================


main:- symbolicOutput(1),!,
       /*planningMonth(Mes),
%    current_prolog_flag(argv,[_,Mes|_]),
       write(planningMonth-Mes), nl,
    retractall(month(_)), assertz(month(Mes)),*/
    writeConstraints, nl, halt.
main:-
%    current_prolog_flag(argv,[_,Mes|_]),
    /*planningMonth(Mes),
    write(planningMonth-Mes), nl,
    retractall(month(_)),
    assertz(month(Mes)),*/
    unix('rm -f solCplex.sol fileForCplex salCplex c.lp cplex.log'),
    write('generating constraints...'),nl,

    tell('c.lp'),
    write('Minimize'    ), nl,   writeObjectiveFunction,
    write('Subject To' ), nl,   writeConstraints,
    write('Bounds'      ), nl,   writeBounds,
    write('Generals'    ), nl,   writeIntegerVars,
```

```
    write('Binary'      ), nl,    writeBooleanVars,
    write('End'         ), nl,    told,
    write('constraints generated'),nl,nl,nl,nl,


    tell(fileForCplex), maxComputationTime(T),
    write('read c.lp'), nl,
    write('set timelimit '), write(T), write(' s'), nl,
    write('set mip tolerance mipgap 0.03.  '), nl,
    write('opt'), nl, write('write solCplex.sol'), nl, write('quit'), nl, told,
   %   unix(' cplex < fileForCplex > salCplex'),
    unix('cplex < fileForCplex ;')
    checkIfSolution, nl,nl,
    halt.
main:-      write('constraints generation failed'),nl, halt.



checkIfSolution:-
    exists_file('solCplex.sol'), !,
    unix('xml2simple.pl solCplex.sol > sol.pl'),
    see('sol.pl'), readModel([],M), seen,
    nl,nl,nl,write('Solution found. Press <enter> to see it'), nl,nl,nl,
    get_char(_),
    identifier(Id),
    tell('sol.txt'), write(Id), nl, nl, displaySol(M), told,
    displaySol(M),!.
checkIfSolution:- shell('grep "Integer infeasible" cplex.log > salgrep', 0), nl, nl, %grep returns 0
    write('Solver: No solution exists'),!.
checkIfSolution:- maxComputationTime(T), nl,nl,
    write('Solver: No solution found under the given time limit of '), write(T),
    write(' s.'),!.

unix(Command):-shell(Command),!.
unix(_).

writeConstraint(C):- C =.. [Op,Sum,K], writeSum(Sum), write('  '), writeOp(Op), write('  '), write(K), nl.
writeSum([]):-  !.
writeSum([M|L]):-  writeMon(M), nl, writeSum(L), !.
writeMon(A*X):- A>=0, !,              write(' + '), write(A ), write(' '), write(X ), !.
writeMon(A*X):- A<0, !,  AB is -A,    write(' - '), write(AB), write(' '), write(X ), !.
writeMon(X):- !,                      write(' + '), write(1 ), write(' '), write(X ), !.

writeClause( Neg, _   ):-  member(Lit,Neg), Lit \= -_,
    write(error('negative lit')), nl, halt.
writeClause(   _, Pos ):-  member(Lit,Pos), Lit  = -_,
    write(error('positive lit')), nl, halt.
```

```
writeClause( Neg, Pos ):-  length(Neg,N), K is 1-N,
    findall( -1*Lit, member(-Lit,Neg), NegLits ), append(NegLits,Pos,Sum),
    writeConstraint( Sum >= K ),!.

readModel(L1,L2):- read(XV), addIfNeeded(XV,L1,L2),!.
addIfNeeded(end_of_file,L,L):-!.
addIfNeeded(XV,L1,L2):- readModel([XV|L1],L2),!.

writeOp(=<):-write('<='),!.
writeOp(Op):-write(Op),!.
```

Listing 24: C++ program to prepare the data from previous sub-problems

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    ifstream inFile;
    inFile.open("sol.pl");
    ofstream outFile;
    outFile.open("previousRounds.pl");
    string input;
    string sign;
    string value;
    inFile >> input >> sign >> value; // x = 0.
    while (inFile >> input) {
        inFile >> sign >> value;
        if (value == "0.") outFile << "not" << input << "." << endl;
        else outFile << "yes" << input << "." << endl;
    }
}
```

Listing 25: File for if no data from previous rounds is needed

```
notassignR(0,0,0,0).
notassign4(0,0,0,0).
notassignVAR(0,0,0,0).
notassignAR(0,0,0,0).
notassignAVAR(0,0,0,0).
notmainRefereeWR(0,0).
notmainAssistantWR(0,0).
notrefereeWR(0,0).
notassistantWR(0,0).
notpunctuation4MR(0,0).
notpunctuation3MR(0,0).
notpunctuation2MR(0,0).
notpunctuation1MR(0,0).
notpunctuation0MR(0,0).
notpunctuation4MA(0,0).
notpunctuation3MA(0,0).
notpunctuation2MA(0,0).
notpunctuation1MA(0,0).
notpunctuation0MA(0,0).
yesassignR(0,0,0,0).
yesassign4(0,0,0,0).
yesassignVAR(0,0,0,0).
yesassignAR(0,0,0,0).
yesassignAVAR(0,0,0,0).
yesmainRefereeWR(0,0).
yesmainAssistantWR(0,0).
yesrefereeWR(0,0).
yesassistantWR(0,0).
yespunctuation4MR(0,0).
yespunctuation3MR(0,0).
yespunctuation2MR(0,0).
yespunctuation1MR(0,0).
yespunctuation0MR(0,0).
yespunctuation4MA(0,0).
yespunctuation3MA(0,0).
yespunctuation2MA(0,0).
yespunctuation1MA(0,0).
yespunctuation0MA(0,0).
```