Combinatorial Decision Making and Optimization

Alessandro Capialbi alessandro.capialbi@studio.unibo.it
Christian Joseph Fiore christian.fiore@studio.unibo.it

1 Introduction

In this project, we address the *Sports Tournament Scheduling* (STS) problem by applying a unified modeling approach across multiple computational paradigms: Constraint Programming (CP), Boolean Satisfiability (SAT), and Mixed Integer Programming (MIP). We focus on the Single Round Robin Tournament Scheduling problem, where each of N teams must play against every other team exactly once. The tournament is structured considering N teams (where N is even), $N-1$ weeks, $\frac{N}{2}$ matches per week (periods/time slots), with a total of $\frac{N(N-1)}{2}$ matches. The problem can be formulated as an assignment problem: let $M = \{(i,j) : 1 \leq i < j \leq N\}$ be the set of all possible matches, where $|M| = \frac{N(N-1)}{2}$. We need to assign each match $m \in M$ to a specific week $w \in \{0,\dots,N-2\}$ and period $p \in \{0,\dots,\frac{N}{2}-1\}$, such that:

$$f : M \to \{0,\dots,\frac{N}{2}-1\} \times \{0,\dots,N-2\} \tag{1}$$

is a valid scheduling function satisfying all tournament constraints. All approaches begin by precomputing the round-robin tournament matrix using the circle method, yielding a $\frac{N}{2} \times (N-1)$ matrix where each column corresponds to a week and contains the set of weekly matchups. Our experimental results show that the precomputation time is negligible (less than a second) for all considered instance sizes. Precomputing the matrix provides several advantages:

- It removes symmetries across weeks and matchup pairings, reducing the search space
- It guarantees that each team plays exactly once per week
- It ensures that each team plays against each other only once

A balanced tournament design is a permutation of the order of games within each week that limits the number of times each team plays in the same period. While the round-robin matrix defines the matchups, our models aim to compute an optimally balanced schedule by assigning games to periods in a way that minimizes the difference between home and away games for each team. All tests were run on a single core of an AMD Ryzen AI 9 HX 370.

1.1 Notation

We introduce the following notation used throughout this work:

- $T = \{0,1,\dots,N-1\}$: set of teams
- $W = \{0,1,\dots,N-2\}$: set of weeks
- $P = \{0,1,\dots,\frac{N}{2}-1\}$: set of periods (time slots) per week
- $M = \{0,1,\dots,\frac{N(N-1)}{2}-1\}$: set of match indices
- $S = \{0,1\}$: home (0) and away (1) designation
- $rb_{p,w,s} \in T$: precomputed round-robin schedule via circle method indicating which team plays in period p, week w, on slot s

- $\text{matches}_{m,s} \in T$: teams involved in match m, where $s \in S$ indicates the slots home/away and m is the match id.

2 CP Model

2.1 Decision Variables

The CP model employs the following decision variables:

- $\texttt{matches\_idx}_{p,w} \in M$: The match index assigned to period p in week w. This variable determines *which* precomputed match from the circle method is scheduled in *which* time slot.
- $\texttt{which\_is\_home}_{p,w} \in \{0, 1\}$: Binary variable indicating which team from the match pair is designated as the home team. If 0, the first team in $\text{matches}_{m,0}$ is home; if 1, the second team in $\text{matches}_{m,1}$ is home.

2.2 Auxiliary Variables

- $\texttt{Cha}_{t,s} \in \{0, 1, \ldots, N-1\}$: Auxiliary count array variable tracking how many times team $t \in T$ plays on slot $s \in S$ (where $s = 0$ represents home games and $s = 1$ represents away games). Its value is derived from the assignment of the decision variables `matches_idx` and `which_is_home`. For any team t: $\texttt{Cha}_{t,0} + \texttt{Cha}_{t,1} = N - 1$.

2.3 Objective function

In addition to the feasibility constraints, we seek to minimize the imbalance between home and away games. The objective function is defined as:

$$\texttt{imbalance} = \sum_{t \in T} |\texttt{Cha}_{t,0} - \texttt{Cha}_{t,1}| \tag{2}$$

and we aim to **minimize** it.
Given the constraints of the problem, there exists a theoretical lower bound on the achievable imbalance. Since each team plays $N - 1$ games total (where $Cha_{t,0} + Cha_{t,1} = N - 1$), and N is even by assumption, $N - 1$ is odd. Therefore, it is impossible for any team to have an equal number of home and away games. The best possible balance for a single team is:

$$|Cha_{t,0} - Cha_{t,1}| \geq 1 \quad \forall i \in T \tag{3}$$

Consequently, the minimum total imbalance across all teams is:

$$\sum_{i=1}^{N} |Cha_{t,0} - Cha_{t,1}| \geq N \tag{4}$$

This lower bound is tight and achievable: a schedule where every team plays exactly $\frac{N}{2}$ home games and $\frac{N-2}{2}$ away games (or vice versa) achieves this minimum imbalance of N. Such schedules are considered *perfectly balanced* in the context of this problem.

2.4 Constraints

2.4.1 Core Constraints

1. **All matches are scheduled exactly once**:

$$\texttt{alldifferent}(\texttt{matches\_idx}) \tag{5}$$

   This global constraint ensures all $\frac{N(N-1)}{2}$ matches are scheduled, without any duplicates.

2. **Each team plays at most twice in the same period**:

$$\forall p \in P, \forall t \in T : \left|\left\{(w, s) \in W \times S \mid \texttt{matches}_{\texttt{matches\_idx}_{p,w},s} = t\right\}\right| \leq 2 \tag{6}$$

For each period p and team t, this constraint counts in how many (week, slot) pairs team t appears during that period, ensuring no team is scheduled more than twice in the same time slot across all weeks. Implemented using the global cardinality constraint.

3. **Week-period structure from circle method**:

$$\forall w \in W, \forall p \in P : w \cdot \frac{N}{2} \leq \texttt{matches\_idx}_{p,w} < (w+1) \cdot \frac{N}{2} \tag{7}$$

This constraint enforces that only matches generated for week w by the circle method (indices in range $[w \cdot \frac{N}{2}, (w+1) \cdot \frac{N}{2})$) can be assigned to periods in that week. Since the circle method pre-generates $\frac{N}{2}$ valid matches per week ensuring each team plays exactly once, this constraint preserves the round-robin property while allowing optimization of period assignments.

2.4.2 Symmetry Breaking Constraints

1. **Period symmetry breaking**: Swapping matches across periods yields equivalent solutions, therefore we impose a *lex_less* ordering.

$$\forall p \in \{0, \ldots, \frac{N}{2} - 2\} : [\texttt{matches\_idx}[p, w] \mid w \in W] <_{lex} [\texttt{matches\_idx}[p+1, w] \mid w \in W] \tag{8}$$

This orders the rows (periods) lexicographically, eliminating period permutation symmetries and reducing the search space.

2. **Fix first match**:

$$\texttt{matches\_idx}[0, 0] = 0 \tag{9}$$

Fixes the first match of the round-robin to be scheduled in period 0 of week 0. This breaks symmetry more effectively than fixing an entire week, as it establishes a single anchor point while allowing the solver maximum flexibility in scheduling the remaining matches. Fixing a full week would over-constrain the problem and potentially eliminate valid solutions.

2.4.3 Implied Constraints

Implied constraints are redundant constraints that do not change the solution set but strengthen propagation and help the solver detect conflicts earlier.

1. **Per-period all-different constraint**:

$$\forall p \in P : \texttt{alldifferent}([\texttt{matches\_idx}[p, w] \mid w \in W]) \tag{10}$$

This constraint is redundant with the global all-different constraint, but posting it per-period creates stronger propagation at the local level. It helps the solver detect conflicts earlier when assigning matches to specific periods by ensuring that each period across all weeks contains distinct match IDs.

2. **Per-week all-different constraint**:

$$\forall w \in W : \texttt{alldifferent}([\texttt{matches\_idx}[p, w] \mid p \in P]) \tag{11}$$

This provides the dual view of the global constraint, ensuring that within each week, all periods have different matches. While also redundant, this strengthens propagation when assigning matches within a single week, complementing the per-period constraint and improving the solver's ability to prune the search space efficiently.

2.4.4 Channeling Constraints

Channeling constraints link different decision variables together, maintaining consistency between alternative views of the problem.

1. **Home/Away channeling constraint**:
   For each slot $s \in S$, we define:

$$\texttt{teams\_in\_slot}_s = [\texttt{team}_{s,p,w} \mid p \in P, w \in W] \tag{12}$$

   where

$$\texttt{team}_{s,p,w} = \begin{cases} \texttt{matches}_{\texttt{matches\_idx}_{p,w},0} & \text{if } (s=0) \equiv (\texttt{which\_is\_home}_{p,w} = 0) \\ \texttt{matches}_{\texttt{matches\_idx}_{p,w},1} & \text{otherwise} \end{cases} \tag{13}$$

   Then, the channeling constraint is enforced as:

$$\forall s \in S, \forall t \in T : \texttt{Cha}_{t,s} = |\{i \mid \texttt{teams\_in\_slot}_s[i] = t\}| \tag{14}$$

   This constraint channels the assignment decisions (`matches_idx` and `which_is_home`) to the counting variables `Cha`. The array $\texttt{teams\_in\_slot}_s$ collects all teams playing in slot s across all periods and weeks, then $\texttt{Cha}_{t,s}$ counts the occurrences of team t in that array, ensuring consistency between the match schedule and home/away statistics.

2.5 Validation

2.5.1 Experimental Design

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

Solvers: Three state-of-the-art CP solvers were employed: *Gecode 6.3.0*, *Chuffed 0.13.2* and *OR-Tools CP-SAT 9.15.6755*. All experiments were conducted with a uniform time limit of 300 seconds per instance and under single-threaded execution to ensure fair comparison.

Model Configurations: To evaluate the impact of different modeling techniques, we tested four configurations of the optimization model: **baseline**, **baseline + symmetry breaking**, **baseline + implied constraints**, **complete**.

Search Strategies: To analyze the impact of search guidance on solver performance, we employed four distinct search strategies:

1. **Int Search Strategy**:
   Uses `int_search` strategy to guide the solver's variable and value selection. This strategy concatenates the match assignment variables (`matches_idx`) with the home/away assignment variables (`which_is_home`), prioritizes variables with the smallest remaining domain and tries the minimum value first for each variable.

2. **Default Search Strategy**:
   Uses MiniZinc's default search annotation `solve minimize function`, allowing each solver to rely entirely on its built-in decision heuristics and restart policies. This serves as a baseline for evaluating the solvers' inherent capabilities.

3. **Sequential Custom Search**:
   A manually defined sequential search strategy that assigns variables in two phases:

   - First phase: **int_search** on `matches_idx` variables using `dom_w_deg` heuristic with `indomain_min` value selection

- Second phase: **int_search** on `which_is_home` variables using `dom_w_deg` heuristic with `indomain_min` value selection

4. **Search with Luby Restarts and LNS**:
   Extends the basic int search by incorporating a Luby restart policy with scale factor 100, enabling the solver to escape local minima more effectively. It also implements a Large Neighborhood Search (LNS) on `matches_idx` variables, preserving 60% of solution values during reconstruction.

2.5.2 Experimental results

The following tables present the times for the satisfaction version problem (*noIMPL* model only) and the objective values for the optimization one obtained by each solver across four model variants:

- **baseline**: baseline model without additional constraints
- **complete**: full model with both implied and symmetry breaking constraints
- **noIMPL**: model with symmetry breaking but without implied constraints
- **noSB**: model with implied constraints but without symmetry breaking

The experimental results in Table 1 reveal that Gecode and CP-SAT have similar behaviors, solving instances up to $N = 16$. Chuffed has the poorest performance. We chose the noIMPL model because it has better results. Tables 2, 3, 4, and 5 reveal that CP-SAT demonstrates superior performance across all configurations, finding optimal solutions up to $N = 22$ with the default search strategy, while other solvers struggle beyond $N = 16$. Chuffed exhibits the weakest performance, frequently timing out on medium-sized instances. An unexpected finding emerges from Table 3: the `baseline` variant without additional constraints achieves optimal solutions for larger instances than the constrained variants with both Gecode and CP-SAT. This suggests that the combination of implied and symmetry breaking constraints, while theoretically beneficial for pruning, may create search bottlenecks that hinder the solver's native heuristics. We also noticed that implied constraints not always increase the number of propagations, especially with the custom search strategies. These (int_search and seq_search) show more uniform behavior across variants but fail to scale beyond $N = 16$ for most solvers. Finally, the application of Luby restarts with Large Neighborhood Search (Table 5) does not yield performance improvements, indicating that this problem structure does not benefit from randomized restart strategies.

N	noIMPL - GECODE	noIMPL - CHUFFED	noIMPL - CP-SAT
6	<1s	<1s	<1s
8	<1s	<1s	<1s
10	<1s	<1s	<1s
12	<1s	<1s	<1s
14	<1s	21	<1s
16	4	300	5
18	300	300	300
20	300	300	300

Table 1: Solving time for satisfaction model without optimization

N	GECODE				CHUFFED				CP-SAT			
	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
8	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
10	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
14	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
16	**16**	**16**	**16**	**16**	N/A	N/A	N/A	N/A	**16**	**16**	**16**	**16**
18	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: Objective values with Int Search Strategy and first_fail

N	GECODE				CHUFFED				CP-SAT			
	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
8	16	16	16	16	**8**	**8**	**8**	12	**8**	**8**	**8**	**8**
10	16	14	14	16	30	18	22	22	**10**	**10**	**10**	**10**
12	**12**	**12**	**12**	**12**	34	N/A	34	30	**12**	**12**	**12**	**12**
14	**14**	**14**	**14**	**14**	N/A	N/A	N/A	N/A	**14**	**14**	**14**	**14**
16	**16**	**16**	**16**	N/A	N/A	N/A	N/A	N/A	**16**	**16**	**16**	**16**
18	**18**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**18**	**18**	**18**	**18**
20	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**20**	**20**	**20**	**20**
22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**22**	**22**	**22**	**22**

Table 3: Objective values with Default Search Strategy

N	GECODE				CHUFFED				CP-SAT			
	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
8	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
10	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
14	**14**	**14**	N/A	**14**	N/A	N/A	N/A	N/A	**14**	**14**	**14**	**14**
16	N/A	**16**	N/A	**16**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
18	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 4: Objective values with Sequential Custom Search Strategy

N	GECODE				CHUFFED				CP-SAT			
	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB	baseline	complete	noIMPL	noSB
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
8	**8**	**8**	**8**	**8**	**8**	**8**	**8**	12	**8**	**8**	**8**	**8**
10	**10**	**10**	**10**	**10**	30	18	24	22	**10**	**10**	**10**	**10**
12	**12**	**12**	**12**	**12**	58	36	48	44	**12**	**12**	**12**	**12**
14	**14**	**14**	**14**	**14**	N/A	N/A	N/A	N/A	**14**	**14**	**14**	**14**
16	**16**	**16**	**16**	**16**	N/A	N/A	N/A	N/A	**16**	**16**	**16**	**16**
18	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 5: Objective values with Luby Restarts and LNS

3 SAT Model

The SAT-based approach uses pure Boolean logic to encode the scheduling problem. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. The constraints are expressed using the following encodings:

- $\texttt{AtMostK}(X, k)$ with sequential encoding
- $\texttt{ExactlyOne}(X) = \texttt{AtMostOne}(X) \wedge \texttt{AtLeastOne}(X)$, where:
  - $\texttt{AtLeastOne}(X) = \bigvee_{i=1}^{n} x_i$
  - $\texttt{AtMostOne}(X)$ with pairwise encoding: $\bigwedge_{1 \le i < j \le n} \neg(x_i \wedge x_j)$
- $\texttt{ExactCount}(X, k)$ links indicator variables to their cardinality using bidirectional implications between count variables and at-most constraints

Decision variables are encoded using one-hot representations, where each possible value is represented by a Boolean variable with exactly one being true. To maintain solver independence, we implement a unified constraint library that automatically selects the appropriate encoding based on the backend solver (Z3 or OR-Tools CP-SAT), exploiting solver-specific optimizations where available (e.g., native `AddAtMostOne` in CP-SAT).

3.1 Decision Variables

The SAT model uses the following Boolean decision variables organized into two categories:

3.1.1 Satisfiability

- **Match Schedule:**
  For each period $p \in P$, week $w \in W$, and match $m \in M$:

$$\texttt{matches\_idx\_vars}_{p,w,m} = \text{true} \iff \text{match } m \text{ is played in period } p \text{ of week } w$$

3.1.2 Optimization

- **Home/Away Assignment (Flip Slot):**
  For each period $p \in P$ and week $w \in W$:

$$\texttt{home\_is\_first\_vars}_{p,w} = \text{true} \iff \text{the first team in the match pair plays at home}$$

- **Home/Away Count:**
  For each team $t \in T$ and count $k \in \{0, \ldots, N-1\}$:

$$\texttt{home\_count\_vars}_{t,k} = \text{true} \iff \text{team } t \text{ plays exactly } k \text{ home games}$$

$$\texttt{away\_count\_vars}_{t,k} = \text{true} \iff \text{team } t \text{ plays exactly } k \text{ away games}$$

  One-hot encoded: exactly one k is true for each team t.

- **Imbalance**:
  For each team $t \in T$ and difference $d \in \{0, \ldots, N-1\}$:

$$\texttt{diff\_vars}_{t,d} = \text{true} \iff |\texttt{home\_count}_t - \texttt{away\_count}_t| = d$$

  One-hot encoded: exactly one d is true for each team t.

3.2 Objective Function

The objective is to minimize the total imbalance across all teams:

$$\text{minimize} \quad \sum_{t \in T} |\text{home\_count}_t - \text{away\_count}_t|$$

Since SAT solvers do not natively support optimization, we implement **iterative optimization**:

1. Start with target imbalance $\tau = N$ (minimum possible total imbalance)
2. Add constraint: $\sum_{t \in T} \texttt{diff}_t = \tau$
3. Check satisfiability
4. If SAT: optimal solution found with imbalance τ
5. If UNSAT: increment τ and repeat
6. Continue until a satisfiable solution is found

The constraint $\sum_{t \in T} \texttt{diff}_t = \tau$ is encoded in pure Boolean logic by enumerating all valid combinations of per-team imbalance values that sum to τ and creating a disjunction over these combinations. For each valid combination, we generate a conjunction of the corresponding one-hot variables $\texttt{diff}_{t,d}$, then combine all such conjunctions with an OR clause.

3.3 Constraints

3.3.1 Core Constraints

- **One-Hot Encoding for Match Positions:** Each position (p, w) must have exactly one match assigned:
$$\forall p \in P, w \in W : \text{exactly\_one}(\{\texttt{matches\_idx\_vars}_{p,w,m} \mid m \in M\})$$
- **All-Different Matches:** Each match m must be assigned to exactly one position:
$$\forall m \in M : \text{exactly\_one}(\{\texttt{matches\_idx\_vars}_{p,w,m} \mid p \in P, w \in W\})$$
- **Week Range Constraint:** Week w can only use matches from the range $[w \cdot (N/2), (w+1) \cdot (N/2))$:
$$\forall w \in W, p \in P, m \in M : m \notin [w \cdot \frac{N}{2}, (w+1) \cdot \frac{N}{2}) \Rightarrow \neg \texttt{matches\_idx\_vars}_{p,w,m}$$
- **At Most Two Games Per Team Per Period:** Each team can play at most twice in any period:
$$\forall t \in T, p \in P : \text{at\_most\_k}(\{\texttt{matches\_idx\_vars}_{p,w,m} \mid w \in W, m \text{ involves team } t\}, k = 2)$$
- **Home/Away Count Linking:** The home and away count variables must accurately reflect actual assignments:
$$\forall t \in T : \texttt{home\_count\_vars}_{t,k} = \text{true} \iff |\{(p, w) \mid t \text{ plays home at } (p, w)\}| = k$$
$$\forall t \in T : \texttt{away\_count\_vars}_{t,k} = \text{true} \iff |\{(p, w) \mid t \text{ plays away at } (p, w)\}| = k$$

3.3.2 Symmetry Breaking Constraints

1. **Fix First Match**

To break symmetry in the search space, we fix the first match:

$$\texttt{matches\_idx\_vars}_{0,0,0} = \text{true}$$

This constraint forces match 0 to be played in period 0, week 0.

3.3.3 Encoding Strategies

The model employs several SAT encoding techniques:

- **Sequential Counter Encoding**: Used for at-most-k constraints. Based on Sinz's sequential encoding, it introduces auxiliary variables $s_{i,j}$ meaning "among the first $i+1$ variables, at least $j+1$ are true."
- **Naive Pairwise Encoding**: Used for small at-most-one constraints, creating pairwise implications $\neg(x_i \wedge x_j)$ for all pairs.
- **One-Hot Encoding**: Integer variables (counts, imbalances) are encoded as Boolean vectors where exactly one element is true, representing the integer value.

3.4 Validation

3.4.1 Experimental design

The SAT model was written in Python by using Z3 Python API and the OR-Tools library. The times for decision mode and objective values for optimization found within the 300 second time limit, are presented in Table 6.

3.4.2 Experimental Results

All solvers find an optimal solution or no solution at all. Overall, CP-SAT can solve larger instances than Z3 and it is faster when considering the basic decision problem. The maximum N reached is 20, with the decision version, while $N = 16$ for optimization.

N	**Z3**		**CP-SAT**	
	Decision	Optimize	Decision	Optimize
6	<1s	**6**	<1s	**6**
8	1s	**8**	<1s	**8**
10	4s	N/A	<1s	**10**
12	11s	N/A	<1s	**12**
14	32s	N/A	<1s	**14**
16	81s	N/A	19s	**16**
18	170s	N/A	14s	N/A
20	N/A	N/A	97s	N/A
22	N/A	N/A	N/A	N/A

Table 6: SAT Model Results - Z3 and CP-SAT

4 MIP Model

4.1 Decision variables

- $\texttt{matches\_one\_hot}_{m,p,w} \in \{0,1\}$: equals 1 if match m is scheduled in period p of week w
- $\texttt{first\_is\_home}_m \in \{0,1\}$: equals 1 if in match m the first team plays at home
- $y_{m,p,w} \in \{0,1\}$: linearization variable, $y_{m,p,w} = \texttt{matches\_one\_hot}_{m,p,w} \cdot \texttt{first\_is\_home}_m$
- $\texttt{home\_count}_t \in \mathbb{Z}_+$: number of home games for team t
- $\texttt{away\_count}_t \in \mathbb{Z}_+$: number of away games for team t
- $\texttt{diff}_t \in \mathbb{Z}_+$: absolute difference between home and away games for team t
- $\texttt{imbalance} \in \mathbb{Z}_+$: total imbalance across all teams

4.2 Objective function

The objective function minimizes the total home/away imbalance across all teams, which is defined as the sum of the absolute differences between the number of home and away games for each team, as in the other cases. This is always captured by the decision variable `imbalance`.

4.3 Constraints

- **Linearization:**

$$y_{m,p,w} \leq \texttt{matches\_one\_hot}_{m,p,w} \qquad \forall m,p,w \tag{15}$$
$$y_{m,p,w} \leq \texttt{first\_is\_home}_{m} \qquad \forall m,p,w \tag{16}$$
$$y_{m,p,w} \geq \texttt{matches\_one\_hot}_{m,p,w} + \texttt{first\_is\_home}_{m} - 1 \qquad \forall m,p,w \tag{17}$$

- **Match assignment:**

$$\sum_{p,w} \texttt{matches\_one\_hot}_{m,p,w} = 1 \qquad \forall m \tag{18}$$
$$\sum_{m} \texttt{matches\_one\_hot}_{m,p,w} = 1 \qquad \forall p,w \tag{19}$$

- **Week integrity:** matches from week w must be scheduled in week w

$$\sum_{m \in W_w} \texttt{matches\_one\_hot}_{m,p,w} = 1 \quad \forall p,w \tag{20}$$

  where W_w is the set of matches originally assigned to week w (in range $[w \cdot \frac{N}{2}, (w+1) \cdot \frac{N}{2})$).

- **Period limit:** each team plays at most twice in the same period

$$\sum_{w,m:t \in m} \texttt{matches\_one\_hot}_{m,p,w} \leq 2 \quad \forall t,p \tag{21}$$

- **Home/away counts:** for each team t

$$\texttt{home\_count}_t = \sum_{m,p,w} \left[y_{m,p,w} \cdot \mathbb{1}_{\{t=m_1\}} + (\texttt{matches\_one\_hot}_{m,p,w} - y_{m,p,w}) \cdot \mathbb{1}_{\{t=m_2\}} \right] \tag{22}$$
$$\texttt{away\_count}_t = \sum_{m,p,w} \left[(\texttt{matches\_one\_hot}_{m,p,w} - y_{m,p,w}) \cdot \mathbb{1}_{\{t=m_1\}} + y_{m,p,w} \cdot \mathbb{1}_{\{t=m_2\}} \right] \tag{23}$$
$$\texttt{home\_count}_t + \texttt{away\_count}_t = n - 1 \tag{24}$$

  where m_1, m_2 are respectively the first and second team in the match m.

- **Imbalance calculation:**

$$\texttt{diff}_t \geq \texttt{home\_count}_t - \texttt{away\_count}_t \qquad \forall t \tag{25}$$
$$\texttt{diff}_t \geq \texttt{away\_count}_t - \texttt{home\_count}_t \qquad \forall t \tag{26}$$
$$\texttt{imbalance} = \sum_t \texttt{diff}_t \tag{27}$$
$$\texttt{imbalance} \geq n \tag{28}$$

- **Symmetry breaking:**

$$\texttt{matches\_one\_hot}_{0,0,0} = 1 \tag{29}$$
$$\sum_{m,w} m \cdot \texttt{matches\_one\_hot}_{m,p,w} \leq \sum_{m,w} m \cdot \texttt{matches\_one\_hot}_{m,p+1,w} \qquad \forall p < |P| - 1 \tag{30}$$

4.4 Validation

The model is written in Python by making use of the PuLP library. The solvers tested on the model were: CBC 2.10.3, HiGHS 1.12.0, all used with their default parameters. Tests were made up to $N = 22$.

4.5 Experimental results

As shown in Table 7, CBC has the worst performance, it isn't able to find the optimal solution for N greater than 12. HiGHS, instead, stops at N = 14, with the best performance.

N	**CBC**	**HiGHS**
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
20	N/A	N/A
22	N/A	N/A

Table 7: MIP Model - Optimization Results

5 Conclusions

This project successfully addressed the sRR STS problem through a unified modeling approach across three computational paradigms: CP, SAT, and MIP. All approaches leveraged the circle method to precompute the round-robin structure, reducing the problem complexity from match generation to optimal period assignment. The experimental results demonstrate that CP-SAT consistently achieves the best performance, reliably finding optimal solutions up to $N = 22$ teams within the 300-second timeout. Among CP solvers, Gecode performs competitively up to $N = 16$ with guided search strategies, while Chuffed frequently times out on small/medium-sized instances. The SAT approach achieves optimal solutions up to $N = 16$ using CP-SAT, while Z3 faces scalability challenges beyond $N = 8$ due to the computational overhead of iterative optimization. The MIP model shows the most limited scalability: HiGHS reaches $N = 14$ and CBC stops at $N = 12$, with linearization constraints and binary variables creating computational bottlenecks. An important finding is that model complexity does not always correlate with solver performance. Notably, for Gecode at $N = 18$, only the baseline model finds the optimal solution while enhanced variants time out. This suggests that combining symmetry breaking and implied constraints can create excessive pruning that interferes with solver-specific propagation, with CP-SAT showing robustness across all variants while Gecode exhibits high sensitivity. Additionally, Luby Restarts with LNS (Table 5) do not improve performance, indicating that restart overhead outweighs potential benefits.

Authenticity and Author Contribution Statement We declare that this work is our own and has not been copied from any other source. All external ideas and resources have been properly cited in the References section. All the modelling decisions were agreed on as a group, while the implementations were carried out more autonomously:

- **Alessandro Capialbi**: CP + decision making for other approaches + report
- **Christian Joseph Fiore**: SAT + MIP

References

[1] A. Dimitsas, C. Gogos, C. Valouxis, A. Tzallas, and P. Alefragis, *A Pragmatic Approach for Solving the Sports Scheduling Problem*, in Proceedings of the 13th International Conference on the Practice and

Theory of Automated Timetabling (PATAT 2022), Volume III, 2022. https://www.patatconference.org/patat2022/proceedings/PATAT_2022_paper_21.pdf

[2] F. Rossi, P. van Beek, and T. Walsh (eds), *Handbook of Constraint Programming*, Elsevier Science, Amsterdam, Netherlands, 2006.