Combinatorial Decision Making And Optimization

Cirone Cono, cono.cirone@studio.unibo.it
Dardini Jacopo, jacopo.dardini@studio.unibo.it
Formichella Gio, gio.formichella@studio.unibo.it
Petrozziello Giulio, giulio.petrozziello@studio.unibo.it

1 Introduction

In this project, we address the sports scheduling (STS) problem by applying a unified modeling approach inspired by [1] across four computational paradigms: Constraint Programming (CP), Boolean Satisfiability (SAT), Satisfiability Modulo Theory (SMT) and Mixed Integer Programming (MIP). We begin by considering the structure of a round robin tournament: for N teams, the tournament spans $N-1$ weeks, with each week consisting of $\frac{N}{2}$ games (or periods), ensuring that every team plays exactly once per week. This structure naturally corresponds to a 1-factorization of the complete graph K_N on N vertices [2].

A balanced tournament design is a permutation of the order of games within each week so as to satisfy the additional constraints [3]. Therefore it is natural to model the problem by considering an $\frac{N}{2} \times (N-1)$ matrix of matches (t_i, t_j) representing the round robin tournament for N teams, where each column represents a week and contains the set of matches scheduled for that round. The objective is to find an indexing of the games along the periods of each week to satisfy the STS problem. Our experimental results show that the time required to precompute the underlying round-robin schedule is negligible (less than a second) with respect to the one needed for all the solved instances. Moreover, by precomputing the round robin tournament matrix, we eliminate symmetries across weeks and fix slot assignments, which significantly reduces the search space for both satisfiability and optimization.

1.1 Notation

- N: number of teams
- $T = \{t \mid t \in [1, N]\}$: team identifiers
- $P = \{p \mid p \in [0, ..., \frac{N}{2} - 1]\}$: period identifiers
- $M = \{m \mid m \in [0, ..., \frac{N}{2} - 1]\}$: weekly matchup identifiers

- $W = \{w \mid w \in [0, ..., N-2]\}$: week identifiers
- $S = \{s \mid s \in [0, 1]\}$: slot identifiers, where $s = 0$ corresponds to playing at home and $s = 1$ to playing away
- $rb_{m,w,s} = t$: team t plays in week w in match m in slot s.

2 CP Model

2.1 Decision Variables

This model utilizes two primary decision variables to construct the tournament schedule and assign home/away teams for each match.

2.1.1 matches

For each period $p \in P$ and week $w \in W$

$$matches_{p,w} = m \in M$$

determines that $rb_{m,w}$ is scheduled in week w and period p.

2.1.2 home_away

For each period $p \in P$ and week $w \in W$

$$home\_away_{p,w} = a \in \{0, 1\}$$

determines which team is assigned as the home team for the match scheduled in week w and period p. Specifically, $a = 0$ means the first team listed plays at home, while $a = 1$ means the second team plays at home.

2.2 Auxiliary Variables

2.2.1 home_games

For each team $t \in T$

$$home\_games_t = h \in [1, \ldots, N-1]$$

determines the total number of home games assigned to team t throughout the tournament. Its value is derived from the assignments of the decision variables *matches* and *home_away*.

2.3 Objective Function

The model's objective is to quantify and minimize disparities in home game assignments through the max_imbalance variable.

2.3.1 Objective Variable: max_imbalance

The integer variable max_imbalance $\in [1, \ldots, N-1]$ quantifies the maximum absolute disparity in home game assignments across all teams. The lower bound of 1 acknowledges that perfect balance (max_imbalance = 0) is not achievable for an odd number of total games ($N-1$ games). The upper bound of $N-1$ represents the theoretical maximum possible deviation, occurring if a team plays all its games either at home or away.

The value of max_imbalance is determined by the following fairness constraint:

$$\forall t \in T : |2 \times \text{home\_games}_t - (N-1)| \leq \text{max\_imbalance}$$

This formulation precisely defines the absolute "imbalance" for each team t. This imbalance is derived from the difference between a team's home games and away games. Since $\text{home\_games}_t + \text{away\_games}_t = N-1$ (total games), substituting the expression for away_games_t yields the imbalance for team t as $2 \times \text{home\_games}_t - (N-1)$. By enforcing that the absolute value of this imbalance for every team must be less than or equal to max_imbalance, this variable effectively captures the largest such deviation among all teams, serving as the direct measure of the overall schedule's fairness.

To further support a fair and computationally efficient solution, the model includes a weekly balancing condition, which helps reduce the search space by ensuring that within each week w, the number of matches where the second team plays at home ($\text{home\_away}_{p,w} = 1$) is roughly half of the total matches $|P|$, with a maximum deviation of 1:

$$\forall w \in W : \left| \sum_{p \in P} \text{home\_away}_{p,w} - \left\lfloor \frac{|P|}{2} \right\rfloor \right| \leq 1.$$

This guides the solver toward balanced assignments week by week and helps prune highly imbalanced configurations.

2.3.2 Objective

The objective is to **minimize max_imbalance**:

$$\text{minimize max\_imbalance}$$

This aims to achieve the fairest possible distribution of home and away games. An optimal solution would ideally yield max_imbalance = 1, meaning each team's home/away game count differs by at most one, which is the best possible outcome for an odd number of total games played.

2.4 Constraints

2.4.1 Core Constraints

These constraints are strictly necessary for defining a feasible round-robin schedule:

1. **Each period must be used exactly once per week:** Ensures that for every week, all matches generated by the round-robin structure's periods are indeed scheduled. Without this, some pairings might be missed, or periods might be duplicated, leading to an incomplete or invalid schedule.

$$\forall w \in W : \texttt{all\_different}([\texttt{matches}[p, w] \mid p \in P])$$

2. **Each team plays at most twice per period:**

$$\forall p \in P, \forall t \in T : \left|\{(w, s) \mid w \in W, s \in S, \mathrm{rb}_{\mathrm{matches}_{p,w},w,s} = t\}\right| \leq 2$$

Implemented using the `global_cardinality` global constraint.

2.4.2 Channeling Constraints

1. **Calculation of Home Games:** This constraint defines the value of $\mathrm{home\_games}_t$.

$$\forall t \in T : \mathrm{home\_games}_t = \sum_{p \in P, w \in W, s \in S} \mathbb{I}\left(\mathrm{rb}_{\mathrm{matches}_{p,w},w,s} = t \wedge \mathrm{home\_away}_{p,w} = s\right)$$

The indicator function $\mathbb{I}(\cdot)$ ensures that 1 is added to the sum if team t is located in slot s and that slot s is designated as the home slot by $\mathrm{home\_away}_{p,w}$.

2.4.3 Implied Constraints

1. **Each team appears exactly once per week:** While this is implicitly ensured by the combination of the `all_different` constraint on `matches` and the construction of `rb`, explicitly stating it helps the solver propagate information earlier and prune the search space more effectively. It is implemented using the `global_cardinality` global constraint:

$$\forall w \in W, \forall t \in T : \left|\{(p, s) \mid p \in P, s \in S, \mathrm{rb}_{\mathrm{matches}_{p,w},w,s} = t\}\right| = 1.$$

2.4.4 Symmetry Breaking Constraints

1. **Break period assignment symmetry using lexicographic ordering:** The order in which matches corresponding to P are assigned within matches for each week is symmetrical. This constraint breaks such symmetries by enforcing a lexicographical ordering, reducing the number of equivalent search paths.

$$(\mathrm{matches}_{p,w})_{p \in P, w \in W} \succeq_{\mathrm{lex}} (\mathrm{matches}_{p,w})_{\mathrm{reversed}(p) \in P, w \in W}$$

This states that the sequence of matches variables, when read in normal (p, w) order, must be lexicographically greater than or equal to when read in (p, w) order with p reversed. This helps to fix one permutation of period assignments.

2. **Fix first match home assignment to break home/away symmetry:** This constraint eliminates global home/away assignment symmetry by fixing the home/away status of the first match.

$$\text{home\_away}_{0,0} = 0$$

2.5 Validation

The model was implemented in MiniZinc and validated through a series of experiments designed to assess solver performance under various model configurations and search strategies.

2.5.1 Experimental Design

To comprehensively evaluate the performance of different solving strategies for the Sports Tournament Scheduling problem, a systematic experimental study was conducted.

Hardware and Software: Experiments were executed on a MacBook Air M1 equipped with an 8-core CPU. The following solvers were employed: *Gecode*, *Chuffed* and *OR-Tools CP-SAT*. A uniform time limit of 300 seconds was imposed for each individual problem instance.

Model Configurations: Four configurations were tested: `baseline (core)`, `baseline+implied`, `baseline+symmetry breaking`, `full model`.

Search Strategies: Three distinct search strategies were employed to analyze solver behavior, focusing on their influence on Gecode, given its often weaker default heuristics compared to modern SAT-based solvers.

1. **Default Search Strategy (Solver's Default):** Each solver relied entirely on its built-in decision heuristics and restart policies, serving as a baseline for their inherent capabilities.

2. **Sequential Custom Search Strategy:** A manually defined sequential search (`seq_search`) was applied, prioritizing `matches` variables with `dom_w_deg` and `home_away` variables with `first_fail`, utilizing a `restart_luby(100)` policy.

3. **Relax-and-Reconstruct (LNS) Strategy:** This higher-level strategy incorporated `relax_and_reconstruct` on the `matches` variables (preserving 60% of solution values), leveraging Large Neighborhood Search (LNS) techniques. It was layered on top of the "Sequential Custom Search Strategy."

Solver-Specific Strategy Application: To ensure a fair and controlled comparison under single-threaded conditions (aligning with project constraints), OR-Tools CP-SAT was run without multi-threading. For both Chuffed and OR-Tools CP-SAT, the `free_search` parameter was explicitly omitted when applying the custom Sequential Custom Search and Relax-and-Reconstruct strategies. This allowed direct evaluation of the user-defined MiniZinc search annotations, rather than the solvers' highly optimized default heuristics.

2.5.2 Experimental Results

In the following tables, we report CPU times for different solvers and four model configurations:

- `bs` denotes the baseline (core) model
- `noIMPL` denotes the baseline with symmetry breaking constraints
- `noSB` denotes the baseline with implied constraints
- `complete` denotes the full model with both implied and symmetry breaking constraints

n	GECODE				CHUFFED				CP-SAT			
	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB
6	0	0	0	0	0	0	0	0	0	0	0	0
8	6	0	0	1	0	0	0	0	0	0	0	0
10	N/A	N/A	N/A	N/A	0	0	0	0	1	1	1	1
12	N/A	N/A	N/A	N/A	5	1	2	4	3	2	2	3
14	N/A	N/A	N/A	N/A	188	20	184	292	4	6	6	6
16	N/A	N/A	N/A	N/A	N/A	N/A	276	N/A	15	28	35	15
18	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	261	34	43	237
20	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	83	66	79	95
22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	234	185	135	N/A

Table 1: CPU time in seconds for finding the *optimal solution* using *Default Search Strategy (Solver's Default)*

n	GECODE				CHUFFED				CP-SAT			
	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB
6	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	1	1	1
12	0	0	0	0	0	0	0	0	54	55	58	81
14	4	7	4	4	N/A	54	33	N/A	N/A	N/A	N/A	N/A
16	N/A	N/A	189	47	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: CPU time in seconds for finding the *optimal solution* using *Sequential Custom Search Strategy*

n	GECODE				CHUFFED				CP-SAT			
	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB	bs	complete	noIMPL	noSB
6	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	1	1	1
12	0	0	0	0	8	1	3	2	54	62	59	81
14	1	5	1	0	N/A	84	183	145	N/A	N/A	N/A	N/A
16	181	19	1	6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
18	N/A	3	N/A	245	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 3: CPU time in seconds for finding the *optimal solution* using *Relax-and-Reconstruct (LNS) Strategy*

3 SAT Model

3.1 Decision variables

Let rb be the round robin tournament, sts a schedule satisfying to the STS problem, and P, W, T the set of periods, weeks and teams respectively. The satisfiability task is formalized by the following categories of propositions:

- $matches\_schedule_{p,w,m} \leftrightarrow$ match m, denoting $rb_{m,w} = (t_1, t_2)$, takes place in period p of week w
- $matches\_to\_periods_{t_1,t_2,p} \leftrightarrow t_1$ plays against t_2 in period p

The optimization task, on the other hand, is expressed as follows.

- $slots\_schedule_{p,w} \leftrightarrow$ team $sts_{p,w,1} = t_1$ plays away
- $matches\_to\_slots_{t_1,t_2} \leftrightarrow t_1$ plays away and t_2 plays at home

3.2 Objective function

Similarly to the other approaches, the objective is to minimize the absolute difference between the number of games played at home and away for each team. Let T be the set of teams and A_t the number of times the team $t \in T$ plays away, the objective function translates into the following:

$$k* = \underset{k \in \mathbb{N}}{\operatorname{argmax}} \left(\forall t \in T, A_t \geq k\right)$$

The chosen CP model allows to solve the optimization task independetly of the STS constraints using the sts schedule computed in the satisfiability process. The optimization consists of a binary search of $k*$ in the interval $[1, \lfloor \frac{N-1}{2} \rfloor]$. Since SAT doesn't directly support optimization, a new set of optimizing constraints is introduced for every instance of k .

The result is encoded in $slots\_schedule_{p,w}$, which cannot express the constraints alone, due to structural limitations of the encoding. Instead $matches\_to\_slots_{t_1,t_2}$ is used: given a week w and a period p, the team $sts_{p,w,1} = t_1$ plays away if, and only if, the team $t_2 = sts_{p,w,2}$ plays at home. Therefore, by definition:

$$\forall p \in P, w \in W.(slots\_schedule_{p,w} \leftrightarrow matches\_to\_slots_{sts_{p,w,1},sts_{p,w,2}})$$

Finally the optimization process is implemented by ensuring that for an instance $k \in [1, \lfloor \frac{N-1}{2} \rfloor]$:

$$\forall p \in P, t_1 \in T.(AtLeastK(matches\_to\_slots_{t_1,t_2} | t_2 \in T/\{t_1\}))$$

3.3 Constraints

3.3.1 Every team plays once a week

In a Round-Robin tournament rb, each team plays exactly once a week. Therefore the only problem is to ensure that for each week w, every match $rb_{m,w} = (t_i, t_j)$ is assigned to exactly one period p.

We enforce that every match is scheduled once:

$$\forall p \in P, w \in W.(ExactlyOne(matches\_schedule_{p,w,m} | m \in P))$$

Finally no two matches are scheduled in the same period.

We ensure that each match is scheduled to a unique period p in the week w.

$$\forall m \in P, w \in W.ExactlyOne(matches\_schedule_{p,w,m} | p \in P)$$

3.3.2 Every team plays at most twice in the same period

The constraint cannot be expressed using directly $matches\_schedule_{p,w,m}$, due to structural limitations of the encoding. Instead, we introduce the literal $matches\_to\_periods_{t_i,t_j,p}$: given a week w, the match $rb_{m,w}$ is scheduled in period p if, and only if, the match $(rb_{m,w,1}, rb_{m,w,2}) = (t_1, t_2)$ takes place in period p. Therefore, by definition:

$$\forall p \in P, w \in W, m \in P.matches\_schedule_{p,w,m} \leftrightarrow matches\_to\_periods_{rb_{m,w,1},rb_{m,w,2},p}$$

Computationally this constraint soundly maps $matches\_schedule$ to $matches\_to\_periods$, which allow to express the main constraint:

$$\forall p \in P, t_1 \in T.AtMost2(matches\_to\_periods_{t_1,t_2,p} | t_2 \in T/\{t_1\})$$

3.4 Validation

3.4.1 Experimental design

The model was written in Python by making use of the Z3 and the CVC5 library, which offers CaDiCaL and MiniSat as the underlying SAT solvers. The time elapsed to find an optimal solution, within the 300 second time limit, was measured and results presented in Fig.1 .

3.4.2 Experimental results

All solvers are able to find the optimal solution, which was much easier to find once the satisfying one was computed, but overall Z3 had the best performance in time and maximal size of the problem, i.e. $N = 20$. On the other hand CaDiCaL was the worst, failing at $N = 12$, while MiniSat stopped at $N = 14$

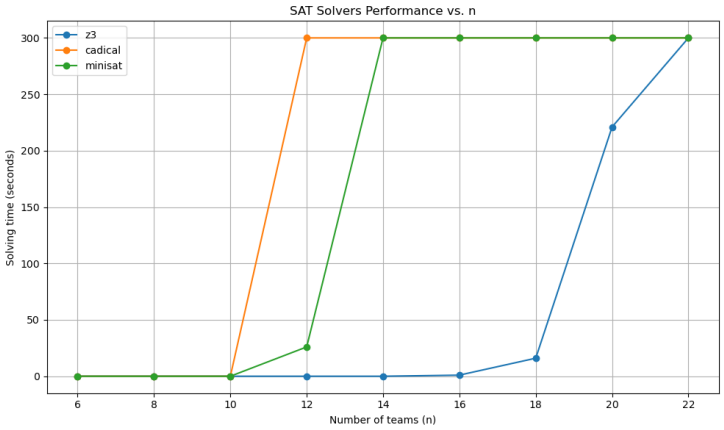

Figure 1: SAT optimization

4 SMT Model

4.1 Decision Variables

The SMT model is divided into two phases:

- **Phase 1**: selects exactly one match pair m for each period p and week w.
- **Phase 2**: reuses the feasible schedule and flips home/away slots to optimize balance.

Phase 1 variables:

- $index_{p,w,m} \in \{\text{true}, \text{false}\}$: true if match pair m is selected for (p, w).
- $home_{p,w} \in T$: home team in (p, w).
- $away_{p,w} \in T$: away team in (p, w).

Phase 2 variables:

- $flip_{p,w} \in \{\text{true}, \text{false}\}$: indicates if match in (p, w) is flipped.
- $homeEff_{p,w}$, $awayEff_{p,w}$: effective teams after flipping.

4.2 Objective Function

Minimize the maximum imbalance k:

$$k^* = \min\Big\{k \mid \forall t \in T : |H_t - A_t| \le k\Big\}, \quad H_t = \sum_{p,w}[\, homeEff_{p,w} = t\,], \quad A_t = \sum_{p,w}[\, awayEff_{p,w} = t\,].$$

Effective teams:

$$homeEff_{p,w} = \text{ite}(flip_{p,w}, away_{p,w}, home_{p,w}), \quad awayEff_{p,w} = \text{ite}(flip_{p,w}, home_{p,w}, away_{p,w}).$$

4.3 Constraints

4.3.1 Unique match assignment per slot

Each period (p, w) must select exactly one match pair:

$$\forall p \in P,\, w \in W : \sum_{m \in M} index_{p,w,m} = 1.$$

4.3.2 Each match pair used exactly once per week

Each pair m must be assigned exactly once within each week:

$$\forall m \in M,\, w \in W : \sum_{p \in P} index_{p,w,m} = 1.$$

4.3.3 Binding decision

If $index_{p,w,m}$ is true, the slot must bind to rb:

$$\forall p \in P,\, w \in W : \bigvee_{m \in M} \Big(index_{p,w,m} \wedge home_{p,w} = rb_{m,w,0} \wedge away_{p,w} = rb_{m,w,1} \Big).$$

4.3.4 Team appears at most twice in same period

A team cannot appear more than twice in the same period over the whole tournament:

$$\forall t \in T,\, p \in P : \sum_{w \in W} \sum_{m \in M} \Big([\, rb_{m,w,0} = t \vee rb_{m,w,1} = t \,] \cdot [\, index_{p,w,m} \,] \Big) \leq 2.$$

4.3.5 Symmetry breaking

Fix first match pair in first slot:

$$index_{0,0,0} = \text{true}.$$

4.3.6 Imbalance constraint

For Phase 2:

$$|H_t - A_t| \leq k.$$

4.4 Validation

4.4.1 Experimental design

Implemented in Python + SMT-LIB. Solvers: Z3 and CVC5, timeout 300s. Phase 1 finds feasible solution; Phase 2 binary-searches the minimal k.

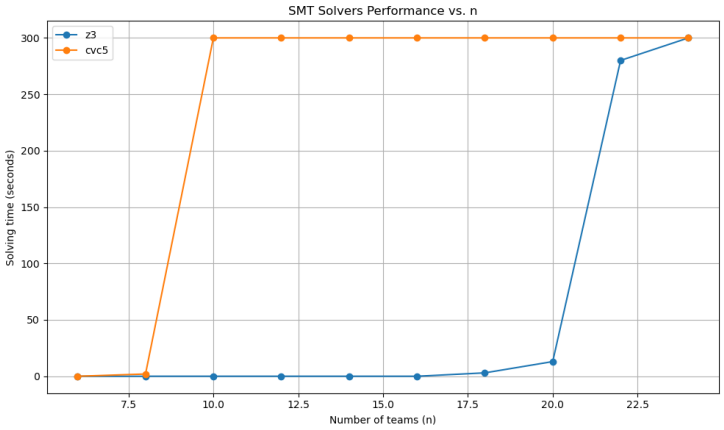

Figure 2: SMT solution example

n	Z3 (s)	CVC5 (s)
6	0.11	0.13
8	0.17	1.85
10	0.23	N/A
12	0.29	N/A
14	0.36	N/A
16	0.74	N/A
18	3.52	N/A
20	14.32	N/A
22	280.17	N/A
24	N/A	N/A

Table 4: SMT solving times

4.4.2 Results

Z3 solves larger instances faster. CVC5 is feasible for small N only.

5 MIP

5.1 Decision variables

5.1.1 matches

The binary decision variables $matches_{w,p,m}$ are equal to 1 if and only if in week $w \in W$ and period $p \in P$ match $m \in M$ is played.

5.1.2 slots

The binary decision variables $home\_away_{w,p}$ determine which team plays at home and which away. $home\_away$ is indexed by weeks and periods, so $home\_away_{w,p}$ corresponds to the matchup (t_1, t_2), $t_1, t_2 \in T$ in week w and period p; if $home\_away_{w,p} = 0$, then t_1 plays at home and t_2 away, if, instead, $home\_away_{w,p} = 1$, the order of the slots is reversed.

5.1.3 team periods of play

The binary variables $TP_{t,w,p}$ are used to constrain each team to playing at most twice in the same period. $TP_{t,w,p} = 1$ if and only if team t plays in week w and period p.

5.1.4 home games counter

To compute the common objective function D in MIP, it is necessary to introduce an array of auxiliary integer variables H, such that H_t is the number of home games team t plays, $H_t \in [0, n-1]$.

5.2 Objective variables

To compute D as the maximum difference between games played at home and away between all teams, we first find for each team the number of games played at home (2), and then constraint D as being greater or equal than the difference of home and away games for each team. Given that we have a minimization problem this is effectively equivalent to computing the max:

$$\begin{cases} \min D \\ D = \max_t |home_t - away_t| \end{cases} = \begin{cases} \min D \\ D \geq |home_t - away_t| \ \forall t \in T \end{cases}$$

The absolute value of the difference is not computed explicitly but is decomposed into 2 inequalities (3)(4). Finally, we look for the minimum of D (1)

$$\min D \tag{1}$$

$$H_t = \sum_{w \in W, m \in M} \mathbb{1}[home\_away_{w,p} = 0 \wedge rb_{m,w,0} = t] + \sum_{w \in W, m \in M} \mathbb{1}[home\_away_{w,p} = 1 \wedge rb_{m,w,1} = t] \qquad \forall t \in T \tag{2}$$

$$D \geq 2H_t - (n-1) \qquad \forall t \in T \tag{3}$$

$$D \geq -(2H_t - (n-1)) \qquad \forall t \in T \tag{4}$$

5.3 Constraints

5.3.1 periods and matches

Due to how the decision variables are defined, it was necessary to impose that each period in each week is assigned a single match (5) and each match is assigned to a single period (6).

$$\sum_{m \in M} matches_{w,p,m} = 1 \qquad \forall p \in P \quad \forall w \in W \tag{5}$$

$$\sum_{p \in P} matches_{w,p,m} = 1 \qquad \forall m \in M \quad \forall w \in W \tag{6}$$

5.3.2 team playing at most twice in the same period

The constraint on teams playing at most twice in the same period is imposed by first linking the variables of *matches* to those in TP based on the values in *rb* (7) and then limiting the sum of periods of play to being smaller than 2 (8).

$$TP_{t,w,p} = matches_{w,p,m} \text{ where } rb_{m,w} = (t, t') \vee (t', t) \qquad \forall t \forall p \forall w \tag{7}$$

$$\sum_{w \in W} TP_{t,w,p} \leq 2 \qquad \forall t \in T \quad \forall p \in P \tag{8}$$

5.4 Validation

Experimental design

The model is written in Python by making use of the PuLP library and the solvers tested on the MIP model were: CBC 2.10.3, HiGHS 1.10.0, CPLEX 22.1.1 and SCIP 5.5.0 with their default parameters. The time elapsed to find an optimal solution, within the 300 second time limit, was measured for each solver. All tests were run on a single core of an Intel i7-10750H CPU.

Experimental results

As shown in Table 5, CBC had the worst performance, it isn't able to find the optimal solution for n greater than 14. CPLEX, instead, is the fastest up to n=14 but after n=16 it stops finding a solution. HiGHS was able to find an optimal schedule up to n=18 and SCIP, the best performer of the four, was able to reach n=20.

It was also verified that the solvers either find an optimal solution or no solution at all within the time limit.

n	CBC	HiGHS	CPLEX	SCIP
6	0	0	0	0
8	0	0	0	0
10	1	0	0	0
12	2	1	0	5
14	45	12	5	17
16	N/A	62	20	15
18	N/A	218	N/A	192
20	N/A	N/A	N/A	101
22	N/A	N/A	N/A	N/A

Table 5: MIP optimization solver results

References

[1] P. Van Hentenryck, L. Michel, L. Perron, and J. C. Régin. Constraint programming in opl. In Gopalan Nadathur, editor, *Principles and Practice of Declarative Programming*, pages 98–116, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[2] J. H. Dinitz, D. Froncek, E. R. Lamken, and W. D. Wallis. Scheduling a tournament. In Charles J. Colbourn and Jeffrey H. Dinitz, editors, *Handbook of Combinatorial Designs*, pages 591–598, Boca Raton, FL, 2006. CRC Press.

[3] E. R. Lamken. Balanced tournament designs. In Charles J. Colbourn and Jeffrey H. Dinitz, editors, *Handbook of Combinatorial Designs*, pages 333–336, Boca Raton, FL, 2006. CRC Press.