CMDO 2025 - Sports Tournament Scheduling (STS) problem

Luca Anzaldi luca.anzaldi@studio.unibo.it
Martino Michelotti martino.michelotti@studio.unibo.it

February 1, 2026

1 Introduction

This report addresses the solution of the well-known *Sports Tournament Scheduling Problem (STS[2])*. The focus is on presenting different modeling approaches and optimization techniques to handle the scheduling constraints and objectives. Throughout the report we describe the common elements that underlie all models: the inputs, symbols, shared objective variable and bounds, global pre-processing, and any assumptions. To avoid repetition, later sections will refer back to this introduction whenever these shared components are required.

Instance parameters Let $T = n$ be the number of teams (even), $W = T - 1$ the number of weeks, and $P = T/2$ the number of periods (parallel timeslots) per week. Each solution is modeled referring to these three main variables.

- Example1 : $T = 6 \rightarrow W = 4$ and $P = 3$
- Example2 : $T = 16 \rightarrow W = 15$ and $P = 8$
- Example3 : $T = 5 \rightarrow$ NOT VALID

For the sake of brevity, whenever w, p and t (or eventually t_1, t_2) are used in formulas and the domains are omitted, they have to be implicit meant to be: $w \in \{1, \dots, W\}$, $p \in \{1, \dots, P\}$ and $t \in \{1, \dots, T\}$.

General constraints The classical formulation of the STS problem requires that:

1. every team plays with every other team exactly once;
2. every team plays exactly once per week;
3. every team plays at most twice in the same period over the whole tournament.

Objective function The main optimization criterion is to achieve a balanced schedule, in which every team plays approximately the same number of home and away games. In particular, given the instance parameters T teams, W weeks, and P periods. The objective is to minimize the overall imbalance:

$$\sum_{t=1}^{T} \Big| \, \#\text{home\_games}(t) - \#\text{away\_games}(t) \, \Big|$$

This formulation depends only on the instance parameters (T, W, P) that define how many games each team must play, and expresses the high-level goal of fairness in the tournament.

Since W is uneven by the definition of the problem, it is clear that no team could ever play the same number of home and away games and therefore the minimal value of $\big|\#\text{home\_games}(t) - \#\text{away\_games}(t)\big|$ for every team t cannot be lower than 1. This implies that the objective function is (non-strictly) bounded below by the number of teams $T = n$, that we will find out that it corresponds to the optimal value we are looking for.

2 SAT Model

The SAT solution was made using two different modeling approaches. Both are included in the report.

Notation. Throughout the constraints we use the following shorthand:

$$\begin{aligned}
\sum x_i &\geq k &\iff& \text{“at least } k \text{ of the } x_i \text{ are true”},\\
\sum x_i &= k &\iff& \text{“exactly } k \text{ of the } x_i \text{ are true”},\\
\sum x_i &\leq k &\iff& \text{“at most } k \text{ of the } x_i \text{ are true”}.
\end{aligned}$$

In the implementation these correspond respectively to the encodings `at_least_k`, `exactly_k`, and `at_most_k`. The special cases with $k = 1$ are captured by the usual constraints: `at_least_one`, `at_most_one`, and `exactly_one`.

Model 1 was implemented using several alternative SAT encodings for the cardinality constraints introduced above. In particular, we considered the **Bitwise** encoding, the **Heule** encoding, the **Pairwise** encoding, and the **Sequential** encoding. Each encoding provides a different trade-off between the number of variables, the number of clauses, and the strength of constraint propagation.

Model 2 extends the formulation of Model 1 by adding further constraints and auxiliary decision variables. In contrast to Model 1, the implementation adopts a single SAT encoding approach (e.g., pairwise-style cardinality constraints) and does not perform a systematic comparison across alternative encodings.

The formulation ideas behind this model emerged partly from conceptual discussions supported by ChatGPT [5] (version 5.2), which helped explore alternative modeling perspectives and refinements.

Moreover, the implementation can optionally enable a **precomputing** phase: a round-robin pairing generator fixes in advance which team pairs are allowed to play in each week, and the corresponding ($M_{t_1,t_2,w}$) variables are forced to either true or false accordingly. This reduces the search space by eliminating week–pair symmetries and infeasible match placements before solving, allowing us to study how much of the performance change comes from a more constrained model versus additional upfront structural pruning.

The purpose of Model 2 is therefore not to benchmark competing encoding techniques, but to assess how a more expressive formulation affects solver performance and scalability.

2.1 Decision variables

2.1.1 Model - 1

The use of multiple encodings within the same model allows for a systematic comparison of their impact on solver performance, while keeping the underlying problem formulation fixed.

Booleans

- $X_{t,h,p,w} \in \{0,1\}$ meaning team t is scheduled with role h in slot (p,w).

2.1.2 Model - 2

Booleans

- $M_{t_1,t_2,w} \in \{0,1\}$ meaning teams t_1 and t_2 play against each other in week w.
- $HOME_{t,w} \in \{0,1\}$ meaning team t plays at home in week w (away otherwise).
- $P_{t,p,w} \in \{0,1\}$ meaning team t is assigned to period p in week w.

Remarks. Model–2 introduces explicit pair variables $M_{t_1,t_2,w}$ to capture which two teams meet in a given week, home assignment variables $HOME_{t,w}$ to distinguish roles, and period variables $P_{t,p,w}$ to ensure consistency of scheduling. It also supports a *precomputing* mode, where a round-robin structure is generated in advance and infeasible pairs are pruned by setting the corresponding $M_{t_1,t_2,w}$ literals to false.

2.2 Objective function

2.2.1 Model - 1

We minimize total home/away imbalance across teams:

$$\sum_{t=1}^{T} \left| \sum_{p=1}^{P} \sum_{w=1}^{W} \left(X_{t,0,p,w} - X_{t,1,p,w} \right) \right| .$$

In the implementation with `Z3`, the optimization is performed by *iterative (binary) search* on a pseudo-Boolean bound z. At each step we introduce a constraint of the form $\sum_t |\cdot| \le z$, encoded using cardinality constraints, and solve for feasibility. If satisfiable, the bound is tightened; otherwise the process stops at the last feasible value.

2.2.2 Model - 2

As in Model–1, the optimization criterion is to balance the number of home and away games for each team. Formally:

$$\sum_{t=1}^{T} \left| \sum_{w=1}^{W} HOME_{t,w} - \Big(W - \sum_{w=1}^{W} HOME_{t,w}\Big) \right|.$$

2.3 Constraints

2.3.1 Model - 1

Main constraints.

(C1) **Every pair of teams plays at most once:**

$$\forall\, t_1, t_2 \text{ with } t_1 \neq t_2 \quad \sum_{p=1}^{P} \sum_{w=1}^{W} \Big(\bigvee_{h \in \{0,1\}} X_{t_1,h,p,w} \wedge \bigvee_{h \in \{0,1\}} X_{t_2,h,p,w} \Big) \;\le 1.$$

(C2) **Each team plays exactly once per week:**

$$\forall\, t, w \quad \sum_{h \in \{0,1\}} \sum_{p=1}^{P} X_{t,h,p,w} \;=\; 1.$$

(C3) **Each team appears at most twice in the same period over the tournament:**

$$\forall\, t, p \quad \sum_{h \in \{0,1\}} \sum_{w=1}^{W} X_{t,h,p,w} \;\le\; 2.$$

Implied constraints.

(C4) **Each game slot has exactly one home and one away team:**

$$\forall\, p, w \quad \sum_{t=1}^{T} X_{t,1,p,w} = 1 \quad \text{and} \quad \sum_{t=1}^{T} X_{t,0,p,w} = 1.$$

2.3.2 Model - 2

Main constraints.

(C1) **Each team plays with every other team only once;**

$$\forall\, t_1 < t_2 \qquad \sum_{w=1}^{W} M_{t_1,t_2,w} \;=\; 1.$$

(C2) **Each team plays exactly once per week**

$$\forall\, t,\, w \qquad \sum_{p=1}^{P} P_{t,p,w} \;=\; 1.$$

(C3) **Each team plays at most twice in the same period**

$$\forall\, t,\, p \qquad \sum_{w=1}^{W} P_{t,p,w} \;\leq\; 2.$$

(C4) **Each slot (p, w) hosts exactly two teams.**

$$\forall\, p,\, w \qquad \sum_{t=1}^{T} P_{t,p,w} \;=\; 2.$$

Implied constraints.

(C5) **Home/away consistency when two teams meet in week w.**

$$\forall\, w,\, t_1 < t_2 \qquad M_{t_1,t_2,w} \;\Rightarrow\; \big(HOME_{t_1,w} \oplus HOME_{t_2,w}\big).$$

(C6) **Link between pair variables and period assignments.**

$$\forall\, w,\, t_1 < t_2 \qquad M_{t_1,t_2,w} \;\Rightarrow\; \bigvee_{p=1}^{P} \big(P_{t_1,p,w} \wedge P_{t_2,p,w}\big),$$

$$\forall\, w,\, p,\, t_1 < t_2 \qquad \big(P_{t_1,p,w} \wedge P_{t_2,p,w}\big) \;\Rightarrow\; M_{t_1,t_2,w}.$$

(Equivalently: $M_{t_1,t_2,w}$ iff the two teams share some period p in week w.)

Simmetry breaking.

(C7) **Precomputed round-robin pruning.** Let $\mathcal{R}_w \subseteq \{\{t_1, t_2\} \mid t_1 < t_2\}$ be the fixed set of pairs for week w produced by the round-robin generator. Then:

$$\forall\, w,\, t_1 < t_2 \qquad M_{t_1,t_2,w} = \begin{cases} 1, & \text{if } \{t_1, t_2\} \in \mathcal{R}_w, \\ 0, & \text{otherwise.} \end{cases}$$

2.4 Validation

Experimental Setup

All experiments were conducted using the `Z3` solver for SAT and optimization ([3]). The implementation is provided in Python, and the solver is invoked through the **Z3 API**.

The experiments were run on two different operating systems, namely *Windows* and *Linux*, to ensure portability of the implementation. The hardware platform was a laptop equipped with a *12th Gen Intel(R) Core(TM) i7-1280P* processor and an *NVIDIA RTX-3060 GPU*. No GPU acceleration was employed for the SAT solving itself, which ran entirely on the CPU.

In order to reproduce the experiments, it is sufficient to follow the instructions provided in the accompanying `README.md` file, which details the installation steps and execution commands. The solver was run without a fixed time limit, but execution times were recorded as reported in the results section.
For reproducibility, a fixed random seed (`SEED_FOR_REPRODUTION`) can be set in the configuration. By default this seed is set to 0, meaning that Z3's internal randomization is left free, but setting it to a positive integer ensures deterministic and repeatable runs across different executions and platforms.

Experimental results

The **following tables reports** the results of the SAT experiments. Each row corresponds to a different value of n. the value of the objective value obtained using different approach. If the instance is solved to optimality is in bold. In addition if the instance is proved to be unsatisfiable, it is indicate as `UNSAT` and if no answer is obtained within the time limit are indicate it with `N/A`. Each modeling approach have is own table (Tab. 1 and Tab. 2).

The **precomputation time** required to generate the round-robin pairs is not reported in the tables, as it is negligible: it remains below one second even for $n \approx 50$, and therefore has no practical impact on the overall computational performance.

n	Bitwise	Heule	Pairwise	Sequential
2	**2**	**2**	**2**	**2**
4	`UNSAT`	`UNSAT`	`UNSAT`	`UNSAT`
6	**6**	**6**	**6**	**6**
8	18	16	20	32
10	28	26	26	24
12	26	40	34	46
14	`N/A`	`N/A`	`N/A`	`N/A`

Table 1: Experimental results of Model 1

n	Standard	Precomputing
2	**2**	**2**
4	`UNSAT`	`UNSAT`
6	**6**	**6**
8	**8**	**8**
10	**10**	**10**
12	**12**	**12**
14	**14**	**14**
16	46	**16**
18	`N/A`	162
20	`N/A`	`N/A`

Table 2: Experimental results of Model 2

3 MIP Model

In this section we will discuss how we implemented a Mixed Integer Linear Program model able to solve the given Tournament Scheduling problem.

The main obstacle in the implementation of linear programs is the fact that the objective function and every constraint have to be linear in our variables; therefore we transformed logic propositions and non-linear functions with linear inequalities, introducing, if needed, some auxiliary variable as for the implementation of the objective function.

3.1 Decision variables

Before defining the variables, we compute the list L of the possible match-ups, without caring about the order of the (i, j) team tuples (we impose $i < j$ for unicity). If $n = T$ is the number of teams, we have $|L| = \binom{n}{2} = \frac{n!}{2!(n-2)!}$, i.e. $\binom{n}{2}$ games to be played.

Our decision variables are then two 3-dimensional tensors of **boolean** values $Y_{w,p,(i,j)}$ and $H_{w,p,(i,j)}$, where

- $Y_{w,p,(i,j)} = 1 \iff$ teams i and j are playing one against the other in week w in period p;
- $H_{w,p,(i,j)} = 1 \iff$ team i (the team with lowest index since $\forall(i, j) \in L$, $i < j$) is playing home in week w in period p.

Clearly these variables have to be linked one with the other, and that's why we'll introduce some variable-linking constraint to our model.

3.2 Objective function

The objective function is essentially the same as in the former cases, but we have to formulate it in a slightly different way because in linear programming the objective function has to be linear and therefore we cannot use absolute values and max / min functions.

We had to find another way to compute

$$\sum_{t=1}^{T} \Big| \, \#\text{home\_games}(t) - \#\text{away\_games}(t) \, \Big|.$$

For brevity we will write h_t and a_t instead of #home_games(t) and #away_games(t) respectively.

Let's rewrite the expression of the objective function in a more efficient way: since

$$h_t + a_t = W = n - 1,$$

we have

$$a_t = W - h_t \text{ and } h_t = W - a_t$$

and therefore

$$\begin{aligned}|h_t - a_t| &= |h_t - (W - h_t)| \\ &= |2h_t - W|,\end{aligned}$$

but equivalently

$$|2h_t - W| = |h_t - a_t| = |a_t - h_t| = |2a_t - W|.$$

Distinguishing two different cases:

$$\begin{aligned}&\text{if } h_t > a_t \implies |h_t - a_t| = h_t - a_t = 2h_t - W \\ &\text{if } a_t > h_t \implies |h_t - a_t| = a_t - h_t = 2a_t - W,\end{aligned}$$

we can then state that

$$|h_t - a_t| = 2\max(h_t, a_t) - W.$$

Once we define a function to sum up the home games and assigning the value to an integer variable h_t for each team t, the away games are computed as $a_t = W - h_t$, where $W = n - 1$ is the total number of games played by each team.

We then compute the maximum between a_t and h_t defining a new integer variable $max(H/A)_t$ and imposing it to be greater or equal than both a_t and h_t, in this way we avoid using a max function, using instead just linear inequalities. Since we look for a minimal value of the objective function we don't need to set an upper bound to the $max(H/A)_t$ values.

Now we can compute the total balance as

$$\sum_{t=1}^{T} (\ 2\, max(H/A)_t - W\ )$$

that is a valid objective function being linear in the $max(H/A)_t$ variables.

3.3 Constraints

Variable linking constraint As already mentioned, we have to link the variables and this has to be done by adding linear inequalities to our linear program.

The only constraint we have to add is the one that ensures that $H_{w,p,(i,j)} \neq 1$ if teams i and j do not play against in week w in period p, i.e.

$$Y_{w,p,(i,j)} = 0 \implies H_{w,p,(i,j)} \neq 1.$$

This constraint can be translated in a linear inequality by imposing

$$H_{w,p,(i,j)} \leq Y_{w,p,(i,j)}.$$

Main constraints

(C1) **Each team plays with every other team only once:**

$$\forall\,(i,j) \in L \qquad \sum_{w,p} Y_{w,p,(i,j)} \;=\; 1.$$

(C2) **Each team plays exactly once per week:**

$$\forall\, t,\, w \qquad \sum_{p=1}^{P} (Y_{w,p,(t,j)} + Y_{w,p,(i,t)}) \;=\; 1.$$

In this case and in the next, the boolean variables that are taken into account are those relative to t, i.e. those where $i = t$ or $j = t$.

(C3) **Each team plays at most twice in the same period:**

$$\forall\, t,\, p \qquad \sum_{w=1}^{W} (Y_{w,p,(t,j)} + Y_{w,p,(i,t)}) \;\leq\; 2.$$

(C4) **Each slot (p, w) hosts exactly one game:**

$$\forall\, p,\, w: \qquad \sum_{t=1}^{T} Y_{w,p,(i,j)} \;=\; 1.$$

With this model we don't need to explicitly ask for exactly two teams in each slot because we already computed the match-ups as couples.

Symmetry breaking constraints

(C5) We fix each match in the first week by imposing:

$$\forall p = 1, \ldots, P \qquad \begin{aligned} &Y_{1,p,(2p-1,\,2p)} = 1 \\ &H_{1,p,(2p-1,\,2p)} = 1 \end{aligned}$$

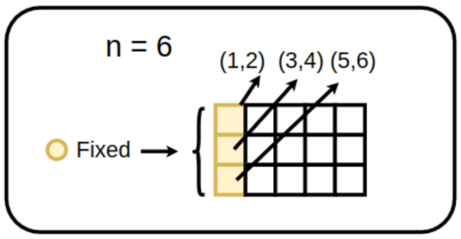

Figure 1: C5 Scheme

(C6) Since the weeks are interchangeable in our formulation (constraints and objective do not depend on the week index), we can break the week-permutation symmetry by fixing the opponent of team 1 in each week, without fixing the period.

$$\forall w = 1, \dots, W \qquad \sum_{p=1}^{P} y_{w,p,(1,\,w+1)} = 1,$$

i.e., in week w team 1 plays against team $w+1$ in some period p.

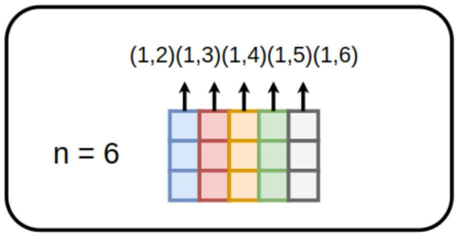

Figure 2: C6 Scheme

(C7) In addition we break symmetry by fixing the main diagonal of the schedule matrix: in each diagonal slot $(w, p) = (p, p)$ (for the first P weeks), team 1 must be scheduled, without fixing the opponent. Formally:

$$\forall p = 1, \dots, P \qquad \sum_{j=2}^{T} y_{p,p,(1,j)} = 1.$$

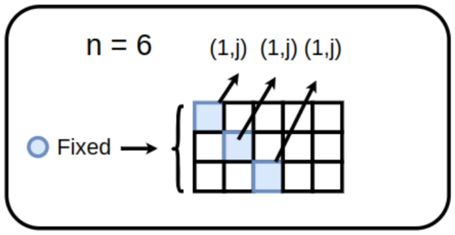

Figure 3: C7 Scheme

3.4 Validation

Experimental Setup

All experiments for the MIP formulation were conducted using a Mixed-Integer Programming solver through the `python-mip` library ([4]). The implementation is provided in Python, and the solver is invoked via the `python-mip` API.

The experimental procedure follows the same workflow adopted for the SAT-based experiments (Section 2.4).

Experimental results

The **following tables report** the results of the MIP experiments. Each row corresponds to a different value of n. For each instance, the tables report the objective value obtained by the different MIP modeling approaches. When an instance is solved to optimality, the corresponding objective value is shown in **bold**. If no feasible solution is found within the allotted time limit, the result is reported as *N/A*.

n	Standard	Standard + SB
2	**2**	**2**
4	`UNSAT`	`UNSAT`
6	**6**	**6**
8	**8**	**8**
10	**10**	**10**
12	**12**	**12**
14	`N/A`	**14**
16	`N/A`	`N/A`
18	`N/A`	`N/A`

Table 3: Experimental results of MIP Model

4 CP Model

In this last section we will discuss how we implemented the Constraint programming model.

With respect to the other two implementations, constraint programming admits a higher range of modeling possibilities and we tried to use them to model less sparse arrays, by avoiding the heavy use of boolean variables as we did in the other models.

4.1 Decision variables

Our decision variables are 3 different arrays:

- $P_{t,w} \in \{1, \dots, P\}$ where
$$P_{t,w} = p \iff \text{team } t \text{ plays in period } p \text{ in week } w$$
- Boolean array $H_{t,w}$ where
$$H_{t,w} = \texttt{true} \iff \text{team } t \text{ plays home in week } w;$$
- $OPP_{t_1,w} \in \{1, \dots, T\}$ where
$$OPP_{t_1,w} = t_2 \iff \text{team } t_1 \text{ plays against team } t_2 \text{ in week } w.$$

4.2 Objective function

For the objective function, we opted for a proxy of the one defined in the introduction. As discussed in section 3.2, we used the `max` function instead of `abs` and the Home/Away imbalance was parametrized using just home games; but in this case, instead of summing up the imbalance for each team, we decided to look for the maximum value of imbalance among all teams and this turned out to be a good choice for performance improvement. The objective function in this section will be

$$\max_{t \in \{1,\dots,n\}} \Big| \, \#\text{home\_games}(t) - \#\text{away\_games}(t) \, \Big|.$$

Such a function can be used instead of the original one since we are looking to the optimal value: n for the original formulation, 1 for the *max* objective function. Knowing a hypothetic value k_{max} of the *max* objective function, defines an upper bound on the values k_{OG} of the original objective function, in fact

$$k_{OG} \le n \, k_{max} \qquad \forall \, n, k_{max}$$

and in case of optimality we clearly get

$$k_{max} = 1 \implies k_{OG} \le n \implies k_{OG} = n.$$

4.3 Constraints

Main constraints.

(C1) **Every pair of teams plays at most once.** For this constraint we used the global constraint `alldifferent` in the following way:

$$\forall\, t \qquad \texttt{alldifferent}(\{\, OPP_{t,w} \,|\, w = 1, \ldots, n-1 \,\}).$$

(C2) **Each team plays exactly once per week.**

This constraint has not to be defined in this model because the array of variables $P_{t,w}$ admits just one value for every team in every week.

(C3) **Each team appears at most twice in the same period over the tournament.**

$$\forall t, p \qquad \big|\{w \,|\, P_{t,w} = p\}\big| \leq 2$$

Implied constraints. These constraints are crucial for the actual definition of the problem and the correct use of the decision variables

(C4) **No team plays against itself:**

$$\forall t, w \qquad OPP_{t,w} \neq t$$

(C5) **Matches are symmetric:**

$$\forall t, w \qquad OPP_{OPP_{t,w}\,,w} = t$$

(C6) **Just one home team in every match:**

$$\forall t, w \qquad H_{t,w} \neq H_{OPP_{t,w}\,,w}$$

(C7) **Matched-up teams play in the same period:**

$$\forall t, w \qquad P_{t,w} = P_{OPP_{t,w}\,,w}$$

(C8) **Every game has to be played by two teams:**

$$\forall p, w \qquad \big|\{\, t \,|\, P_{t,w} = p\}\big| = 2$$

Symmetry breaking The problem has different symmetries and we tried to reduce the search space as much as possible, but maintaining some flexibility. Some of the subsequent constraints seem redundant, but we experimentally found out they were improving performance by helping propagation, especially focusing on the cases in which n was higher. The constraints are imposed especially on team 1 and week 1 for simplicity, but they could've been imposed on every other week and team (singularly) because of the week and team symmetries.

(C9) We fix each match in the first week by imposing three combined constraints:

- fixing match-ups:

$$\forall p \qquad OPP_{2p-1,1} = 2p \ \wedge \ OPP_{2p,1} = 2p - 1 \ ;$$

- fixing periods:

$$\forall p \qquad P_{2p-1,1} = p \ \wedge \ P_{2p,1} \ ;$$

- fixing home/away teams:

$$\forall p \qquad H_{2p-1,1} = \texttt{true} \ \wedge \ H_{2p,1} = \texttt{false} \ .$$

The result is a completely fixed first week, as illustrated for the MIP case for n=6 in Figure 1.

(C10) We impose an increasing order of the opponents for team 1 with the following constraint:

$$\forall w = 1, \dots, n-2 \qquad OPP_{1,w} < OPP_{1,w+1}.$$

This constraint helps, along with (C9), to break the team symmetry of the problem by defining a total order on the teams' labels.

We initially implemented a constraint that fixed the home and away games for team 1 and the model was faster in finding the optimal solution; but we avoided to apply constraints for the home/away alternation (except for week 1), letting the solver find the optimal value of the objective function without external hints. The extreme case of these restrictions on the objective function is the constraint

$$\forall\, t \qquad \big|\ \#\text{home\_games}(t) - \#\text{away\_games}(t)\ \big| = 1$$

or alternatively

$$\sum_{t=1}^{n} \big|\ \#\text{home\_games}(t) - \#\text{away\_games}(t)\ \big| = n.$$

By strictly constraining the objective function, the model was finding a solution (optimal by problem definition) in significant less time; but this would've changed the optimization problem in a satisfaction problem.

4.4 Validation

Experimental design. All experiments for the constraint programming model were done using MiniZinc IDE and then implemented on the docker container. The model is runnable using Gecode solver, but with poor performance; we had some slight improvement when used with Optimization level O2 (two pass compilation). The model was instead optimized for the Chuffed solver (v 0.13.2), since we noticed it's efficiency already from the early stages of modeling. For reproducibility we fixed the random seed 42.

The experiment were ran on a *Windows 11* OS and the hardware platform was a laptop equipped with a *AMD RYZEN 7 5800H* processor. No GPU acceleration was employed for CP solvers, which ran entirely on the CPU.

Experimental results. In the following table we reported the imbalance values of the best solution found by the solver within the time limit of 5 minutes. Is important to notice that the values in the table are the values of the *max* objective function implemented for CP, introduced in section 4.2. Optimal results are marked in **bold**, `UNSAT` if the problem is proven infeasible and `N/A` if no result was found after the time limit. For non-optimal solutions, also the value of the original objective function is shown in brackets. A curious result showed up for n=18, where the additional constraint worsened the best solution found within the time limit. This could be due to the specific random seed we set, but maybe the specific number of teams can be a determinant factor, since for n=20 we had no solutions at all without SB constraints.

Table 4: Results with/without symmetry breaking

n	**Gecode+SB**	**Gecode–SB**	**Chuffed+SB**	**Chuffed–SB**
2	**1**	**1**	**1**	**1**
4	`UNSAT`	`UNSAT`	`UNSAT`	`UNSAT`
6	**1**	**1**	**1**	**1**
8	**1**	**1**	**1**	**1**
10	**1**	`N/A`	**1**	**1**
12	`N/A`	`N/A`	**1**	**1**
14	`N/A`	`N/A`	**1**	**1**
16	`N/A`	`N/A`	**1**	**1**
18	`N/A`	`N/A`	13 (84)	3 (42)
20	`N/A`	`N/A`	15 (156)	`N/A`
22	`N/A`	`N/A`	`N/A`	`N/A`

5 Conclusions

The diversity of the programming languages and their limitations and strengths gave us different views on the problem and tools to handle it, letting us develop different models, having diverse variables and therefore also diverse constraints; without limiting ourselves to just develop three implementations of the same model.

Among all models we developed, unless one specific case (n=18 with Chuffed solver), we had a good improvement for each of them when symmetry breaking (SB) constraints were applied. For all three models, the respective SB constraints allowed us to find a solution when the problem was scaled further by increasing n by one (+2) step. Execution times were not reported, but they also had an improvement by adding SB constraints, outputting faster solutions by reducing the search tree.

The **SAT** approach has a clear limitation in its restriction to purely Boolean variables, which significantly constrained how we could define decision variables. Despite the considerable effort of implementing and testing four different cardinality encodings, the improvements in the final results remained limited for our formulation. The SAT experiments highlighted that, in our case, performance is influenced more by the modeling choices than by symmetry-breaking techniques. While symmetry breaking yielded incremental improvements, reformulating the model produced substantially larger performance gains.

Linear Programming allowed us to set integer variables in the problem, but it would have been trickier to define the constraints, especially when summing up variables. Further issues encountered with MIP were the linear limitations we had defining functions and constraint, bringing us to define new auxiliary variables and reformulate them as linear inequalities.

Constraint programming was for sure the more flexible method, admitting in the formulation every function we needed and different kind of variables; but it's exactly here where we had to think and experiment plenty of different options to find the best combination that could facilitate the constraint propagation of the solver and consequently the reduction of the search tree.

Authenticity and Author Contribution Statement

We declare that the work reported here is our own and properly cites all external ideas and sources. Parts of this project (text/code/experiments) were assisted by AI tools where indicated; we disclose the tools used and the exact sections where AI assistance was applied expect for the part where it was used to improve the quality of the report, such as grammar, sentence formulation and for code debugging.

Author contributions. Luca Anzaldi developed the SAT-based formulations, including both Model 1 and Model 2. He was responsible for the reformulation of the MIP constraints, the configuration and setup of all scripts, Docker environments, and auxiliary tools required to run the experiments. Martino Michelotti developed the complete CP model and designed the core of the MIP model. The report was written by both students, mostly reporting the part of the project they were responsible of.

References

[1] MiniZinc: The Modeling Language.
`https://www.minizinc.org/`

[2] CSPLib, *Problem 026: Sports Tournament Scheduling (prob026)*.
`https://www.csplib.org/Problems/prob026/` (Accessed: 2026-01-29)

[3] Leonardo de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 4963, pp. 337–340, Springer, 2008.
`https://github.com/Z3Prover/z3`

[4] Henrique R. Lopes, Thiago A. F. S. Santos, and Tallys H. Y. Lopes. *Python-MIP: A Python Toolbox for Modeling and Solving Mixed-Integer Linear Programs*. SoftwareX, Volume 12, 2020.
`https://www.python-mip.com/`

[5] OpenAI. *ChatGPT (GPT-5.2)*. 2026. Available at: `https://openai.com/`