Modelling & solving the Sports Tournament Scheduling (STS) problem

Leonardo Billi leonardo.billi@studio.unibo.it,
Leonardo Massaro leonardo.massaro2@studio.unibo.it,
Giorgio Scavello giorgio.scavello@studio.unibo.it

November 28, 2025

1 Introduction

The aim of this project is to resolve a particular instance of the Sports Tournament Scheduling problem. Starting from the work of Kendall et al. (2010), where a variety of models for different methodologies is exposed. The two similarities the models share are the preprocessing method employed which is the circle method and the objective function employed.

1.1 Circle method

The circle method is described as follows in Lambrechts et al. (2018): imagine having n teams where n is even. Then team 1 is fixed at the center of the circle while the remaining teams are placed on the border of it. The team at the top position of the circle is matched with the team at the center. After this we match the first team we find clockwise to the first team counterclockwise. When all the teams are matched then the teams on the circle rotate clockwise and the process is repeated until the teams on the circle are on the initial position.

1.2 Objective function

The function is defined as the largest difference between the home and away games of the teams. Defining the total number of home games for each team t as H_t, being W the number of weeks equal to $N-1$, and reducing the number of away games for team t to A_t, the objective function can be defined as:

$$f(H_t) = \max_t(|2H_t - W|) \tag{1.1}$$

The lower bound of this function is 1 as the minimum difference between home and away games is 1 for each team. The upper bound is obtained when a team plays all the games at home or away, leading the function to be equal to W.

1.3 Machine for experiments

All the experiments for each approach were conducted on a machine with an AMD Ryzen 7 8-core processor running at 4.2 GHz. The computational environment was a Docker container using a slim Debian Linux distribution, and each experimental run was limited to 300 seconds as stated in the problem specs.

2 CP Model

The CP approach began with a naive baseline model, later refined through improved variable encoding, symmetry-breaking constraints, and preprocessing techniques. As outlined in Kendall et al. (2010), the problem can be divided into two stages, scheduling of matches and home/away pattern generation, this allows to reduce the search space providing the home/away model with a correct but unbalaced solution.
The scheduling models progress from the **naive_model** with basic encoding, to **better_model** with improved variable encoding and symmetry breaking constraints, going to **better_model_chan** introducing an alternative channeled encoding, finally to the **circle_met_model** & **circle_met_model_chan**, which incorporate circle method preprocessing to further shrink the search space. Any valid schedule produced by these models can then serve as input to the **HAP_v1_model** & **HAP_v2_model**, which assign balanced home/away patterns minimizing two different versions of ojective functions.

2.1 Decision Variables

- **home_team, away_team** in naive_model: are the most simple decision variables, representing respectively the full calendar of teams that play at home and play away, indexed with the tuple (period, week), such that $home\_team[p, w] = t$, with $t \in [1, n]$.

- **calend** in better_model and better_model_chan: which is the stack of the two home_team and away_team arrays, with the advantage that having a single 3d array doesn't impose an order when search annotations are used.

- **team_at_p** in channeled models better_model_chan and circle_met_model_chan: which is used as an alternative domain representation, which indexes the periods at which a given team plays in a given week. For example $team\_at\_p[t, w] = p$, where $t \in [1, n]$, $w \in [1, n-1]$ and $p \in [1, \frac{n}{2}]$. This will make the constraint of "at most twice in the same period" more efficient.

- **permutation_calendar** used in the circle method models: this decision variable consists in a 2d array of $Periods \times Weeks$ that instead of containing the actual teams, it contains a permutation index $i \in [1, \frac{n}{2}]$, such that with the support of the preprocessed circle method calendar, it can decide at which position each match should appear in a single week. This

means that the degree of freedom of this model is just about permuting the matches inside each week in such a way that it does not break the constraint of "at most twice in the same period".

2.2 Objective Function

The HAP_v1_model and HAP_v2_model both contain an objective function that should be minimized to balance the home/away pattern of the resulting calendar. The v1 model refers to the Section 1.2 function and should be minimized to 1.
The v2 objective function instead evaluates the home/away balance by considering only the number of home matches played by each team.

$$obj\_func2 = \max_{t \in \text{TEAMS}}(\text{home\_games}_t) - \min_{t \in \text{TEAMS}}(\text{home\_games}_t),$$

where

$$\text{home\_games}_t = \sum_{p=1}^{\text{periods}} \sum_{w=1}^{\text{weeks}} \mathbf{1}_{\{\text{calendar}[p,w,\text{home\_team\_index}_{p,w}]=t\}},$$

and $\mathbf{1}_{\{.\}}$ is the indicator function, equal to 1 if the condition is true and 0 otherwise. This objective minimizes the difference between the maximum and minimum number of home games assigned to any team.

Since each team plays an odd number of matches $(n-1)$, the minimum possible value of the function is 1.

2.3 Constraints

In the initial model, all constraints were explicitly stated according to the problem description:

1. **A match requires two different teams**: this is encoded in two different ways, the first is that $\forall p \in PERIODS$ and $\forall w \in WEEKS$, $calendar[p, w, 1] \neq calendar[p, w, 2]$, the second encoding is in the next subsection relative to symmetry breaking constraints.

2. **Each team plays exactly once per week**: this is encoded explicitly in all the models without the circle method preprocessing, where instead it is enforced by design. The encoding of this constraint is achieved through the global constraint alldifferent, which checks that in each week the teams are all different from each other.

3. **Two teams can play against each other just once per tournament**: this constraint, just like the one before, is encoded explicitly only when the circle method is not applied. In the other models, is achieved with the same alldifferent global constraint, generating a unique ID for each match

with the math operation $ID_i = teamA_i \cdot n + teamB_i$ where the i index indicates that there is an ID for each match in the calendar. This method works only if paired with the symmetry breaking constraint that imposes an ordering between indexes of teamA versus teamB.

Symmetry-Breaking Constraints. Many symmetry breaking constraints were tested, but some of them worsened the performances, maybe because they did not propagate well. The following ones have been useful instead:

1. **Fixing the order of teams in each match**: imposing $teamA_i < teamB_i$, $\forall i \in MATCHES$, makes the solver consider only the part of the tree where this condition is met, without invalidating the existence of a solution.
2. **Team 1 plays in the first match of the first week**: this cuts off all the other branches, without compromising the correctness of the search.

2.4 Validation

All models were implemented in MiniZinc 2.9.3 and tested with the bundled solvers `Gecode` and `Chuffed`.

Experimental design. The experiments were performed sequentially, with a main.py entry point that manages the execution of Minizinc models, extracting the results along with the execution time, creating a pipeline of two models in the case of HAP optimization experiments. The main.py deals also with the generation of preprocessed data, executing a circle method algorithm that will feed the circle_met_model .

Experimental Results. In the following table we present the experimental results for each 'n' instance, for a selection of interesting CP models. The column names are the following:

- **v1**: naive_model not balanced, with 'gecode' solver
- **v2**: better_model not balanced, with 'gecode' solver
- **v3**: better_model not balanced, with 'chuffed' solver
- **v4**: better_model_chan not balanced, with 'chuffed' solver
- **v5**: circle_met_model not balanced, with 'chuffed' solver
- **v6**: circle_met_model_chan, not balanced, with 'chuffed' solver
- **v7**: circle_met_model_chan, balanced with HAP_v1, with 'chuffed' solver
- **v8**: circle_met_model_chan, balanced with HAP_v2, with 'chuffed' solver

n	**v1**	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v8**
4	US	US	US	US	US	US	US	US
6	0s	0s	0s	0s	0s	0s	**1**(0s)	**1**(0s)
8	0s	0s	0s	0s	0s	0s	**1**(0s)	**1**(0s)
10	—	0s	0s	0s	0s	0s	**1**(0s)	**1**(0s)
12	—	—	6s	3s	1s	0s	**1**(0s)	**1**(0s)
14	—	—	34s	160s	208s	1s	**1**(1s)	2(299s)
16	—	—	—	—	4s	14s	**1**(14s)	**1**(14s)
18	—	—	—	—	—	39s	**1**(39s)	**1**(39s)
20	—	—	—	—	—	44s	**1**(40s)	3(299s)

Table 1: Experimental results for different values of n. Where the result is in bold it indicates that it was solved to optimality.

3 SMT Model

Starting from a model where the circle method was not used and where a larger number which reached satisfying results in decision for a specific solver, a model was developed using the circle method. Both these were derived from the work of Larson et al. (2014).

3.1 Decision Variables

Linear Integer Arithmetic theory was employed as it was considered to work well in this framework. The used variables are:

- $\text{Per}_{tw} = p$ $t = 0, \cdots, N-1$ and $w = 0, \cdots, W-1$ a series of sort Int variables, which define that team i+1 in week j+1 will play in period p. The domain for this variable is $[1, \cdots, P]$, where $P = \frac{N}{2}$.
- Home_{tw} for $t = 0, \cdots, N-1$ and $w = 0, \cdots, W-1$ is a series of sort Bool variables, which is true when the team i+1 is playing at home in week j+1 and it is false when that team is playing away in a given week.

3.2 Objective function

The function used is the same as the one described in section 1.2. However as the max function introduces non linearity, it was necessary to resort to a particular encoding using minimax optimization imposing the following ordering constraint:

$$f(H_1) \geq f(H_2) \geq \cdots \geq f(H_n) \tag{3.1}$$

where $f(H_1)$ is the function at 1.1 applied on the first team. This constraint, which allows also to break the symmetry obtained by permuting the differences between home and away games for each team, permits to obtain an optimal solution by just minimizing $f(H_1)$ with the procedure explained in 3.4.1.

3.3 Constraints

3.3.1 Main problem constraints

The two models have constraints which are defined in a similar way. The following are the one which were explicitly defined:

- One constraint of the problem says every team plays with every other team only once. This is encoded by definition when using the circle method.
- The second constraint says that every team must play once a week. This is encoded by definition when using the circle method..
- Another constraint says that every team plays at most twice in the same period over the tournament. An indicator function q_{tw}^{p} which takes value 1 if $\text{Per}_{tw} = p$ and 0 otherwise was used to define the following constraint:

$$\bigwedge_{t=0}^{N-1} \bigwedge_{p=1}^{P} \left(\sum_{w=0}^{W-1} q_{tw}^{p} \leq 2 \right) \tag{3.2}$$

There are then structural constraints which must be imposed in order to obtain a correct solution, they are the following:

- When two teams play against each other, they must occupy the same period. Let opp_{iw} denote the opponent of team $i+1$ in week w, as defined by the circle method. Then the constraint is defined as:

$$\bigwedge_{w=0}^{W-1} \bigwedge_{t=0}^{N-1} \left(\text{Per}_{tw} = \text{Per}_{(\text{opp}_{tw}-1)w} \quad \wedge \bigwedge_{\substack{u=0 \\ u \neq t, u \neq \text{opp}_{tw}-1}}^{N-1} \text{Per}_{tw} \neq \text{Per}_{uw} \right) \tag{3.3}$$

  This ensures teams playing together share a period, while teams not matched occupy different periods.

- When two team play against each other then one should play at home and the other away. Similarly to constraint at 3.3, being matches_w the games defined by the circle method for week w, the constraint is the following:

$$\bigwedge_{w=0}^{W-1} \bigwedge_{\text{(u.v) in matches}_w} \text{Home}_{(u-1)w} \oplus \text{Home}_{(v-1)w} \tag{3.4}$$

3.3.2 Implied constraint

The only implied constraint across the two models is in the second model. Here, using the indicator function defined at 3.2, the following constraint which was employed in a previous model is kept:

$$\bigwedge_{w=0}^{W-1} \bigwedge_{p=1}^{P} \left(\sum_{t=0}^{N-1} q_{tw}^{p} = 2 \right) \tag{3.5}$$

This fixes that there should be just two teams which share the same period, this could be obtained thanks to the constraint 3.3 and the definition of the circle method.

3.3.3 Symmetry breaking constraints

The two models have similar symmetry breaking constraints:

- Break the home away flip by fixing the first team at home, meaning that Home_{00} should always hold.
- Break the symmetry obtained from permuting the order of rows by fixing the matches of the first week in given periods. Being u_p and v_p the teams which play against in the p-th match of the first week, the constraint is:

$$\bigwedge_{p=1}^{P} (\text{Per}_{(u_p-1)0} = p \ \wedge \text{Per}_{(v_p-1)0} = p) \tag{3.6}$$

  This fixes the periods of the matches for the first week, preventing row permutations that would yield equivalent solutions.

It is also necessary to say that the symmetry obtained by permuting the weeks of the table is broken by using the circle method which fixes the game which are played in a given week in advance, not allowing the a specific game to be inserted in a week different from the one defined by the circle method.

3.4 Validation

3.4.1 Experimental design

The goal of this approach was to use SMT-LIB in order to make a solver independent encoding which could be used to compare the performances of three different solvers: Z3, CVC5 and OptiMathSAT. It was preferred to use two different packages to write the model in Python: Z3py and PySMT and then use specific functions which are in this packages to translate the code in SMT-LIB. The necessity of using two different packages, and therefore encodings, arose from the fact that Z3py favored Z3 in the performance leading to a biased comparison.
Since not all solvers support optimization, an offline routine was implemented: after each iteration, an upper bound on the objective function defined at 1.1 was imposed, requiring it to be strictly less than the value obtained at previous iteration until the minimum of 1 is reached.
Different configurations for the solver were investigated, after this Z3 phase selection is set to 5 (random phase selection) for instances larger than 14 for both encodings, in all the other cases it was set equal to 4 (caching phase selection). For CVC5 and optimathsat the default configuration was kept for all runs. The seed was set to 90 for all solvers as it was considered the seed which gave the best performances along the three different solvers.

3.4.2 Experimental results

Here a table for the results of the model with each of the 3 solvers in each of the 2 encodings both for decision mode and optimization mode is shown:

n	**pzd**	**pzo**	**zzd**	**zzo**	**pcd**	**pco**	**zcd**	**zco**	**pod**	**poo**	**zod**	**zoo**
4	US	US	US	US	US	US	US	US	US	US	US	US
6	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)
8	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)
10	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)	0s	**1**(0s)
12	0s	**1**(0s)	0s	**1**(0s)	3s	**1**(10s)	2s	**1**(7s)	0s	**1**(0s)	0s	**1**(0s)
14	0s	**1**(4s)	0s	**1**(6s)	7s	**1**(224s)	38s	**1**(77s)	0s	**1**(1s)	0s	**1**(0s)
16	14s	**1**(30s)	1s	**1**(13s)	-	-	-	-	9s	**1**(9s)	7s	**1**(16s)
18	32s	**1**(68s)	1s	7	-	-	-	-	29s	**1**(72s)	38s	**1**(212s)
20	113s	19	35s	19	-	-	-	-	227s	19	98s	19
22	-	-	26s	21	-	-	-	-	-	-	-	-

Table 2: The 12 versions here are all for the model with preprocessing. The first letter is for the encoding: p for PySMT, z for Z3py; the second letter is for the solver: z for Z3, c for CVC5, o for OptiMathSAT; the third letter is either if the result was for decision or optimality. US stands for UNSAT.

OptiMathSAT is the solver which allows the best performance in optimization in both encodings reaching an optimal solution for N=18, it should not be a surprised as it is specifically built for this. Surprisingly Z3 reaches optimization in the encoding with PySMT while timing out for optimization on the Z3py enconding. Z3 is also the best solver in decision as it finds a solution for $n = 22$ with the Z3py encoding.

4 MIP Model

The implementation supports four different variable encodings, but only three are relevant (referred to later as `base/v1`, `i<j/v3`, `prepro/v4`).

4.1 Decision variables

- Full oriented variables (`base/v1`) Binary variables $x_{w,p,i,j} \in \{0,1\}$ for all $w \in W$, $p \in P$, $i \in T$, $j \in T$. Interpretation: $x_{w,p,i,j} = 1$ means in week w, period p, team i plays team j (orientation matters: i at home, j away).

- Unordered pairs (`i<j/v3`) Binary variables only for unordered pairs. When the optimization requires orientation (balanced objective), additional binary orientation variables are introduced, for each variable x:
$x_{w,p,i,j} \in \{0,1\}, h_{w,p,i,j} \in \{0,1\} \quad \text{for } i < j.$

- Preprocessed (`circle method/v4`) Instead of deciding the full oriented assignment $x_{w,p,i,j}$, we precompute the weekly pairs. For each week w and

precomputed pair index k, we introduce binary variables $y_{w,k,p} \in \{0,1\}$ for every period p, and only decide a permutation assigning each pair to exactly one period. When the optimization requires orientation, the additional variable is $h_{w,k,p} \in \{0,1\}$.

Other variables In the model are used also other three continuous variables, with lower bound = 0. s d_t(Imbalance variable for each team),$weight\_week$ and $weight\_period$. They are used to evaluate some constraint and not for branching, their use is later explained.

4.2 Objective function

Two operational modes are implemented:

- **Feasibility mode**: no objective (dummy objective = 0).
- **Optimization/Balanced mode**: During the validation phase, I observed through several experiments that when the objective function's minimum value is set to 0 instead of n or 1, the solver requires less time to reach an optimal solution. Therefore, I reformulated the objective function as follows, introducing d_t and 1:

$$\sum_{t \in T} |diff_{t(home-away)}| \quad = \quad \sum_{t \in T} d_t \quad where \quad -d_t \leq (|diff_{t(home-away)}| - 1) \leq d_t.$$

I did not use the $f_{obj} min - max$ 1.1 because I initially worked with the f_{obj} above, and later validation experiments showed that CBC does not perform significantly better (run MIP $v7$, see README). Only GLPK shows a clear improvement (see Table 4).

4.3 Constraints

4.3.1 Main problem constraints

Let the set of unordered pairs be $E = \{(i,j) : i < j\}$. The main linear constraints are:

(1) One match per (week,period). For every $w \in W$ and $p \in P$, we have respectably for v1, v3 and v4:

$$\sum_{i \in T} \sum_{j \in T} x_{w,p,i,j} = 1 \quad , \quad \sum_{(i,j) \in E} x_{w,p,i,j} = 1 \quad , \quad \sum_{k=0}^{p-1} y_{w,k,p} = 1.$$

(2) Each pair meets exactly once. For every pair/match $(i,j)/k$, for v1, v3 and v4 (v4 is also for each week):

$$\sum_{w \in W} \sum_{p \in P} \left(x_{w,p,i,j} + x_{w,p,j,i}\right) = 1 \quad , \quad \sum_{w \in W} \sum_{p \in P} x_{w,p,i,j} = 1 \quad , \quad \sum_{p \in P} y_{w,k,p} = 1.$$

(3) Each team plays exactly once per week. For every team $t \in T$ and week $w \in W$, for v1 and v3:

$$\sum_{p \in P} \sum_{j \in T,\, j \neq t} \left(x_{w,p,t,j} + x_{w,p,j,t}\right) = 1 \quad , \quad \sum_{p \in P} \sum_{j \in T,\, j \neq t} \left(x_{w,p,t,j}\right) = 1$$

(4) Period cap (balance of periods). For every team $t \in T$ and period $p \in P$, for v1, v3 and v4:

$$\sum_{w \in W} \sum_{j \in T,\, j \neq t} \left(x_{w,p,t,j} + x_{w,p,j,t}\right) \leq 2, \sum_{w \in W} \sum_{j \in T,\, j \neq t} \left(x_{w,p,t,j}\right) \leq 2, \sum_{(w,k) \in \text{map}[t]} y_{w,k,p} \leq 2,$$

(Linking orientation) For $v3$ and $v4$ when balancing (or opt) is enabled, we also introduce:

$$h_{w,p,i,j} \leq x_{w,p,i,j} \qquad \forall w, p, i, j, \qquad \text{or} \qquad h_{w,k,p} \leq y_{w,k,p} \qquad \forall w, k, p.$$

When $x = 1$ or $y = 1$, h may be 0 or 1 to set the match orientation; if $x = 0$, then $h = 0$. If pair k is placed in (w, p), $h_{w,k,p} = 1$ means the *first* team of the pair is at home; otherwise the second team is at home.

4.3.2 Symmetry breaking

A (anchor). Anchor a team $k = 1$ to occupy a match in $(w = 1, p = 1)$: $\sum_{\{i,j\} \ni k} x_{1,1,i,j} = 1$.

B (fix week 1 composition). Fix the set of unordered pairs appearing in week 1 to the circle-method composition: for each pair (a, b) fixed, $\sum_{p \in P} x_{1,p,a,b} = 1$.

C (canonical order of weeks). Define a continuous weight variable $weight\_week = \sigma_w^{\text{week}}$ for each week (weighted sum of team ids participating) and enforce $\sigma_w^{\text{week}} \leq \sigma_{w+1}^{\text{week}}, \quad \forall w$, to order weeks to one canonical representative.

Other symmetry braking, like the permutation of teams of possihypble solution, are taken into consideration by creating the variables in a fixed way, like for the unordered pairs (`i<j`) and `preprocessed` models.

4.3.3 Warm-Start

For the CBC solver there is the possibility to build MIP warm-starts (MIP-Start) by setting initial values for selected binary variables corresponding to the preprocessed schedule. And i tested this implementations:

- **week1**: set only the first week.
- **bal_full** (or for preprocessing version **random_half**): set half of the possible solution found by circol method.
- **half_full** (or for preprocessing version **random_full**): set all slots of the solution found.

4.4 Validation

Before starting, note that all tables come from `source/MIP/TABLES_summeryze.txt`, and solver behaviors are documented in JSON files under `res/MIP/MIP_old`. Also, CBC uses a fixed seed by default; enabling randomness leads to highly variable runtimes for the same model (run MIP $v5$, see README), likely due to internal heuristics, probing, cut selection, and randomized tie-breaking. For CBC I also tried disabling/enabling presolve, but for length reasons these tests were not included and are assumed to be set to `True`.

4.4.1 Test 1

Table 3 illustrates the limitations of the two model, v1 and v3. Key observation

Table 3: Comparison of v1 and v3 across different n values.

n	v1-feas	v1-bal	v3-feas	v3-bal
12	10/10 (4–108s)	**7**/10(obj = **0**)	10/10 (10–60s)	**10**/10(obj = **0**)
14	5/10 (2–279s)	0/10	2/10 (21–229s)	**3**/10(obj = **0**)

is that v3 performs well for balanced-optimization, whereas v1 is superior for feasibility (*realtive json files are stored in res/MIP/MIP_old/table_1*).

4.4.2 Test 2-3

These tests aim to find the best additional constraint and best worm-starts to get better performance from the models. *Realtive json files are stored in res/MIP/MIP_old/table_2 and table_3.*
Best combination found are:
CBC_base_feasible_week1 (**v1C-f**)
CBC_i<j_balanced_week1 (**v3C-o**).

4.4.3 GLPK

Unlike CBC, GLPK (which uses a random seed by default) produced more stable results across runs (*relative json files in res/MIP/MIP_old/test_4*). I tried different flags of GLPK (– cuts, –goromy, –clique – dual –prima (defoult –simplex), run MIP $v6$, see README, to test different behaviors) and the best configurations are:
GLPK_base_feasible_dual_B (**v1G-f**)
GLPK_i<j_balanced_dual_cuts_A (***v3G-o***).

4.4.4 Preprocessing

Finally, I tested another implementation, the preprocessing model (v4). As expected, it outperformed all the previous models in both the CBC/GLPK and feasible/optimal versions. CBC showed strong sensitivity to preprocessing and

warm-start hints. (*realtive json files are stored in res/MIP/MIP_old/test_5*).
CBC_prepro_anchor_balanced_random_half (**v4C-o**),
CBC_prepro_anchor_feasible_week1 (**v4C-f**),
GLPK_prepro_anchor_feasible_dual (**v4G-f**),
GLPK_prepro_anchor_balanced_dual (**v4G-o**).
I tested also the usage of the $f_{obj}min - max$ 1.1 and the relevant test are:
bon_CBC_prepro_anchor_balanced_random_half (**v4C-o2**),
bon_GLPK_prepro_anchor_balanced_dual (**v4G-o2**).

As shown in Table 4, preprocessing, warm-start initialization, and variable

Table 4: Best models

n	v3C-o	**v4C-o**	**v4C-o2**	v3G-o	v4G-o	v4G-o2	v1C-f	**v4C-f**	v1G-f	v4G-f
4	US	US	US	US	US	US	US	US	US	US
6	**0**(0s)	**0**(0s)	**0**(0s)	**0**(0s)	**0**(0s)	**0**(0s)	0s	0s	0s	0s
8	**0**(1s)	**0**(0s)	**0**(0s)	**0**(0s)	**0**(0s)	**0**(0s)	0s	0s	0s	0s
10	**0**(4s)	**0**(1s)	**0**(0s)	**0**(18s)	**0**(0s)	**0**(0s)	4s	0s	5s	0s
12	**0**(37s)	**0**(1s)	**0**(1s)	–	**0**(27s)	**0**(1s)	3s	0s	15s	2s
14	**0**(83s)	**0**(21s)	**0**(18s)	–	–	**0**(67s)	5s	20s	–	93s
16	–	**0**(119s)	**0**(283s)	–	–	–	–	44s	–	–
18	–	–	–	–	–	–	–	184s	–	–

reduction proved to be crucial. Consequently, the model employing v4 and CBC solver achieved the best performance.

5 Conclusions

In conclusion, preprocessing with the circle method allowed to reach better performance for all approaches as the matches are already partially scheduled in advance. This permitted to reduce greatly the difficulty of the problem as the encodings needed to just respect the constraints about the maximum number of games in each period while the circle method in few calculations created a schedule which respected the constraint of only one game for each team in the same week and the constraint of having always different opponents for each team.
Another improvement, for some approaches, was given by the implementation of symmetry breaking constraint.
The two best results were obtained by SMT for the decision version ($n = 22$) while for the optimization one by CP ($n = 20$).
Each approach had its key aspects which allowed to improve the performance as the methods used were quite different even though some similarities were present. For CP a huge role was played by separating the home/away assignment from scheduling; the use of symmetry breaking constraints and implied constraints while also using an offline routine was fundamental to reach similar performances across different solvers and encodings; introducing a warm start allowed MIP to reach satisfying performances.

6 Acknowledgments

We declare that the described work is our own and that we did not copy it from someone else.
All the design choices were searched and approved by all the members of the group but, Leonardo Massaro focused on CP, Giorgio Scavello on SMT and Leonardo Billi on MIP.

References

Kendall, G., S. Knust, C. C. Ribeiro, and S. Urrutia (2010). Scheduling in sports: An annotated bibliography. *Computers & Operations Research 37*(1), 1–19.

Lambrechts, E., A. M. Ficker, D. R. Goossens, and F. C. Spieksma (2018). Round-robin tournaments generated by the circle method have maximum carry-over. *Mathematical Programming 172*(1), 277–302.

Larson, J., M. Johansson, and M. Carlsson (2014). An integrated constraint programming approach to scheduling sports leagues with divisional and round-robin tournaments. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 144–158. Springer.