

STS - Sport Tournament Scheduling

Samuele Centanni, Tomaž Cotič, Mattia Lodi, Artem Uskov
{samuele.centanni, tomaz.cotic, mattia.lodi, artem.uskov}@studio.unibo.it

CDMO – September, 2025

1 Introduction

The problem addressed in this project is the the Sports Tournament Scheduling (STS). We approach it using Constraint Programming (CP), SAT solvers, Satisfiability Modulo Theory (SMT), and Mixed Integer Programming (MIP). All the developed models share a common formalization, described in this section. All experiments were conducted respecting the given timeout of 300 s, with the solvers in their sequential version and fixed random seeds.

1.1 Input Parameters.

The main input parameter of the model is the number of teams (an even integer) n . From it, we define the following quantities used throughout the project and this report: (i) number of weeks $w = n - 1$ (ii) number of periods $p = \frac{n}{2}$ (iii) set of teams $T = \{1, \dots, n\}$ (iv) set of weeks $W = \{1, \dots, n - 1\}$ (v) set of periods $P = \{1, \dots, n/2\}$.

1.2 Objective variable.

The objective function is the same across all proposed approaches. Specifically, we aim to minimize the maximum absolute difference between the number of home and away matches played by any team:

$$O = \max_{t \in T} |H_t - A_t|,$$

where H_t denotes the number of home games of team t , A_t denotes the number of away games of team t . The objective variable is bounded between a lower limit of 1 and an upper limit of $n - 1$.

Moreover, initially, we considered an alternative formulation:

$$O = \sum_{t \in T} |H_t - A_t|,$$

but empirical evaluation showed that minimizing the maximum imbalance consistently produced significantly better results across all the techniques used.

1.3 Pre-solving with the Circle Method.

In each of the four approaches addressed in this report we used a pre-processing step, *circle method*, that helped improving the quality of the found solutions avoiding additional decision variables [1].

The circle method generates round-robin schedules by fixing one team as a pivot and rotating after every round, or week, the others around it in a circle. In each week, the teams are paired according to their positions in the circle, where each team plays against the team directly opposite to it. By construction two core constraints of the problem are satisfied:

- every team plays against every other team exactly once
- every team plays once a week.

2 CP Model

2.1 Decision Variables

The CP model relies on the following decision variables:

- For each unordered pair of teams (i, j) we define $\text{period}_{i,j} \in P$ as the slot in which team i plays against team j .
- Similarly, for each unordered pair of teams (i, j) we define the variables $\text{home}_{i,j} \in \{0, 1\}$ in such a way that $\text{home}_{i,j} = 1$ when i plays against j at home.

Moreover, using the previously described *circle method*, all week assignments are precomputed and passed to the model in the variables $\text{week}_{i,j} \in W$, where $\text{week}_{i,j} = w$ means that i plays against j in week w .

2.2 Objective Function

To implement the objective function described in Section 1.2 in the CP model, the number of games each team plays at home and away are stored in two arrays:

$$\forall t \in T \quad \text{home_count}[t] \in \{1, \dots, n - 1\}, \quad \text{away_count}[t] \in \{1, \dots, n - 1\}.$$

The imbalance for each team is stored in

$$\text{imbalance}[t] = |\text{home_count}[t] - \text{away_count}[t]|$$

and the optimization variable is taken as the max over the array.

2.3 Constraints

Thanks to the previously described *circle method* only two constraints had to be implemented to solve the problem in a correct way:

Each team plays at most twice in the same period:

$$\forall t \in T, \forall p \in P : \sum_{j \in T \setminus \{t\}} \chi_{\{\text{period}_{t,j}=p\}} \leq 2,$$

with χ being the characteristic function. We implemented this constraint using the global constraint `count_geq`, which yielded better results than both the intuitive formulation and the alternative global constraint `global_cardinality`.

Each period in each week contains exactly one match:

$$\forall w \in W, \forall p \in P : \sum_{(i,j) \in M} \chi_{\{\text{week}_{i,j}=w \wedge \text{period}_{i,j}=p\}} = 1,$$

where M is the set of all matches $M = \{(i, j) : i, j \in T, i < j\}$.

We tried to tackle this constraint using the global constraint `alldifferent` but it weakened the performance on basically all instances, which is why we didn't implement this constraint using the available global one.

Additional constraints were required due to the way we defined the decision variables. Specifically, we allowed the slot variable to take the value 0, even though it does not correspond to a valid slot in which teams can play. Furthermore, the slot matrix must be symmetric, and the home team assignment also exhibits a particular structure by design:

- $\forall t \in T : \text{period}_{t,t} = 0$
- $\text{period}_{i,j} \neq 0,$
- $\forall i, j \in T, i \neq j : \text{period}_{i,j} = \text{period}_{j,i},$
 $\text{home}_{j,i} = 1 - \text{home}_{i,j}$

Symmetry Breaking Constraints

We identified symmetries that enlarge the solver's search space, so we added symmetry-breaking constraints to avoid redundant exploration:

SB1: Fixing the slots of the first team:

$$\forall t \in T : \text{period}_{1,t} \leq \text{period}_{1,t+1}$$

This constraint ensures that the slots assigned to the first team are strictly increasing, effectively removing symmetries that arise from permuting slot orders. We implemented it using the MiniZinc global constraint `increasing`.

SB2: Team 1's home/away pattern is fixed:

$$\forall w \in W : \text{home}_{1,w+1} = \begin{cases} 1, & 1 \leq w \leq \lfloor \frac{n}{2} \rfloor, \\ 0, & \lfloor \frac{n}{2} \rfloor + 1 \leq w \leq n - 1 \end{cases}$$

This constraint eliminates symmetries that come from flipping home-away status forcing the first team to play the first half of the season at home and the second away.

2.4 Validation

The models were implemented in MiniZinc and coupled with a Python script that determines week assignments and executes the MiniZinc model. We used **Gecode** and **Chuffed** with different search strategies.

2.4.1 Experimental Results (Decision version)

In Figure 1, we present the results for the decision version of the problem. The best configuration combines Chuffed with the random-order variable selection heuristic and Luby restarts, using a restart scale of 50. Overall, the results are

very similar to those of the optimization version, which is why the discussion of the chosen approaches is deferred to the following section. In both sections smaller instances (4, 6, 8) are left out as all the approaches solved them instantly.

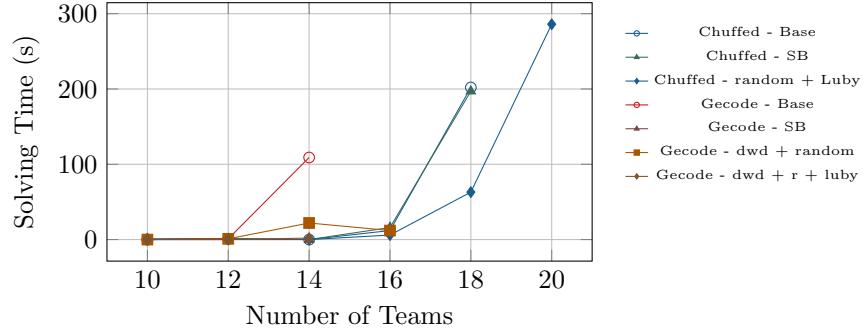


Figure 1: Runtime in seconds for the decision version.

2.4.2 Experimental Results (Optimization version)

Our first objective was to evaluate the impact of the proposed symmetry-breaking constraints. As shown in Table 1, the inclusion of symmetry breaking yields clear benefits for both Gecode and Chuffed.

Having established the advantages of symmetry breaking, we next investigated alternative search strategies to further enhance performance of both solvers. All subsequent experiments were conducted with symmetry breaking enabled.

For the optimization version with Chuffed, we were not able to identify search strategies that improved the model’s solutions, highlighting the effectiveness of Lazy Clause Generation. Among the tested configurations, the best was `random_order` for variable selection combined with a `Luby(50)` restart strategy. We also experimented with `first_fail`, but regardless of the chosen value-assignment heuristic, no improvement was observed, although `indomain_split` performed slightly better than the others.

We additionally tested enabling free search, allowing the solver to switch between the specified search annotations and its default search. However, this significantly degraded performance.

For Gecode, we initially experimented with the `first_fail` variable selection heuristic, but it did not yield any performance improvement. Switching to the domain over weighted degree heuristic, combined with random value assignment, we obtained a significant improvement.

We also attempted to combine this strategy with restarts, specifically Luby, but the results weakened. This trend was consistent across several restart scales arbitrarily chosen within [20, 150].

Additionally, we evaluated the large neighborhood search (LNS) strategy; however, it provided no benefit regardless of the retention percentage. As to

be expected, this approach was only tested for the optimization version of the problem.

Overall, Chuffed consistently outperformed Gecode, producing better results across nearly all non-trivial instances, with the exception of some faster solutions that Gecode was able to find for $n = 16$.

n	Chuffed			Gecode			
	Base	SB	random+luby(50)	Base	SB	dwd+random	dwd+r+luby(50)
10	0 1	0 1	0 1	0 1	0 1	0 1	0 1
12	1 1	1 1	2 1	1 1	0 1	1 1	1 1
14	1 1	6 1	6 1	300 7	0 1	36 1	12 1
16	300 3	22 1	87 1	N/A	N/A	14 1	300 9
18	300 11	300 5	300 7	N/A	N/A	N/A	N/A

Table 1: Optimization version: runtime in seconds and found objective value.

3 SAT Model

3.1 Decision Variables

To encode the Sports Tournament Scheduling (STS) problem, we define the following decision variables:

1. **match_period_vars**[i, j, p] $\in \{\text{True}, \text{False}\}$, for all $i, j \in T$ with $i < j$, and $p \in P$.
This variable is true if and only if team i plays against team j during period p . The week for each match-up (i, j) is precomputed (see Section 1.3) and fixed.
2. **home_vars**[i, j] $\in \{\text{True}, \text{False}\}$, for all $i, j \in T$ with $i < j$.
This variable is true if and only if team i plays at home against team j .

3.2 Objective Function

The implemented objective function is defined in Section 1.2.

Since SAT solvers handle only boolean variables, we solve this optimization problem via a *binary search* on the maximum allowed imbalance k .

For each team $t \in T$, the number of home games H_t is defined as

$$H_t = \sum_{\substack{(i,j) \in \text{pair_to_week} \\ p \in P}} \begin{cases} 1 & \text{if } t = i \wedge \text{match_period_vars}_{i,j,p} \wedge \text{home_vars}_{i,j} \\ 1 & \text{if } t = j \wedge \text{match_period_vars}_{i,j,p} \wedge \neg \text{home_vars}_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

To enforce that the home-away imbalance for each team t does not exceed k , we use two pseudo-boolean constraints. For a given k , the constraints are:

1. $H_t \geq \text{lower_bound}$, where $\text{lower_bound} = \lceil \frac{w-k}{2} \rceil$,
2. $H_t \leq \text{upper_bound}$, where $\text{upper_bound} = \lfloor \frac{w+k}{2} \rfloor$.

These bounds ensure that the number of home games for each team respects the maximum imbalance k . Since the maximum number of matches a team can play is $|W|$, which is odd, then the optimal value we want to achieve is exactly $k = 1$. For other details, see section 3.4.

3.3 Constraints

Thanks to the *circle method*, some constraints are inherently satisfied (see Section 1.3). This reduces the constraints needed, leaving the solver to decide only the specific period for each match (i, j) , where $(i, j) \in \mathcal{M}_w$ (\mathcal{M}_w is the set of matches scheduled in week $w \in W$). In particular, we implemented the following constraints:

Each match is assigned to exactly one period:

$$\forall (i, j) \in \text{pair_to_week} : \sum_{p \in P} \text{match_period_vars}_{i,j,p} = 1$$

Each period in each week contains exactly one match:

$$\forall w \in W, \forall p \in P : \sum_{(i,j) \in \mathcal{M}_w} \text{match_period_vars}_{i,j,p} = 1$$

Each team plays at most twice in the same period:

$$\forall t \in T, \forall p \in P : \sum_{\substack{(i,j) \in \text{pair_to_week} \\ t \in \{i,j\}}} \text{match_period_vars}_{i,j,p} \leq 2$$

Symmetry Breaking Constraints

To reduce redundant search caused by symmetric solutions, we experimented with several symmetry breaking (SB) constraints.

SB1: Match between teams 0 and $n - 1$ is in the first period:

$$\text{match_period_vars}_{0,n-1,0} = 1$$

This fixes the match between the pivot team $n - 1$ and team 0 in period 0, breaking rotational symmetry.

We also investigated two further constraints, which are commented in the code and reported here only for completeness. Since they introduced additional complexity that outweighed any potential benefits, they were not considered in the final results.

SB2: Team 0's home/away pattern is fixed: This enforces a fixed home/away assignment for team 0.

SB3: Lexicographical ordering of matches in week 0: This constraint enforces a lexicographical order on the period assignments of matches in week 0.

Encoding Methods

We implemented all the encoding methods covered in class:

1. For the *exactly-one* constraint: Pairwise, Bitwise, Sequential, and Heule encodings.
2. For the *at-most-k* constraint: Pairwise, Sequential, and Totalizer encodings.

Additionally, we employed the **Totalizer encoding** [2] to efficiently model cardinality constraints. This encoding builds a balanced binary tree of adders and is known for scalability and efficiency in SAT formulations. It introduces $O(n \log n)$ auxiliary variables and up to $O(n^2)$ clauses in the worst case.

3.4 Validation

The model was implemented in Python using Z3’s API.

In the following, we present tables and plots to compare the Z3 model using different encoding techniques (used for both optimization and decision versions), both with and without symmetry breaking constraints:

1. Pairwise + Pairwise encodings (p-p in the Table)
2. Heule + Sequential encodings (h-s in the Table)
3. Heule + Totalizer encodings (h-t in the Table)

3.4.1 Experimental Results (Decision version)

As shown in Table 2, even a simple symmetry breaking (SB) constraint can effectively guide the solver, leading to noticeably faster search times.

Table 2: Runtime in seconds for the SAT (decision) version using the Z3 solver, with and without symmetry breaking (SB).

N	p-p + SB	p-p w/out SB	h-s + SB	h-s w/out SB	h-t + SB	h-t w/out SB
12	1.0	1.0	0.0	0.0	0.0	0.0
14	3.0	2.0	0.0	0.0	1.0	0.0
16	4.0	26.0	7.0	3.0	5.0	10.0
18	17.0	80.0	8.0	11.0	7.0	15.0
20	N/A	N/A	79.0	N/A	150.0	151.0

3.4.2 Experimental Results (Optimization version)

As shown in Table 3, the results confirm the expected performance of the encodings. The Heule + Totalizer combination generally proved to be very efficient,

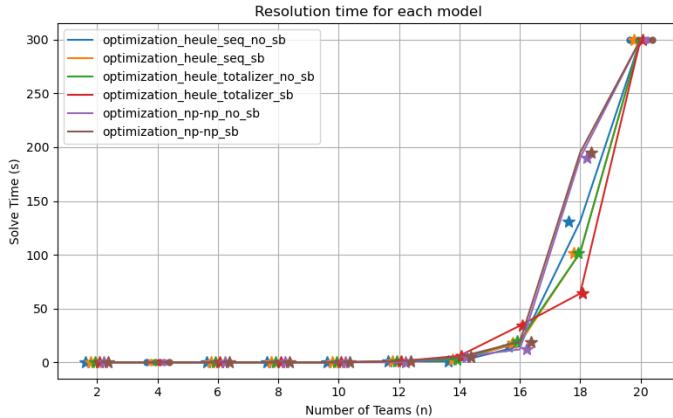
particularly for larger instances. Conversely, the Pairwise + Pairwise approach, known for its larger number of clauses, was the slowest.

Generally, except for some rare cases, the SB constraint proved effective at guiding the solver to an optimal solution faster, significantly reducing the search space.

The table also shows that all tested configurations successfully found an **optimal solution** for $N \leq 18$.

Table 3: SAT results. Results are of the type "time|**optimal value**" or "time|suboptimal value".

N	p-p + SB	p-p w/out SB	h-s + SB	h-s w/out SB	h-t + SB	h-t w/out SB
10	1 1	1 1	0 1	0 1	0 1	0 1
12	2 1	3 1	0 1	1 1	1 1	1 1
14	7 1	7 1	3 1	3 1	2 1	5 1
16	46 1	35 1	37 1	25 1	36 1	34 1
18	207 1	292 1	143 1	160 1	150 1	98 1
20	N/A	N/A	300 5	N/A	300 10	N/A



(a) Resolution time

Figure 2: Runtimes for the optimization version of the Sports Tournament Scheduling (STS) problem. Data points are marked to indicate the solution status: a star (*) indicates that a solution (optimal or sub-optimal) was found, while a circle (●) indicates that no solution was found within the time limit.

4 SMT Model

4.1 Decision Variables

Let n be the number of teams, and let $W = \{1, \dots, n - 1\}$ denote the set of tournament weeks. Each week is divided into $n/2$ periods. Let \mathcal{M}_w be the set of matches scheduled in week $w \in W$.

We define an integer decision variable:

$$p_{w,i,j} \in \{1, \dots, n/2\}$$

for each match between any pair of teams (i, j) , where $(i, j) \in \mathcal{M}_w$, indicating the period during which this match takes place in week w . We avoid additional decision variables by precomputing which teams (i, j) play in each week w using the fast and classical *circle method* for round-robin tournaments [1], which was already explained above in 1.3.

The model is encoded using the theory of Linear Integer Arithmetic (LIA), extended with pseudo-Boolean constraints for cardinality.

4.2 Objective Function

For the sake of solving the home-away balancing task, there is a simple mathematical algorithm. It does not need optimization with SMT. If (i, j) is a game between any two teams i and j then the modulo arithmetic's algorithm is this:

$$i, j \in \{1, \dots, n\}, \quad d \equiv j - i \pmod{n}, \quad 1 \leq d \leq n - 1, \quad \text{home}(i, j) = \begin{cases} i, & d < \frac{n}{2}, \\ j, & d \geq \frac{n}{2}, \end{cases}$$

By construction, it achieves an absolute imbalance of home/away games equal to 1 per each team. In total, the sum of the absolute imbalance values will be n . Theoretically, it is minimal for even n .

We use this mathematical approach in the decisional version, so the json results for the decisional version are also home/away optimal.

But to study the solver overhead of enforcing this balance via optimization, we performed a dedicated experiment. We used the same $\max_t \text{abs_diff}_t$ objective function, as it was found to be the best performing in other parts of our report too.

4.3 Constraints

Domain Constraints. Each decision variable must be assigned a valid period:

$$\forall(w, i, j) : \quad 1 \leq p_{w,i,j} \leq n/2$$

Unique Match per Period per Week. For each week $w \in W$ and period $k \in \{1, \dots, n/2\}$, exactly one match must be assigned to that period:

$$\sum_{(i,j) \in \mathcal{M}_w} [p_{w,i,j} = k] = 1$$

Period Load per Team. Each team must appear at most twice in each period across the tournament:

$$\forall t \in [n], \forall k \in \{1, \dots, n/2\} : \sum_{(w,i,j):t \in \{i,j\}} [p_{w,i,j} = k] \leq 2$$

Symmetry Breaking Constraints

Fixed Periods in First Week. To reduce the search space, we fix the period assignment of all matches in the first week:

$$\forall k \in \{1, \dots, n/2\}, \quad p_{1,i_k,j_k} = k$$

where (i_k, j_k) is the k -th match in a lexicographically sorted list of matches in week 1. This removes the permutation symmetry for the period labels.

In our SMT tests more complicated constraints - like lexicographical constraints on period assignment vectors for chosen pairs of teams - did not bring any noticeable additional improvements, as well as adding redundant constraints.

For optimization version, we added following additional symmetry-breaking constraint (w.r.t to team numbering) on the per-team imbalances:

$$abs_diff_{t_1} \geq abs_diff_{t_2} \geq \dots \geq abs_diff_{t_n},$$

4.4 Validation

Solver. The model was implemented in Python using the Z3 SMT solver. A custom tactic pipeline was used to enable pseudo-Boolean cardinality constraints translation into bit-vector constraints for speed.

Experimental Results (Decision version, SMT in Z3)

We tested the SMT model (without home-away optimization) using Z3 on increasing values of n , with and without symmetry breaking. The results in Table 4 show that enabling symmetry breaking significantly improves performance, especially for larger instances. Since all times were zero for $n < 12$, these cases are not shown in both of the tables here.

Table 4: Runtime in seconds for SMT (decision version), Z3 solver, with and without symmetry breaking (SB).

Number of Teams	SB Enabled	SB Disabled
12	0	1
14	0	0
16	0	2
18	8	17
20	5	42

For our SMT model, we observed that the simple version (first week periods were fixed) of symmetry breaking consistently reduced runtime for bigger n , especially beyond $n = 16$. Without SB, solving $n = 20$ takes 8 times longer.

Though we need to mention that for a fixed n there is still some variation between runs - due to the solver's heuristic choices adding some randomness. In some cases $n = 22$ gets solved on our hardware, while often it times out, so it was not included in the final table. Thanks to circular matching pre-solving, we could always solve those instances with up to including $n = 20$ teams. This is 2 to 4 teams more than our first SMT models that generated all possible matches as decision variables.

Experimental Results (Optimization version, SMT in Z3)

We extended the SMT model to solve the optimization version of the problem, where the objective is to minimize the maximal (w.r.t to all teams) absolute discrepancy in home/away balance. Theoretically, our objective - maximal absolute discrepancy - being equal to 1 is equivalent to reaching the optimum, and indeed this optimum value was reached in all tested instances.

Table 5 reports the total solving time (in seconds) for various team sizes, both with and without symmetry breaking (SB). All runs were performed with Z3 under a 5-minute timeout. In all cases, the solver found and proved optimal solutions within the time limit.

Table 5: Runtime in seconds for SMT (optimization version), Z3 solver, with and without symmetry breaking (SB). The objective function, $\max_t \text{absdiff}_t$, in every case was optimal, and the value is shown in bold.

Number of Teams	SB Enabled	SB Disabled
12	3.00 1	6.00 1
14	4.00 1	11.00 1
16	127.00 1	44.00 1
18	189.00 1	257.00 1
20	149.00 1	154.00 1

We observe that for the SMT optimization version, symmetry breaking has no such dramatic impact for bigger n , though it helped in every case except for $n = 16$. For $n = 12$ and $n = 14$, SB yields a 2 times and 3 times improvement.

Though there is a noticeable variance due to solver heuristic choices, the observations made above are true for the majority of the re-runs, if made.

5 MIP Model

The MIP formulation was developed with Pyomo, a Python library for writing solver-independent models. Two models were developed: a base version composed of a 4d array with symmetry breaking and implied constraints, and a better one, optimizing the number of variables featuring circle matching.

5.1 Variables

Other than the constants defined in 1, the MIP formulation needs two more. Let $wp = (w, p)$ be the week/period slot w, p ; let $m_{i,j} = (i, j)$ be the match played by team i, j with $i < j$:

$$WP = (\{0, \dots, n-2\}, \{0, \dots, \frac{n}{2}-1\}), \quad M = (\{0, \dots, n-1\}, \{0, \dots, n-1\})$$

Decision Variables

To encode the STS problem we define:

1. $\mathbf{Y}_{wp,m} \in \{0, 1\} \quad \forall wp \in WP, \forall m = (i, j) \in M \text{ with } i < j.$
 $\mathbf{Y}_{wp,m} = 1$ if and only if match m is scheduled in slot wp .
2. $\mathbf{H}_{i,j} \in \{0, 1\} \quad \forall (i, j) \in M, i < j.$
 $\mathbf{H}_{i,j} = 1$ if team i plays at home against j , 0 otherwise.

Other Variables

For the efficiency constraints, we also track whether a team plays in a given period:

$$\mathbf{Q}_{i,p} \in \{0, 1\}, \quad \forall i \in T, p \in P,$$

where $\mathbf{Q}_{i,p} = 1$ if team i has at least one match in period p .

5.2 Objective Function

Variables In order to minimize the maximum imbalance, we need 3 extra variables: $\mathbf{Home}_i \in [0, n-1]$, the number of home matches played by team i $\mathbf{Away}_i \in [0, n-1]$, the number of away matches played by team i and $Z \in [0, n-1]$, the maximum imbalance across teams

Constraints

1. **Home games:**

$$\forall i \in T : \quad \mathbf{Home}_i = \sum_{j \in T}^j \mathbf{H}_{i,j} + \sum_{j \in T}^j (1 - \mathbf{H}_{j,i}) \quad \text{with } i < j$$

2. **Away games:**

$$\forall i \in T : \quad \mathbf{Away}_i = \sum_{j \in T}^j (1 - \mathbf{H}_{i,j}) + \sum_{j \in T}^j \mathbf{H}_{j,i} \quad \text{with } i < j$$

3. **Maximum imbalance:**

$$\forall i \in T : \quad |\mathbf{Home}_i - \mathbf{Away}_i| \leq Z.$$

5.3 Constraints

With circle matching we get a matrix:

$$\text{presolved}_{w,i,j} = \begin{cases} 1 & \text{if the match } (i,j) \text{ is scheduled for week } w \\ 0 & \text{o.w.} \end{cases}$$

To take advantage of the precomputed matching schedule, we enforce:

$$\forall w \in W, \forall p \in P, \forall m \in M, m = (i, j) : \mathbf{Y}_{(w,p),m} \leq \text{presolved}_{w,i,j}$$

Circle matching already takes care of some of the problem constraints, but we have to enforce:

1. **One match per w/p slot:**

$$\forall wp \in WP : \sum_{m \in M}^m \mathbf{Y}_{wp,m} = 1$$

2. **One w/p slot per match:**

$$\forall m \in M : \sum_{wp \in WP}^{wp} \mathbf{Y}_{wp,m} = 1$$

3. **At most 2 matches in the same period:**

$$\forall p \in P, \forall k \in T : \sum_{w \in W}^w \sum_{j \in \{k+1 \dots n\}}^j \mathbf{Y}_{(w,p),(k,j)} + \sum_{w \in W}^w \sum_{i \in \{0 \dots k\}}^i \mathbf{Y}_{(w,p),(i,k)} \leq 2$$

Constraints for efficiency

Especially useful in an optimization environment, this constraint aims to spread the matches of a team in different periods over the scheduling, to get a more balanced result. It also help with symmetry breaking.

$$\begin{aligned} \forall p \in P, \forall i \in T : 2 \cdot \mathbf{Q}_{i,p} &\geq \sum_{w \in W}^w \sum_{j \in \{i+1 \dots n\}}^j \mathbf{Y}_{(w,p),(i,j)} + \sum_{w \in W}^w \sum_{j \in \{0 \dots i\}}^j \mathbf{Y}_{(w,p),(j,i)} \\ \forall i \in T : \sum_{p \in P}^p \mathbf{Q}_{i,p} &\geq \text{ceil}((n - 1)/2) \end{aligned}$$

Symmetry breaking constraints

In this particular model the addition of symmetry breaking constraints did not improve the results, as circle matching already breaks most of the symmetries, and the addition of other constraints only increased the model weight.

5.4 4D array model

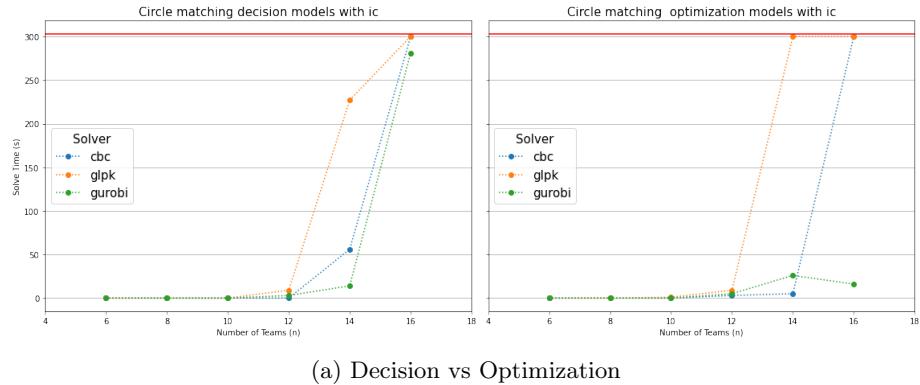
The simplest possible implementation of the STS problem, developed only for comparison as it is not nearly as efficient as the previous one. The only decision variable is X , a 4d array ($n - 1 \times n/2 \times n \times n$) of binary values, where each cell represents a match, described as a combination of week, period, team1 and team2. The first team is the one playing home. The constraints, apart from the necessary ones, include two symmetry-breaking constraints and two implied constraints. The optimization is performed in the same way as in the previous model.

5.5 Validation

The model was implemented in Python using Pyomo 6.9.2. The results were obtained by running the models for 3 different solvers: **Glpk**, works well for small instances, but does not scale well; **Cbc**, the most advanced open-source solver; **Gurobi**: a very efficient commercial solver, used under an academic license.

5.5.1 Experimental Results

We tested the MIP formulations on different configurations, using all solvers and both model variants (4D array and Circle Matching, CM).



(a) Decision vs Optimization

Decision version Somewhat counterintuitively, the decision version required on average *more time* to solve (see Figure 3a). This behavior can be explained by the branch-and-bound exploration of MIP solvers: once the objective is removed, the search tends to focus on constraint satisfaction, which often leads to a larger feasible space and slower convergence. Moreover, efficiency-oriented constraints, designed to balance the schedule, turned out to be detrimental in this setting.

n	Model	Solver	Time	n	Model	Solver	Time
8	4D	Glpk	1s	12	CM	Gurobi	2s
10	4D	Glpk	N/A	14	4D	Gurobi	24s
10	4D	Cbc	4s	14	CM	Glpk	N/A
12	4D	Cbc	N/A	14	CM	Cbc	27s
12	4D	Gurobi	23s	14	CM	Gurobi	36s
12	CM	Glpk	17s	16	CM	Cbc	N/A
12	CM	Cbc	14s	16	CM	Gurobi	72s

Table 6: Results for the decision problem
Efficiency constraints were not enabled here

Optimization version As expected, the CM model consistently outperformed the 4D array formulation in both solution quality and runtime. This highlights the benefit of a stricter model, with a smaller associated search space. Table 7 shows how solver performance followed the usual hierarchy: **Gurobi** was the most efficient, followed closely by **Cbc**, while **Glpk** struggled with scalability and often reached the time limit for larger instances. Interestingly, though, for some smaller instances Glpk performs better than Cbc.

n	Model	Solver	Time Obj	n	Model	Solver	Time Obj
10	4D	Cbc	298 1	12	CM	Gurobi	5 1
10	4D	Glpk	35 1	14	CM	Cbc	5 1
10	4D	Gurobi	5 1	14	CM	Glpk	N/A
12	4D	Cbc/Glpk	N/A	14	CM	Gurobi	26 1
12	CM	Cbc	4 1	16	CM	Cbc	N/A
12	CM	Glpk	10 1	16	CM	Gurobi	16 1

Table 7: Results for the optimization problem
Efficiency constraints were enabled here

6 Conclusion

Across CP, SAT, SMT, and MIP, we used the same circular matching method to fix weekly pairings and focus the search on periods and home/away. This allowed us to scale up to including $n = 20$ in the most effective approaches. Also, significant gains came from symmetry breaking: fixing rotations, team labels, and a simple home/away pattern prevented exploration of equivalent schedules. Quantitatively, it enabled us to solve $n = 16$ to optimality in CP, and for some cases gave 4-7x speedups in SAT, around 2-3x in SMT, while in MIP the circular formulation already removed most symmetries. In short, circular matching plus symmetry breaking plus a max imbalance objective minimization is a clear, reliable recipe that performed consistently across all reported approaches.

Authenticity and Author Contribution Statement

We declare that this work is our own and not copied from anyone else. The work has been roughly split in the following way: Cotič did the CP part, Centanni worked on SAT, Uskov did the SMT part and Lodi completed the MIP part.

References

- [1] Dominique de Werra. Scheduling Round-Robin Tournaments: An Overview. *Discrete Applied Mathematics*, 91(1–3):241–277, 1999.
- [2] Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In: *Principles and Practice of Constraint Programming (CP 2003)*, Lecture Notes in Computer Science, 2003.