Project Title

Dounia, Benkherfallah dounia.benkherfallah@unibo.it
of the authors, in alphabetical order

Introduction

This report details the modelling and solving of the Sports Tournament Scheduling (STS) problem using Combinatorial Optimization techniques. The core challenge involves constructing a round-robin tournament schedule for n teams (n even) over $n-1$ weeks, where each week consists of $n/2$ periods. Each period hosts a single game between two teams, designating one as the home team and the other as the away team. The primary goal is to find a feasible schedule that satisfies these constraints. Furthermore, we consider an optimization variant aimed at enhancing fairness by balancing the number of home and away games for each team. The objective is to minimize the maximum imbalance (the absolute difference between home and away games) across all teams.

Our project involved approaching this problem using Constraint Programming (CP), Propositional Satisfiability (SAT), and Mixed-Integer Linear Programming (MIP).

1 CP Model

This section describes the Constraint Programming model implemented in MiniZinc to solve the Sports Tournament Scheduling problem.

1.1 Decision variables

The model is built around two core decision variables that define the schedule:

- `home`$[w, p]$: Represents the team playing at **home** in period p of week w. Domain: TEAMS = $1..n$.
- `away`$[w, p]$: Represents the team playing **away** in period p of week w. Domain: TEAMS = $1..n$.

These variables are organized into 2D arrays of size WEEKS $\times$ PERIODS, where WEEKS = $1..(n-1)$ and PERIODS = $1..(n/2)$.

1.2 Objective function

The primary goal of this model is to find a **feasible** schedule that satisfies all hard constraints of the problem. Therefore, the objective is to satisfy the constraint model:

```
solve satisfy;
```

An optimization variant to minimize the imbalance between home and away games per team was initially considered but is not the focus of this model. The feasibility problem itself is computationally challenging for larger instances of n.

1.3 Constraints

The constraints enforce the core rules of the tournament and break symmetries to prune the search space effectively.

1.3.1 Main constraints

C1. No Self-Play: A team cannot play against itself.

$$\forall w \in \text{WEEKS},\ \forall p \in \text{PERIODS} :\ \texttt{home}[w,p] \neq \texttt{away}[w,p]$$

Explanation: This fundamental constraint ensures the obvious requirement that the two participants in any game are distinct teams.

C2. One Game Per Week: Every team must play exactly once per week.

$$\forall w \in \text{WEEKS} :\ \texttt{alldifferent}\left(\bigcup_{p \in \text{PERIODS}} \{\texttt{home}[w,p], \texttt{away}[w,p]\} \right)$$

Explanation: The `alldifferent` constraint over all home and away slots in a given week w ensures that each of the n teams appears exactly once in that week, fulfilling the requirement of one game per team per week.

C3. Round-Robin: Every pair of distinct teams (a,b) must play exactly once across the tournament.

$$\forall a,b \in \text{TEAMS},\ a < b :\ \sum_{\substack{w \in \text{WEEKS} \\ p \in \text{PERIODS}}} \mathbf{1}(\texttt{home}[w,p] = a \wedge \texttt{away}[w,p] = b) + \mathbf{1}(\texttt{home}[w,p] = b \wedge \texttt{away}[w,p] = a) = 1$$

Explanation: For every unique unordered pair $\{a,b\}$, the sum of occurrences of the match (a,b) or (b,a) across all weeks and periods must be exactly one. This enforces the classic round-robin requirement.

C4. Period Capacity: No team plays more than twice in the same period over the entire tournament.

$$\forall t \in \text{TEAMS},\ \forall p \in \text{PERIODS} :\ \sum_{w \in \text{WEEKS}} \mathbf{1}(\texttt{home}[w,p] = t) + \sum_{w \in \text{WEEKS}} \mathbf{1}(\texttt{away}[w,p] = t) \leq 2$$

Explanation: This counts the number of times a team t appears in a specific period p (either as home or away) over all weeks and constrains this count to be at most two, preventing overexposure in any given time slot. Note that we found in litterature some similar exemples that would consraint this number to only one

1.3.2 Symmetry breaking constraints

C5. Fixed Initial Week: The pairings for the first week are fixed to a canonical configuration.

$$\texttt{home}[1,1] = 1 \quad \wedge \quad \texttt{away}[1,1] = 2 \quad \wedge \quad \texttt{home}[1,2] = 3 \quad \wedge \quad \texttt{away}[1,2] = 4$$

Explanation: This breaks the symmetry of relabeling the first week by fixing it to a specific, arbitrary state. Without this, the solver would waste time exploring schedules that are identical modulo a permutation of the first week's games.

C6. Game Representation: Within any game, the home team is the smaller-numbered team.

$$\forall w \in \text{WEEKS},\ \forall p \in \text{PERIODS}:\ \texttt{away}[w,p] > \texttt{home}[w,p]$$

Explanation: This breaks the symmetry of swapping the home and away designation for any game. For any possible pairing $\{a, b\}$, it forces the game to be represented uniquely as $(min(a, b)$ v $max(a, b))$, effectively halving the search space for each game.

C7. Lexicographical Ordering:

$$\forall j \in \{2, \ldots, P\}:\ \texttt{home}[1,j] > \texttt{home}[1,j-1]$$

$$\forall i \in \{2, \ldots, W\}:\ \texttt{home}[i,1] \geq \texttt{home}[i-1,1]$$

Explanation: The first constraint orders the home teams in the first week by increasing team number, breaking the symmetry of permuting the periods within that week. The second constraint imposes a weak ordering on the first period of each week, helping to break the symmetry of permuting the weeks themselves.

1.3.3 Implied constraints

An implied constraint to bound the number of home games per team was tested. The rationale was that, over the $n-1$ weeks, each team must play $n-1$ games. Therefore, the number of home games H_t for any team t must be close to half of that total. The constraint was formulated as:

$$\forall t \in \text{TEAMS}:\ \left\lfloor \frac{n-1}{2} \right\rfloor - 1 \leq H_t \leq \left\lceil \frac{n-1}{2} \right\rceil + 1$$

where $H_t = \sum_{w \in \text{WEEKS}} \sum_{p \in \text{PERIODS}} \mathbf{1}(\texttt{home}[w,p] = t)$.

Explanation and Result: While logically redundant (as the round-robin and weekly constraints already define the total number of games), this constraint was intended to provide tighter bounds for the solver's propagation engine, potentially pruning invalid assignments to the 'home' variables earlier. However, in practice, its addition did not yield a measurable improvement in solver performance for the instances we tested. The significant performance gains were already achieved by the powerful symmetry-breaking constraints, particularly **C6**. The overhead of propagating this additional global cardinality constraint for all teams likely offset any potential benefits, leading to the decision to omit it from the final model to maintain clarity and solving speed.

1.4 Validation

1.4.1 Experimental design

The model was implemented in MiniZinc. The Gecode solver was used as the primary solver for validation, with Chuffed, coin-bc, cp-sat and highs also tested for comparison. A time limit of 300 seconds was set for each run.

A comprehensive evaluation of search strategies was conducted. The base strategy was defined as:

```
solve :: int_search(
    [home[i,j] | i in WEEKS, j in PERIODS] ++
    [away[i,j] | i in WEEKS, j in PERIODS],
    input_order,
    indomain_min
) satisfy;
```

Beyond this base configuration, we empirically tested several alternative strategies known to be effective for scheduling problems:

- **Smallest Domain + Random Value:** Using `first_fail` variable ordering (choosing the variable with the smallest domain size first) combined with `indomain_random` value selection to diversify the search.
- **Domain-Over-Weighted-Degree:** Using `dom_w_deg` variable ordering, which prioritizes variables involved in the most constraints that are currently "tight," to trigger more effective constraint propagation early.
- **Two-Phase Search:** Decomposing the search to first assign all `home` variables before assigning any `away` variables, leveraging the structure of our model where home assignments heavily constrain the possible away assignments.

Furthermore, the performance of the model was assessed with and without the key symmetry-breaking constraint **C6** (`away > home`) to isolate and quantify its critical contribution.

1.4.2 Experimental results

The performance of the CP model was evaluated on instances with n teams, where $n \in \{6, 8, 10, 12, 14, 16\}$. A time limit of 300 seconds was imposed for each run. The results, showing the runtime in seconds required to find a feasible schedule, are presented in Table 1. A runtime of 300 seconds indicates that the solver failed to find a solution within the time limit. The objective was to find a feasible schedule (`satisfy`), so the `obj` field is `null` and the `optimal` field is `false` in all cases.

Table 1: CP Model Results: Runtime (seconds) for Finding a Feasible Schedule

Solver	**Number of Teams (n)**					
	6	8	10	12	14	16
Gecode	1.27	0.57	88.69	300	300	300
Chuffed	0.59	0.72	5.48	16.09	300	300
COIN-BC	1.28	261.87	289.80	227.34	300	300
CP-SAT	0.95	1.64	13.85	73.21	300	300
HiGHS	1.83	78.15	300	300	300	300

Discussion: The results demonstrate the scalability challenges of the STS problem. For small instances ($n = 6, 8$), all solvers found a solution quickly, often in under two seconds. The Chuffed solver consistently performed well, finding solutions for $n = 10$ and $n = 12$ in 5.48 and 16.09 seconds, respectively. The CP-SAT solver also showed strong performance on these mid-sized instances. In contrast, the Gecode solver struggled with instances where $n \geq 12$, and the HiGHS (MIP) solver failed to find a solution for $n \geq 10$ within the time limit. For larger instances ($n \geq 14$), the problem became intractable for all tested solvers under the given constraints and time limit, highlighting the combinatorial complexity of the Sports Tournament Scheduling problem.

2 SAT Model

2.1 Constraints

We express the problem requirements using logical clauses built on Boolean variables. Then, we defined the encodings, main problem, implied, and symmetry breaking constraints.

2.1.1 Encoding for Cardinality Constraints

To handle "exactly one" and "at most k" constraints, we use sequential encoding [?] that introduces helper Boolean variables to represent the sum of literals more efficiently. For a set of m Boolean variables $y_1, \ldots, y_m$, sequential encoding for the "at most one" case only needs $O(m)$ clauses, compared to the $O(m^2)$ required by pairwise encoding. This linear growth makes it far more scalable for larger problems.

2.1.2 Main Problem Constraints

1. **Every pair of teams plays exactly once**: For any unique pair of distinct teams (t_i, t_j) where $i \neq j$, exactly one game between them must occur over the entire tournament. This means that across all weeks w and all periods p, the disjunction of x_{w,p,t_i,t_j} (team t_i plays at home against t_j) and x_{w,p,t_j,t_i} (team t_j plays at home against t_i) must evaluate to true exactly once.

   Encoding: We used "exactly one" encoding on the set of literals $\{x_{w,p,t_i,t_j} \mid \forall w, p\} \cup \{x_{w,p,t_j,t_i} \mid \forall w, p\}$.

2. **Every team plays exactly once per week**: In each week w, every team t must participate in exactly one game, either as the home or away team.

   Encoding: For each team t_1 and week w, we used "exactly one" encoding on the set of literals $\{x_{w,p,t_1,t_2} \mid \forall p, t_2 \neq t_1\} \cup \{x_{w,p,t_2,t_1} \mid \forall p, t_2 \neq t_1\}$.

3. **Every team plays at most twice in the same period over the tournament**: For a given team t and a given period p, team t cannot be involved in more than two games across all $n - 1$ weeks.

   Encoding: We used "at most two" encoding on the set of literals $\{x_{w,p,t_1,t_2} \mid \forall w, t_2 \neq t_1\} \cup \{x_{w,p,t_2,t_1} \mid \forall w, t_2 \neq t_1\}$.

2.1.3 Implied Constraints

1. **Each period in each week has exactly one game**: For every week w and every period p, exactly one game must be scheduled [**?**].

   Encoding: We used "exactly one" encoding on the set of literals $\{x_{w,p,t_1,t_2} \mid \forall t_1, t_2, t_1 \neq t_2\}$.

2. **No team plays against itself** [**?**]:

   Encoding: For all w, p, t, we added the unit clause $\neg x_{w,p,t,t}$.

2.1.4 Symmetry Breaking Constraints

To enhance the solver's performance by reducing redundant search, we introduce the following symmetry-breaking constraints:

1. **Canonical first week assignments**: We fix the schedule of the first week (week 0) in a predefined, canonical manner. For each period p in week 0, we enforce that team $2p$ plays at home against team $2p + 1$. This eliminates solutions that are merely permutations of teams in the first week.

   $\forall p \in \{0, \ldots, n/2 - 1\} : x_{0,p,2p,2p+1} = \text{true}$.

2. **Enforcing home team index smaller than away team**: For every game, we ensured that the numerical index of the home team is always less than the numerical index of the away team. This removes symmetrical solutions where only the home and away roles are swapped (e.g., Team A vs. Team B is equivalent to Team B vs. Team A in terms of matchup, but this constraint makes one representation canonical).

   $\forall t_1, t_2 \in (t_1 \geq t_2) : x_{w,p,t_1,t_2} = \text{false}.$

2.2 Validation

The SAT model is implemented using the Z3 solver on Python. Z3 provides powerful tools for building and solving SAT and SMT problems. Setup:

- **Solver**: Z3 version 4.15.3 - 64 bit
- **CPU**: Intel i7-12700H
- **Software**: PyCharm Community Edition 2023.2.3 for Windows 11
- **Time limit**: 300 seconds
- **Instances**: from $n = 2$ until $n = 20$
- **Output format**: JSON structure containing the solution

Each problem instance is solved once, and the solution is carefully checked using a separate tool to make sure it meets all the required constraints.

2.2.1 Experimental Results

To assess the performance of the SAT model, we tested it on tournament instances with increasing numbers of teams. Table 2 shows the solving times using the Z3 solver, both with and without symmetry breaking (SB) enabled:

Table 2: SAT solving times in seconds

Number of teams	Z3 + SB	Z3 w/out SB
2	0	0
4	UNSAT	UNSAT
6	0	0
8	0	0
10	0	23
12	2	29
14	N/A	N/A

The results show something interesting: it's impossible to create a valid schedule for 4 teams using the current rules. With only 3 weeks and 2 time slots per week, the rule that limits how often a team can appear in the same time slot (no more than twice) makes it impossible to fit everything in.

The symmetry breaking constraints show significant impact on harder instances:

- For $n = 10$: Reduces runtime from 23 seconds to instantaneous solution
- For $n = 12$: Reduces runtime from 29 seconds to 2 seconds

While for lower n values no significant change is observed.

The model hits a clear performance limit around 12 teams. For smaller tournaments, it finds solutions in just a few seconds. But once the number of teams goes beyond 12, things get much harder. That's mainly because the number of variables grows $O(n^3)$. The constraints become more complex, and even with symmetry-breaking techniques, the search space becomes too large for the solver to handle efficiently.

3 MIP Model

This section describes the Mixed-Integer Programming (MIP) formulation of the Sports Tournament Scheduling problem. The model is implemented in Python using the PuLP library and solved with both CBC and HiGHS solvers. The approach relies on precomputing weekly pairings through the classical round-robin "circle method" so that the solver focuses on deciding the allocation of matches to periods and the assignment of home and away teams. The objective is to ensure fairness by balancing the number of home and away games across all teams.

3.1 Decision variables

The MIP formulation uses the following decision variables:

- $y_{(i,j),p} \in \{0,1\}$: binary variable, equal to 1 if the match between teams i and j is scheduled in period p of its assigned week.
- $h_{(i,j)} \in \{0,1\}$: binary variable, equal to 1 if team i (with $i < j$) plays at home against j, and 0 otherwise.
- `home_count`$[i] \in R_{\geq 0}$: continuous variable that counts the total number of home games played by team i.
- $z_{\min}, z_{\max} \in R$: continuous variables that represent respectively the minimum and maximum number of home games across all teams.
- $d_i^+, d_i^- \in R_{\geq 0}$: continuous deviation variables used to express the difference between `home_count`$[i]$ and the ideal value $(n-1)/2$.

3.2 Objective function

The objective is to promote fairness among teams by balancing the number of home and away games. Ideally, each team should have $(n-1)/2$ home games. To achieve this, we minimize the sum of absolute deviations:

$$\min \sum_{i=1}^{n} (d_i^+ + d_i^-)$$

subject to

$$\texttt{home\_count}[i] - \frac{n-1}{2} = d_i^+ - d_i^- \quad \forall i \in \text{TEAMS}.$$

This formulation drives the solution toward schedules where all teams have nearly the same number of home games.

3.3 Constraints

The following constraints guarantee a feasible tournament and remove unnecessary symmetries:

C1. One period per match: each match is assigned to exactly one period in its fixed week:

$$\sum_{p=1}^{n/2} y_{(i,j),p} = 1 \quad \forall (i,j).$$

C2. One game per (week, period): each week has exactly one game scheduled in every period:

$$\sum_{(i,j)\in \text{week}(w)} y_{(i,j),p} = 1 \quad \forall w, p.$$

C3. Period capacity: no team plays more than twice in the same period over the whole season:

$$\sum_{(i,j)\ni t} y_{(i,j),p} \le 2 \quad \forall t, p.$$

C4. Home count definition: the total number of home games for each team is defined by the home/away variables:

$$\texttt{home\_count}[i] = \sum_{j>i} h_{(i,j)} + \sum_{k<i} (1 - h_{(k,i)}).$$

C5. Symmetry breaking: to reduce equivalent solutions, the match of team 1 in week 1 is fixed to period 1.

3.4 Validation

The MIP model was implemented in Python using PuLP. Two solvers were tested: CBC and HiGHS . Both were given a time limit of 300 seconds. The experiments were run on instances with $n \in \{6, 8, 10, 12, 14\}$ teams.

Table 3 reports runtime (in seconds), optimality (1 if the solver proved optimality, 0 otherwise), and the fairness objective value (sum of deviations).

Table 3: MIP Model Results (time in seconds, obj = total deviation, opt=1 if optimal).

	HiGHS			CBC		
n	time	opt	obj	time	opt	obj
6	0	1	3	1	1	3
8	0	1	4	0	1	4
10	0	1	5	1	1	5
12	2	1	6	28	1	6
14	8	1	7	140	1	7

The results show that both solvers solved small instances ($n = 6, 8, 10$) almost instantly. For larger sizes ($n = 12, 14$), HiGHS was much faster: 2s vs 28s at $n = 12$, and 8s vs 140s at $n = 14$. In all cases, both solvers produced the same objective value and proved optimality. The objective values increase linearly with n, confirming that the fairness norm is satisfied and scale predictably along the number of teams.

4 Conclusions

Authenticity and Author Contribution Statement

References