# Combining a Decomposition Scheme with Metaheuristics for Round-Robin Tournament Scheduling

**Jan-Erik Hein**

A thesis presented for the degree of
Bachelor of Science

Combinatorial Optimization and Graph Algorithms
Technische Universität Berlin
December 21, 2021

## Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, am ........................ ............................................

(Unterschrift)

# Zusammenfassung

Die Spielplanerstellung für Sportwettbewerbe stellt oftmals ein schweres kombinatorisches Problem für die Verantstalter dar. Vor allem Spielpläne für professionelle Wettbewerbe, welche eine hohe ökonomische Bedeutung haben, unterliegen einer großen Anzahl an Regulierungen, um die Fairness und Attraktivität des Wettbewerbes für alle Interessengruppen zu gewährleisten. Aufgrund der hohen Komplexität sind generelle Lösungsverfahren meist nicht geeignet um gute Spielpläne zu generieren. Stattdessen werden oft problemspezifische heuristische Ansätze benötigt.

In dieser Arbeit betrachten wir speziell das Problem der Spielplanerstellung für Rundenturniere bei denen jedes Team einmal zuhause und einmal auswärts gegen jedes andere Team spielt. Um ein möglichst universelles Lösungsverfahren zu formulieren betrachten wir eine Reihe von abstrakten Nebenbedingungen, welche eine Modellierung von vielen Sportwettbewerben erlauben. Hierbei nutzen wir die Problemstellung der *21. International Timetabling Competition (ITC2021).* Unser Ansatz basiert auf einer in der Literatur verbreiteten Zerlegung des Problemes in Teilprobleme, welche wir mittels Integer Programmierung und lokaler Suche lösen.

Wir testen unseren Ansatz auf allen Instanzen die für den Wettbewerb bereit gestellt wurden und vergleichen unsere Ergebnisse mit den besten Ergebnissen die während des Wettbewerbes gefunden wurden. Als zusätzlichen Vergleich betrachten wir auch eine Implementierung des Problems als Integer Programm.

Für den Großteil der Instanzen ist bei der Generierung von zulässigen Startlösungen die Laufzeit unseres Ansatzes vergleichbar mit der des Integer Programmes. Für manche Instanzen gibt es jedoch drastische Unterschiede, welche wir auf die Struktur der jeweiligen Instanzen zurückführen. Bei der Verbesserung der Startlösungen stellt sich unser heuristischer Ansatz als überlegen heraus, sodass wir, verglichen mit dem Integer Programm, wesentlich bessere Lösungen finden. Jedoch sind unsere Lösungen nur in Einzelfällen vergleichbar mit den besten Lösungen die im Wettbewerb gefunden wurden. Es ist unklar ob dies an unserem Ansatz liegt oder der begrenzten Rechenzeit, welche wir für unsere Experimente genutzt haben, liegt.

# Abstract

Scheduling a real world sport tournament is a hard combinatorial problem, which usually involves a large number of constraints imposed by a multitude of stakeholders. Due to a large variety of constraints that occur in sport tournaments, problem-specific approaches are often needed to find suitable schedules. The goal of our work is to advance the research on general purpose solvers, which can be easily applied to a wide range of sport tournaments. Specifically, we consider the problem of scheduling round-robin tournaments under a large variety of hard and soft constraints. We propose a method that incorporates a tabu search heuristic in a known decomposition scheme and compare it with an integer programming approach. Our proposed method performs similar to the integer program for finding feasible start solutions but performs significantly better at finding optimal solutions, allowing us to generate good solutions for a series of hard instances, where the integer programming approach fails to find significant improvements.

# Contents

# Chapter 1

# Introduction

Sport tournaments attract a lot of international attention and have a major economic relevance. Organizers of sport tournaments often have to balance the conflicting interests of a diverse group of stakeholders, including teams, broadcasters, fans, security, airlines and more, while still ensuring a fair competition for the participating teams. This leads to challenging optimization tasks, such as revenue maximization and logistic optimization.

An important and often highly constrained aspect of a sport tournament is the schedule of the tournament. The structure of the schedule not only has direct revenue implications for stakeholders but also significantly affects the performance of the teams. Broadcasters for example want to maximize their revenue and match high profile games with good broadcasting slots, while teams often want to minimize traveling distances or avoid consecutive high profile games. Additionally, schedules often need to conform to a multitude of other regulations, due to infrastructure limitations or security concerns. In some cases finding any schedule that follows all imposed regulations, even without taking the optimization aspect into consideration, is a difficult task in itself.

The difficult nature and economic relevance of tournament scheduling problems led to a multidisciplinary research effort in the past 40 years. Applications to real world problems, such as the scheduling of football, basketball, baseball and cricket tournaments are common in the literature. These applications usually involve a large number of conflicting requirements and require specialized solution methods. To that end, different optimization techniques were applied, including integer and constraint programming, metaheuristic approaches and decomposition schemes. While specialized methods become more and more effective they are often based on problem specific constraints and objective functions, making it difficult to apply them to other tournaments. This motivates the research into general purpose methods, that can easily be adapted to a wide range of real world sport tournaments.

In this thesis we will consider the problem of scheduling *round-robin tournaments* where each pair of teams plays a fixed number of games. The aim of our research is to extend and combine common approaches from the literature to formulate a general purpose approach, that could be applied to various real world tournaments. To that end we consider generic constraint types that can be either *hard* or *soft*, where hard constraints describe necessary properties of the schedule and soft constraints are used to model the conflicting interests of stakeholders. Our proposed approach combines a known decomposition scheme with a local search heuristic. For comparison we will also consider a straight-forward integer programming formulation of the problem.

The rest of the thesis is organized as follows. Section 1.1 will introduce the sport scheduling terminology used throughout this thesis. Section 2 defines the tournament structure, constraint types and objective function of the problem instances that we consider. In section 3 we will review common approaches from the literature and discuss their limitations. In section 4 we analyze how these approaches can be applied to our problem and explain our proposed approach. In section 5 we present and discuss our computational results and finally, the last section summarizes our conclusions.

## 1.1 Terminology of Sport Scheduling

In this section we introduce the sport scheduling terminology used in the following chapters. A *k-round-robin tournament* ($k$RR) is a tournament in which each pair of teams meet exactly $k$ times. In this thesis we consider *double round-robin* (2RR) and *single round-robin tournaments* (1RR) as they are the most common tournament formats in real sport leagues, however other formats including *triple* or *quadruple round-robin tournaments* do also occur [1]. Sport leagues that play a 2RR often utilize a *phased* tournament structure, where the tournament is split into two parts and a 1RR is played in each part.

When scheduling a tournament, a *game allocation* assigns all games to *time slots* in such a way that no team plays twice during a single slot. We refer to the set of games that is assigned to a specific slot as a *round*. A tournament is called *time-constrained* if it uses the minimal number of rounds needed and *time-relaxed* otherwise. Note that when the number of teams $n$ is even, a timetable for a time-constrained $k$-round-robin tournament has exactly $kn - k$ rounds, where each team must play exactly once in each round.

Teams often have a *home venue* associated with them, where playing a game at their own venue is referred to as playing *at home*, while playing at a different venue is referred to as as playing *away*. In real sport tournaments that play a 2RR it is common to require each pair of teams to meet once at each of their respective home venues, resulting in each team having the same number of home and away games over the course of the tournament.

A *break* occurs if a team has the same home/away status in consecutive games. We differentiate between *home* and *away breaks* respectively. If a team does not play in a given slot it is said to have a *bye*. The sequence of byes, home and away games a team $t$ has over the course of the tournament is referred to as the *home/away pattern* of $t$. Home/away patterns can be combined into a *pattern set*, where a pattern is assigned to each team. The pattern set is said to be *feasible* if a game allocation exists, that assigns each game to a slot in which one of the corresponding teams plays at home and the other plays away.

A feasible pattern set together with a corresponding game allocation form the *schedule* or *timetable* of the tournament. In this thesis we will represent a schedule by a table indicating the opponents and patterns of the teams, where each row corresponds to a team and each column corresponds to a time slot. The opponent of team $t_i$ in the $j$-th round is represented by the absolute value of the element $(i, j)$ and the home/away status is represented by the sign of the the element. A value greater than zero indicates the

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | -6 | 2 | -5 | 4 | -3 | 3 | -4 | 5 | -2 | 6 |
| **2** | -3 | -1 | 4 | -6 | 5 | -5 | 6 | -4 | 1 | 3 |
| **3** | 2 | -4 | 6 | -5 | 1 | -1 | 5 | -6 | 4 | -2 |
| **4** | -5 | 3 | -2 | -1 | 6 | -6 | 1 | 2 | -3 | 5 |
| **5** | 4 | -6 | 1 | 3 | -2 | 2 | -3 | -1 | 6 | -4 |
| **6** | 1 | 5 | -3 | 2 | -4 | 4 | -2 | 3 | -5 | -1 |

Figure 1.1: Example schedule for a time-constrained phased 2RR with 6 teams, with breaks being highlighted for each team. Team $t_2$ has away and home breaks in the second and tenth time slot respectively.

team playing at home and a value less than zero indicates the team playing away. Fig. 1.1 provides an example for a timetable of a time-constrained double round-robin tournament, where each pair of teams meet once at each of their respective home venues.

# Chapter 2

# Problem Description

In this section we describe the problem statement that we consider in this thesis. The problem considered is based on the problem statement of the *21.International Timetabling Competition (ITC2021)* [2]. All instances presented in the competition were expressed using the standardized RobinX XML data format developed by Van Bulck et al. [3]. The data format uses a 3-field notation describing the tournament format, the constraints in use and the objective function.

## 2.1 Tournament Structure

We consider the problem of scheduling a time-constrained double round-robin tournament for a set of teams $\mathcal{T} = \{t_1, \dots, t_n\}$ and a set of time slots $\mathcal{S} = \{s_1, \dots, s_m\}$. We consider the case that the number of teams $n$ is even, where the number of slots $m$ is given by $2n - 2$ and each team has to participate exactly once in each round. Note, that requiring $n$ to be even is not restrictive, as time-constrained round-robin tournaments with an odd number of teams can be modeled by adding a "dummy" team. Then, we solve the model for an even number of teams and reinterpret a game between a team $i$ and the dummy team as a bye of $i$ in the respective time slot.

We assume each team to have a corresponding home venue and each pair of teams to play one game at each of their respective venues. Furthermore, we consider tournaments that require a phased tournament structure as well as tournaments without any additional structural requirements.

## 2.2 Constraints

In the competition two kinds of constraints were given, *hard* constraints that represent essential properties of the schedule that must always be satisfied and *soft* constraints which infer a penalty if violated and should be satisfied whenever possible. As an example, a hard constraint could enforce venue restrictions, e.g. requiring a team to play away in a specific slot if its home venue is used for another event at that time. A soft constraint on the other hand could be used to limit the total number of breaks, as schedules with a high number of breaks are often undesirable.

The set of constraints will be denoted by $C$ and the sets of hard and soft constraints will be denoted by $C_h$ and $C_s$ respectively. For each constraint $c \in C$ we define a *deviation* $\Delta_c(x)$, where $\Delta_c(x) = 0$ if $c$ is satisfied in a solution $x$ and $\Delta_c(x) > 0$ if $c$ is violated. For simplicity of notation we abbreviate $\Delta_c(x)$ to $\Delta_c$ if the solution $x$ is implicitly given.

Throughout this thesis we will use a partition of the set of constraints into *pattern*, *pattern set* and *general constraints*. Pattern constraints constrain only the pattern of a single team, whereas pattern set constraints can restrict patterns of multiple teams.

However, pattern and pattern set constraints only constrain the home/away statuses of teams, but not the game allocation. General constraints on the other hand can restrict both the pattern set and the game allocation of the schedule. As an example, a constraint limiting the number of home games a given team plays in the first three slots would be seen as a pattern constraint. However, if the constraint would only limit home games played against a subset of teams, it would be seen as a general constraint instead. Furthermore, if a constraint only needs to hold for a subset of teams, we say it is *team-specific.*

The RobinX format used in the competition defines 5 different constraint types, each with their own subtypes, which can model an exhaustive range of constraints occurring in real world tournaments. For a more detailed description of the constraint types, the different subtypes and their applications we refer to Van Bulck et al. [3]. In the competition simplified versions of these constraint types were given, which can be defined as follows.

**Capacity Constraints** Capacity constraints (CA) impose home/away restrictions on teams and regulate the total number of games played between two groups of teams in a given set of time slots. Formally, a capacity constraint is characterized by a 5-tuple

$$\mathrm{CA}(T_1, T_2, S, ub, md), \quad T_1, T_2 \subseteq \mathcal{T},\ S \subseteq \mathcal{S}, ub \in \mathbb{N},\ md \in \{\mathrm{H}, \mathrm{A}, \mathrm{HA}\},$$

enforcing an upper bound $ub$ on the number $N$ of games ($md$ = HA), home games ($md$ = H), or away games ($md$ = A), teams in $T_1$ play collectively against teams in $T_2$ during time slots in $S$. The deviation of a capacity constraint $c$ is then defined as

$$\Delta_c = \max\{0, N - ub\}.$$

A capacity constraint is a general constraint if $T_2 \neq \mathcal{T}$. However, in the special case, that $T_2 = \mathcal{T}$ the constraint can be seen as a pattern constraint for $|T_1| = 1$ and as a pattern set constraint for $|T_1| > 1$. We consider hard and soft capacity constraints.

**Game Constraints** Game constraints (GA) enforce or forbid specific assignments of games to time slots and are characterized by a 4-tuple

$$\mathrm{GA}(G, S, lb, ub), \quad G \subseteq \{(i, j) : i, j \in \mathcal{T}, i \neq j\}, S \subseteq \mathcal{S}, lb, ub \in \mathbb{N},$$

enforcing an upper bound $ub$ and a lower bound $lb$ on the number of games $N$ from $G$ taking place during time slots in $S$. The deviation of a game constraint $c$ is then defined as

$$\Delta_c = \max\bigl\{\max\{lb - N, 0\}, \max\{N - ub, 0\}\bigr\}.$$

Although the definition of a game constraint also allows modeling other constraints, including pattern and pattern set constraining capacity constraints, we always consider them as general constraints throughout this thesis. We consider hard and soft game constraints.

**Break Constraints** Break constraints (BR) regulate the frequency and timing of breaks in the competition and are defined by a 4-tuple

$$\mathrm{BR}(T, S, ub, md), \quad T \subseteq \mathcal{T},\ S \subseteq \mathcal{S},\ ub \in \mathbb{N},\ md \in \{\mathrm{H}, \mathrm{A}, \mathrm{HA}\},$$

setting an upper bound $ub$ on the total number $N$ of breaks ($md$ = HA), home breaks ($md$ = H), or away breaks ($md$ = A), teams in $T$ collectively have during time slots in $S$. The deviation of a break constraint $c$ is given by

$$\Delta_c = \max\{N - ub, 0\}.$$

A break constraint is pattern constraining if $|T| = 1$ and pattern set constraining otherwise. We consider hard and soft break constraints.

**Fairness Constraints** Similar to capacity constraints, fairness constraints (FA) impose home/away restrictions for a given set of teams. However, instead of limiting the total number of games played, they limit the difference in home games played between pairs of teams. Formally, a fairness constraint is given by a 3-tuple

$$\mathrm{FA}(T, S, ub), \quad T \subseteq \mathcal{T},\ S \subseteq \mathcal{S}, ub \in \mathbb{N},$$

requiring each pair of teams in $T$ to have a difference in home games played of at most $ub$ after each time slot in $S$. Let $\delta_{ijs}$ be the absolute difference in home games played by teams $i$ and $j$ after slot $s$. We then define the deviation of a fairness constraint $c$ as

$$\Delta_c = \sum_{i,j \in \mathcal{T}} \max\bigl\{\max\{\delta_{ijs} : s \in S\} - ub, 0\bigr\}.$$

A fairness constraint is always a pattern set constraint and assumed to be soft.

**Separation Constraints** Separation constraints regulate the number of rounds between the two meetings of a pair of teams and are defined by

$$\mathrm{SE}(T, lb), \quad T \subseteq \mathcal{T},\ lb \in \mathbb{N},$$

requiring each two teams in $T$ to have their mutual games separated by at least $lb$ time slots. Let $\alpha_{ij}^{(1)}$ be the first slot and $\alpha_{ij}^{(2)}$ be the second slot in which teams $i$ and $j$ meet. The deviation of a separation constraint $c$ is then defined as

$$\Delta_c = \sum_{i,j \in \mathcal{T}, i<j} \max\{\alpha_{ij}^{(1)} - \alpha_{ij}^{(2)} + lb + 1, 0\}.$$

A separation constraint is always a general constraint and assumed to be soft.

## 2.3 Objective Function

In the competition it was guaranteed that a feasible solution satisfying all hard constraints exists. However, finding a solution that also satisfies all soft constraints is usually impossible due to contradicting constraints.

For each soft constraint $c \in C_s$ we have an associated penalty $p_c > 0$. Then, the objective is to find a schedule that satisfies all hard constraints and minimizes the weighted deviations of all soft constraints. More precisely, the objective can be formulated as

$$\underset{x}{\text{minimize}} \sum_{c \in C_s} p_c \, \Delta_c(x), \quad \text{s.t.} \sum_{c \in C_h} \Delta_c(x) = 0.$$

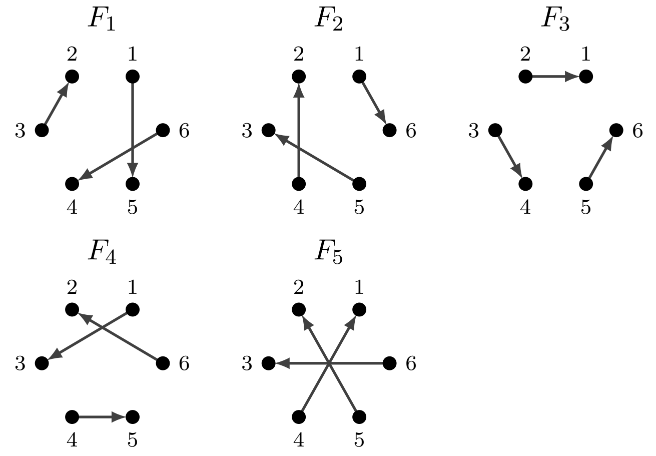
# Chapter 3

# Related Work

In this section we will give a brief overview on the history of tournament scheduling research as well as introduce the work that motivated our approach. For a more in-depth review of past contributions in the field we refer to a survey by Rasmussen and Trick [1].

The structure of round-robin tournaments has been extensively studied, with early research focusing on the close relationship to *1-factorizations* of complete graphs. Given an undirected graph $G = (V, E)$, a 1-factorization of $G$ is a partition of $E$ into pairwise disjoint *1-factors*, where a 1-factor is a subset $F$ of $E$, such that each vertex in $V$ is incident to exactly one edge in $F$. De Werra [4] describes the relationship between 1-factorizations and tournaments in the following way. Let $(F_1, \ldots, F_{2n-1})$ be a 1-factorization of $K_{2n}$, the complete graph on $2n$ vertices. This factorization can then be associated with a time constrained single round-robin tournament for $2n$ teams, by letting each node correspond to a team and letting each edge in $F_s$ correspond to a game played during the time slot $s$. De Werra also notes that tournaments that differentiate between home and away games can be associated with *oriented 1-factorizations*, where 1-factors consisting of directed edges are considered. Fig. 3.1 shows an example for an oriented 1-factorization with a corresponding timetable.

Different constructive methods were developed for generating 1-factorizations and corresponding round-robin tournament schedules. In the above mentioned paper De Werra defines a method for obtaining a 1-factorization of $K_n$, called the *canonical 1-factorization*, where the corresponding schedule has $n-2$ breaks. He observes that the number of breaks in the corresponding *canonical schedule* is minimal, since at most two teams can have a pattern with no breaks. Furthermore, he proposes a method on how a canonical schedule



| **T\S** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| **1** | 5 | 6 | -2 | 3 | -4 |
| **2** | -3 | -4 | 1 | -6 | -5 |
| **3** | 2 | -5 | 4 | -1 | -6 |
| **4** | -6 | 2 | -3 | 5 | 1 |
| **5** | -1 | 3 | 6 | -4 | 2 |
| **6** | 4 | -1 | -5 | 2 | 3 |

Figure 3.1: Oriented 1-factorization of $K_6$ and the corresponding timetable for 1RR

can be modified to satisfy place constraints, where the home/away status of some teams is fixed in some time slots.

Another approach is to form a schedule slot-by-slot. However, Rosa and Wallis [5] show that such a greedy approach might fail. They prove the existence of *premature sets*, which are sets of pairwise disjoint 1-factors that can not be extended into a 1-factorization. Thus, they conclude that forming the schedule of a round-robin tournament slot-by-slot can lead to the corresponding one-factors forming a premature set, resulting in some games not being able to be scheduled.

In the 1990s the research focus moved from graph theoretical results on 1-factorizations towards practical applications of scheduling sport leagues, where a wider range of constraints had to be considered. To accommodate the various constraints that occur in real sport tournaments, integer and constraint programming [6, 7] as well as metaheuristic approaches [8–10] were considered. When a tournament involves home/away restrictions, a common approach in the literature is the use of hybrid methods [11,12], where the problem is split into subproblems, which can then each be solved separately using problem-specific approaches. In the following sections we review these approaches and discuss their suitability for our problem.

## 3.1 Integer and Constraint Programming Approaches

Integer and constraint programming were among the first methods studied to improve on the previous constructive methods for generating 1-factorizations. Trick [6] and Henz, Müller and Thiel [7] examine integer and constraint programming approaches for real sport scheduling problems. Based on the earlier graph theoretical research they identify two major types of constraints needed to classify round-robin tournaments: *one-factor* and *all-different* constraints. One-factor constraints ensure that each team participates exactly once in each slot, whereas all-different constraints enforce the opponents of each team $i$ over all slots to be exactly the set of teams excluding $i$.

In the above mentioned paper, Trick formulates the following basic integer program for a time-constrained 1RR, using binary variables $x_{ijs}$ which indicate whether teams $i$ and $j$ play each other in slot $s$.

$$\begin{aligned}
x_{iis} &= 0, && i \in \mathcal{T},\ s \in \mathcal{S} && \text{(1a)}\\
x_{ijs} &= x_{jis}, && i, j \in \mathcal{T}, s \in \mathcal{S} && \text{(1b)}\\
\sum_{j \in \mathcal{T}} x_{ijs} &= 1, && i \in \mathcal{T},\ s \in \mathcal{S} && \text{(1c)}\\
\sum_{s \in \mathcal{S}} x_{ijs} &= 1, && i, j \in \mathcal{T},\ i \neq j && \text{(1d)}\\
x_{ijs} &\in \{0, 1\}, && i, j \in \mathcal{T},\ s \in \mathcal{S}
\end{aligned}$$

Constraints (1a) and (1b) ensure that no team plays itself and force the symmetry of $x$. The one-factor constraints and all-different constraints are defined by (1c) and (1d) respectively. Trick also notes, that this formulation can easily be adapted to model 2RR with home/away restrictions, by reinterpreting $x_{ijs}$ to indicate that team $i$ plays at home against team $j$ in slot $s$. Removing the symmetry constraint (1b) and replacing the one-factor constraints (1c) by

$$\sum_{j\in\mathcal{T}} x_{ijs} + x_{jis} = 1, \qquad i \in \mathcal{T},\ s \in \mathcal{S},$$
$$\sum_{i,j\in\mathcal{T}} x_{ijs} = \frac{n}{2}, \qquad s \in \mathcal{S},$$

we thus get an integer program for 2RR. We will refer to this model as the *base IP-formulation.*

Henz et al. [7] consider scheduling round-robin tournaments in a constraint programming context. They show that using propagation methods by Régin [13] for the all-different and one-factor constraint is crucial for tournaments without home/away restrictions, leading to a search tree reduction of up to two orders of magnitude. However, they observe a much smaller reduction for tournaments with home/away restrictions, making the additional computational effort of the propagation methods not justifiable in those cases.

Trick [6] finds that for highly constrained pattern sets, decomposition approaches outperform both the integer programming and the constraint programming approach. On the other hand the IP approach is often faster compared with a decomposition approach for tournaments with only few pattern and pattern set constraints. This is due to many decomposition schemes utilizing an expanded search space in which at first feasible patterns are combined into pattern sets, which leads to a very large search space if only few restrictions on patterns and pattern sets are given.

## 3.2 Hybrid Methods and Decomposition Schemes

Scheduling a round-robin tournament with home/away restrictions naturally decomposes into 4 subproblems: generating feasible patterns, combining these patterns into pattern sets, assigning teams to patterns and finding a game allocation. Most decomposition approaches in the literature adopt these steps, with the main differences being the order in which the subproblems are solved. In the so-called *schedule-then-break* approach the algorithm first finds a suitable game allocation and then assigns home/away statuses to the teams. The *break-then-schedule* approach on the other hand reverses the order of these steps, where the algorithm first computes a pattern set and then finds a corresponding game allocation.

**Schedule-then-break** Trick [11] proposes a schedule-then-break approach for scheduling 1RR with a highly constrained game allocation. Specifically, he considers tournaments where some games are pre-assigned to specific time slots as well as the problem of minimizing *carry-over* effects. If a team $A$ plays consecutive games against teams $B$ and $C$, team $C$ gets a carry-over effect from $B$. If $B$ is a very strong team $C$ might gain an advantage in the game against $A$. The motivation for utilizing a schedule-then-break approach in that case stems from the observation that a good decomposition structure allows the most critical constraints to be added at an early stage of the decomposition. If most constraints restrict the game allocation of the schedule, adding them in the first phase of the decomposition can thus lead to a significantly smaller search space that needs to be considered.

The approach proposed by Trick consists of two phases. In the first phase an optimal game allocation is computed, ignoring all constraints on home/away assignments. In the second phase a constraint programming formulation is used to assign the home/away roles. Trick finds that this approach proves very effective for 1RR with up to 20 teams. However he notes that solving 2RR is computationally much harder if each pair of teams is required to play once at each of their home venues. In that case the pattern set is much more restricted and finding a pattern set that matches the optimal game allocation found in the first phase might not be possible. Furthermore it is unclear how more complicated pattern and pattern set constraints can be incorporated in such an approach.

**Break-then-schedule** The most common decomposition approach in the literature is the break-then-schedule approach, where at first a suitable pattern set is found. One essential aspect of this approach is characterizing feasible pattern sets, which are pattern sets that allow all games to be scheduled. Miyashiro et al. [14] present the following necessary condition for feasible pattern sets of double round-robin tournaments.

**Theorem 1** (Pattern Diversity Condition)**.** *Let $\mathcal{T}$ be the set of teams and $\mathcal{S}$ be the set of time slots in a time-constrained 2RR with an even number of teams. For a given pattern set let $H(T, s)$ denote the number of home games and $A(T, s)$ denote the number of away games teams in $T \subseteq \mathcal{T}$ play collectively during time slot $s \in \mathcal{S}$. Then the pattern set is feasible only if*

*(i) no two teams have the same pattern assigned,*

*(ii) $H(\mathcal{T}, s) = A(\mathcal{T}, s)$ for all $s \in \mathcal{S}$*

*and for all $T \subseteq \mathcal{T}$ it holds that*

$$\sum_{s\in\mathcal{S}} \min\{A(T, s), H(T, s)\} \geq |T|(|T| - 1).$$

While (i) and (ii) describe trivial properties of a feasible pattern set, the reasoning behind the main condition is that any subset of teams $T$ must play $|T|(|T| - 1)$ mutual games in a 2RR and that the number of mutual games that can be scheduled during slot $s$ is bounded above both by $H(T, s)$ and $A(T, s)$. In the above mentioned paper, Miyashiro et al. also present the following analogous condition for 1RR:

$$\sum_{s\in\mathcal{S}} \min\{A(T, s), H(T, s)\} \geq \frac{|T|(|T| - 1)}{2}, \qquad \text{for all } T \subseteq \mathcal{T}.$$

Furthermore, Miyashiro et al. show that the pattern diversity condition is a sufficient criterion for the feasibility of pattern sets with a minimal number of breaks. However finding sufficient criteria for general pattern sets is still an open research question.

Rasmussen and Trick [12] propose a break-then-schedule approach, called the *Pattern Generating Benders Approach (PGBA)*, that incorporates the pattern diversity condition as part of a *logical Benders decomposition.* The concept of a logical Benders decomposition was first introduced by Hooker and Ottosson [15] and extends the general cut generation strategy of a traditional Benders decomposition. Instead of solving a linear subproblem, a logical Benders decomposition uses so-called *inference duals* to derive Benders cuts. In case of a feasibility problem, an inference dual is simply a sufficient condition for infeasibility of a given solution. Hooker and Ottoson note, that the main advantage of a logical Benders decomposition is that it allows for generating cuts through problem-specific methods of logical inference.

In the PGBA a 4-phase approach is employed, that closely follows the typical four decomposition steps. However, instead of solving the subproblems sequentially, the PGBA iterates between all four steps. The approach starts by using a CP model to generate patterns with a limited number of breaks. Then an IP model combines these patterns into pattern sets for placeholder teams. Once a candidate solution for a pattern set is found, teams are assigned to placeholders and different inference duals are checked. If the candidate solution is proven infeasible, a Benders cut is added to cut of the candidate and similar infeasible solutions. Otherwise another IP is used to find a corresponding game allocation. If no game allocation exists, the pattern set is infeasible and a cut is added to the model. Finally, if a game allocation was found, the resulting schedule is checked for optimality. If the solution is not proven to be optimal, additional patterns are generated.

Rasmussen [16] also applies a modified version of PGBA to scheduling a triple round-robin tournament for the Danish football league. Notably, in order to allow for team-specific constraints in the competition, Rasmussen combines the team assignment and pattern generation steps by initially finding feasible patterns for each team. We will adopt this idea in our approach.

We note that, to our knowledge, no work has been done on how hard general constraints can effectively be added in a decomposition approach. Since general constraints can constrain the pattern set and the game allocation at the same time, adding them in a decomposition approach poses a difficult challenge. In a break-then-schedule approach general constraints must be added at the game allocation step of the decomposition. However, hard general constraints are likely to cause the game allocation subproblem to be infeasible, due to the pattern set being fixed at that point. A similar issue arises in a schedule-then-break approach, where general constraints must be added when the pattern set is computed and the game allocation is fixed.

## 3.3 Metaheuristic Solution Methods

For many practical applications finding optimal solutions is not viable due to the NP-hardness of the underlying combinatorial problem. However, finding good solutions in a reasonable amount of time is often more important then finding optimal solutions. In those cases, heuristic approaches can significantly reduce the runtime and still be able to find near optimal solutions.

Although the characteristics of a good heuristic are often highly problem specific, there are common strategies that can be adapted to various combinatorial problems. Gendreau and Potvin [17] divide these so-called *metaheuristics* into two categories: *population-based* and *trajectory-based* heuristics. A population-based heuristic incrementally improves an initial population of solutions through search iterations. In each iteration the algorithm substitutes part of its population with newly generated best solutions, which are usually obtained by identifying and combining desirable features of the current population. A trajectory-based heuristic on the other hand starts with a single solution and replaces it with a new one at each iteration, often making use of problem specific neighborhood structures.

Although various metaheuristics were applied to tournament scheduling problems, including population-based methods like genetic programming [18], most approaches in the literature utilize trajectory-based *local search* methods. The basic idea of local search is to start with a heuristically generated solution and to iteratively improve this solution by applying small modifications. Most local search methods also involve randomization techniques to prevent the search from stagnating. Hoos and Tsang [19] formally define the components of such a *stochastic local search (SLS)* algorithm as follows.

**Definition 3.3.1.** *Given a combinatorial problem* $\Pi$*, a* stochastic local search algorithm *for solving an arbitrary problem instance* $\pi \in \Pi$ *is defined by the following components.*

*(i) the* search space $S(\pi)$ *of instance* $\pi$*, which is a finite set of candidate solutions — for tournament scheduling problems, the search space is typically given by all schedules satisfying the one-factor and all-different constraints.*

*(ii)* *a* set of feasible solutions $S'(\pi) \subseteq S(\pi)$ *— in our problem, this set consists of solutions satisfying all hard constraints of* $\pi$*.*

*(iii)* *a* neighborhood relation $R_N(\pi) \subseteq S(\pi) \times S(\pi)$ *— the relation is usually implicitly defined by a set of* moves $M$*, where* $(s, s') \in R_N(\pi)$*, if* $s'$ *can be obtained by applying a move* $m \in M$ *to* $s$*. The set of* neighbors $\{s' : (s, s') \in R_N\}$ *will be denoted by* $N(s)$*;*

*(iv)* *a finite* set of memory states $M(\pi)$ *— memory structures are often used to hold information beyond the current search position (e.g. tabu lists in the case of Tabu Search), however for memory-less SLS algorithms* $M(\pi)$ *may consist of a single state only;*

*(v)* *an* initialization function*, which specifies a probability distribution over initial search positions and memory states and is often based on a problem specific* start heuristic*;*

*(vi)* *an* evaluation function $g : S(\pi) \to \mathbb{R}$*, where global minima of* $g$ *correspond to optimal solutions for* $\pi$*;*

*(vii)* *a* step function *mapping each search position* $x \in S(\pi)$ *and memory state onto a probability distribution over its neighboring search positions and states — the function is applied at each iteration to find a new configuration;*

*(viii)* *a probabilistic* termination predicate*, which indicates for each search position and memory state the probability of terminating the search.*

In the above mentioned paper Hong and Tsang describe the simplest local search method, which is called *iterative improvement*, where at each iteration the current solution $s$ is replaced by an *improving solution* $s' \in N(s)$ with $g(s') < g(s)$ and the algorithm terminates if no improving solutions are found. A special case of iterative improvement is the *steepest descent* approach, where at each iteration the entire neighborhood $N(s)$ is evaluated and the best improving solution is selected. The main weakness with the iterative improvement approach however is its inability to escape local minima of the evaluation function. In order to guide the search towards a global minimum, it is therefore essential to allow for some non-improving moves in the search process. In the context of sport scheduling, the following two local search methods that allow non-improving moves have been extensively studied.

**Tabu Search** First introduced by Glover [20], *tabu search* extends the steepest-descent approach to allow for non-improving solutions. One difficulty in allowing non-improving solutions in the search process is that it might lead to the search cycling near local optima. The key idea behind tabu search is the use of a short-term memory to prevent the search from returning to previous solutions when moving away from local optima, thus preventing the search from cycling. This is typically achieved by marking specific features of solutions visited in the last few iterations as *tabu*, where moves that restore these features are excluded from the neighborhood. Alternatively, when the neighborhood relation is symmetric, a common approach is to mark the last few moves itself as tabu.

As an undesirable side-effect, marking partial solutions or moves as tabu can sometimes lead the search to rule out good solutions. To circumvent this issue Glover proposes the use of so-called *aspiration criteria*, which specify conditions under which the tabu status of a solution should be overridden. A common criterion is to consider a solution marked as tabu, only if it is the best solution found throughout the search process and thus leads to a global improvement on the evaluation function.

Furthermore, Glover also proposes more advanced search strategies that can be used to improve the basic tabu search method. He points out the importance of incorporating *intensification* and *diversification* strategies in the search process. Intensification strategies encourage move combinations and solution features that have previously led to good solutions. Diversification strategies on the other hand guide the search towards new areas of the search space. These search strategies often make use of longer-term frequency-based memory structures, e.g. by counting how often specific solution features have occurred in the solutions visited so far.

**Simulated Annealing** Another widely used local search approach that allows for non-improving moves is *simulated annealing.* Similar to tabu search, simulated annealing can be seen as an extension of the iterative improvement approach. However, instead of evaluating the entire neighborhood, a stochastic approach for neighbor selection is utilized, where at each iteration a neighboring solution $s'$ is selected at random from the neighborhood of the current solution $s$. Then an *acceptance function* defines the probability of accepting $s'$ as the new solution. For minimization problems the acceptance function $a$ is often given by the *Metropolis condition* and defined as

$$a(T, s, s') = \begin{cases} 1 & \text{if } g(s') < g(s), \\ \exp\left(\frac{g(s)-g(s')}{T}\right) & \text{otherwise,} \end{cases} \qquad T > 0.$$

Note that the value of $T$ has a crucial effect on the performance of the algorithm. For $T \to \infty$ the algorithm performs a random walk, whereas for $T \to 0$ the algorithm closely resembles the iterative improvement approach. The core idea of simulated annealing is to dynamically change the value of $T$ throughout the search, starting with a high value which is steadily decreased over time.

In the literature local search methods have been applied to scheduling both time-relaxed and time-constrained round-robin tournaments. Van Bulck et al. [8] apply a tabu search method to scheduling a time-relaxed 2RR with home/away restrictions. They employ a simple neighborhood definition, where at each iteration the home games of a selected team are rescheduled. In order to speed up the neighbor selection they solve a transportation problem instead of iterating over the entire neighborhood. However, more sophisticated neighborhood structures are needed for time-constrained tournaments, due to their restricted structure. Observe, that in a time-constrained tournament rescheduling the games of a single team or the games in a single time slot always leads to violating at least one one-factor or all-different constraint.

Anagnostopoulos et al. [9] propose a simulated annealing approach for the time-constrained 2RR using a complex neighborhood structure with $O(|\mathcal{T}|^3)$ moves, which we will introduce in more detail in section 4.2.1. Similar to our problem they consider instances with hard and soft constraints, which can constrain both the pattern set and the game allocation of the schedule. They incorporate two notable search strategies in their approach. First, they use a *strategic oscillation* strategy for exploring both the feasible and infeasible search space, where an additional penalty for violated hard constraints is introduced. Increasing this penalty leads the search towards the feasible region of the search space, whereas decreasing the penalty encourages the search to consider infeasible configurations. Thus, dynamically changing the penalty can balance the time spent in the feasible and infeasible region of the search space. Secondly, they use *reheats*, where upon search stagnation (e.g. due to very low values of $T$) the search is restarted with the best solution found so far. Gaspero and Schaerf [10] consider the neighborhood definition by Anagnostopoulos et al. in a tabu search context. As evaluating the entire neighborhood at each iteration is computationally expensive they compare various restrictions of the neighborhood and apply them to the traveling tournament problem. We will adopt this neighborhood definition and some of the search strategies in our approach.

# Chapter 4

# Methodology

In this section we describe the methodology used for our approach. The aim of our work was to adapt and improve the common decomposition approach for scheduling round-robin tournaments in order to allow for more general constraints, that can then be used to schedule a wide range of real sport tournaments. As a benchmark for our approach, we use the problem instances of the *21. International Timetabling Competition (ITC2021).*

Our proposed approach can generally be split into two phases. In the first phase we only consider hard constraints of the problem instance and compute a feasible start solution by combining a decomposition approach with a tabu search heuristic. In the second phase we add the soft constraints to the model and again use a tabu search method to improve on the start solution.

As shown in section 2 the set of constraints in our problem can be partitioned into *pattern*, *pattern set* and *general constraints.* This motivated the choice of a break-then-schedule approach in which at first a feasible pattern set is computed and then games are allocated to time slots, as pattern and pattern set constraints can thus be added at an earlier stage in the decomposition than if pattern set generation and game allocation were reversed. We chose to adapt the *Pattern Generating Benders Approach (PGBA)* by Rasmussen [12], which was introduced in section 3.2. The approach follows the common 4 decomposition steps, and utilizes multiple feasibility checks that heuristically determine the feasibility of a given pattern set. However, in order to allow for *team-specific* pattern and pattern set constraints to be added in their respective step of the decomposition, our approach combines pattern generation and team assignment by initially generating feasible patterns for each team.

A challenging part of adapting the decomposition approach was the question of how general constraints could be added to the model. The difficulty with general constraints is that they can constrain both the pattern set and the game allocation of the schedule, thus making it impossible to add them directly to the pattern set model. However when added to the game allocation model, they often lead to infeasibility due to the pattern set being fixed. To deal with this issue two approaches were considered. First, we define necessary feasibility conditions on pattern sets for general constraints to be satisfied, which are then either directly added to the pattern set model or verified as part of the feasibility checks. Secondly, instead of solving the game allocation subproblem as a feasibility problem we solve a *feasibility relaxation*, where the deviations of hard general constraints are minimized. Then, if the relaxation is feasible but did not find a feasible schedule (i.e. the relaxation has a lower objective bound greater than zero), a tabu search heuristic is used to resolve all conflicts in the infeasible schedule found. Otherwise, if the relaxation is infeasible, the fixed pattern set is proven to be infeasible as well and a cut is added to the pattern set model. The diagram in Figure 4.1 gives an overview of how the algorithm for the first phase works.

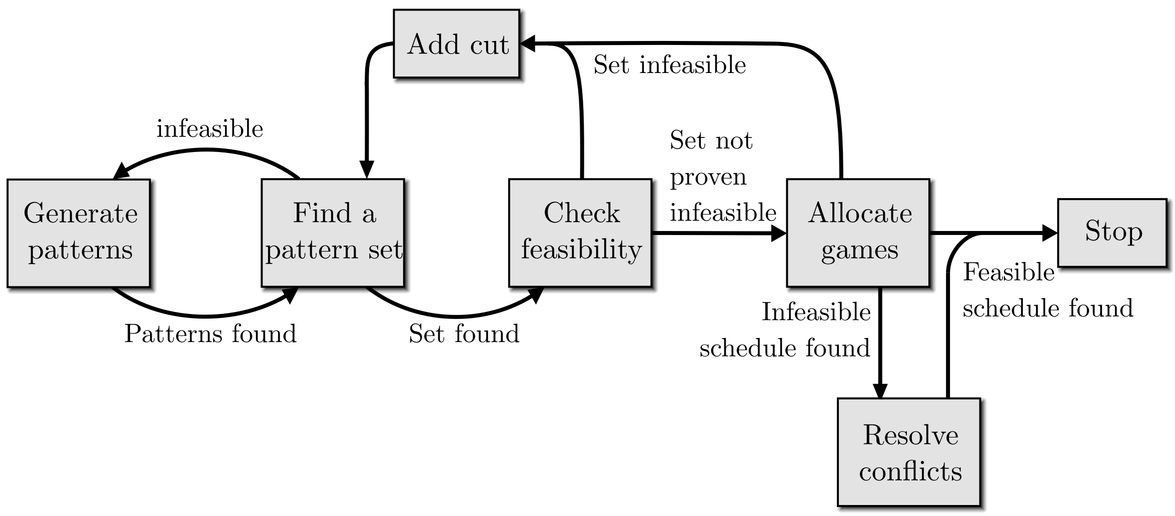


Figure 4.1: The modified PGBA for finding a feasible start solution

## 4.1 Generating Feasible Pattern Sets

In the first phase of our approach we generate a feasible pattern set satisfying all hard pattern and pattern set constraints. At this stage we only consider capacity, break and game constraints but no fairness and separation constraints, as they are always assumed to be soft.

We start by identifying all pattern and pattern set constraints of the given instance. For a given team $t_i$ let $C_i$ be the set of constraints on the pattern of $t_i$ and let $C_P$ be the set of pattern set constraints. Note that $C_i$ and $C_P$ do not contain any game constraints, since game constraints always constrain both the pattern set and the game allocation. However, in order to include hard game constraints during the pattern set generation phase we define relaxations for some game constraints and add them to the corresponding constraint sets. Specifically, we consider game constraints that enforce a lower bound on a number of games to be played in a given set of time slots. In that case we can relax the constraint into two capacity constraints using the following theorem.

**Theorem 2** (Game Constraint Relaxation)**.** *Let $c$ be a game constraint, requiring at least $lb > 0$ games from $G = \{(x_i, y_i) : i = 1, \dots, l, x_i \neq y_i\}$ taking place during time slots in $S \subseteq \mathcal{S}$. Let $T_x = \{x_1, \dots, x_l\}$ and $T_y = \{y_1, \dots, y_l\}$. Then $c$ is violated if one of the following conditions is violated.*

*(i) $T_x$ plays at most $|T_x||S| - lb$ away games during time slots in $S$.*

*(ii) $T_y$ plays at most $|T_y||S| - lb$ home games during time slots in $S$.*

*Proof.* In order to schedule $lb$ games from $G$ during time slots in $S$, teams in $T_x$ must play at least $lb$ home games and teams from $T_y$ must play at least $lb$ away games in $S$. From this (i) and (ii) can be directly deduced, based on the observation that any subset of teams $T$ collectively play $|T||S|$ games during time slots in $S$.

For each game constraint with a lower bound $lb > 0$ we add capacity constraints for (i) and (ii) to the corresponding constraint sets, depending on whether (i) and (ii) are pattern or pattern set constraints. We note that this simple relaxation is effective for constraints on a small number of games. As an example, consider *fixed game assignments* which are a special case of game constraints and require a game $(t_i, t_j)$ to be played in a specific time slot $s$. In that case the above relaxation would add capacity constraints to $C_i$ and $C_j$, enforcing $t_i$ playing at home and $t_j$ playing away in slot $s$.

### 4.1.1 Pattern Generation

Our decomposition approach starts by first generating feasible patterns for each team. Specifically, for each team $t_i$ we generate a set of patterns $P_i$, such that each pattern in $P_i$ satisfies all constraints in $C_i$. Since the number of feasible patterns can be exponentially large it is crucial to limit the number of patterns generated. Therefore, we add an upper bound on the number of breaks allowed in a feasible pattern. Note that for any set of time slots $S$ there are either zero or two patterns with its breaks occurring in slots $S$. Thus the number of patterns with length $m$ and $k$ breaks is bounded by $2\binom{m}{k}$.

The motivation for limiting the number of breaks is twofold. First, breaks are generally considered undesirable in a schedule, therefore constraints on breaks usually involve only an upper bound and no lower bound. Thus, by limiting the total number of breaks in a pattern break constraints are likely to be fulfilled. Secondly, pattern sets with a minimal number of breaks have been extensively studied in the literature. Notably, the pattern diversity condition introduced in section 3.2 is proven to be sufficient for the feasibility of pattern sets with a minimal number of breaks. Therefore, applying the pattern diversity condition on patterns with a limited number of breaks can potentially improve the runtime of the algorithm if a feasible schedule with a minimal number of breaks exists.

Contrary to the PGBA we opt for a simple enumeration algorithm for generating patterns instead of a more sophisticated constraint programming approach. In order to generate all patterns with $k$ breaks, our algorithm iterates through all possible combinations of breaks of size $k$. For a given combination of breaks $B \subset \mathcal{S} \backslash \{s_1\}$ we generate two bit strings $x^{(0)}$ and $x^{(1)}$ of length $m$, such that

$$x_1^{(i)} = i, \qquad x_k^{(i)} = \begin{cases} x_{k-1}^{(i)} & \text{if } s_k \in B \\ \left(1 - x_{k-1}^{(i)}\right) & \text{otherwise} \end{cases}, k = 2, \dots, m,\ i = 0, 1\,,$$

where $x_k^{(i)}$ is the $k$-th bit of $x^{(i)}$. Note that $x^{(0)}$ and $x^{(1)}$ are bitwise inverses of each other. If the number of ones equals the number of zeros in $x^{(0)}$ we can interpret both bit strings as patterns with breaks during slots in $B$, by letting a one correspond to a home game and a zero correspond to an away game. Then, for each team $t_i$, we validate all pattern constraints $C_i$ on both patterns and if a pattern is feasible it is added to $P_i$. As all of our instances are limited to at most 20 teams this approach proved sufficient, with the algorithm being able so generate all feasible patterns with up to 6 breaks in a few seconds.

### 4.1.2 Pattern Set Model

Given a set of feasible patterns $P_i$ for each team we want to select one pattern from each set to form a feasible pattern set. In the following let $\mathcal{P}$ be the set of all patterns (i.e the union of the sets $P_i$) and $h_{ps}$ indicate whether the pattern $p \in \mathcal{P}$ has a home game ($h_{ps} = 1$) or an away game ($h_{ps} = 0$) in slot $s$. Similar to the PGBA we use an IP model called PSM, to first generate candidate solutions, which are then heuristically checked for feasibility. The model uses binary variables $x_{ip}$ indicating whether pattern $p$ is assigned to team $i$ and can be formulated as follows.

$$
\text{(PSM)}\qquad
\begin{aligned}
x_{ip} &= 0, && i \in \mathcal{T},\, p \in \mathcal{P}\backslash P_i && \text{(2a)}\\
\sum_{p\in\mathcal{P}} x_{ip} &= 1, && i \in \mathcal{T} && \text{(2b)}\\
\sum_{i\in\mathcal{T}} x_{ip} &\leq 1,, && p \in \mathcal{P} && \text{(2c)}\\
\sum_{i\in\mathcal{T}}\sum_{p\in\mathcal{P}} h_{ps}\, x_{ip} &= \frac{n}{2}, && s \in \mathcal{S} && \text{(2d)}\\
x_{ip} &\in \{0,1\}, && i \in \mathcal{T},\, p \in \mathcal{P}
\end{aligned}
$$

Constraints (2a) and (2b) ensure that each team $i$ is assigned exactly one pattern from its set of feasible patterns $P_i$. Note that it is not possible to schedule the games between two teams if they have the same pattern. Therefore, (2c) enforces that each pattern is assigned to at most one team. Finally, (2d) ensures that the number of teams that play at home equals the number of teams that play away in each slot.

Furthermore we add all pattern set constraints to PSM. For a pattern set constraining capacity constraint CA$(T_1, \mathcal{T}, S, ub, md)$, requiring teams in $T_1$ to collectively play at most $ub$ home ($md$ = H) or away ($md$ = A) games during slots in $S$, we add the following constraint to (PSM).

$$
ub \geq \begin{cases} \sum_{s\in S}\sum_{i\in T_1}\sum_{p\in\mathcal{P}} h_{ps}x_{ip} & \text{if } md = \text{H},\\ \sum_{s\in S}\sum_{i\in T_1}\sum_{p\in\mathcal{P}} (1-h_{ps})x_{ip} & \text{if } md = \text{A}. \end{cases}
$$

Note that pattern set constraining capacity constraints of mode HA (home and away games) are redundant and do not need to be considered in the model.

For all break constraints BR$(T, S, ub, md)$, limiting the number of breaks ($md =$ HA), home breaks ($md$ = H) or away breaks ($md$ = A) teams in $T$ have in slots in $S$, we add the following constraint to PSM.

$$
ub \geq \begin{cases} \sum_{s\in S}\sum_{i\in T_1}\sum_{p\in\mathcal{P}} b_{ps}h_{ps}x_{ip} & \text{if } md = \text{H},\\ \sum_{s\in S}\sum_{i\in T_1}\sum_{p\in\mathcal{P}} b_{ps}(1-h_{ps})x_{ip} & \text{if } md = \text{A},\\ \sum_{s\in S}\sum_{i\in T_1}\sum_{p\in\mathcal{P}} b_{ps}x_{ip} & \text{if } md = \text{HA}. \end{cases}
$$

If PSM is infeasible the limit on the number of breaks is increased and additional patterns are generated. Otherwise, a candidate solution is obtained and checked for feasibility, using the conditions described in the next section.

### 4.1.3 Feasibility Conditions

A pattern-set satisfying all pattern and pattern-set constraints of the model is not guaranteed to allow all games to be scheduled. Therefore a solution found by PSM might not allow for a corresponding game allocation. Similar to the PGBA we utilize a cut generation approach by heuristically determining infeasibility using a logic-based Benders decomposition. Contrary to a traditional Benders decomposition a logic-based Benders decomposition does not derive cuts from a linear subproblem, but from inference duals describing necessary feasibility conditions. A good feasibility condition not only recognizes infeasibility in the current solution but also identifies which specific solution features cause the solution to be infeasible. Then, a cut can be introduced that excludes these features from future solutions, where the quality of the cut is determined by the number of infeasible solutions cut off.

We differentiate between *general cuts* and *team-specific cuts* as follows. Let $P = (p_1, \dots, p_n)$ be a candidate solution obtained by PSM where team $t_i$ is assigned pattern $p_i$. If a feasibility check determines that a subset $P'$ of $P$ leads to an infeasible pattern set, indifferent to which teams are assigned to patterns in $P'$, we add the general cut

$$\sum_{p \in P'} \sum_{i \in \mathcal{T}} x_{ip} \leq |P'| - 1 \tag{3}$$

to PSM, cutting of all pattern sets that contain $P'$. However if $P'$ is proven to not have a corresponding feasible game allocation due to the assignment of teams to patterns, we add the weaker team-specific cut

$$\sum_{i : p_i \in P'} x_{t_i p_i} \leq |P'| - 1, \tag{4}$$

requiring at least one team in $\{t_i : p_i \in P'\}$ to have a different pattern assigned. Note, that the size of $P'$ has a big impact on the quality of the cut, with smaller subset sizes leading to more infeasible solutions cut off. Therefore, we first consider small subsets of patterns in each of the following feasibility checks and stop once a cut was generated.

**Phased Constraints** In order to schedule the games between two teams $t_i$ and $t_j$ there must be one slot $s^{(1)}$ in which $t_i$ plays at home and $t_j$ plays away, as well as another slot $s^{(2)}$ in which $t_j$ plays at home and $t_i$ plays away. Note that this is always the case if the pattern set consists of unique patterns. However if the tournament requires a phased structure $s^{(1)}$ and $s^{(2)}$ must not be selected from the same half of the tournament, as both teams must meet exactly once in each half. In that case we can define the following feasibility condition for each pair of patterns $(p_i, p_j)$.

$$\begin{aligned} &\Big(\min\big\{k : h_{p_i s_k} = 1 \wedge h_{p_j s_k} = 0\big\} \leq n-1 < \max\big\{k : h_{p_i s_k} = 0 \wedge h_{p_j s_k} = 1\big\}\Big) \\ \vee\quad &\Big(\min\big\{k : h_{p_i s_k} = 0 \wedge h_{p_j s_k} = 1\big\} \leq n-1 < \max\big\{k : h_{p_i s_k} = 1 \wedge h_{p_j s_k} = 0\big\}\Big) \end{aligned}$$

For each pair of patterns $(p_i, p_j)$ violating this condition we add the general cut

$$\sum_{t \in \mathcal{T}} x_{tp_i} + x_{tp_j} \leq 1$$

to PSM.

**Pattern Diversity** For subsets of teams with similar patterns scheduling all games might not be possible. Similar to the PGBA we utilize the pattern diversity condition to check for feasibility. The condition, which was introduced in section 3.2, states that $P$ is not feasible if a subset of teams $T \subseteq \mathcal{T}$ exists, such that

$$\sum_{s \in \mathcal{S}} \min\left\{ \sum_{i : t_i \in T} h_{p_i s}, \sum_{i : t_i \in T} (1 - h_{p_i s}) \right\} < |T|(|T| - 1). \tag{5}$$

In that case not all games between teams in $T$ can be scheduled and the general cut

$$\sum_{i:t_i \in T} \sum_{j \in \mathcal{T}} x_{jp_i} \leq |T| - 1 \tag{6}$$

is added to PSM.

Checking all subsets of $\mathcal{T}$ is however not computationally viable. Therefore the PGBA uses an IP model called UBM which, for a given subset size $k$, finds the subset of teams that minimizes the left hand side in (5). The model uses binary variables $\alpha_i$, which indicate whether the team $t_i$ is contained in the subset, as well as continuous variables $\beta_s$ which are used to count games in slot $s$. A binary variable $\delta_s$ indicates whether $\beta_s$ counts home games ($\delta_s = 1$) or away games ($\delta_s = 0$).

$$\begin{aligned}
&& \underset{p}{\text{minimize}} \quad & \sum_{s \in \mathcal{S}} \beta_s && && \text{(7a)}\\
&& \text{subject to} \quad & \sum_{i \in \mathcal{T}} \alpha_i = k, && && \text{(7b)}\\
&\text{(UBM)} && \beta_s + k(1-\delta_s) - \sum_{i \in \mathcal{T}} h_{p_i s}\alpha_i \geq 0, && s \in \mathcal{S}, && \text{(7c)}\\
&& & \beta_s + k\delta_s - \sum_{i \in \mathcal{T}} (1-h_{p_i s})\alpha_i \geq 0, && s \in \mathcal{S}, && \text{(7d)}\\
&& & \alpha_i, \delta_s \in \{0,1\}, && i \in \mathcal{T}, s \in \mathcal{S}, && \text{(7e)}\\
&& & \beta_s \in \mathbb{R}_{>0}, && s \in \mathcal{S} && \text{(7f)}
\end{aligned}$$

Constraint (7b) ensures that exactly $k$ teams are selected and constraints (7c) and (7d) are used to bound $\beta_s$ from below by the minimum of the number of home games and the number of away games played in slot $s$. Note that depending on the value of $\delta_s$ one of the constraints (7c) and (7d) is always satisfied. If the optimal solution of UBM has a value less than $k(k-1)$ the pattern set $P$ is infeasible. In that case the cut 6 is added for $T = \{t_i : \alpha_i^* = 1\}$, where $\alpha_i^*$ is the value of $\alpha_i$ in the optimal solution.

For phased tournaments we can strengthen the feasibility check by considering both halves of the tournament as two separate single round-robin tournaments and utilizing the pattern diversity condition for 1RR. In that case we solve UBM for both halves separately and add a cut if the optimal objective value is less than $\frac{k(k-1)}{2}$ in one half.

**General Constraints** The previous two feasibility conditions are used to prove that the given pattern set has no corresponding game allocation. However, in some cases the pattern set is feasible but only allows for an infeasible game allocation, where some general constraints can not be satisfied. To identify these pattern sets, we employ additional feasibility checks for general capacity and game constraints.

Consider a constraint $c$ enforcing a lower bound $lb$ on the number of games from $G = \{(x_i, y_i) : i = 1, \ldots, k\}$ taking place during slots in $S$. Let $T_x = \{x_1, \ldots, x_k\}$ be the set of home teams and $T_y = \{x_1, \ldots, x_k\}$ be the set of away teams. If $|T_x| = 1$ or $|T_y| = 1$ the one-factor constraints ensure that at most one game from $G$ can take place during each slot in $S$. In that case we consider the bipartite graph $B$ with vertices $G \cup S$ where an edge $(g, s) \in G \times S$ is given, if and only if $g$ can be scheduled during $s$, i.e for $g = (x, y)$ we have that $x$ plays at home and $y$ plays away in slot $s$. Then $c$ is violated in any game allocation if a maximum matching in $B$ has a size of at most $lb - 1$. In that case we add the team-specific cut

$$\sum_{i:t_i \in T_x \cup T_y} x_{t_i p_i} \leq |T_x \cup T_y| - 1$$

to PSM. We note that a maximum bipartite matching can be easily obtained, e.g by utilizing the Hungarian method.

While expressing a game constraint in the above form is straight-forward, capacity constraints are defined with an upper bound and no lower bound on the number of games to be played. However, since all games must be scheduled, we can reformulate a capacity constraint $c = CA(T_1, T_2, S, ub, md)$ in one of the following two ways:

(i) Let $X$ be the total number of games of mode $md$ that teams in $T_1$ play against teams in $T_2$ over the course of the season. Then $c$ is satisfied, if and only if teams in $T_1$ play at least $X - ub$ games of mode $md$ against teams in $T_2$ during slots in $\mathcal{S} \backslash S$.

(ii) Let $Y$ be the total number of games of mode $md$ that teams in $T_1$ play during time slots in $S$. Then $c$ is satisfied, if and only if teams in $T_1$ play at least $Y - ub$ games of mode $md$ against teams in $\mathcal{T} \backslash T_2$ during slots in $S$.

### 4.1.4 Game Allocation Model

The feasibility conditions presented in the previous section are not exhaustive, where a pattern set solution obtained by PSM can pass all feasibility checks yet not have a corresponding game allocation. In a similar fashion to the PGBA, we utilize an IP model to find a corresponding game allocation if the pattern set is feasible. If the IP model is infeasible a general cut is added to PSM cutting off the infeasible pattern set and all its permutations.

Furthermore, at this stage of the decomposition, we add all hard general constraints to the model. However since general constraints can also constrain the pattern set, they are likely to cause the model to be infeasible due to the pattern set being fixed. Therefore it is crucial to allow some general constraints to be violated in a solution. Thus, contrary to the PGBA, we do not solve the game allocation problem as a feasibility problem but as a feasibility relaxation. Our model then finds a game allocation that minimizes the deviations of the general constraints.

We use an IP formulation that is based on the formulation for 2RR by Trick [6]. In addition to binary variables $z_{ijs}$, indicating whether the game $(i, j)$ is scheduled in slot $s$, we use continuous variables $\Delta_c$ for modeling the deviation of each general constraint $c \in C_G$. The resulting base formulation for the game allocation model (GAM) is as follows.

$$\text{(GAM)} \qquad \begin{array}{lll}
\text{minimize} & \displaystyle\sum_{c \in C_G} \Delta_c & \\
\text{subject to} & z_{iis} = 0, & i \in \mathcal{T},\ s \in \mathcal{S}, \\
& z_{ijs} = 0, & i, j \in \mathcal{T}, s \in \mathcal{S} : h_{is} = 0 \vee h_{js} = 1, \\
& \displaystyle\sum_{j \in \mathcal{T}} z_{ijs} + z_{jis} = 1, & i \in \mathcal{T},\ s \in \mathcal{S}, \\
& \displaystyle\sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{T}} z_{ijs} = \frac{n}{2}, & s \in \mathcal{S}, \\
& \displaystyle\sum_{s \in \mathcal{S}} z_{ijs} = 1, & i, j \in \mathcal{T},\ i \neq j, \\
& z_{ijs} \in \{0, 1\} & i, j \in \mathcal{T}, s \in \mathcal{S}, \\
& \Delta_c \in \mathbb{R}_{\geq 0}, & c \in C_G
\end{array}$$

For each general capacity constraint $c = \text{CA}(T_1, T_2, S, ub, md)$, limiting the number of games ($md = \text{HA}$), home games ($md = \text{H}$) or away games ($md = \text{A}$) teams in $T_1$ play collectively against teams in $T_2$ during slots in $S$, we add the following constraint to GAM.

$$ub + \Delta_c \geq \begin{cases} \sum_{i \in T_1} \sum_{j \in T_2} \sum_{s \in S} z_{ijs}, & \text{if } md = \text{H} \\ \sum_{i \in T_1} \sum_{j \in T_2} \sum_{s \in S} z_{jis}, & \text{if } md = \text{A} \\ \sum_{i \in T_1} \sum_{j \in T_2} \sum_{s \in S} z_{ijs} + z_{jis}, & \text{if } md = \text{HA}. \end{cases}$$

For each game constraint $c = \text{GA}(G, S, lb, ub)$, requiring at least $lb$ and at most $ub$ games from $G$ to be played during slots in $S$, we first add two additional variables $c_l, c_u \in \mathbb{R}_{\geq 0}$, which are used to relax the lower and upper bound of $c$ respectively. We then and the following constraints to GAM.

$$lb - c_l \leq \sum_{(i,j) \in G} \sum_{s \in S} x_{ijs} \leq ub + c_u$$
$$\Delta_c = c_l + c_u.$$

Finally, if the tournament requires a phased structure, we add the following constraint for each pair of teams $(i, j)$, requiring $i$ and $j$ to play each other exactly once in the first half of the tournament.

$$\sum_{k=1}^{m/2} x_{ijs_k} + x_{jis_k} = 1.$$

If GAM is infeasible the pattern set $P$ must be infeasible as well. In that case we add the general cut

$$\sum_{p \in P} \sum_{i \in \mathcal{T}} x_{ip} \leq n - 1$$

to PSM. Otherwise $P$ is feasible and GAM provides a schedule that satisfies all pattern and pattern set constraints, but potentially violates some general constraints. In that case we use a local search heuristic to resolve all conflicts in the schedule. We will introduce this heuristic in the next section.

## 4.2 Tabu Search

The tabu search heuristic used in our approach can be split into two phases. First we resolve all conflicts in the schedule obtained by the decomposition approach, by minimizing the deviations of all hard constraints. Then if a feasible start solution is found, we add the penalties of all soft constraints to the objective and find improving solutions. Since defining a neighborhood that connects the feasible solution space is difficult, we consider feasible and infeasible configurations in the second phase of the search.

Our approach employs a large neighborhood with $O\left(|\mathcal{T}|^3\right)$ moves. In order to improve the overall speed of the procedure, we dynamically restrict the neighborhood at each iteration to decrease the number of moves that need to be considered as well as to guide the search towards favorable solutions. Generally speaking, we first identify the entries in the schedule that are constrained by the violated constraints at each iteration. Then we restrict the neighborhood to moves that change at least one of these entries, which leads to fast iterations if only few constraints are violated. Furthermore, we employ a *first-improvement* strategy, where any improving move is accepted during the neighbor selection phase. This subsequently leads to faster iterations when a large number of constraints is violated and multiple improving solutions exist in the neighborhood of the current configuration.

To guide the search towards good solutions we utilize two different memory structures. A short-term memory is used to prevent the search from cycling, where moves that are likely to return to solutions previously visited are marked as tabu. Additionally, during the first phase we use a frequency based long-term memory structure for diversification. We will describe these memory structures as well as the neighbor selection strategy in more detail in section 4.2.2.

### 4.2.1 Neighborhood Definition for 2RR

We use the neighborhood definition for 2RR by Anagnostopoulos et al. [9], which defines 5 types of moves that can be applied to a schedule. To illustrate these moves we consider the following example schedule for a phased 2RR with 6 teams.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 3 | -2 | 6 | -4 | -3 | 2 | -6 | -5 | 4 |
| **2** | -3 | -6 | 1 | -4 | -5 | 6 | -1 | 5 | 4 | 3 |
| **3** | 2 | -1 | 4 | -5 | -6 | 1 | 5 | -4 | 6 | -2 |
| **4** | -6 | 5 | -3 | 2 | 1 | -5 | 6 | 3 | -2 | -1 |
| **5** | -1 | -4 | 6 | 3 | 2 | 4 | -3 | -2 | 1 | -6 |
| **6** | 4 | 2 | -5 | -1 | 3 | -2 | -4 | 1 | -3 | 5 |

Figure 4.2: Example schedule for 2RR with 6 teams

***SwapHomes*($t_i, t_j$)** Let $s_i$ and $s_j$ be the slots in which teams $t_i$ and $t_j$ meet in the schedule $S$. The move then simply swaps the home/away roles of teams $t_i$ and $t_j$, by inverting the entries of $t_i$ and $t_j$ in the slots $s_i$ and $s_j$.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 3 | -2 | 6 | -4 | -3 | 2 | -6 | -5 | 4 |
| **2** | -3 | -6 | 1 | 4 | -5 | 6 | -1 | 5 | -4 | 3 |
| **3** | 2 | -1 | 4 | -5 | -6 | 1 | 5 | -4 | 6 | -2 |
| **4** | -6 | 5 | -3 | -2 | 1 | -5 | 6 | 3 | 2 | -1 |
| **5** | -1 | -4 | 6 | 3 | 2 | 4 | -3 | -2 | 1 | -6 |
| **6** | 4 | 2 | -5 | -1 | 3 | -2 | -4 | 1 | -3 | 5 |

Figure 4.3: Resulting schedule of the move *SwapHomes*($t_2, t_4$)

***SwapRounds*($s_i, s_j$)** This move swaps the games played in slots $s_i$ and $s_j$ for each team. Note, that when a phased structure of the tournament should be maintained moves on slots from different phases must be excluded from the neighborhood.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 2 | -2 | 6 | -4 | -3 | 3 | -6 | -5 | 4 |
| **2** | -3 | -1 | 1 | -4 | -5 | 6 | -6 | 5 | 4 | 3 |
| **3** | 2 | 5 | 4 | -5 | -6 | 1 | -1 | -4 | 6 | -2 |
| **4** | -6 | 6 | -3 | 2 | 1 | -5 | 5 | 3 | -2 | -1 |
| **5** | -1 | -3 | 6 | 3 | 2 | 4 | -4 | -2 | 1 | -6 |
| **6** | 4 | -4 | -5 | -1 | 3 | -2 | 2 | 1 | -3 | 5 |

Figure 4.4: Resulting schedule of the move *SwapRounds*($t_2, t_7$)

***PartialSwapRounds***$(t_i, s_k, s_l)$ This move generalizes the previous move, by identifying a minimal subset of teams $T$ containing the team $t_i$, such that swapping the games of $T$ in slots $s_k$ and $s_l$ produces a round-robin schedule. Finding the minimal subset can be achieved by considering the union of the two one-factor graph representations of rounds $k$ and $l$, where two teams are connected via an edge if they meet in slot $s_k$ or $s_l$. Then, $T$ is given by the connected component containing $t_i$.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 3 | -6 | 6 | -4 | -3 | 2 | -2 | -5 | 4 |
| **2** | -3 | -6 | 5 | -4 | -5 | 6 | -1 | 1 | 4 | 3 |
| **3** | 2 | -1 | 4 | -5 | -6 | 1 | 5 | -4 | 6 | -2 |
| **4** | -6 | 5 | -3 | 2 | 1 | -5 | 6 | 3 | -2 | -1 |
| **5** | -1 | -4 | -2 | 3 | 2 | 4 | -3 | 6 | 1 | -6 |
| **6** | 4 | 2 | 1 | -1 | 3 | -2 | -4 | -5 | -3 | 5 |

Figure 4.5: Resulting schedule of the move *PartialSwapRounds*($t_5, s_3, s_8$)

***SwapTeams***$(t_i, t_j)$ This move swaps the games of teams $t_i$ and $t_j$ in all slots except for the ones in which they play each other. In order to obtain a round-robin schedule the games of the opponents of teams $t_i$ and $t_j$ must also be swapped, resulting in four changed entries in each round in which $t_i$ and $t_j$ do not meet.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 3 | 5 | -2 | 6 | -4 | -5 | 2 | -6 | -3 | 4 |
| **2** | -5 | -6 | 1 | -4 | -3 | 6 | -1 | 3 | 4 | 5 |
| **3** | -1 | -4 | 6 | -5 | 2 | 4 | 5 | -2 | 1 | -6 |
| **4** | -6 | 3 | -5 | 2 | 1 | -3 | 6 | 5 | -2 | -1 |
| **5** | 2 | -1 | 4 | 3 | -6 | 1 | -3 | -4 | 6 | -2 |
| **6** | 4 | 2 | -3 | -1 | 5 | -2 | -4 | 1 | -5 | 3 |

Figure 4.6: Resulting schedule of the move *SwapTeams*($t_3, t_5$)

***PartialSwapTeams***$(t_i, t_j, s_k)$ This move generalizes the previous move by identifying a minimal subset of slots containing $s_k$, where swapping the games of teams $t_i$ and $t_j$ produces a round-robin schedule. The minimal subset can be identified recursively using the following observation. Let $g_i$ and $g_j$ be the entries in the schedule for teams $t_i$ and $t_j$ in slot $s_k$. Then, swapping the games of both teams in slot $s_k$ also requires swapping the games of both teams in the slot in which $t_i$ has the entry $g_j$ and the slot in which $t_j$ has the entry $g_i$. We again need to change four entries in the schedule for each slot in which the games of $t_i$ and $t_j$ are swapped. Observe, that partially swapping teams can lead to violating phased constraints. Therefore we exclude this move from the neighborhood if a phased tournament structure should be preserved.

| **T\S** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 3 | -6 | 6 | -4 | -3 | 2 | -2 | -5 | 4 |
| **2** | -3 | -6 | -5 | -4 | 3 | 6 | -1 | 1 | 4 | 5 |
| **3** | 2 | -1 | 4 | -5 | -2 | 1 | 5 | -4 | 6 | -6 |
| **4** | -6 | 5 | -3 | 2 | 1 | -5 | 6 | 3 | -2 | -1 |
| **5** | -1 | -4 | 2 | 3 | 6 | 4 | -3 | -6 | 1 | -2 |
| **6** | 4 | 2 | 1 | -1 | -5 | -2 | -4 | 5 | -3 | 3 |

Figure 4.7: Resulting schedule of the move *PartialSwapTeams*$(t_2, t_6, s_8)$

Note, that applying different moves does not guarantee different resulting schedules. If applying the moves $m_1$ and $m_2$ to the schedule $S$ results in the same schedule $S'$ we call $m_1$ and $m_2$ *equivalent* (on $S$). As an example, consider the move *PartialSwapRounds* for the parameter tuples $(t_5, s_3, s_8)$ and $(t_2, s_3, s_8)$, which result in the same schedule (Fig. 4.5) when applied to the schedule in Fig. 4.2.

Identifying all equivalent moves is crucial to prevent the search from returning back to previous solutions, since whenever a move is marked tabu we must also mark all equivalent move as tabu. Furthermore excluding equivalent moves during the neighbor selection leads to significant performance improvements, since it prevents evaluating the same neighbor multiple times. We use a simple criterion for identifying equivalent moves, by restricting the neighborhood to moves that change a unique set of entries in the schedule.

### 4.2.2 Search Strategies

In this section we describe the details of our proposed Tabu Search method. We use very similar heuristics for resolving conflicts in the solution obtained by the decomposition and for minimizing the penalties of soft constraints. In fact, the former can be seen as a special case of the general heuristic described in this section, where the set of hard constraints is assumed to be empty (i.e. the heuristic is applied to the *relaxed problem instance*, where soft constraints are removed and hard constraints are replaced by soft constraints).

Two memory structures are used in our approach. A tabu matrix $T \in \mathbb{N}^M$, where $M$ is the set of moves, is used to mark the last few moves and all equivalent moves as tabu. Specifically, $T_m$ indicates the last iteration at which the move $m$ is tabu. Thus, in the $k$-th iteration the move $m$ is tabu, if and only if $T_m \leq k$. Then, if the move $m'$ should be marked as tabu for $l$ iterations, $T_{m'}$ is set to $k + l$. Furthermore a long-term frequency matrix $F \in \mathbb{N}^{\mathcal{T} \times \mathcal{S}}$ is used, where $F_{(i,s)}$ is the number of times the entry of the team $i$ in slot $s$ was changed throughout the search.

**Search phases** A common strategy in applications of local search methods is to split up the search into phases, where the search is restarted with the best solution found so far at the beginning of each phase. We utilize this approach to prevent the search from diverging to far from the optimal solution, where a phase ends if no globally improving feasible solution has been found for a given number of iterations $k$. We will refer to $k$ as the *phase length*. Note, that if no new best solution was generated during a search phase, the previous best solution is restored again. In that case the initial moves performed in the last phase are also marked as tabu, in order to guide the search towards new regions of the search space. This can lead to all moves being marked tabu at the start of a new phase if the same solution was recovered multiple times, in which case the algorithm terminates.

Another issue arises from evaluating both feasible and infeasible solutions, where the search might diverge to far from the feasible search space, thus preventing the search from finding new feasible solutions. To prevent this issue we impose strong restrictions on when infeasible solutions should be accepted in the neighbor selection. Generally, we only allow infeasible configurations if no feasible non-tabu neighbor exists in the neighborhood of the current solution. However, at the start of each phase except for the initial one, we perform a small number of iterations in which infeasible solutions are accepted without any restrictions. This allows the search to diversify and explore new areas of the search space without diverging to far from the feasible search space.

**Neighbor selection** Evaluating the entire neighborhood at each iteration is computationally expensive, due to the large number of moves that have to be evaluated. In addition to restricting the neighborhood to pairwise non-equivalent moves at each iteration, we use two ideas presented by Hoos and Stützle [21] to reduce the number of moves that have to be evaluated in each search step.

First, we further restrict the neighborhood by excluding moves that are proven to not change the evaluation function. To that end, for each violated constraint $c$ we identify the set of *constrained entries* $E_c$ in the schedule, where changing an entry not in $E_c$ is guaranteed to not change the deviation of $c$. As an example, for a capacity constraint $c = \mathrm{CA}(T_1, T_2, S, ub, md)$ the set of constrained entries would be given by

$$E_c = (T_1 \cup T_2) \times S.$$

At each iteration we dynamically restrict the neighborhood to moves that change at least one constrained entry in a violated constraint. This leads to fast search iterations when only few constraints are violated.

Secondly, we use a *first-improvement* selection strategy. For a given solution $y$ let $g_h(y)$ and $g_s(y)$ be given by

$$g_h(y) = \sum_{c \in C_h} \Delta_c(y), \qquad g_s(y) = \sum_{c \in C_s} p_c \Delta_c(y),$$

i.e $g_h$ and $g_s$ denote the sums of deviations of all hard constraints and penalties of all soft constraints respectively. Instead of evaluating the entire neighborhood we select the first move $m$ that fulfills one of the following two conditions:

(i) Applying $m$ decreases the value of $g_h$, and does not increase the value of $g_s$.

(ii) Applying $m$ does not increase the the value of $g_h$, and decreases the value of $g_s$.

The first improvement strategy generally leads to faster iterations when a large number of constraints is violated and multiple improving solutions exist in the neighborhood of the current solution.

At each iteration we randomize the order in which the moves are considered, to ensure that each move satisfying the above conditions is equally likely to be selected. If no improving solution was found, the entire neighborhood $N(x)$ of the current solution $x$ is evaluated and the best feasible non-tabu solution $x'_{\text{feas}}$ is identified. If no feasible solution exists or the search is at the start of a new phase where infeasible solutions are accepted, the best infeasible non-tabu solution $x'_{\text{infeas}}$ is identified as well. To prevent the search from diverging from the feasible search space infeasible solutions that reduce the deviations of hard constraints are preferred if the current configuration is infeasible. However, if the current solution is feasible an infeasible solution that minimizes the penalty of all soft constraints is selected instead. Formally we define the best feasible and infeasible solutions as

$$x'_{\text{feas}} = \operatorname*{arg\,min}_{y\in\bar{N}(x):\, g_h(y)=0} g_s(y), \qquad x'_{\text{infeas}} = \begin{cases} \operatorname*{arg\,min}\limits_{y\in\bar{N}(x)} \big(g_h(y), g_a(y)\big), & \text{if } g_h(x) > 0, \\ \operatorname*{arg\,min}\limits_{y\in\bar{N}(x)} \big(g_a(y), g_h(y)\big), & \text{if } g_h(x) = 0, \end{cases}$$

where $\bar{N}(x)$ denotes the restriction of $N(x)$ to non-tabu solutions and the tuples in the right-hand side of the second equation are compared in lexicographic order. If infeasible solutions should be accepted or no feasible non-tabu solution exists, $x'_{\text{infeas}}$ is selected as the new solution and otherwise $x'_{\text{feas}}$ is selected.

**Diversification** Restricting the search to moves that change constrained entries of violated constraints can potentially prevent the search from modifying some parts of the schedule if only very few constraints are violated, which is often the case when the tabu search method is used to resolve the conflicts in the schedule obtained by the decomposition approach. To deal with this issue in the first phase, we first return to the best solution found so far once the search stagnates (i.e. no improving solution was found for a given number of iterations). Then we apply a very short diversification phase in which moves on rarely changed entries of the schedule are encouraged. To that end, the frequency matrix $F$ is used to add an additional penalty to each entry of the schedule, depending on the number of times the entry was changed during the search process. This leads to an modified evaluation function

$$g'(y) = g_s(y) + \sum_{e:y_e\neq x_e} F_e,$$

where $x_e$ and $y_e$ denote the value of the current solution $x$ and the neighboring solution $y$ on the entry $e$ respectively. Note, that $g_h(y) = 0$ since this diversification strategy is only applied to the relaxed instance, where no hard constraints are present in the instance.

In order to not not diverge to much from the best solution, only very few search steps are performed during a diversification phase. Furthermore during diversification we always evaluate the entire neighborhood and select the solution that minimizes $g'$.

# Chapter 5

# Results

In this section we describe our computational experiments and present our results. First we introduce an IP formulation of the problem, that we use as an additional benchmark for our approach. Then we analyze how the modified PGBA performs in generating feasible start solutions and compare the results of our tabu search method with the best solutions found in the competition.

Our approach was tested on all 45 instances provided in the competition, which include phased and non-phased double round-robin tournaments for 16 to 20 teams. All experiments were performed on a Google Cloud C2 instance with 8 vCPUs and 32GB RAM. The algorithms are implemented in the Julia programming language and Gurobi is used for solving the IP problems [1]. All computation times presented in this section are in seconds.

## 5.1 Integer Programming Formulation

We extend the base IP-formulation by Trick, which we presented in section 3.1, for modeling the problem as an integer program. In addition to binary variables $z_{ijs}$ indicating whether the game $(i, j)$ is scheduled during the slot $s$ we add auxiliary binary variables $b_{is}^{(h)}$ and $b_{is}^{(a)}$ which indicate whether team $i$ has a home break or away break in slot $s$ respectively. Furthermore we use continuous variables $\Delta_c \geq 0$ for modeling the deviations of the constraints. We explain below how the different constraint types can be added to the following model.

$$\underset{p}{\text{minimize}} \quad \sum_{c \in C_s} p_c \Delta_c$$

$$\text{subject to} \quad \Delta_c = 0, \qquad c \in C_h, \tag{9a}$$

$$z_{iis} = 0, \qquad i \in \mathcal{T},\ s \in \mathcal{S}, \tag{9b}$$

$$\sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{T}} z_{ijs} = \frac{n}{2} \qquad s \in \mathcal{S}, \tag{9c}$$

$$\sum_{j \in \mathcal{T}} z_{ijs} + z_{jis} = 1 \qquad i \in \mathcal{T},\ s \in \mathcal{S}, \tag{9d}$$

$$\sum_{s \in \mathcal{S}} z_{ijs} = 1, \qquad i, j \in \mathcal{T},\ i \neq j, \tag{9e}$$

$$b_{is_1}^{(h)} + b_{is_1}^{(a)} = 0 \qquad i \in \mathcal{T}, \tag{9f}$$

[1]The source code of the implementation is available under *github.com/janerikhein/RR2timetabling*

$$
\begin{aligned}
& b_{is}^{(h)} + b_{is}^{(a)} \le 1 && i \in \mathcal{T}, s \in \mathcal{S}, && \text{(9g)}\\
& b_{is_k}^{(a)} - b_{is_k}^{(h)} + \sum_{j \in \mathcal{T}} z_{ijs_k} + z_{ijs_{k-1}} = 1 && i \in \mathcal{T}, k \in \{2, \dots, |\mathcal{S}|\}, && \text{(9h)}\\
& z_{ijs} \in \{0,1\} && i, j \in \mathcal{T}, s \in \mathcal{S}, && \text{(9i)}\\
& b_{is}^{(h)}, b_{is}^{(a)} \in \{0,1\} && i \in \mathcal{T}, s \in \mathcal{S}, && \text{(9j)}\\
& \Delta_c \in \mathbb{R}_{\ge 0} && c \in C && \text{(9k)}
\end{aligned}
$$

Constraint (9a) enforces all hard constraints to be satisfied in a feasible solution and constraint (9b) ensures that no team plays itself. Constraint (9c) enforces that the number of teams playing at home equals the number of teams playing away in each round. Constraints (9d) and (9e) model the one-factor and all-different constraints respectively. Breaks are modeled by the constraints (9f)–(9g) as follows. (9f) forbids a break occurring in the first slot and (9g) ensures that no team has a home and away break in the same slot. The break values are then defined by (9h), since the value of $\sum_{j \in \mathcal{T}} z_{ijs_k} + z_{ijs_{k-1}}$ is always between 0 and 2, where the value is 0 if and only if $i$ has an away break in slot $s_k$ and 2 if and only if $i$ has a home break in slot $s_k$.

Capacity and game constraints are added to the model in the same way as in (GAM). Break constraints can also be added analogously on the auxiliary break variables. For each fairness constraint $c = \mathrm{FA}(T, S, ub)$ we add the constraints

$$
\sum_{t \in \mathcal{T}} \sum_{l=1}^{k} z_{its_l} - z_{jts_l} \le ub + \Delta_c, \qquad i, j \in T, k \in \{x : s_x \in S\}, \tag{10}
$$

which limit the difference in home games played by teams $i$ and $j$ after the first $k$ rounds. Furthermore, for each separation constraint $c = \mathrm{SE}(T, lb)$ we first add binary variables $h_{ij}$ indicating whether team $i$ plays the first game against team $j$ at home. Then for each pair of teams $\{i, j\} \subset T$ we add continuous variables $\delta_{ij} > 0$ and the following constraints to the model.

$$
\delta_{ij} - 1 - \sum_{k=1}^{|\mathcal{S}|} k z_{ijs_k} - k z_{jis_k} \ge lb + 2(h_{ij} - 1)|\mathcal{S}| \tag{11a}
$$

$$
\delta_{ij} - 1 + \sum_{k=1}^{|\mathcal{S}|} k z_{ijs_k} - k z_{jis_k} \ge lb - 2h_{ij}|\mathcal{S}| \tag{11b}
$$

Assume that team $i$ plays team $j$ at home in the $k$-th round and away in the $l$-th round. Observe that the sum in (11a) and (11b) is then given by $k - l$, where its absolute value is thus bounded by $|\mathcal{S}|$. Therefore one of the two constraints is always satisfied depending on the value of $h_{ij}$. Then $\Delta_c$ can be defined by

$$
\Delta_c = \sum_{\{i,j\} \subset T} \delta_{ij}.
$$

## 5.2 Finding Feasible Solutions

The first part of our experiments was to compare the performance of the modified PGBA with the integer programming approach for finding feasible start solutions. Implementing the IP approach for finding feasible solutions is straight-forward, as it is sufficient to solve the IP model from the previous section after removing all soft constraints from the instances. However, for implementing the modified PGBA the following considerations had to be made.

**Pattern selection** As the number of feasible patterns for each team can be exponentially large it is crucial to only generate a limited number of patterns for each team. In addition to generating patterns with a limited number of breaks, two other heuristic approaches were considered: generating patterns randomly and generating the best patterns for each team, i.e. patterns with minimal penalties for soft pattern constraints. For the second heuristic a constraint programming model based on the pattern generation model used by Rasmussen [12] was modified to generate all feasible patterns with an overall penalty on soft pattern constraints not exceeding some threshold.

Both of these approaches were tested empirically and showed a worse performance compared with selecting patterns with a limited number of breaks. One issue is that generating the best patterns for each team leads to a much larger number of patterns being added to PSM, where most patterns are only feasible for a small number of teams. This however decreases the quality of the general cuts that are added to PSM during the feasibility checks, as less potential pattern set solutions are cut off. Selecting best patterns for each team could potentially lead to good initial solutions if the number of soft pattern constraints exceeds the number of pattern set and general constraints. This however is not the case in any of the instances provided in the competition.

**Phased tournaments** The neighborhood definition for 2RR that is used in our tabu search method includes moves that do not preserve a phased tournament structure. Two approaches were considered to deal with this issue. First, we considered restricting the neighborhood to moves that preserve a phased tournament structure, as outlined in section 4.2.1. This proved to be too restrictive and did not lead to feasible solutions. Therefore, we allowed the search to violate the phased tournament structure during the search, by considering an unrestricted neighborhood and expressing the phased structure by hard game constraints, which require each pair of teams to play exactly one of their games in the first half of the tournament.

**Parameter estimation** We performed empirical tests for different parameter combinations of the tabu search method. We compared the performance of the method for phase lengths of 100 and 500 iterations and found that the performance for a phase length of 100 iterations proved to be slightly better. This is due to the common occurrence of deadlock situations, where the current solution is very close to feasibility but restricting the neighborhood to moves that change constrained entries leads to no improvements. In those cases the algorithm was usually able to find a feasible solution shortly after applying a diversification phase.

A small number of additional tests were performed with different values for the lengths of tabu tenures and the length of a diversification phase. While the former did not have a meaningful effect effect on the performance, we found that applying a very small number of iterations in a diversification phase is sufficient to escape the deadlock situations. Therefore we fixed the length of tabu tenures and diversification phases to 20 and 5 iterations respectively.

**Time limits** A time limit of 3 hours was enforced for both the PGBA and the IP approach. An additional time limit of 20 minutes was enforced for the game assignment subproblem GAM, where the model was terminated after 20 minutes only if an infeasible schedule was already found.

Table 5.1 shows the computational results for the comparison of the modified PGBA and the IP approach for finding a feasible start solution, with instances for which neither of the two approaches managed to find a solution being omitted. For the modified PGBA the table contains the runtimes of the three steps of the algorithm. The first column (PSM) denotes the runtime for finding a feasible pattern set and the second column (GAM) shows the time spent in finding a game allocation with minimal deviations of hard constraints for the pattern-set obtained by PSM. Finally, if the schedule found by GAM is not optimal, the third column (TS) shows the time spent resolving all conflicts in the schedule using our tabu search method.

The computational results show that neither the IP or PGBA is a superior method for finding feasible start solutions, as both approaches find solutions for instances where the other fails to find any. The tabu search method proves to be suitable for resolving conflicts if the solution found by GAM is close to feasibility. However, if the solution found by GAM violates a large number of general constraints the tabu search methods either fails to find feasible solutions or performs significantly worse than the IP approach. A potential improvement would be to generate multiple pattern sets

We observe that both approaches either failed to find a solution in 3 hours or obtained a solution in less than an hour. This presumes the existence of specific instance characteristics that make the instances much harder to solve, as the sizes of the instances itself are evenly distributed. We analyzed whether the distribution of constraint types in the instances is correlated to the performance of the approaches and found that instances for which both approaches failed to find a solution generally have a higher percentage of hard general constraints. As seen in figure 5.1, the IP approach performs better for these instances, whereas the PGBA performs better on large instances with a high percentage of pattern and pattern set constraints.

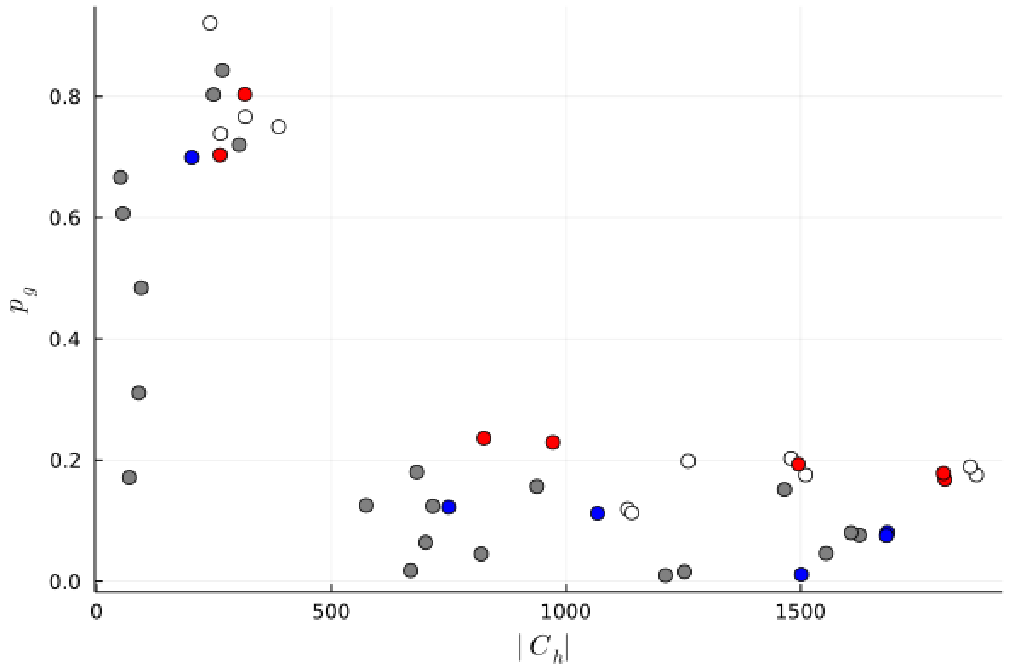


Figure 5.1: Performance comparison of the IP and the PGBA depending on the number of hard constraints $|C_h|$ and the percentage of hard general constraints $p_g$. Blue and red dots correspond to instances only solved by the PGBA or IP respectively, while gray dots correspond to instances solved by both approaches. White dots correspond to instances for which no solution was found.

| Instance | PGBA | | | | IP |
|---|---|---|---|---|---|
| | PSM | GAM | TS | total | |
| early-01 | 19.77 | 0.89 | 0.45 | 21.11 | – |
| early-02 | 163.47 | 2.09 ∗ | | 165.56 | – |
| early-03 | 61.37 | 734.51 | 0.50 | 796.38 | 17.13 |
| early-06 | – | | | – | 97.86 |
| early-07 | 119.42 | 206.44 ∗ | | 325.86 | – |
| early-08 | 5.84 | 0.56 ∗ | | 6.40 | 20.45 |
| early-09 | 15.32 | 1.02 | 0.41 | 16.75 | 21.73 |
| early-11 | 23.76 | 6.32 | 30.08 | 60.16 | – |
| early-12 | 4864.7 | 170.30 | – | – | 151.30 |
| early-13 | 159.24 | 2.72 | 18.99 | 180.95 | 839.21 |
| early-14 | 8.98 | 0.37 | 0.44 | 9.35 | 39.35 |
| early-15 | 8.47 | 10.33 ∗ | | 18.80 | 204.00 |
| middle-04 | – | | | – | 32.35 |
| middle-05 | 45.84 | 33.65 ∗ | | 79.49 | 45.01 |
| middle-06 | 28.35 | 1200 † | 2326.10 | 3554.45 | 81.44 |
| middle-07 | 5.22 | 5.04 | 958.31 | 968.57 | 55.02 |
| middle-08 | 7.98 | 0.29 | 15.72 | 23.99 | 65.14 |
| middle-09 | 8.85 | 0.43 ∗ | | 9.28 | 41.51 |
| middle-11 | – | | | – | 235.25 |
| middle-12 | – | | | – | 113.94 |
| middle-13 | 7.23 | 7.21 | 40.20 | 54.64 | 89.72 |
| middle-14 | 4.59 | 0.57 ∗ | | 5.16 | – |
| middle-15 | 15.77 | 1.83 | 0.46 | 18.06 | 37.79 |
| late-01 | 6.93 | 185.27 ∗ | | 192.20 | 97.21 |
| late-03 | 15.20 | 0.56 ∗ | | 15.76 | 19.13 |
| late-04 | 15.20 | 0.56 | 3.42 | 19.18 | 30.43 |
| late-06 | 96.18 | 488.95 | – | – | 140.23 |
| late-07 | 47.44 | 1.07 | 1.59 | 50.01 | 201.55 |
| late-08 | – | | | – | 16.64 |
| late-09 | 6.79 | 0.80 ∗ | | 7.59 | 64.30 |
| late-12 | 38.36 | 3.16 ∗ | | 41.52 | – |
| late-13 | 1.41 | 19.46 | 321.17 | 342.04 | 136.18 |
| late-14 | 10.53 | 2.27 ∗ | | 12.80 | 119.86 |
| late-15 | 10.52 | 0.34 | 0.83 | 11.69 | 35.24 |

∗ relaxation solved to optimality and feasible schedule found
† relaxation reached time limit but infeasible schedule found
– time limit reached and no solution found

Table 5.1: Performance comparison of the modified PGBA and the IP approach for finding a feasible start solution

## 5.3 Optimal Solutions

In the second part of our experiments we analyze how the tabu search methods compares with the integer program for finding optimal solutions. For the integer program, the starting solution is provided as a MIP start to the solver. However, to ensure a fair comparison between the two approaches we set the solver to utilize only one thread, as the implementation of our tabu search method does not utilize multithreading. We then solve all instances for which a feasible solution was obtained with a time limit of 8 hours for each instance.

Table 5.2 compares the best solutions found by the two approaches with the best solutions found in the competition. Except for instance late-04, all experiments reached the time limit of 8 hours and were terminated prematurely. Note that the tabu search method performs better than the IP approach on all instances. However, the best solutions obtained by the tabu search method are in many instances significantly worse compared with the optimal solutions found in the competition. It is unclear, whether this is due to our method or due to the time limit that was enforced. It is possible, that further improvements could be found with an extended time limit.

Another potential for improvement would be finding more suitable search parameters and adjusting the neighbor selection strategy. We analyzed the effect different search phase lengths have on the performance by again considering phase lengths of 100 and 500 iterations. The effect proves to be minimal, with the overall performance for longer phase lengths being slightly better. We also considered two strategic oscillation strategies based on the ideas by Agagnostopoulos et al. [9] and Gaspero and Schaerf [10]. The core idea is to consider feasible and infeasible search configurations without any restrictions and to use an adaptive objective function to guide the search towards desirable solutions. To that end, an additional penalty $p_h$ is introduced for hard constraints, which is dynamically changed throughout the search. In the first approach $p_h$ is multiplied by a constant $\alpha > 0$ after each iteration if the new configuration is infeasible, and divided by $\alpha$ if the new configuration is feasible. The second approach extends this simple strategy by using a *shifting* mechanism, where, for a given $k$, the penalty $p_h$ is multiplied by $\alpha$ if no feasible solution have been found in the last $k$ iterations, and divided by $\alpha$ if the solution was feasible in all of the last $k$ iterations. However, both of these approaches lead the search to quickly diverge from the feasible search space and were not able to find any new feasible solutions.

| Instance | OPT | TS | | IP |
|---|---|---|---|---|
| | | phase 100 | phase 500 | |
| early-01 | 362 | 642 | 593 | 1437 |
| early-02 | 145 | 443 | 405 | 626 |
| early-03 | 992 | 1539 | 1203 | 3878 |
| early-06 | 3325 | 5261 | 4953 | 5281 |
| early-07 | 4763 | 10243 | 9584 | 11142 |
| early-08 | 1051 | 1634 | 1623 | 2302 |
| early-09 | 108 | 803 | 507 | 5447 |
| early-11 | 4426 | 8602 | 8611 | 10812 |
| early-12 | 380 | 1910 | 1907 | 2025 |
| early-13 | 121 | 380 | 352 | 1361 |
| early-14 | 4 | 288 | 249 | 4129 |
| early-15 | 3362 | 4138 | 4161 | 4687 |
| middle-04 | 7 | 86 | 88 | 117 |
| middle-05 | 413 | 317 | 309 | 334 |
| middle-06 | 1120 | 2925 | 2981 | 3395 |
| middle-07 | 1783 | 4997 | 4851 | 7194 |
| middle-08 | 129 | 302 | 341 | 933 |
| middle-09 | 450 | 1465 | 745 | 3665 |
| middle-11 | 2446 | 4393 | 3856 | 4993 |
| middle-12 | 911 | 1524 | 1469 | 5132 |
| middle-13 | 252 | 1851 | 2626 | 4918 |
| middle-14 | 1172 | 1630 | 1585 | 3437 |
| middle-15 | 485 | 1608 | 1367 | 5999 |
| late-01 | 1922 | 2713 | 2693 | 3212 |
| late-03 | 2369 | 3123 | 2803 | 5649 |
| late-04 | 0 | 0 | 0 | 231 |
| late-06 | 923 | 1514 | 1499 | 2320 |
| late-07 | 1558 | 2966 | 2919 | 2382 |
| late-08 | 934 | 1725 | 1734 | 3685 |
| late-09 | 563 | 1549 | 800 | 3525 |
| late-12 | 3428 | 5861 | 5908 | 9991 |
| late-13 | 1820 | 2840 | 2910 | 8109 |
| late-14 | 1202 | 1830 | 1831 | 3798 |
| late-15 | 20 | 400 | 160 | 3090 |

Table 5.2: Comparison of the best solutions found in the competition (OPT), the best solutions found by Tabu Search (TS) for phase lengths of 100 and 500 iterations and the best solutions found by the integer program (IP).

# Chapter 6

# Conclusion

We considered the problem of scheduling a time-constrained round-robin tournament under a large variety of hard and soft constraints. The goal of our work was to combine a decomposition approach with local search methods to formulate a general purpose solver, that can be used to effectively schedule many real-life sport tournaments.

We considered various constraint types that occur in real-life sport tournaments and partitioned them into three classes: pattern and pattern set constraints which constrain only the home/away statuses of the teams and general constraints which can also constrain the game allocation of the schedule. Our proposed method incorporates a tabu search heuristic in a common decomposition scheme, in which at first a feasible pattern set satisfying all hard pattern and pattern set constraints is generated. The decomposition approach was modified to generate solutions that are close to feasibility and only violate a small number of hard general constraints. Then, a tabu search heuristic was used to resolve all conflicts and find improving solutions. We tested our approach on all problem instances provided in the *21.International Timetabling Competition*.

The computational results for finding feasible start solutions are ambivalent. On one hand, our approach performs similar or sometimes significantly worse then the straightforward IP approach for many problem instances. On the other hand, our approach manages to find solutions in a matter of minutes for some of the large instances, where the IP fails to find a solution in three hours. We find, that problem instances with a large number of general constraints generally lead to a worse performance in both approaches. It is questionable, whether the common decomposition approach is suitable for these types of constraints. Compared with the IP approach, our tabu search heuristic however performs significantly better at improving a given start solution, as the IP often fails to find meaningful improvements.

Further experiments would be necessary to potentially produce solutions which are comparable with the best solutions found in the competition. In the future we would like to analyze whether the decomposition approach can be further improved for finding feasible solutions and consider alternative start heuristics. Another interesting consideration would be to somehow combine the decomposition approach with local search for improving feasible start solutions.

# Bibliography

[1] R. V. Rasmussen and M. A. Trick, "Round robin scheduling – a survey," *European Journal of Operational Research*, vol. 188, pp. 617–636, 2008.

[2] D. Van Bulck, D. Goossens, J. Belien, and M. Davari, "The fifth international timetabling competition (itc 2021): Sports timetabling," in *MathSport International 2021*, pp. 117–122, University of Reading, 2021.

[3] D. Van Bulck, D. Goossens, J. Schönberger, and M. Guajardo, "Robinx: A three-field classification and unified data format for round-robin sports timetabling," *European Journal of Operational Research*, vol. 280, no. 2, pp. 568–580, 2020.

[4] D. d. Werra, "Scheduling in sports," in *Annals of Discrete Mathematics (11)* (P. Hansen, ed.), vol. 59 of *North-Holland Mathematics Studies*, pp. 381–395, North-Holland, 1981.

[5] A. Rosa and W. D. Wallis, "Premature sets of 1-factors or how not to schedule round robin tournaments," *Discrete Applied Mathematics*, vol. 4, no. 4, pp. 291–297, 1982.

[6] M. A. Trick, "Integer and constraint programming approaches for round-robin tournament scheduling," in *Practice and Theory of Automated Timetabling IV* (E. Burke and P. De Causmaecker, eds.), pp. 63–77, Springer Berlin Heidelberg, 2003.

[7] M. Henz, T. Müller, and S. Thiel, "Global constraints for round robin tournament scheduling," *European Journal of Operational Research*, vol. 153, no. 1, pp. 92–101, 2004.

[8] D. Van Bulck, D. R. Goossens, and F. C. Spieksma, "Scheduling a non-professional indoor football league: a tabu search based approach," *Annals of Operations Research*, vol. 275, no. 2, pp. 715–730, 2019.

[9] A. Anagnostopoulos, L. Michel, P. V. Hentenryck, and Y. Vergados, "A simulated annealing approach to the traveling tournament problem," vol. 9, no. 2, pp. 177–193.

[10] L. Di Gaspero and A. Schaerf, "A composite-neighborhood tabu search approach to the traveling tournament problem," *Journal of Heuristics*, vol. 13, no. 2, pp. 189–207, 2007.

[11] M. A. Trick, "A schedule-then-break approach to sports timetabling," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 242–253, Springer, 2000.

[12] R. V. Rasmussen and M. A. Trick, "A benders approach for the constrained minimum break problem," *European Journal of Operational Research*, vol. 177, no. 1, pp. 198–213, 2007.

[13] J.-C. Régin, "A filtering algorithm for constraints of difference in csps," in *AAAI*, vol. 94, pp. 362–367, 1994.

[14] R. Miyashiro, H. Iwasaki, and T. Matsui, "Characterizing feasible pattern sets with a minimum number of breaks," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 78–99, Springer, 2002.

[15] J. N. Hooker and G. Ottosson, "Logic-based benders decomposition," *Mathematical Programming*, vol. 96, no. 1, pp. 33–60, 2003.

[16] R. V. Rasmussen, "Scheduling a triple round robin tournament for the best danish soccer league," *European Journal of Operational Research*, vol. 185, no. 2, pp. 795–810, 2008.

[17] M. Gendreau and J.-Y. Potvin, "Metaheuristics in combinatorial optimization," *Annals of Operations Research*, vol. 140, no. 1, pp. 189–213, 2005.

[18] D. Costa, "An evolutionary tabu search algorithm and the NHL scheduling problem," vol. 33, no. 3, pp. 161–178.

[19] H. H. Hoos and E. Tsang, "Local search methods," in *Handbook of Constraint Programming* (F. Rossi, P. van Beek, and T. Walsh, eds.), vol. 2 of *Foundations of Artificial Intelligence*, pp. 135–167, Elsevier, 2006.

[20] F. Glover and E. Taillard, "A user's guide to tabu search," *Annals of operations research*, vol. 41, no. 1, pp. 1–28, 1993.

[21] H. H. Hoos and T. Stützle, "Sls methods," in *Stochastic Local Search* (H. H. Hoos and T. Stützle, eds.), The Morgan Kaufmann Series in Artificial Intelligence, pp. 61–112, San Francisco: Morgan Kaufmann, 2005.