Handling fairness issues in time-relaxed tournaments with availability constraints

David Van Bulck, Dries Goossens

Faculty of Economics and Business Administration
Ghent University, Ghent, Belgium

Abstract

Sports timetables determine who will play against whom, where, and on which time slot. In contrast to time-constrained sports timetables, time-relaxed timetables utilize (many) more time slots than there are games per team. This offers time-relaxed timetables additional flexibility to take into account venue availability constraints, stating that a team can only play at home when its venue is available, and player availability constraints stating that a team can only play when its players are available. Despite their flexibility, time-relaxed timetables have the drawback that the rest period between teams' consecutive games can vary considerably, and the difference in the number of games played at any point in the season can become large. Besides, it can be important to timetable home and away games alternately. In this paper, we first establish the computational complexity of time-relaxed timetabling with availability constraints. Naturally, when one also incorporates fairness objectives on top of availability, the problem becomes even more challenging. We present two heuristics that can handle these fairness objectives. First, we propose an adaptive large neighborhood method that repeatedly destroys and repairs a timetable. Second, we propose a memetic algorithm that makes use of local search to schedule or reschedule all home games of a team. For numerous artificial and real-life instances, these heuristics generate high-quality timetables using considerably less computational resources compared to integer programming models solved using a state-of-the-art solver.

Keywords: Time-relaxed sports scheduling, Fix-and-optimize, Evolutionary algorithm, Round-robin tournament, Bipartite tournament, Complexity theory

1. Introduction

The success of a sports competition heavily depends on its timetable, which defines who will play whom, when, and where. Sports timetables, also (imprecisely [47, 48, 58]) called schedules, need to be fair and organizationally practical. Many of the organizational requirements concern the scarce availability of venues and teams [47]. This is reflected in the literature by two types of availability constraints (e.g. [31, 45, 47]). First, venue availability constraints restrict a team from playing at home on a given time slot, typically because the team's venue is already in use for another activity (e.g. a concert). Second, player availability constraints state that a team cannot play at all (i.e. neither home, nor away) on a given time slot. In non-professional competitions, this type of constraint occurs when players should be able to combine their sport with work and family. In professional competitions, this constraint occurs when a team has a game in an international competition, and so it cannot be scheduled for its national tournament.

Most of the sports timetabling literature can be categorized either as time-constrained or as time-relaxed. Time-constrained timetables utilize the minimum number of time slots required to play all games. Time-relaxed timetables, on the other hand, utilize (many) more time slots than games per team and are therefore particularly well suited in settings where availability constraints are prominent. The most extreme case example is asynchronous round-robin timetabling where no two games take place at the same time (see [53]). This tournament structure occurs when there is only one venue, as is the case in the top-tier national football league of Gibraltar, or when fans need to be able to watch all games live. Despite the large amount of contributions for sports timetabling (see [29, 58] for an overview) only a small minority of these studies deal with time-relaxed timetabling (e.g. [5, 18, 31, 34, 47, 59]). This is somewhat surprising since several well known competitions such as the National Basketball Association league (NBA, e.g. [6]) and the North American National Hockey League (NHL, e.g. [12]) are in fact time-relaxed. Moreover, most non-professional competitions use a time-relaxed format since this offers more flexibility to incorporate player and venue availability.

Due to their temporal structure, time-relaxed timetables may face fairness issues that do not occur in time-constrained timetabling. Suksompong [53] identifies three issues. First, the rest time between consecutive games of a team can vary substantially, which can result in congested periods or long periods without any game. Second, the difference in the rest time allocated to opponents in a game can disadvantage the less rested team (see also [2, 3]). Third, the difference in the number of games played per team after each time slot can vary considerably, which causes tournament rankings to become inaccurate. In addition, time-relaxed timetables face fairness issues

Email addresses: `david.vanbulck@ugent.be` (David Van Bulck), `dries.goossens@ugent.be` (Dries Goossens)

This is the peer-reviewed author-version of `https://doi.org/10.1016/j.cor.2019.104856`, *published in Computers & Operations Research.*

which also occur in time-constrained timetabling. As an example, Bao and Trick [5] study how to minimize the total distance traveled in time-relaxed timetabling. Knust [31] minimizes the total number of breaks in a time-relaxed timetable; a team has a break when it plays two consecutive games with the same home-away status, irrespective of the number of time slots between these two games. Kyngäs et al. [34] simultaneously minimize the distance traveled and the number of breaks in the timetable.

The contributions and remainder of this paper are as follows. First, Section 2 formalizes the time-relaxed availability constrained tournament problem and gives an overview of the related literature. Subsequently, Section 3 presents sufficient conditions for the existence of a feasible timetable and proves that it is $\mathcal{NP}$-complete to decide whether a feasible timetable exists as soon as player or venue availability is considered. This settles the complexity status for several problems studied in the literature that involve availability constraints (e.g. [31, 46, 47, 59]). In the second part of this paper, we additionally take into account fairness issues. Section 4 describes how to measure fairness and provides several integer programming (IP) formulations to construct fair timetables. Since these IP formulations consume a considerable amount of computational resources, Section 5 introduces two heuristic algorithms. First, we propose an adaptive large neighborhood search method that repeatedly destroys and repairs a timetable. Second, we propose a memetic algorithm that makes use of local search to schedule or reschedule all home games of a team. Section 6 compares the performance of the two heuristic algorithms with the IP formulations solved using a state-of-the-art IP solver for a set of real-life and artificial instances. Finally, Section 7 concludes the paper.

2. Problem description and related work

In the time-relaxed availability constrained k-tournament problem (RAC-kTP), the input consists of a set of *time slots* S, a set of *teams* T, and a multiset of *games* M. Each game in M consists of an ordered pair (i, j) in which $i \in T$ is the home team providing the venue where the game is played, and $j \in T \setminus \{i\}$ is the away team. If we denote with $m_{i,j}$ the multiplicity of game $(i, j) \in M$, a tournament is called a k-tournament if $\max_{i,j \in T: i \neq j}\{m_{i,j} + m_{j,i}\} = k$. For simplicity, we denote with m_i the total number of games team $i \in T$ plays, i.e. $m_i = \sum_{j \in T\setminus\{i\}}(m_{i,j} + m_{j,i})$. Each team $i \in T$ also provides a venue availability set $H_i \subseteq S$ containing all time slots during which i's venue is available, and a player availability set A_i containing all time slots during which i's players are available. Since a team can only play (at home or away) when its players are available, we assume without loss of generality that $H_i \subseteq A_i$ for each $i \in T$. This makes that a team can play at home on all time slots in H_i, and that it can play away on all time slots in A_i. RAC-kTP consists of finding a feasible timetable, i.e. an assignment of games to time slots such that:

(C1) each game in M is mapped to exactly one time slot $s \in S$,

(C2) the venue availability H_i with $i \in T$ is respected (i.e. no game (i, j) is planned on a time slot $s \notin H_i$),

(C3) the player availability A_i with $i \in T$ is respected (i.e. no game (i, j) or (j, i) is planned on a time slot $s \notin A_i$), and

(C4) each team plays at most one game per time slot $s \in S$.

Van Bulck et al. [59] propose a three-field notation to describe a sports timetabling problem by means of the tournament format, the constraints in use, and the objective. In terms of this classification framework, RAC-kTP corresponds with the notation NRR, R, ∅ | CA1 | ∅. The first field denotes that $m_{i,j}$ may take any arbitrary value for each pair $i, j \in T, i \neq j$, the tournament is time-relaxed ('R'), and there are no symmetry requirements ('∅'). The second field denotes that all constraints are special cases of capacity constraint CA1. The third field denotes that RAC-kTP is a constraint satisfaction problem, i.e. there is no objective function ('∅').

A popular format of sports timetables is the so-called k round-robin tournament in which teams play each other exactly k times. Numerous real-life examples of round-robin tournaments can be found in sports such as basketball [38], table-tennis [31, 47], and indoor football [59]. In the time-relaxed availability constrained k round-robin tournament problem (RAC-kRRT), $m_{i,j} + m_{j,i}$ equals k for each $i, j \in T$ with $i \neq j$.

Alternatively in a k bipartite round-robin tournament, the teams can be partitioned into two equally sized groups $T = T_1 \cup T_2$ with $|T_1| = |T_2|$ such that each team meets each other team from the competing group exactly k times. Bipartite tournaments for example occur in multi-conference competitions, also called divisions or leagues, in which teams play inter-conference games against teams outside their conference. Examples include the Nippon Professional Baseball league that organizes inter-conference play near the middle of the season [27], and pre-2013 Major League Baseball inter-conference play that was held before the start of the regular season [55]. In the time-relaxed availability constrained k bipartite tournament problem (RAC-kBT), $m_{i,j} + m_{j,i}$ equals k if $i \in T_1$ and $j \in T_2$, and 0 otherwise. In the time-relaxed availability constrained k *partial* bipartite tournament problem (RAC-kPBT), $m_{i,j} + m_{j,i}$ is at most k if $i \in T_1$ and $j \in T_2$, and 0 otherwise. Although it is easy to construct round-robin (e.g. [14]) and bipartite round-robin tournaments (e.g. [1]), Section 3 proves that timetabling becomes $\mathcal{NP}$-complete once player or venue availability is considered.

Availability constraints in the context of time-relaxed timetables have been studied by few researchers. Bean and Birge [6] reduce the total travel distance in the time-relaxed NBA league in which venue availability needs to be considered. Costa [12] studies the NHL in which teams provide a list of time slots in which they can play a home game and a list of time slots in which they cannot play at all. Additionally, teams request an even distribution of games throughout the season, travel distance must be minimized, and long sequences of consecutive away games are to be avoided. To tackle this problem, Costa [12] proposes a genetic algorithm in which the mutation phase is replaced by tabu search; an adaptive large neighborhood search method was recently proposed by Bueno [9]. Kostuk and Willoughby [33] discuss the Canadian football league in which participants submit specific time slots, known as stadium blocks, during which teams cannot play at home because a competing event may stifle ticket sales or the multiple-tenant facilities are used for other activities. Moreover, the timetable must provide the team with an appropriate number of days-off between consecutive games to avoid competitive imbalance. Schönberger et al. [47] propose a memetic algorithm backed by a constraint propagation method to timetable a non-professional time-relaxed double round-robin tournament with availability constraints. Games should be evenly distributed over the season, and teams request a minimal rest time between two consecutive games. Moreover, Schönberger et al. [47] provide evidence of the limited suitability of constraint programming to construct time-relaxed timetables. Knust [31] additionally restricts the assignments of some games to a subset of time slots, and teams should play home and away games alternately (i.e. breaks should be avoided). Knust [31] models the problem as a multi-mode resource-constrained project scheduling problem, for which an IP formulation and a two-stage heuristic solution algorithm are proposed, involving local search and a genetic algorithm. Van Bulck et al. [59] consider a similar problem faced by a non-professional indoor football competition. They propose a tabu search based algorithm employing a novel move operator that solves a transportation problem to schedule or reschedule all home games of a team. Finally, Schauz [45] examines the minimum number of time slots needed to construct a timetable for the availability constrained tournament problem without venue availability constraints (i.e. $H_i = A_i, \forall i \in T$). Besides, Schauz [45] investigates how this number changes when player availability is not known in advance.

This work extends the existing literature in two ways. First, we show for the first time the computational complexity of time-relaxed tournament timetabling problems with availability constraints. Second, we generalize fairness measures used in asynchronous timetabling (see [53]) and propose exact models and two heuristic algorithms to incorporate these measures in the timetabling process. Practitioners may use these algorithms to construct fair timetables. An extensive set of computational experiments not only shows the performance of our algorithms, but also reveals how the presence of availability constraints impacts the fairness of timetables.

3. Theoretical results

This section provides theoretical results with regard to RAC-kTP and its variants. First, we show that RAC-kTP is a special case of list-edge coloring. This allows us to derive sufficient conditions for the existence of a feasible timetable (Sec. 3.1). Next, we settle the computational complexity of RAC-kTP (Sec. 3.2).

3.1. Feasibility results

Consider the problem of list-edge coloring (LECOL).
LECOL *Instance*. A multigraph G with a set of vertices V, a set of edges E, a set of colors C, and for each edge $e \in E$ a list of colors $L(e) \subseteq C$. *Output*. A proper list-edge coloring $f : E \rightarrow C$ in G, that is a mapping from each edge $e \in E$ to a color in $L(e)$ such that no two adjacent edges get the same color.

The list-edge chromatic index $\chi'_l(G)$ denotes the minimum list size that guarantees a list-edge coloring in G. In the special case that $L(e) = C$ for all $e \in E$, the problem is known as the edge coloring problem. The smallest number of colors needed to construct a proper edge coloring in G is the edge chromatic index $\chi'(G)$. Denote with $\delta(v)$ the degree of vertex v, with $\Delta(G)$ the maximal vertex degree, and with $\mu(G)$ the maximum number of edges joining any two vertices. Then, Vizing's theorem for multigraphs [60] states that $\chi'(G) \leq \Delta(G) + \mu(G)$. The list-edge coloring conjecture states that $\chi'_l(G) = \chi'(G)$, and was independently proposed by several researchers (see e.g. [26] for an overview).

It is well known that a sports timetable without availability constraints can be constructed via edge coloring techniques (e.g. [11, 28, 29, 45]). Lemma 1 shows the relationship between RAC-kTP and list-edge coloring.

Lemma 1. *RAC-kTP is a special case of list-edge coloring.*

Proof. We prove the lemma by restricting LECOL to RAC-kTP. For any instance of RAC-kTP, we create an instance of LECOL as follows. First, we construct a set of vertices V containing vertex v_i for each team $i \in T$, and a set of colors C containing color c_s for each time slot $s \in S$. Then, we construct a set of edges E containing an edge $\{v_i, v_j\}$ for each game $(i, j) \in M$ and we set $L(\{v_i, v_j\}) = \{c_s : s \in H_i \cap A_j\}$. This results in the multigraph $G(V, E)$ in which there are exactly $m_{i,j} + m_{j,i}$ edges between two vertices $v_i, v_j \in V$, and the vertex degree $\delta(v_i)$ corresponds to m_i. Observe that $\mu(G) = k$, and that G is a complete graph in case of RAC-kRRT and a bipartite graph in case of RAC-kBT or RAC-kPBT.

Now, we show that the RAC-kTP instance is feasible if and only if the corresponding LECOL instance is feasible. Suppose first that we have a feasible timetable for the RAC-kTP instance. For each game $(i, j) \in M$ scheduled on $s \in S$, we select an uncolored edge $\{v_i, v_j\}$ for which $L(\{v_i, v_j\}) = \{c_p : p \in H_i \cap A_j\}$ and we color $\{v_i, v_j\}$ with c_s. This results in a proper list-edge coloring since it follows from *(C1)* that all edges are colored, from *(C2)*, *(C3)* and the edge-selection strategy that only feasible colors are used, and from *(C4)* that adjacent edges get different colors.

Conversely, assume that we are given a proper list-edge coloring of G. For each edge $\{v_i, v_j\} \in E$ colored with $c_s \in C$, we then select an unscheduled game $(i, j) \in M$ if $L(\{v_i, v_j\}) = \{c_p : p \in H_i \cap A_j\}$ or $(j, i) \in M$ if $L(\{v_i, v_j\}) = \{c_p : p \in H_j \cap A_i\}$, and we schedule this game on time slot s. This makes that all games are timetabled *(C1)*. Moreover it follows from the game-selection strategy and the construction of the edge-color lists that availability constraints *(C2)* and *(C3)* are respected. Finally, each team plays at most once per time slot *(C4)* since all edges incident to the same vertex get different colors. □

Corollary 1. *An instance of RAC-kTP has a feasible solution that can be constructed in polynomial time if for each game* $(i, j) \in M$ *it holds that* $|H_i \cap A_j| \geq \max\{m_i, m_j\} + \lfloor 1/2 \min\{m_i, m_j\} \rfloor$.

Proof. Recall from Lemma 1 that RAC-kTP is a special case of list-edge coloring in a multigraph $G(V, E)$. In this graph team $i \in T$ is associated with vertex $v_i \in V$, time slot $s \in S$ is associated with color $c_s \in C$, and game $(i, j) \in M$ is associated with an edge $\{v_i, v_j\} \in E$ carrying colors $L(\{v_i, v_j\}) = \{c_p : p \in H_i \cap A_j\}$. Also recall that $\delta(v_i) = m_i$ for all $i \in T$. Borodin et al. [8] show that G can be list-edge colored in polynomial time if $|L(\{v_i, v_j\})| \geq \max\{\delta(v_i), \delta(v_j)\} + \lfloor 1/2 \min\{\delta(v_i), \delta(v_j)\}\rfloor, \forall \{v_i, v_j\} \in E$. □

If the list-edge coloring conjecture holds, that is if $\chi'_l(G) = \chi'(G)$, the right-hand side in Corollary 1 can be changed to $\max_{i\in T} m_i + k$. Indeed, by definition it suffices that $L(e) \geq \chi'_l(G), \forall e \in E$. Moreover, Vizing's theorem for multigraphs [60] guarantees that $\chi'(G) \leq \Delta(G) + \mu(G)$. Galvin [24] proves that the list-edge coloring conjecture holds for bipartite multigraphs for which we derive an even stronger result.

Corollary 2. *An instance of RAC-kPBT or RAC-kBT has a feasible solution if for each game* $(i, j) \in M$ *it holds that* $|H_i \cap A_j| \geq \max\{m_i, m_j\}$.

Proof. From Lemma 1, it follows that RAC-kPBT or RAC-kBT is a special case of list-edge coloring in a bipartite multigraph $G(V_1, V_2; E)$. In this graph team $i \in T_1$ ($j \in T_2$) is associated with vertex $v_i \in V_1$ ($v_j \in V_2$), time slot $s \in S$ is associated with color $c_s \in C$, and game $(i, j) \in M$ is associated with an edge $\{v_i, v_j\} \in E$ carrying colors $L(\{v_i, v_j\}) = \{c_p : p \in H_i \cap A_j\}$. Also recall that $\delta(v_i) = m_i$ for all $i \in T$. Borodin et al. [8] show that bipartite graphs can be list-edge colored if $|L(\{v_i, v_j\})| \geq \max\{\delta(v_i), \delta(v_j)\}, \forall \{v_i, v_j\} \in E$. □

Corollary 3. *RAC-1RRT has a feasible solution if for each game* $(i, j) \in M$ *it holds that* $|H_i \cap A_j| \geq |T|$.

Proof. From Lemma 1, it follows that RAC-1RRT is a special case of list-edge coloring in a complete simple graph $K_{|T|}(V, E)$. In this graph team $i \in T$ is associated with vertex $v_i \in V$, time slot $s \in S$ is associated with color $c_s \in C$, and game $(i, j) \in M$ is associated with an edge $\{v_i, v_j\} \in E$ carrying colors $L(\{v_i, v_j\}) = \{c_p : p \in H_i \cap A_j\}$. By definition, the corresponding LECOL instance is feasible if $L(e) \geq \chi'_l(G), \forall e \in E$. Häggkvist and Janssen [26] show that $\chi'_l(K_{|T|}) \leq |T|$. □

Without player availability, this means that a feasible solution for RAC-1RRT exists if the venue availability set of each team contains at least $|T|$ time slots. Similarly, if venues are always available, i.e. $H_i = A_i, \forall i \in T$, this means that a feasible solution exists if the player availability set of each team contains at least $(|S| + |T|)/2$ time slots. If the list-edge coloring conjecture holds and $|T|$ is even, the lower bound improves to $|T| - 1$. This is for example the case for $|T| = p + 1$, with p any odd prime number [44]. Moreover, we can then generalize the corollary to RAC-kRRT by requiring that $|H_i \cap A_j| \geq k|T|, \forall i \in T$. This follows from Vizing's theorem for multigraphs [60] stating that $\chi'(K_{|T|}) \leq \Delta(K_{|T|}) + \mu(K_{|T|}) = k(|T| - 1) + k$.

3.2. Complexity results

In the remainder of this section, we refer with RAC-kTP and its variants to the decision version of the original problem, i.e. does a feasible timetable exist? First, in the special case of RAC-kPBT, we show that answering this question is $\mathcal{NP}$-complete by showing a reduction from the classic class-lecturer timetabling problem. To the best of our knowledge, this is the first proven relation between course timetabling and sports timetabling. Subsequently, we show that RAC-kBT is $\mathcal{NP}$-complete by showing a reduction from completing a Latin square (that is an $m \times m$ array filled with symbols $\{1, \ldots, m\}$ in such a way that that each row and each column contains every symbol exactly once). Finally, we show a reduction from bipartite tournaments to round-robin tournaments to prove that RAC-kRRT is also $\mathcal{NP}$-complete. In the remainder of this section, we say that venues are always available if for each team $i \in T$ it holds that $H_i = A_i$. In addition, a team $i \in T$ is called tight if $|A_i| = m_i$, i.e. i must play a game whenever its players are available.

Theorem 1. *RAC-kPBT is* $\mathcal{NP}$*-complete, even if* $k = 1$*, venues are always available, teams in* T_2 *are always available, teams in* T_1 *are tight, and there are only three time slots or three teams in the second group* ($|S| = 3$ *or* $|T_2| = 3$).

Proof. We prove the theorem by presenting a reduction from restricted class-lecturer timetabling (RTT) [21].

RTT. *Instance*: A set of time slots P, a set of lecturers L where each lecturer $l \in L$ has lecturer availability set $A_l \subseteq P$, a set of classes C where each class $c \in C$ has class availability set $A_c \subseteq P$, and for each pair $(l, c) \in L \times C$ there is an integer $R_{l,c}$ that indicates the number of time slots lecturer l needs to teach class c. *Question*: Is there a timetable, i.e. a mapping $f : L \times C \times P \rightarrow \{0, 1\}$ with $f(l, c, p) = 1$ whenever lecturer l meets class c on time slot p and 0 otherwise, such that: (i) lecturer l teaches class c exactly $R_{l,c}$ times, (ii) the lecturer availability A_l with $l \in L$ is respected, (iii) the class availability A_c with $c \in C$ is respected, (iv) each lecturer teaches at most one class per time slot, and (v) each class follows at most one lecture per time slot.

For any instance of RTT, we let $S = P$ and add a lecturer team $t_l \in T_1$ for each lecturer $l \in L$ and a class team $t_c \in T_2$ for each class $c \in C$. The availability of lecturer team t_l (class team t_c) corresponds to the availability of lecturer l (class c) in the input of RTT. For all teams, venues are always available ($H_i = A_i, \forall i \in T$). Furthermore, we add $R_{l,c}$ games between home team t_l and away team t_c, i.e. $m_{t_l,t_c} = R_{l,c}, \forall l \in L, c \in C$. Finally, to make sure that $|T_1| = |T_2|$, we can always add dummy teams in T_1 or in T_2 that do not have to play any game.

Now, we show that the RTT instance is feasible if and only if the RAC-kPBT instance is feasible. Suppose first that we have a feasible timetable for the RTT instance. We can then construct a feasible timetable for the RAC-kPBT instance by timetabling a game between home team t_l against away team t_c on time slot p whenever $f(l, c, p) = 1$. It follows from the feasibility of the RTT timetable that all games are planned *(C1).* Since a lecturer meets with at most one class per time slot, and vice versa, it also follows that a team plays at most once per time slot in S *(C4)*. Finally, the availability constraints are respected since venues are always available *(C2)*, and the availability of lecturers and classes directly correspond with the availability of teams in the RAC-kBT instance *(C3)*.

Conversely, assume that we have a feasible timetable for the RAC-kPBT instance. Then, it suffices to set $f(l, c, s) = 1$ whenever team $t_l \in T_1$ plays a game against team $t_c \in T_2$ on time slot $s \in S$, which will timetable all lectures. This function also respects the availability constraints since the player availability of teams in T_1 and T_2 directly correspond to the availability of lecturers and classes. Finally, from the feasibility of the RAC-kPBT timetable, it follows that this function will never match a lecturer with more than one class, or vice versa, per time slot.

Even et al. [21] prove that RTT is $\mathcal{NP}$-complete, even if $R_{l,c} \in \{0, 1\} \forall l \in L, c \in C$, $|P| = 3$, and $A_c = S, \forall c \in C$. Costa et al. [13] prove that RTT remains $\mathcal{NP}$-complete when $R_{l,c} \in \{0, 1\} \forall l \in L, c \in C$, $|C| = 3$ and $A_c = S, \forall c \in C$. In both proofs, lecturers are tight, i.e. $|A_l| = \sum_{c \in C} R_{l,c} \forall l \in L$, implying that teams in T_1 are also tight. □

RAC-kPBT remains difficult when teams are always available but the availability of venues is limited. Indeed, in the proof of Theorem 1 class teams are always available and lecturer teams only play home games. Hence, venue availability of teams in T_1 can be modeled as player availability, and Corollary 4 follows.

Corollary 4. *RAC-kPBT is $\mathcal{NP}$-complete, even if $k = 1$, teams are always available, venues of teams in T_2 are always available, and there are only three time slots or three teams in the second group.*

Theorem 2. *RAC-1BT is $\mathcal{NP}$-complete, even if venues are always available and all teams are tight.*

Proof. We prove the theorem by presenting a reduction from completing a partial Latin square (CLS) to RAC-1BT.
CLS. *Instance*: A partial Latin square P, that is an $m \times m$ array filled with $3m$ symbols from $\psi = \{1, \ldots, m\}$ in such a way that each symbol occurs at most once in every row and column. *Question*: Is it possible to fill the empty cells in P so that every symbol occurs exactly once in every row and column?

Easton and Parker [19] prove that CLS is $\mathcal{NP}$-complete, even if only $3m$ cells are filled in any $m \times m$ partial Latin square.

For any instance of CLS, we create a set of m row teams T_1 and m column teams T_2 representing each row and column in P. Furthermore, we add exactly one game between home team $i \in T_1$ and away team $j \in T_2$, i.e. $m_{i,j} = 1$. In addition, we partition the set of time slots into two subsets $S = S_1 \cup S_2$. The first set contains a time slot for each symbol in CLS ($S_1 = \psi$), the second set contains a time slot $s_{i,j}$ for each filled cell $P_{i,j}$. The player availability set A_i of a row team $i \in T_1$ (or column team $j \in T_2$) is such that this team is unavailable during time slots in S_1 corresponding to symbols that are already present in row i (or column j) of P, unavailable in all time slots in S_2 not corresponding to filled cells of row i (or column j) of P, and available in all other time slots. For all teams, venues are always available ($H_i = A_i, \forall i \in T$). Remark that this transformation can be realized in polynomial time since the number of fixed cells is $3m$, resulting in $|S_1| + |S_2| = 4m$. Moreover, it follows from the transformation that all teams are tight.

Now, we show that the CLS instance is feasible if and only if the corresponding RAC-1BT instance is feasible. Suppose first we have a feasible solution for the CLS instance, that is a Latin square L that fills every empty cell of P. Then, we can construct a feasible timetable for the RAC-1BT instance by timetabling game $(i, j), i \in T_1, j \in T_2$ on time slot $L_{i,j}$ whenever $P_{i,j}$ is empty, or on time slot $s_{i,j} \in S_2$ if $P_{i,j}$ is filled *(C1)*. Since each symbol in a Latin square occurs at most once in every row and column, and since teams in the RAC-1BT instance are only unavailable during time slots in S_1 that correspond to a filled symbol in the corresponding row or column of P, this transformation respects constraints *(C3)* and *(C4)* for time slots of S_1. Clearly, *(C3)* and *(C4)* are also respected in S_2 since only teams i and j are available on a time slot $s_{i,j}$ belonging to S_2. Finally, since venues are always available, constraints *(C2)* are also respected.

Conversely, assume that we have a feasible timetable for the RAC-1BT instance. Then, exactly one game between teams $i \in T_1$ and $j \in T_2$ must be planned in S_1 whenever cell $P_{i,j}$ is unfilled since, by construction, i and j are in this case never simultaneously available in S_2. Therefore, we can always set the value of an unfilled cell $P_{i,j}$ to the time slot in which i plays against j. This results in a feasible Latin square since teams are unavailable during time slots that already occur in the corresponding row or column of P and teams in the RAC-1BT instance can play at most once per time slot. □

In contrast to RAC-kPBT, RAC-kBT can be answered in polynomial time if there are no venue availability constraints and teams in T_2 are always available.

Proposition 1. *RAC-kBT can be answered in polynomial time if venues are always available, and teams in T_2 are always available.*

Proof. Corollary 2 guarantees feasibility of an instance if for each game $(i, j) \in M$ it holds that $|H_i \cap A_j| \geq \max\{m_i, m_j\}$. Because in RAC-kBT we have $m_i = k|T_2|$ for all $i \in T$ and since venues and teams in T_2 are always available, the condition simplifies to $|A_i| \geq k|T_2|$ for all $i \in T_1$. Clearly, this is also a necessary condition since each team must be available during no fewer time slots than the total number of games it has to play. Hence, return 'yes' if the condition holds, and 'no' otherwise. □

We now show that RAC-1RRT, and by extension RAC-kRRT, is also $\mathcal{NP}$-complete.

Theorem 3. *RAC-1RRT is $\mathcal{NP}$-complete, even if venues are always available and all teams are tight.*

Proof. We prove the theorem by presenting a reduction from RAC-1BT to RAC-1RRT. Recall, in RAC-1BT we are given a set of time slots S', and a set of teams T' that are partitioned into two disjoint groups T_1 and T_2 each containing m teams.

Moreover, each team $i \in T'$ provides a venue availability set $H'_i \subseteq S'$, and a player availability set $A'_i \subseteq S'$. Finally, we are given a set of games M', with $m'_{i,j} + m'_{j,i} = 1$, $\forall i \in T_1, \forall j \in T_2$.

We construct an instance of RAC-1RRT from any instance of RAC-1BT in which venues are always available, and all teams are tight. This setting is proven to be $\mathcal{NP}$-complete in Theorem 2. To begin, we set $T = T_1 \cup T_2$. Next, we partition the set of time slots S into three sets of time slots $S = S_1 \cup S_2 \cup S_3$, with $S_1 = S'$. The player availability of a team in S_1 corresponds to the player availability of the corresponding team in the RAC-1BT instance. The second set S_2 contains $|T_1| - 1$ time slots if $|T_1|$ is even, and $|T_1|$ time slots otherwise. If $|T_1|$ is even, players of T_1-teams are always available in S_2, otherwise the players of team $i \in T_1$ are unavailable in the i-th time slot of S_2 and available in all other time slots of S_2. Players of T_2-teams are never available in S_2. We apply a similar construction for S_3 with the role of the T_1-teams and T_2-teams inverted. The game set M consists of all games in M', appended with exactly one game (i, j) for each $i, j \in T_1, i < j$, and for each $i, j \in T_2, i < j$. It follows from the input of RAC-1BT instance and the construction of the RAC-1RR instance that all teams are tight and that venues are always available. In summary, if the number of teams is even, the instance of RAC-1RRT is completely specified by:

$$\begin{aligned}
&S = S' \cup S_2 \cup S_3 \\
&T = T_1 \cup T_2 \\
&A_i = A'_i \cup S_2 && \forall\, i \in T_1 \\
&A_j = A'_j \cup S_3 && \forall\, j \in T_2 \\
&H_i = A_i && \forall\, i \in T \\
&M = M' \cup \{(i, j),\ \forall i, j \in T_1, i < j\} \cup \{(i, j),\ \forall i, j \in T_2, i < j\}
\end{aligned}$$

Remark that this transformation can be realized in polynomial time since the total number of time slots is at most $|S'| + |T_1| + |T_2|$. Now, we show that the RAC-1BT instance is feasible if and only if the RAC-1RRT instance is feasible. Suppose first that we have a feasible timetable for the RAC-1BT instance. We can then construct a feasible timetable for the RAC-1RRT instance in which T_1-teams play against T_2-teams in S_1, T_1-teams play against T_1-teams in S_2, and T_2-teams play against T_2-teams in S_3. To realize this, we first copy all game to time slot assignments from the corresponding RAC-1BT solution. This makes that all games between T_1-teams and T_2-teams are planned in S_1 with the correct home-away status. Since teams in the RAC-1BT instance play at most once per time slot, it follows that teams in the RAC-1RRT instance also play at most once per time slot in S_1 *(C4)*. Moreover, this construction respects *(C3)* because availability of teams during S_1 in the RAC-1RRT instance fully correspond to the availability of the teams in the RAC-1BT instance. Second, we use the circle method [14] to construct, in $O(|T_1|^2)$ time, a (time-constrained) single round-robin tournament between the T_1-teams in S_2. We thereby assign the home-away status to the team with the smallest index. It follows from the circle method that teams play at most once per time slot *(C4)*; constraints *(C3)* are also respected since T_1-teams are always available in S_2 if $|T_1|$ is even. If $|T_1|$ is odd, relabeling of the teams in the circle method always result in the necessary bye for the unavailable team of each time slot. Finally, we use the same technique to plan all games between the T_2-teams in S_3. This makes that all games of the single round-robin tournament are planned *(C1)*. Finally, since venues are always available, constraints *(C2)* are also respected.

Conversely, assume that we have a feasible timetable for the RAC-1RRT instance. Then, we can always timetable game (i, j) or (j, i) with $i \in T_1$ and $j \in T_2$ on a time slot $s \in S_1$ *(C1)*. This follows from the fact that j is unavailable during S_2, i is unavailable during S_3, and M contains all games with the correct home-away status since it includes M'. The resulting timetable also respects constraints *(C3)* since the availability of the teams during S_1 in the RAC-1RRT instance fully corresponds to the availability of the teams in the RAC-1BT instance. From the feasibility of the RAC-1RRT solution, it also follows that each team is involved in at most one game per time slot *(C4)*. Finally, constraints *(C2)* are also respected since venues are always available. □

Corollary 5. *RAC-kRRT is $\mathcal{NP}$-complete for any fixed k, even if venues are always available and all teams are tight.*

Proof. We prove the corollary by presenting a reduction from RAC-1RRT to RAC-kRRT. Recall, in RAC-1RRT we are given a set of time slots S', and a set of teams T' with for each team $i \in T'$ a venue availability set $H'_i \subseteq S'$ and a player availability set $A'_i \subseteq S'$. Finally, we are given a set of games M', with $m'_{i,j} + m'_{j,i} = 1, \forall i, j \in T', i \neq j$. RAC-1RRT is proven to be $\mathcal{NP}$-complete in Theorem 3, even if venues are always available and all teams are tight.

We construct an instance of RAC-kRRT from any instance of RAC-1RRT. To begin, we set $T = T'$. Next, we partition the set of time slots into k sets $S = S_1 \cup S_2 \cup \cdots \cup S_k$, with $S_1 = S'$. The player availability of a team in S_1 corresponds to the player availability of the corresponding team in the RAC-1RRT instance. All other intervals contain exactly one time slot for each unordered team pair $\{i, j\}, i \neq j$; only team i and j are available in this time slot. Venues are always available. Finally, the game set M consists of all games in M', appended with exactly $k - 1$ games (i, j) for each $i, j \in T, i < j$. This makes that all teams are tight.

Now, we show that the RAC-kRRT instance is feasible if and only if the RAC-1RRT instance is feasible. Suppose first that we have a feasible timetable for the RAC-1RRT instance. We can then construct a feasible solution for the RAC-kRRT instance in which each team plays exactly once against each other team in each of the intervals. For S_1, we copy all game to time slot assignments and the corresponding home-away status from the RAC-1RRT solution. For each of the other intervals it suffices to schedule a game between the two available teams of each time slot. We assign the home-away status to the team with the smallest index. This makes that all games are timetabled *(C1)*, that player availability is respected *(C3)*, and that each team plays at most once per time slot *(C4)*. Finally, since venues are always available, constraints (C2) are also respected.

Conversely, assume that we have a feasible timetable for the RAC-kRRT instance. Since any pair of teams meets k times, each interval from S_2 to S_k contains only one feasible time slot for a pair to meet, and all teams are tight, it follows that every pair of teams meets exactly once in S_1. Since venues are always available, both teams can play either at home or away in this time slot. Therefore, a feasible timetable for the RAC-1RRT instance can be constructed by copying the game assignments in S_1, and swapping the home-away status of game $(i, j), i, j \in T, i \neq j$, if $(i, j) \notin M'$. □

4. Fairness measures

Section 3 proves that it is $\mathcal{NP}$-complete to decide whether an availability constrained sports timetable exists. Clearly, the problem remains difficult when we additionally optimize fairness measures. This section proposes several of these measures and provides mathematical models to optimize them.

Consider first Equations (1)-(4) that provide a base model to solve RAC-kRRT. In this model, the variable $x_{i,j,s}$ is 1 if team $i \in T$ and team $j \in T \setminus i$ meet at the venue of i on time slot $s \in S$. The first set of constraints ensures that each team plays the required number of home games against each other team *(C1)*. The next set of constraints enforces that a team plays at most once per time slot *(C4)*. Constraints (3) reduce the number of variables in the system by explicitly stating that two teams can only meet when the venue of the home team and the players of the away team are simultaneously available *(C2), (C3)*; when implementing this formulation, these variables need not be created. Finally, constraints (4) are the binary constraints on the x-variables.

Base model

$$\sum_{s \in H_i \cap A_j} x_{i,j,s} = m_{i,j} \qquad \forall i, j \in T : i \neq j \quad (1)$$

$$\sum_{j \in T \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) \leqslant 1 \qquad \forall i \in T, s \in A_i \quad (2)$$

$$x_{i,j,s} = 0 \qquad \forall i, j \in T : i \neq j, s \in S \setminus \{H_i \cap A_j\} \quad (3)$$

$$x_{i,j,s} \in \{0, 1\} \qquad \forall i, j \in T : i \neq j, s \in H_i \cap A_j \quad (4)$$

The remainder of this section enhances this base model to handle several fairness issues.

4.1. Aggregated rest time penalty

In time-relaxed timetables, the rest time between consecutive games of a team can vary substantially. This is problematic for two reasons. First, if the difference in rest time between two opposing teams is large, teams might blame the timetable for losing the game. This is the topic of research in [2, 3] where the authors show how to minimize the total number of rest time differences in a time-relaxed timetable in which the total number of games per time slot is limited. Second, the absolute rest time needs to be monitored since several researchers (e.g. [7, 16]) found a relation between fixture congestion and higher injury rates. Interestingly, none of the authors in [7, 16] found a decrease in technical or physical performance during congested periods. Nevertheless, Scoppa [50] finds that differences in rest time in professional football have a positive and significant impact on performance when at least one of the two teams enjoys a very short period of rest, whereas no impact is found when the rest time of both teams is sufficiently long. This hints that the impact of days of rest on performance is non-linear: 'it turns out to be important if rest time is equal or below three days between consecutive matches, whereas it becomes irrelevant when teams are allowed with at least four days of rest [50]'.

Related to the rest time of teams, Suksompong [53] defines the *guaranteed rest time* of a timetable as the maximum integer g such that any team in the tournament has at least g time slots of rest between two consecutive games.

In Proposition 2, we derive an upper bound of the guaranteed rest time in a k round-robin tournament; we refer to this bound as GRT.

Proposition 2. *The guaranteed rest time of any k round-robin tournament is at most* $GRT = \lfloor \frac{|S|-1}{k(|T|-1+|T| \bmod 2)-1} \rfloor - 1$ *time slots.*

Proof. If $|T|$ is even, each team needs at least $k(|T|-1)$ time slots to play all its games, and $(k(|T|-1)-1)$GRT time slots to have at least GRT time slots recovery between any two consecutive games. Rewriting the inequality $k(|T|-1)+(k(|T|-1)-1)\text{GRT} \leqslant |S|$ results in the bound. If $|T|$ is odd, there will always be a team that has no game during the first $1+\text{GRT}$ slots. Indeed, the total number of teams that play a game in the same time slot must be even, and each team needs at least GRT days of rest before it can play its next game. If we apply this reasoning k times, we get: $k(|T|-1)+(k(|T|-1)-1)\text{GRT}+k(1+\text{GRT}) \leqslant |S|$. Rewriting this inequality results in the bound. □

As an example, in a double round-robin tournament with 15 teams and 150 time slots, the guaranteed rest time is at most four time slots. If the organizers prefer a guaranteed rest time of five time slots, the season should contain at least 175 time slots. Without considering availability constraints, this bound is easily achievable by first constructing a time-constrained timetable and then inserting GRT empty time slots between any two consecutive games. Nonetheless, this bound is not necessarily achievable if we include availability constraints.

To avoid injuries, time-relaxed timetables usually limit the number of games per team in a given period of time. As an example, Knust [31] limits the total number of games per week to at most two, and Van Bulck et al. [59] limit the total number of games in a given number of consecutive time slots to at most two. Hence, this section additionally requires that a team plays at most twice per GRT + 1 time slots; we refer to this constraint with the symbol *(A1)*.

A potential drawback of the guaranteed rest time is that it only considers the worst-case rest time. For this reason, we propose a new fairness measure, the aggregated rest time penalty (ARTP), that penalizes the timetable with a positive value of $p_r \geq p_{r+1}$ each time a team has only r time slots between two consecutive games. With $p_r = 0$ for $r \geq$ GRT, which we assume in the remainder of this paper, a timetable with a guaranteed rest time of GRT has an ARTP of 0. What is more, Proposition 3 shows that penalties can always be chosen such that a k round-robin timetable which is optimal for the ARTP is also optimal for the guaranteed rest time. This is an interesting result when lexicographic minimax fairness is pursued.

Proposition 3. *In a k round-robin timetable ARTP optimality implies guaranteed rest time optimality when* $\frac{p_{r-1}}{p_r} > |T|(k(|T|-1)-1)$ *for all r with* $0 < r <$ GRT *and* $p_r = 0$ *for all* $r \geq$ GRT.

Proof. Assume that no timetable exists which achieves the GRT-bound from Proposition 2 (otherwise, the proof is trivial), and assume, without loss of generality, $p_{\text{GRT}-1} = 1$. To prove the proposition, we need to show that the ARTP of a timetable F_1 with a guaranteed rest time of $0 < r <$ GRT is always smaller than the ARTP of a timetable F_2 with a guaranteed rest time of $r - 1$. For this, note that the ARTP of timetable F_1 is at most $|T|p_r(k(|T|-1)-1)$, i.e. all teams have exactly r time slots between any two consecutive games. Similarly, the ARTP of timetable F_2 is at least p_{r-1}, i.e. there is one team that plays two consecutive games in $r-1$ time slots. Therefore, it suffices that $p_{r-1} > |T|p_r(k(|T|-1)-1)$. □

To minimize the ARTP of a timetable, the mathematical model below uses an auxiliary variable $y_{i,s,t}$ which is 1 if team i plays a game on time slot s followed by its next game on time slot t, and 0 otherwise. Constraints (5) regulate the value of the $y_{i,s,t}$ variables by considering the number of time slots between two consecutive games of the same team. Additionally, constraints (6) model *(A1)*. We note that it follows from *(A1)* that the games are consecutive if team i plays on time slot s and t and $|t - s| \leq$ GRT. Hence, if constraints *(A1)* are present and $p_r = 0$ for $r \geq$ GRT, we can strengthen the formulation by dropping the negative summation term of Equation 5. Finally, constraints (7) state that the y-variables are non-negative; integrality follows from the objective function and the integrality of the x-variables.

ARTP model

minimize $\displaystyle\sum_{i\in T}\sum_{s\in A_i}\sum_{t=s+1}^{s+\text{GRT}} p_{(t-s-1)}y_{ist}$

subject to

(1) – (4)

$$\sum_{j\in T\setminus\{i\}}\Big(x_{i,j,s}+x_{j,i,s}+x_{i,j,t}+x_{j,i,t} - \sum_{k=s+1}^{t-1}(x_{i,j,k}+x_{j,i,k})\Big) - 1 \leqslant y_{i,s,t} \quad \forall i\in T, s,t\in A_i : s<t, t-s\leqslant \text{GRT} \quad (5)$$

$$\sum_{j\in T\setminus\{i\}}\sum_{k=s}^{s+\text{GRT}}(x_{i,j,k}+x_{j,i,k}) \leq 2 \quad \forall i\in T, s\in A_i \quad (6)$$

$$y_{i,s,t} \geq 0 \quad \forall i\in T, s,t\in A_i : s<t, t-s\leqslant \text{GRT} \quad (7)$$

4.2. Games played difference index

To ensure that all teams have roughly played the same number of games at any point in time, Suksompong [53] defines the games played difference index (GPDI) as 'the minimum integer p such that at any point in the timetable, the difference between the number of games played by any two teams is at most p'. A timetable with a low GPDI is desirable since this results in more accurate tournament rankings and since this may reduce the opportunities for match fixing. In contrast to time-constrained round-robin timetables in which at least $n-1$ teams have always played the same number of games, the GPDI of a time-relaxed k round-robin can be as high as $k(n-2)$: one team has played k times against all other teams except for a single team that has not played any game yet. The GPDI model listed below makes use of a variable $g_{i,s}$ that represents the number of games played by team i up to and including time slot s. Equations (8), (9) and (10) recursively model the number of games a team played up to any time slot. Next, equations (11) calculate the GPDI value that is minimized by the objective function. Note that the integrality of $g_{i,s}$ follows from (4), (8), (9), and (10).

GPDI model

minimize GPDI

subject to

(1) – (4)

$$g_{i,1} = \sum_{j\in T\setminus\{i\}}(x_{i,j,1}+x_{j,i,1}) \quad \forall i\in T \quad (8)$$

$$g_{i,s} = g_{i,s-1} + \sum_{j\in T\setminus\{i\}}(x_{i,j,s}+x_{j,i,s}) \quad \forall i\in T, s\in A_i : s>1 \quad (9)$$

$$g_{i,s} = g_{i,s-1} \quad \forall i\in T, s\in S\setminus A_i : s>1 \quad (10)$$

$$g_{i,s} - g_{j,s} \leqslant \text{GPDI} \quad \forall i,j\in T : i\neq j, s\in S \quad (11)$$

4.3. Break minimization

In time-relaxed timetabling, a team has a home (away) break whenever it plays two consecutive home (away) games, irrespective of the number of time slots between these two games. There are several reasons why it matters for a team to be either the home team or the away team. First, for a large variety of sports it is believed that the team playing at home has a *home-field advantage* due, among others, to crowd effects, travel effects, psychological effects, and referee bias (see [41, 42]). Second, many sports require one of the two teams to start the game, for example, by serving some object which the opponent tries to return. The starting team usually has a *first-mover advantage or disadvantage*. As an example, in a round-robin chess tournament each player plays an equal number of times with the white (home) and black (away) pieces since the player with the white pieces may open the game thereby having greater flexibility to control the game. In softball 'a team prefers to be the home team, not because there is a home-field advantage in recreational softball, but because the home team bats last; therefore, it knows what it needs to do to win the game in the last inning [25]'. In terms of fairness, breaks are undesirable as they mean (not) having the advantage related to the home-away status for two consecutive games (see e.g. [17, 34]). Moreover, in professional leagues home breaks may have an adverse impact on game attendance (see [22]) whereas away breaks may result in long periods without any home game (and corresponding revenues) when there are many days-off between consecutive games (e.g. [17]).

Professional teams may prefer a timetable with less travel distance but slightly more breaks (see e.g. [5, 34]). Indeed, if the distance between every pair of teams is constant and teams do not return home when playing consecutively away, Urrutia and Ribeiro [56] show that travel minimization is equivalent with break maximization. Nevertheless, if teams do not return home after playing away (e.g. because there are too many days-off in between, hotels are too expensive, or teams are located close to one another as is typical the case in non-professional regional leagues (see also [54])) travel minimization may not be a major concern and breaks may be minimized.

It is well known that a timetable for a time-constrained single round-robin tournament with n teams contains at least $n-2$ breaks when n is even [14]. However, a single round-robin timetable without any break can be constructed in time $\mathcal{O}(n^2)$ when n is odd, or when there is one additional time slot available [23]. Nevertheless, as soon as player availability is considered, Corollary 5 proves that no polynomial-time algorithm exists to schedule the tournament, unless $\mathcal{P} = \mathcal{NP}$.

To minimize breaks, we present a mathematical model that uses a binary variable $b_{i,s}$ (as proposed in [31]) that is one if team i has a break on time slot s, and 0 otherwise. Constraints (12) model the home breaks of a team, whereas constraints (13) model the away breaks of a team. Finally, constraints (14) reduce the number of break variables by stating that a team cannot have a break when its players are unavailable; in practice these variables are not created.

Break model

minimize $\sum_{i \in T} \sum_{s \in A_i} b_{i,s}$

subject to

(1) – (4)

$$\sum_{j \in T \setminus \{i\}} \Big(x_{i,j,s} + x_{i,j,t} - \sum_{u \in S : s<u<t} (x_{i,j,u} + x_{j,i,u}) \Big) - b_{i,t} \leqslant 1 \;\; \forall i \in T, s,t \in H_i : s < t \quad (12)$$

$$\sum_{j \in T \setminus \{i\}} \Big(x_{j,i,s} + x_{j,i,t} - \sum_{u \in S : s<u<t} (x_{i,j,u} + x_{j,i,u}) \Big) - b_{i,t} \leqslant 1 \;\; \forall i \in T, s,t \in A_i : s < t \quad (13)$$

$$b_{i,s} = 0 \qquad \forall i \in T, s \in S \setminus A_i \quad (14)$$

Notice that (12) and (13) potentially include respectively $\mathcal{O}(|T||H_i|^2)$ and $\mathcal{O}(|T||A_i|^2)$ constraints. Since the venue availability set of a team is usually small, especially the large number of type (13) constraints is problematic. We significantly reduce this number by noticing that team i must play at least one home game in any subset of $|H_i| - \sum_{j \in T} m_{i,j} + 1$ home slots (i.e. time slots belonging to H_i). Therefore, it suffices in constraints (12) and (13) to consider time slots s and t only if $s < t$ and team i has less than $|H_i| - \sum_{j \in T} m_{i,j}$ home slots between s and t.

5. Heuristic algorithms

The IP formulations from Section 4 require a considerable amount of computational resources using a state-of-the-art IP solver (see Sec. 6). This section therefore proposes an adaptive large neighborhood search (Sec. 5.1) and a memetic algorithm (Sec. 5.2). Unless otherwise stated, the remainder of this paper focuses on double round-robin tournaments in which each team plays one home game against every other team, i.e. $m_{i,j} = 1$ for all $i, j \in T$ with $i \neq j$. This assumption enables us to exploit problem specific properties in the memetic algorithm of Section 5.2.

5.1. Adaptive large neighborhood search

Instead of optimizing a difficult problem at once, large neighborhood search (LNS [51]) gradually improves an incumbent solution by alternately destroying and repairing this solution. This method differs from traditional local search methods by destructing (exponentially) large parts of the solution in the hope that the local optima are of higher quality. This implies that the repair stage is also much more complex.

The approach in this paper was based on a variant of LNS which is called adaptive large neighborhood search (ALNS). The general structure of the ALNS algorithm is given in Figure 1. ALNS extends LNS in two ways. First, as in variable neighborhood search (VNS [36]), ALNS makes use of multiple destruction and repair operators in order to profit from a simultaneous search in multiple neighborhoods. However, when compared with VNS, ALNS has the advantage that it self-adaptively determines which of these operators to select by maintaining an operator-specific weight that is updated during the execution of the algorithm. Second, unlike LNS, ALNS also accepts a worse solution in a way that is very similar to simulated annealing. For an introduction to ALNS and related methods, we refer to Pisinger and Ropke [40].

5.1.1. Initial solution

ALNS is an improvement heuristic that needs an initial solution for its first iteration. Similar to Dorneles et al. [15], we generate this solution by disregarding the optimization criteria and solving the corresponding feasibility problem. Using a state-of-the-art IP solver, this results in an initial solution within a few seconds.

5.1.2. Destroy and repair

When used in combination with IP solvers, ALNS can be seen as performing a sequence of fix-optimize operations: first the solver selects a subset of free variables and fixes all other variables at their value in the current solution, then it re-optimizes all free variables. The main variable in the models of Section 4 are the x variables: once the x variables are fixed, the value of the other variables can easily be inferred. We consider two operators to select the free x variables: a team-based and time-based destructor. The team-based destructor initially selects d_1 teams with a probability of selection for each team that is proportionate to the contribution of this team to the total cost. All x variables regulating the home and away games of the selected teams are then free to be optimized. The time-based destructor initially selects d_2 consecutive time slots with a probability of selection that is proportionate to the total cost induced by all games scheduled within this period.

In order to repair the destroyed solution, we fix the value of all but the free x variables to their value in the incumbent solution and optimize the resulting model using an IP solver. Note that the use of a IP solver implies that the output of the

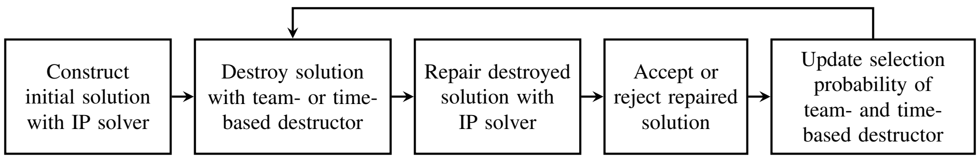
Construct initial solution with IP solver
Destroy solution with team- or time-based destructor
Repair destroyed solution with IP solver
Accept or reject repaired solution
Update selection probability of team- and time-based destructor

Figure 1: General structure of the adaptive large neighborhood search algorithm.

Table 1: Overview of the parameters for the ALNS heuristic.

Parameter	Explanation	Range	ARTP	GPDI	Break
w	Initial acceptance rate (%)	0-100	71	24	45
c	Temperature cooling rate (%)	0-100	26	78	28
β	Minimum timetable difference	0-18	4	7	0
ω	Weight factor	1-100	22	11	91
σ	Decay in exponential moving average (%)	0-100	44	52	90
d_1	Initial team destroy size	1-10	2	4	2
d_2	Initial time slot destroy size (×10)	1-10	1	1	8
iter	Number of iterations before neighborhood size is increased	0-100	87	55	61
Time limit	Max. time to solve subproblem with IP solver	5-30	5	28	24

repair phase is never worse since the solver can always return the solution before it was destroyed. In order to diversify the search, we therefore add constraint (15).

$$\sum_{\{i,j,s\}\in C} x_{i,j,s} \leqslant |C| - \beta \tag{15}$$

This constraint enforces that the candidate solution differs in at least β values from the incumbent solution, from which the free variables C were selected by the destructor.

If β is strictly positive, we need to verify whether we accept the newly generated solution. As advised in [43], we always accept solutions that are better than the incumbent and accept a worse solution with a probability of $\exp(-\Delta/T)$. In this formula, Δ refers to the difference in solution quality between the incumbent and the candidate solution. Each iteration, the temperature T decreases with $c\%$ such that the probability of accepting a worse solution is reduced accordingly. We set the initial temperature so that a solution that is $w\%$ worse than the initial solution is accepted with a probability of 50% [43].

In order to select an appropriate neighborhood, ALNS rewards the selected destruction operator at the end of each iteration. It thereby selects the highest applicable reward from the following list: a reward of (*i*) 1 if the candidate solution is not accepted, (*ii*) ω if we accept a solution worse than the incumbent, (*iii*) ω^2 if the candidate solution is better than the incumbent, or (*iv*) ω^3 if the candidate solution is better than the best so far solution. The selection probability of each destructor is initialized with a weight of 1 and is then updated according to an exponential moving average with smoothing factor σ. In other words, the selection probability of a destructor in iteration $i+1$ is equal to σ times the selection probability in iteration i plus $(1-\sigma)$ times the reward earned by the destructor in iteration i.

For each of the two operators, we additionally keep track of the number of destructions that did not result in an improvement over the incumbent solution and increase the size of the neighborhood each time this number is higher than a threshold (iter). The idea is to gradually increase the search space, since this likely results in better solutions, while keeping computation time manageable. Table 1 gives an overview of all parameters in the model.

5.2. *Memetic algorithm*

Genetic algorithms and variants thereof have previously been used to successfully generate time-relaxed sports timetables (e.g. [31, 47]). This motivated us to develop a genetic algorithm backed by a local improvement heuristic, resulting in a memetic algorithm (also known as hybrid genetic algorithm, see [37]). A general overview of the algorithmic flow is depicted in Figure 2. The remainder of this section explains the different components of the algorithm. An overview of all parameters in the model and the range of values that were considered can be found in Table 2.

5.2.1. *Solution representation and evaluation*

Schönberger et al. [47] propose to encode a timetable via a string composed of different segments each carrying the time slot assignments for all home games of a team. This representation enables to define crossover operators segment wise, ensuring that a team plays at most one home game per time slot in the timetable resulting from crossover. Alternatively, Knust [31] encodes a solution as a permutation of games that can be decoded by planning each game at its first allowable time slot. A fictive overflow interval ensures that such a time slot is always found but assigning games in this interval results in a large penalty value.

In our approach, a double round-robin timetable corresponds to a matrix in which each cell (i, j) carries time slot $r_{i,j}$ on which home team $i \in T$ plays against away team $j \in T \setminus \{i\}$. Similar to Knust [31], we allow the algorithm not to plan a game by leaving the corresponding cell blank, but this results in a high penalty cost P. In other words, constraints *(C1)* are transformed into a soft constraint for which violations are penalized in the objective function. This makes that the fitness of an individual equals P times the total number of unplanned games, plus the value of the fairness measure being optimized.

5.2.2. *Recombination and mutation*

To improve solutions and to avoid getting trapped in local optima, genetic algorithms vary candidate solutions via crossover

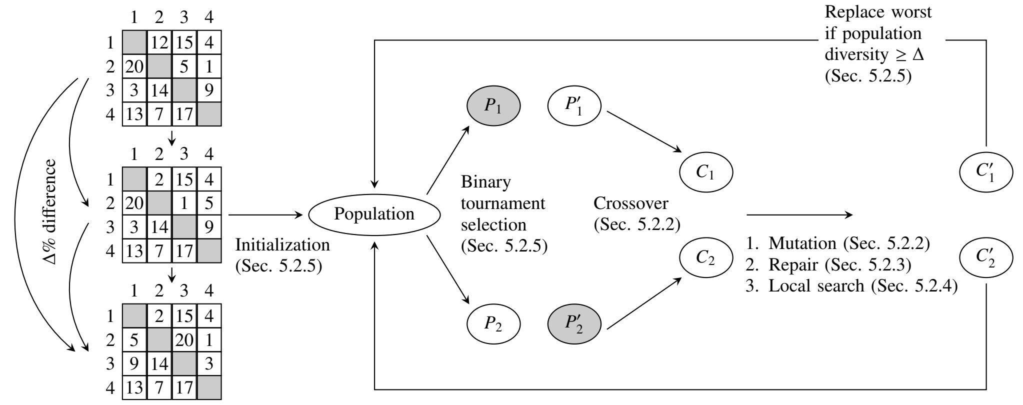
Δ% difference
Initialization (Sec. 5.2.5)
Population
Binary tournament selection (Sec. 5.2.5)
P_1
P'_1
P_2
P'_2
Crossover (Sec. 5.2.2)
C_1
C_2
1. Mutation (Sec. 5.2.2)
2. Repair (Sec. 5.2.3)
3. Local search (Sec. 5.2.4)
C'_1
C'_2
Replace worst if population diversity ≥ Δ (Sec. 5.2.5)

Figure 2: Illustration of the memetic algorithm with population management (MA|PM). Grey ellipses represent the parents selected by the binary tournament operator.

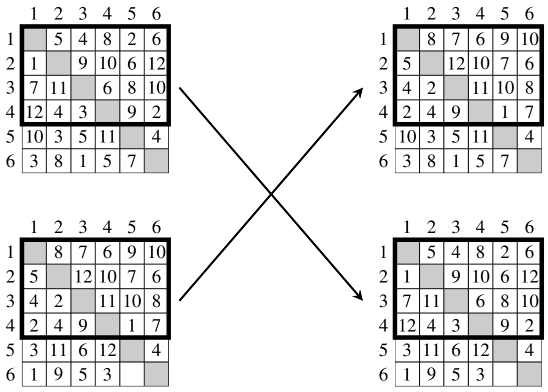

Figure 3: The row-wise crossover picks a crossover point and swaps the parents' rows (left) to form two new offspring (right).

and mutation. Crossover operators combine the genetic information of two parent solutions to create two new offspring solutions. Schönberger et al. [47] mention the importance for crossover operators for timetabling problems to induce as few constraint violations as possible. Therefore, we generalize the segment-wise concept of [47] to row-wise and column-wise crossover operators. The row-wise (column-wise) single-point crossover uniformly draws a number $1 \leq t \leq |T| - 1$, and swaps the upper (leftmost) t rows (columns) between the two parent solutions (see Fig. 3, with $t = 4$).

In addition, we propose a third crossover operator that, in contrast to the row- and column-wise operators, allows for altering the values of cells within the same row of the parent solutions. We adapt the cycle crossover operator (see e.g. [20]) as originally developed to recombine permutations in which the absolute position of cell values must be preserved. Indeed, if the venue availability set of a team is only slightly larger than the number of its home games, the rows in the matrix will resemble permutations. Given two parent solutions, we generate two offspring solutions by applying the operator row by row. More specifically, for each row, we use the traditional cycling procedure to generate different *cycles* (see Figure 4). If the two rows do not contain the same set of elements, the cycling process potentially bumps into a time slots which does not occur in the other parent. If this happens, we extend the cycle in the other direction, which will ultimately result in another time slot which can be used to close the cycle. After splitting the parents' row into cycles, the cycle crossover operator generates the offspring's row by alternately selecting cycles from each parent solution.

Note that all three operators fully respect the availability constraints *(C2)* and *(C3)* when given two feasible parent solutions. Moreover, the row-wise and the adapted cycle crossover operator fully respect *(C4)* with regard to the home games, whereas the column-wise crossover operator fully respects *(C4)* with regard to the away games.

After crossover, each offspring solution undergoes mutation. The algorithm decides with a probability of p_m for each cell (i, j) independently whether cell value $r_{i,j}$ is mutated. If a cell is mutated, the algorithm uniformly draws a time slot from $\{H_i \cap A_j\} \setminus r_{i,j}$. This is similar to the mutation operator proposed in [47].

5.2.3. Repair

Despite our effort so far, newly created offspring solutions may (partially) violate constraints *(C4)* and constraints *(A1)* in case of ARTP optimization. One possibility would be to convert *(C4)* and *(A1)* into soft constraints by penalizing the violations in the objective function. However, under these circumstances, Schönberger et al. [46] provide computational evidence that genetic algorithms without any repair mechanism are inefficient to construct feasible timetables. For this reason, we propose the following heuristic based on [59] that completely repairs offspring solutions resulting from recombination and mutation.

In case there are no constraints of type *(A1)*, we gradually repair a solution in polynomial time for all teams in a randomly chosen order by reassigning all home games of each team. In order to minimize the impact of this repair on the genetic search, we additionally minimize the total number of changes with regard to the original game assignment. More specifically, we repair the home game assignments of team $i \in T$ by constructing and solving the following transportation problem (see Fig. 5). First, we add a unit-supply vertex u_s for each time slot

Table 2: Overview of the parameters for the memetic algorithm. If a parameter is not applicable, we write n/a.

Parameter	Explanation	Range	ARTP	GPDI	Break
μ	Size population	1-250	16	183	11
	Available crossover operators	{col, cycle, row, all}	col	row	col
p_c	Crossover probability (%)	0-100	40	72	17
p_m	Mutation probability (%)	0-100	1	51	1
n_{iter}	No. local search improvements	0-750	117	724	648
Δ	Diversity parameter (%)	0-100	2	13	2
β	Beam width	1-500	n/a	19	n/a
P	Penalty for not scheduling a game	$1,000,000$		$1,000,000$	

Steps to generate a cycle in row R_1 and R_2:

1. Go to the first unused position in R_1. Label its cell value as v_1. Go to 2.
2. Go to the same position in R_2 and label its cell value as v_2. Go to 3.
3. Is v_2 present in R_1? Yes: go to 4. No: go to 5.
4. Go to a position in R_1 with cell value v_2. Add v_2 to the cycle. Go to 2.
5. Is v_1 present in R_2? Yes: go to 6. No: return the cycle.
6. Go to a position in R_2 with cell value v_1. Add v_1 to the cycle. Go to 7.
7. Go to the same position in R_1 and store its cell value in v_1. Go to 5.

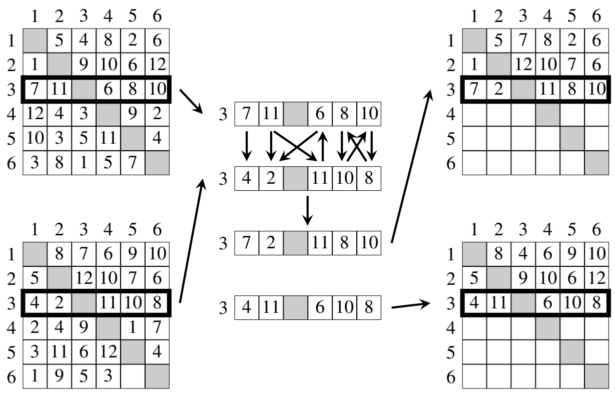

Figure 4: The adapted cycle crossover recombines each row of the two parents (left) separately. First the operator identifies cycles, next it distributes the cycles over the new offspring (right).

$s \in H_i$, and add a vertex q with supply equal to $|T|-1$. Similarly, we add a unit-demand vertex v_j for each opponent $j \in T \setminus \{i\}$, and a vertex d with demand equal to $|H_i|$. For each time slot $s \in H_i$, we then check whether assigning game (i, j) to time slot s would result in a conflict with the already planned games in the partial timetable after removing all home games of i. If this is not the case, we draw an edge between vertex u_s and vertex v_j and set its weight equal to 0 if game (i, j) was originally unassigned or assigned to time slot s, and to 1 otherwise. In the other case, the two vertices remain unconnected. Furthermore, we draw an edge between vertex q and every vertex v_j with $j \in T \setminus \{i\}$ and set its cost equal to P since sending flow over this edge results in not timetabling game (i, j). Finally, we draw an edge between every supply vertex and vertex d and set its cost to zero since these edges are merely used for balancing purposes. For P sufficiently high, this transportation problem assigns a maximal number of i's home games, thereby changing as few assignments as possible.

Nevertheless, in the case of ARTP optimization, solving the transportation problem does not necessarily respect constraints *(A1)*. Indeed, when H_i contains two or more time slots with less than GRT time slots in between, one of the following two events can happen: *(i)* i plays two or more home games combined with one away game within a period of GRT+1 time slots, or *(ii)* i plays three or more home games within a period of GRT + 1 time slots. We solve this problem by constructing a branching tree such that in every vertex the transportation problem outlined above needs to be solved (see Fig. 6). If the resulting home game assignment violates constraints *(A1)*, we apply

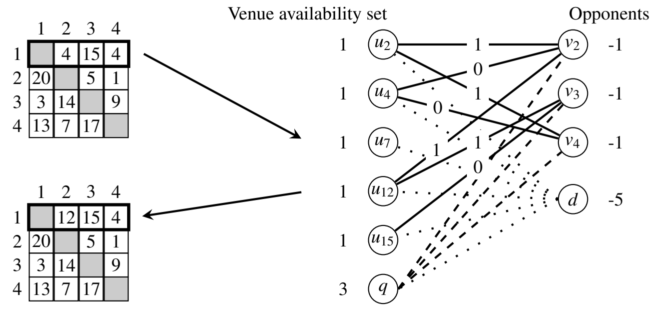

Figure 5: Illustration of the repair operator, assuming $S = \{1, 2, \ldots, 21\}$, $H_1 = \{2, 4, 7, 12, 15\}$, $S \setminus A_2 = \{15\}$, $S \setminus A_3 = \{2, 4, 7\}$, and $S \setminus A_4 = \{12, 15\}$. The timetable in the top-left is infeasible since team 1 plays two home games on time slot 4. Dashed lines have a cost of P, dotted lines have a cost of zero. Solving the transportation problem on the right results in a feasible assignment for all home games of team 1.

branching. More specifically, there is a branch for each of the first two (or three) time slots in case the first found conflict is of type *(i)* (or *(ii)*). In the resulting vertex from each branch, we resolve the transportation problem but with the time slot excluded from H_i. In the root vertex, all time slots in H_i are available. Using the cost resulting from solving the transportation problem as a lower bound, we traverse the tree with a best-bound vertex-selection strategy where we fathom a vertex when its lower bound is dominated by the current best found feasible solution. In real-life instances, the need for this branching is rather rare since the venue availability set of teams is usually well spread over the season (see [59]).

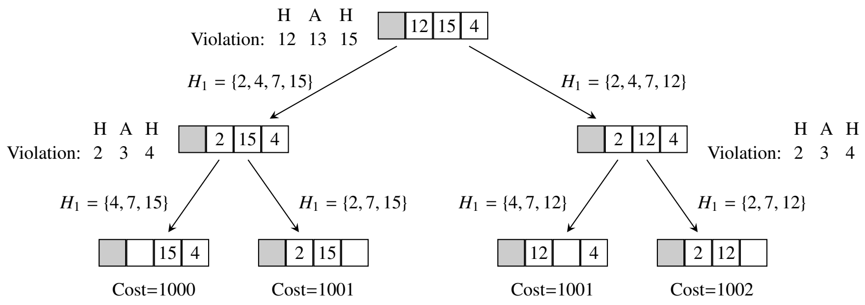

Figure 6: Illustration of the full branching tree for the example in Figure 5, additionally assuming $P = 1000$. Each vertex represents the outcome of the home game assignment resulting from solving the transportation problem from Figure 5. If this assignment violates constraints *(A1)*, we apply branching to solve the conflict.

5.2.4. Local search

Although genetic algorithms are quite effective in locating attractive regions of the search space, they are less efficient in finding the final refinements needed for high quality solutions. This is especially relevant for timetabling problems where crossover operators, mainly responsible for exploitation, act rather disruptively (see [47]). In order to accelerate convergence, memetic algorithms therefore hybridize genetic algorithms with local search operators that are able to quickly improve solutions. This section proposes a dedicated local search procedure for each fairness measure, additionally being able to improve the penalty cost associated with not scheduling games.

ARTP. First, assume that there are at least GRT time slots between any two elements of team i's venue availability set H_i. In this case, given a partial timetable, Van Bulck et al. [59] show how to adapt the transportation problem from Section 5.2.3 to optimally reassign all home games of i with regard to the ARTP measure. They set the weight associated with the edge connecting vertex u_s and vertex v_j equal to the cost of playing game (i, j) on time slot $s \in H_i, j \in T \setminus \{i\}$. This cost will depend on the previous and next game of team i and j in the partial timetable after removing all home games of i, with respect to time slot s. For instance, consider the cost of playing game $(1, 2)$ on time slot 12 in the partial timetable of Figure 7. Observing that GRT = 3, the cost induced by team one is equal to p_0 as its next away game is on time slot 13. The cost induced by team two is equal to p_1 as its next game is on time slot 14. As neither of the two teams has a previous game within GRT + 1 time slots, the cost of the edge is set to $p_0 + p_1$.

Nevertheless, if team i's venue is available during two or more time slots within a period of GRT + 1 time slots, the transportation problem does not take into account costs related to timetabling two successive home games of i within less than GRT time slots (only away games of team i are considered for this). Moreover, as already noted in Section 5.2.3, solving the transportation problem can then result in an infeasible home game assignment.

To cope with these issues, the supply vertices in the transportation network only contain a subset of the time slots in H_i. We choose this subset in two phases. First, we uniformly select an unconsidered time slot on which i plays a home game in the partial timetable. Next, we examine all other time slots in H_i. Consider for example an arbitrary time slot $s \in H_i \setminus H'_i$, and assume that we already inserted time slots $H'_i \subset H_i$. Then, if H'_i does not contain any time slot s' with $|s' - s| \leq$ GRT, we add vertex u_s to the transportation network (see Fig. 7 for $s = 4$ and $H'_1 = \emptyset$). Otherwise, if H'_i contains one such time slot, we check whether i already plays an away game during an interval of GRT + 1 consecutive time slots containing s and s', in the partial timetable. If we do not find such an interval, i can play on both time slots for an additional cost of $p_{|s'-s|-1}$. Therefore, we add vertex v_s, a dummy unit-supply vertex c_1, and a dummy unit-demand vertex c_2 to the network (see Fig. 7 for $s = 7, s' = 4$, $H'_1 = \{4\}$). Next, we connect c_1 to demand vertex d for a cost of zero and to c_2 for a cost equal to $p_{|s'-s|-1}$. Similarly, we connect c_2 to vertices u_s and $u_{s'}$ for a cost of zero. In the other case that we did find such an interval, we know that i can play at most one game during s and s'. Therefore, we add a dummy unit-demand vertex c_3 that is solely connected to vertices u_s and $u_{s'}$ for a cost of 0 (see Fig. 7 for $s = 15, s' = 12, H'_1 = \{4, 7, 12\}$). In the case that there is another time slot $s' \in H'_i$ with $|s' - s| \leq$ GRT that is already connected to a dummy vertex, or in the case that there is a third time slot s'' with $|s'' - s| \leq$ GRT and $|s'' - s' >$ GRT$|$, we try to insert u_s with its worst case cost. If this is not possible, or if H'_i already contains more than one time slot s' with $|s' - s| \leq$ GRT, we simply refrain from adding time slot s to the network (e.g. time slot 2 in Fig. 7 with $H'_1 = \{4, 7\}$). This way, the resulting assignment is always feasible and never undervalues the ARTP. However, since we do not make use of all time slots in the venue availability set and since some slots are added at worst case cost, the reassignment of games may be worse than the original solution. Therefore, we only accept the new solution if it is better, and repeat this procedure n_{iter} times, each time with a randomly chosen team from T.

Breaks. Given a partial timetable, we employ a transportation problem to reassign all home games of team $i \in T$, thereby minimizing the total number of breaks in the timetable. We first partition the time slots in H_i into subsets H_i^k, $1 \leq k \leq g$, as follows. The first group H_i^1 consists of all time slots in H_i up to and including the time slot on which i plays its first away game. The last group H_i^g consists of all time slots in H_i after i plays its last away game. For all other groups, H_i^k consists of all time slots in H_i after i plays its $(k - 1)$-th away game up to and including the time slot in which i plays its (k)-th away game (see

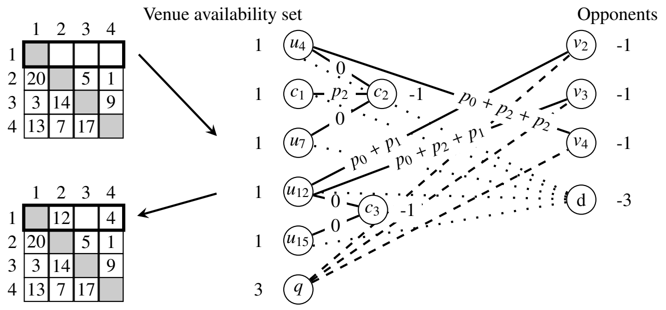

Figure 7: Illustration of the ARTP operator, assuming $S = \{1, 2, \ldots, 21\}$, $H_1 = \{2, 4, 7, 12, 15\}$, $S \setminus A_2 = \{15\}$, $S \setminus A_3 = \{2, 4, 7\}$, and $S \setminus A_4 = \{12, 15\}$. Note that GRT = 3; dashed lines have a cost of P, dotted lines have a cost of zero. Furthermore, assume that we try to add the time slots from H_i in the order $(4, 7, 2, 12, 15)$.

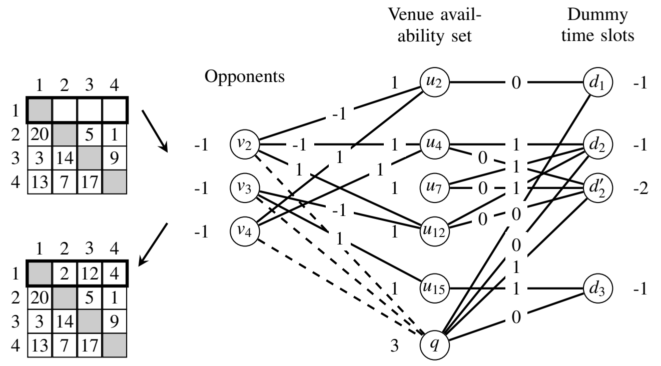

Figure 8: Illustration of the break operator, assuming $S = \{1, 2, \ldots, 21\}$, $H_1 = \{2, 4, 7, 12, 15\}$, $S \setminus A_2 = \{15\}$, $S \setminus A_3 = \{2, 4, 7\}$, and $S \setminus A_4 = \{12, 15\}$. Dashed lines have a cost of P.

Fig. 8). Next, we add a unit-supply vertex u_s for each time slot $s \in H_i$, a unit-demand vertex v_j for each opponent $j \in T \setminus \{i\}$, and a vertex q with supply equal to $|T| - 1$. Finally, we add for each group H_i^k a first unit-demand vertex d_k and a second demand vertex d'_k with demand equal to $|H_i^k| - 1$.

As in Section 5.2.3, we draw an edge between vertex q and every vertex v_j with $j \in T \setminus \{i\}$ and set its cost equal to P since sending flow over this edge results in not timetabling game (i, j). Similarly, we draw an edge between vertex u_s and vertex v_j if assigning game (i, j) to time slot $s \in H_i$ does not result in any conflict with the already planned games in the partial timetable after removing all home games of i. However, this time we set the cost of the edge between vertices u_s and v_j equal to 1 if j has one additional break when playing away on time slot s, to -1 if j has one break less when playing away on time slot s, and to 0 otherwise. To model the costs associated with the home breaks of i, note that the total number of home breaks in H_i^k equals the total number of time slots used in H_i^k minus one. Therefore, we draw an edge between q and d_k with cost zero, and an edge between q and d'_k with cost 1 for each group H_i^k. To model the costs associated with the away breaks of i, note that i can never have an away break before its first away game, or after its last away game. Therefore, we draw edges $\{u_s, d_1\}$, $\{u_s, d'_1\}$, $\{u_s, d_g\}$ and $\{u_s, d'_g\}$ for all $s \in H_i^1 \cup H_i^g$ and set all costs to 0. In all the other groups, i has an away break when it does not play any home game. Therefore, we draw an edge $\{u_s, d_k\}$ with cost 1, and an edge $\{u_s, d'_k\}$ with cost 0 for each time slot $s \in H_i^k, 1 < k < g$. As advised by Burkard et al. [10], we increment all costs with a value of 1 to get rid of the negative costs.

Solving this transportation problem reassigns all home games of team i, thereby minimizing the total number of breaks. We repeat this procedure n_{iter} times, each time with a randomly chosen team from T.

GPDI. Unlike the the ARTP and break settings, we do not use a transportation problem to reassign all home games of a team, since the impact on GPDI of assigning a game to a time slot heavily depends on the other assignments. Instead, we use a simple local search operator based on the ruin-and-recreate paradigm (see [49]). First, this operator destructs a timetable by removing all home game assignments of team $i \in T$. Next, it repairs the timetable by constructing a branching tree that enumerates over all possible home game assignments. Each level of this tree corresponds to a home game of i such that a vertex at level l corresponds to a partial timetable in which l home games of i are either assigned to a time slot or are definitively left unassigned. Branching on a vertex corresponds to exploring the options for the game of the next level, taking into account the assignments in the partial timetable. In the root vertex, no home game of i is timetabled.

Since the size of the neighborhood is exponentially large in the number of opponents, resulting in $O(|H_i|^{|T|-1})$ vertices in the branching tree, we employ beam search (see e.g. [4]) to partially explore the tree with a breadth-first vertex-selection strategy. The main idea is to keep the size of the tree at each level manageable. We label at each level at most β, the beam width, vertices as parents thereby considering all child vertices of the parents of the previous level. We always select the vertex corresponding with the original assignment as the first parent and select the remaining $\beta - 1$ parents by applying the following sorting criteria in decreasing order. The first criterion orders the child vertices according to the total number of unassigned games; if this number is higher than the total number of unassigned games in the original timetable, the vertex is pruned. Similarly, the second criterion orders the child vertices in decreasing GPDI value. If ties still remain, the third criterion assigns a score to each child vertex by summing for each $s \in S$ over the largest difference in the number of games played by any two teams up to and including time slot s. Next, it sorts the vertices in decreasing order of this score. Initial experimentation revealed that this scoring was crucial since otherwise the potential value of moving a team's game to a less congested period, and thus freeing up the timetable for other moves, is not propagated if the GPDI value remains the same. Any remaining tie is broken at random. Our implementation of beam search will never result in a worse solution since, in worst case, it returns the original assignment. We repeat this procedure n_{iter} times, each time with a uniformly chosen team from T. Since the order in which beam search fixes the home game assignments influences the expansion of the tree, we uniformly choose

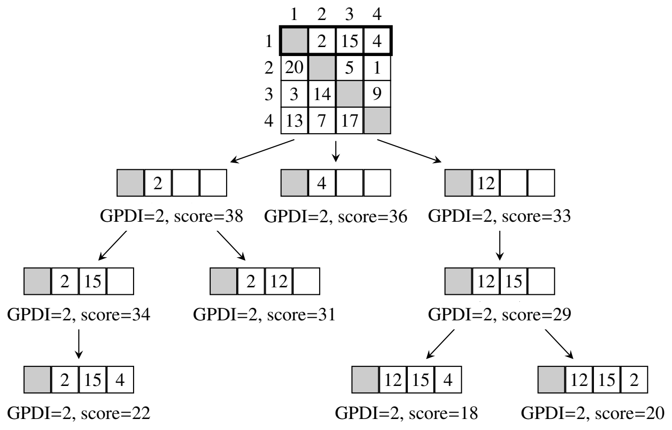

Figure 9: Illustration of beam search for GPDI optimization, assuming $S = \{1, 2, \dots, 21\}$, $H_1 = \{2, 4, 12, 15\}$, $S \setminus A_2 = \{15\}$, $S \setminus A_3 = \{2, 4, 7\}$, $S \setminus A_4 = \{12, 15\}$, and $\beta = 2$.

an unassigned home game at each level of the tree.

5.2.5. *Population management*

Maintaining the right level of population diversity in genetic algorithms so to avoid premature convergence is challenging. This becomes even more complicated in genetic algorithms hybridized with local search, since there is a risk that the local search maps all the solutions to the same local optima. For this reason, we enhance the memetic algorithm with a population management strategy (MA|PM, see [52]) that explicitly preserves the diversity of the population by employing a distance measure expressing the similarity between newly created offspring and the current population (for an overview, see Fig. 2).

We first create the initial population by repeatedly solving the corresponding IP model without the objective function. To ensure a high level of diversity, we enhance the model by adding for each new initial solution constraints of type (15) enforcing that the game assignments must differ in at least Δ% of the assignments with regard to each solution already in the population pool. In case that Δ is high and teams are often unavailable, the process might become computationally expensive or it could be that no additional solutions exist satisfying constraints (15). Though rare, to avoid this, we run the IP solver with a global time limit of 10 seconds, after which we create the remaining solutions by mutating an empty timetable 100 times.

Genetic algorithms need a parent and survivor selection scheme to guide the evolution of the population. As proposed by Sörensen and Sevaux [52], our algorithm uses a binary tournament operator (see [20]) with uniform probabilities to select two parent solutions. With a probability of p_c the two parents mate in which case two offspring are generated by a uniformly chosen operator from the available set of crossover operators. In the other case, the two offspring are identical to the two parents. In each of the two offspring solutions, each cell independently undergoes mutation with a probability of p_m. After recombination and mutation, the local improvement heuristic is applied, and the improved offspring replaces the original offspring (i.e. this algorithm is a Lamarckian memetic algorithm). Subsequently, the algorithm calculates for each of the two offspring and each member of the population the distance expressed in terms of the percentage of different game to time slot assignments. If the distance between the offspring and each member of the population is greater than the diversity parameter Δ, the offspring replaces the worst solution in the current population. Otherwise, the offspring is simply discarded. Although more complex diversification strategies exist (see [52]), we choose to keep Δ constant over the entire run of the algorithm.

6. Computational experiments

This section experimentally evaluates the IP models and heuristics proposed respectively in Section 4 and 5. First, Section 6.1 describes a benchmark of real-life and artificial problem instances. Section 6.2 then explains how we tuned the different parameters of the heuristics, and derives some insights from the parameter space. Section 6.3 employs the algorithms to construct timetables for the benchmark instances and compares the performance of the different solution methods. Finally, Section 6.4 analyzes the contribution of the local search procedures and the population management strategy.

6.1. *Experimental setup*

A first problem instance set consists of 53 real-life double round-robin problem instances originating from a non-professional indoor football competition in Belgium (see [59]). These instances have between 13 and 15 teams and contain 273 or 274 time slots. This makes that the maximum achievable guaranteed rest time, GRT, is either 8 or 9. On average teams can play at home during 4.5 time slots more than the number of opponents in the tournament and cannot play any game during 14.8 time slots.

Corollary 5 states that it is $\mathcal{NP}$-complete to decide whether a feasible solution respecting all availability constraints exists. Nevertheless, for all real-life instances, a state-of-the-art IP solver was able to solve the base model from Equations (1)-(4) within less than a second. In total there were 9 infeasible instances. These instances were discarded, leaving us with 44 feasible real-life double round-robin problem instances.

In order to control for specific problem characteristics, a second benchmark consists of artificial double round-robin problem instances. A problem instance in this set is of type $(|T|, |S|, h, a)$ if it contains $|T|$ teams and $|S|$ time slots, and for each team $i \in T$ it holds that $|H_i| = \sum_{j \in T} m_{i,j} + h$ and $|S| - |A_i| = a$. To construct these instances, we build an instance generator, conforming to the one in [47], which warrants feasibility by constructing each instance around an initial timetable such that $A_i = S$ and H_i contains all time slots on which i plays its home games. Next, we respectively add h time slots from a uniform distribution to H_i and remove a time slots from A_i, thereby ensuring that $H_i \subseteq A_i \forall i \in T$. We consider $|T|$ in the set $\{10, 14, 18, 22\}$, $|S|$ in $\{100, 175, 250\}$, h in $\{0, 5, 10, 15\}$, and a in $\{0, 10, 20, 30, 40, 50\}$. Note that configurations $(22, 100, 10, 50)$ and $(22, 100, 15, 50)$ do not admit a feasible instance since more time slots would be needed. Moreover, since we analyze the computational results with regard to instance parameter a in triples (see Sec. 6.3), we do not consider configurations $(22, 100, 10, 30)$, $(22, 100, 10, 40)$, $(22, 100, 15, 30)$, and $(22, 100, 15, 40)$. For all other problem types, we generate 10 random instances resulting in a total of 2,840 artificial double round-robin problem test instances.

To train the parameters of the heuristics, we use the instance generator described above to construct a separate training set consisting of two random double round-robin problem instances for each of the configurations with $|T| = 22$. This results in 136 artificial problem training instances. A similar training set is constructed with $|T| = 14$ to train the parameters of the ALNS heuristic for break minimization.

We generate a final instance set to evaluate the effect of the tournament structure on the computational performance of the algorithms. To this end, we generate 10 artificial single round-robin problem instances of type $(14, 250, 5, 20)$. This configuration closely resembles the real-life double round-robin problem instances, and has a maximum achievable guaranteed rest time, GRT, of 19. For fairness reasons, it is usually required that each team plays approximately half of its games at home. To this end, we choose the game set M such that for each team the number of home and away games differ by at most one (see [32] for more information).

For all real-life and artificial instances, the penalty values for playing two consecutive games within $r <$ GRT time slots were set to $p_r = 2^{\text{GRT}-r-1}$ whereas p_r was set to 0 for $r \geq$ GRT.

Van Bulck et al. [59] propose XML-based file templates to store sports timetabling problem instances and their solutions. The instances used in the computational experiments of Section 6 are available in this format (see [57]).

The ALNS and memetic algorithm were implemented in C++, compiled with g++ 4.8.5 using optimization flag -O3. To solve the transportation problems, we use an $O(n^3)$ implementation of Kuhn-Munkres algorithm (see [30]). The ALNS and memetic algorithm were granted 2 minutes of computation time and one thread with 2GB of RAM on a CentOS 7.4 GNU/Linux based system with an Intel E5-2680 processor, running at 2.5 GHz. The full IP formulations were solved with the state-of-the-art IP solver Gurobi Optimizer 7.5.2 (for more information about Gurobi, see [39]) using one thread on the same machine but with 8GB of RAM and three hours of computation time. The choice for Gurobi was motivated by Van Bulck et al. [59], who found no performance difference with ILOG CPLEX on similar IP models.

6.2. Parameter tuning

The ALNS and memetic algorithm feature different parameters that need to be set to define a search strategy. To calibrate the heuristics for best performance, we use a dedicated R package called `irace` (see López-Ibáñez et al. [35]) which performs an iterated racing procedure in three steps. First, a number of parameter configurations are sampled from a particular distribution. Second, the best configurations are determined by means of racing: at each step of the race the candidate parameter configurations are tested on a single instance, after which the candidates that perform statistically worse are discarded. Third, the surviving parameter configurations are used to update the sampling distributions.

We independently tuned the parameters of both heuristics and each objective function on the artificial double round-robin problem training instances with $|T| = 22$ using 10,000 evaluations (referred as training budget in `irace`). Since the performance of the ALNS method for break minimization is affected by the size of the IP formulation when the number of teams is large (see Sec. 6.3), the ALNS method for break minimization was tuned using the artificial double round-robin problem training instances with $|T| = 14$. An overview of the best found parameters and the range of parameter values that were considered can be found in Tables 1 and 2. To get a better understanding of the parameter space of the different heuristics, Figures 10 and 11 plot the frequency of the algorithmic parameter values as sampled by `irace` during tuning for the ARTP objective.

When considering the ALNS algorithm for ARTP optimization, Figure 10 advises to destroy only small parts of the solution (d_1, d_2), and to gradually increase the destruction size over the course of the run (iter). With regard to the temperature management, the initial temperature should be chosen as such that a solution that is around 70% (w) worse than the initial solution is accepted with a probability of 50%. The rate at which the temperature is cooled (c) seems to be less important. With regard to the selection probability of the destroy operators, the difference in rewards should be rather high (by choosing a relatively large ω) and more importance should be placed on past iterations (large smoothing factor σ). It seems to be advisable to keep the minimum timetable difference (β), rather low, or even equal to zero in which case the IP solver will never return a worse solution. This may explain why the time limit to repair the destroyed timetable is rather low.

The parameter distributions of the memetic algorithm (Figure 11) propose to work with small population sizes (μ), individuals that display a rather low diversity (Δ), and to apply crossover (p_c) much more frequently than mutation (p_m). With regard to the crossover mode, the column-wise crossover is sampled most often. This may be explained by the local search operator which reschedules all home games of a team. Hence, by using a crossover operator which recombines columns, parent solutions can focus on transferring promising away game assignments. Finally, the high parameter value for the number of times that the local search operator is applied (n_{iter}), indicates that the local search operator enhances the performance of the algorithm.

6.3. Performance analysis

For each heuristic, the best configuration found by `irace` was run 10 times on the real-life double round-robin test set, each time using a different random seed and granted 2 minutes of computation time. Besides, we solved the IP formulations (strongest versions) of Section 4 using Gurobi with a time limit of 60 and 180 minutes. Figures 12 to 14 display the absolute gaps for the various algorithms defined as the best upper bound found by the algorithm minus the best lower bound found by

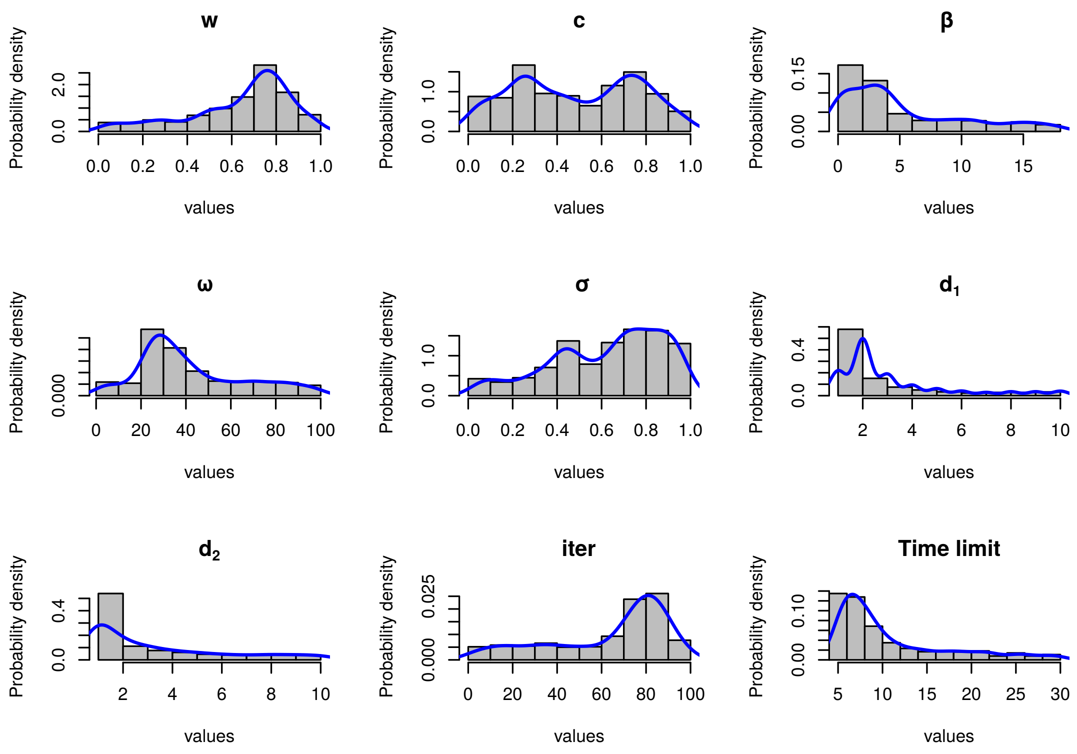

Figure 10: Frequency of the parameters as sampled by `irace` to tune the ALNS algorithm for the ARTP objective.

Gurobi run with a time limit of 180 minutes. When comparing the performance of the algorithms on the ARTP objective, Figure 12 hints that the ALNS and memetic algorithm are the best algorithms in terms of the absolute gap and the variance thereof, despite being given considerably less computational resources. The absolute ARTP gap remains strictly positive, implying that better solutions exist or that the best lower bound found by Gurobi can be improved. When comparing the absolute GPDI gap, Figure 13 hints that the memetic algorithm performs slightly worse than the other algorithms. Nevertheless, if expensive licenses are not an option (see e.g. [59]), the memetic algorithm may be an interesting alternative: the absolute gap of the memetic algorithm always remains below 2. The figure also reveals that Gurobi cannot improve the bounds when run with a time limit of three hours instead of one hour. Finally, Figure 14 shows that Gurobi and the memetic algorithm regularly find the optimal solution value when minimizing the total number of breaks in the timetable. The ALNS algorithm, however, seems less suitable to minimize breaks as the absolute gap values and the variance thereof are higher.

Table 3 provides a more detailed overview of the absolute gap values for the different solution methods on the real-life test instances. The first column in this table displays the mean of the best lower bounds found by Gurobi run with three hours of computation time. The low values for the GPDI and break measures explain why the table shows the absolute gap instead of the more popular relative gap. The second column represents the mean absolute gap when solving the IP model without considering any objective. Since the heuristics make use of the model without objective to generate an initial solution, this column hints that the quality of the initial solution in both heuristics is rather poor when compared to the best found solution. The four last columns display the average gap for each solution approach. To verify the null hypothesis stating that the population mean of the absolute gap differs between two solution methods, Table 4 reports the p-values resulting from a pairwise Wilcoxon rank sum test with Bonferonni's correction for multiple testing. The p-value gives the smallest level of significance at which the null hypothesis would be rejected. A small p-value therefore indicates strong evidence that one solution method systematically performs better in terms of the mean absolute gap; a large p-value, on the contrary, hints that the two solutions methods perform equally well. For the ARTP objective, the table shows that the memetic algorithm statistically outperforms the other solution methods. Contrarily, when considering the GPDI objective, the IP formulation performs better than the memetic algorithm; there is no statistical difference

	LB	No obj.	Gur. 1h	Gur. 3h	ALNS	Memetic
ARTP	761	9,856	1,180	958	324	286
GPDI	1.15	7.84	0.86	0.86	0.87	1.11
Breaks	0.11	144.59	1.13	0.14	32.3	0.14

Table 3: Mean absolute gaps for the different solution methods on the real-life double round-robin test instances. Gaps are based on the best lower bound found by Gurobi run with 3 hours of computation time (see column 'LB'). The 'No obj.' column represents the mean gap when solving the IP model without considering any objective.

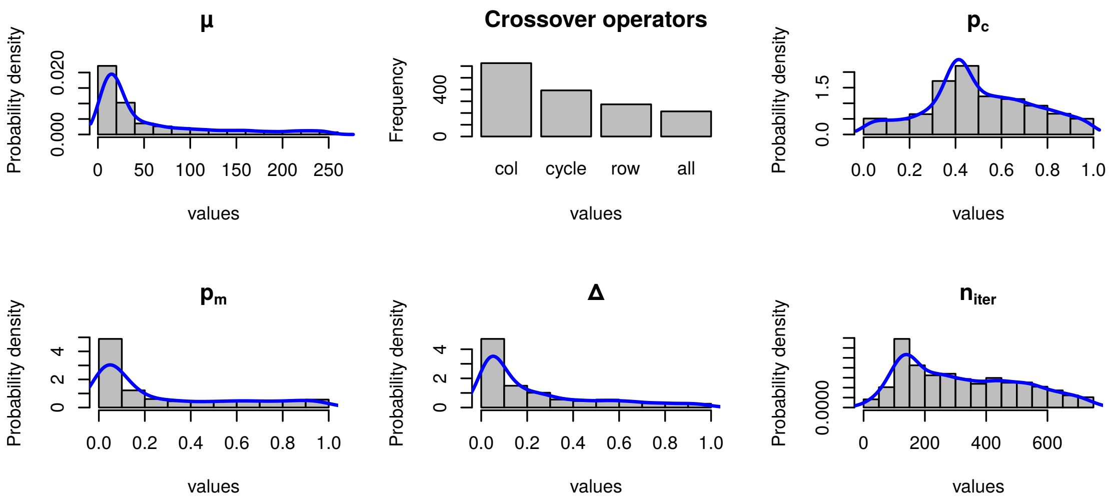

Figure 11: Frequency of the parameters as sampled by `irace` to tune the memetic algorithm for the ARTP objective.

	ARTP			GPDI			Breaks		
	Gur. 1h	Gur. 3h	ALNS	Gur. 1h	Gur. 3h	ALNS	Gur. 1h	Gur. 3h	ALNS
Gurobi 3h	< 2e-16	n/a	< 2e-16	n/a	n/a	1	< 2e-16	n/a	< 2e-16
ALNS	< 2e-16	< 2e-16	n/a	1	1	n/a	< 2e-16	< 2e-16	n/a
Memetic	< 2e-16	< 2e-16	< 2e-16	< 2e-16	< 2e-16	< 2e-16	< 2e-16	1.000	< 2e-16

Table 4: Pairwise Wilcoxon signed rank values for the mean objective values of Table 3. For the GPDI objective, no p-value is available (n/a) comparing Gurobi 1h with Gurobi 3h as the objective value for each instance was exactly the same.

between the IP formulation and the ALNS model. With regard to the break objective, the memetic algorithm outperforms the ALNS algorithm and Gurobi run with one hour but it does not statistically differ from Gurobi run with three hours.

Tables 5 to 7 illustrate how venue and player availability constraints impact algorithmic performance by analyzing the artificial double round-robin instances. For the ARTP objective, the IP solver performs fairly well when the number of time slots is low but performance decreases when $|S|$ increases. Intuitively, this can be explained by the ARTP measure which only penalizes a timetable if two consecutive games are scheduled within GRT time slots. When the number of time slots increases, the value of GRT increases, and hence more variables play a role in the objective function ($p_r > 0$ for $r <$ GRT). A similar observation with regard to the performance of the IP solver can be made when h, which controls the size of the venue availability set, increases. Indeed, in the most extreme case, $h = 0$, and hence a team has to play a home game whenever its home venue is available. In this case, the transportation network used in the local search operator of the memetic algorithm can only decide against whom a team plays its home game, which curtails its strength. Nonetheless, when h is small, Table 5 shows that the memetic algorithm is competitive with the other methods, especially when the total number of teams or time slots is large. When h is large, for most of the configurations, the absolute gap of the memetic algorithm is considerably lower when compared with the IP formulations and slightly lower when compared with the ALNS algorithm.

Table 6 shows that the IP formulation for the GPDI objective performs particularly well. The gap only becomes somewhat larger when the number of teams and time slots are simultaneously high. When comparing the two heuristics, the ALNS method seems to achieve better mean absolute gaps for most of the configurations; only when the number of teams and time slots are large, the memetic algorithm regularly achieves better performance. Nonetheless, the mean absolute gap of the memetic algorithm is never larger than 3.5.

Section 4.3 explained how to reduce the total number of constraints of type (12) and (13). For the real-life test instances, this resulted in an average reduction of 41% of type (12) constraints and 56% of type (13) constraints. Nevertheless, when the number of teams and time slots are large, the size of the IP formulation becomes problematic for Gurobi and the ALNS algorithm and out-of-memory errors frequently occur (see Table 7). In contrast to the ARTP and GPDI gap, the absolute break gap of the ALNS algorithm is large. A similar observation was made during parameter tuning where we observed that the choice of parameters of the ALNS algorithm heavily depends on the problem configuration that is being solved. The memetic algorithm, in contrast, performs more consistently, even when the number of teams and time slots are large.

Finally, Table 8 analyzes the mean gap performance of the solution methods when considering single round-robin problem instances. As in a single round-robin tournament the total number of games halves, we double the team (d_1) and time destruction (d_2) sizes in the ALNS algorithm. All other parameter

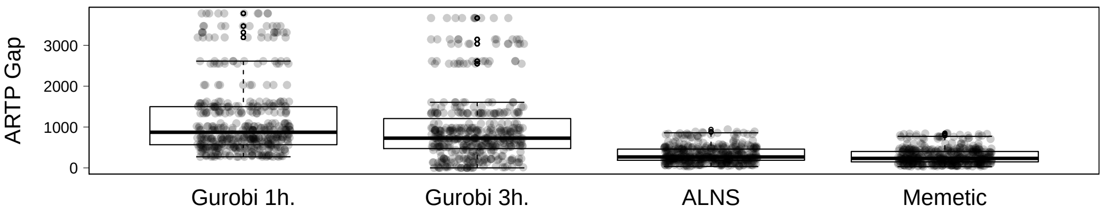

Figure 12: Boxplots of the absolute ARTP gap for 10 independent runs on all real-life instances. Boxes represent the three quartiles, whiskers are drawn at 1.5 times the interquartile range, gray dots represent the solution quality for each solution, and black circles represent outliers. To enhance the comparability of the figure, the bounds found by Gurobi are displayed 10 times for each instance.

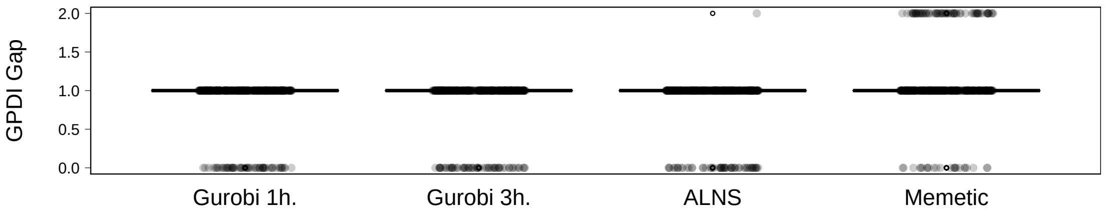

Figure 13: Boxplots of the GPDI gap for 10 independent runs on all real-life instances.

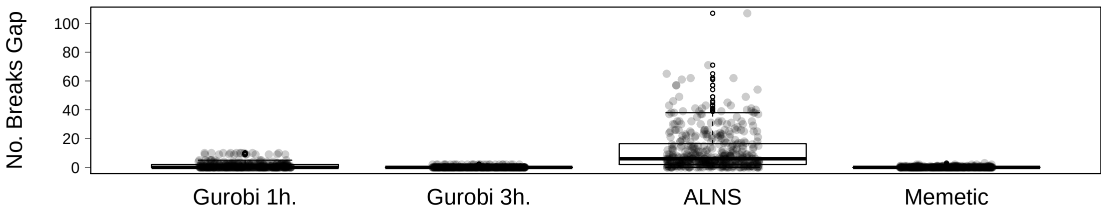

Figure 14: Boxplots of the number of breaks gap for 10 independent runs on all real-life instances.

values, including those of the memetic algorithm, remain unchanged. When comparing the mean absolute ARTP gap, the ALNS and memetic algorithm turn out to perform much better than Gurobi run with 1 hour. Despite the reduced number of variables (compared to a double round-robin tournament), Gurobi is still not able to fully close the ARTP gap. We note that this is partly explained by an increasing GRT which causes many more variables to play a role in the objective function ($p_r > 0$ for $r <$ GRT). With regard to the GPDI, Gurobi achieves a slightly lower mean absolute gap; the ALNS and memetic algorithm perform equally well. Finally, when optimizing the total number of breaks, we observe that Gurobi and the memetic algorithm find an optimal solution for all considered single round-robin instances. In contrast, the use of the ALNS method results in considerably higher mean absolute break gaps.

Based on the computational results of this section, we advise practitioners to minimize the ARTP using the memetic algorithm, unless h and the number of teams are small in which case we advise the use of IP models. If enough time is available, the GPDI objective may be minimized using Gurobi. Alternatively, the ALNS method can be used to generate timetables more quickly and the memetic algorithm can be used as a good alternative when expensive IP solver licenses are not an option. Finally, when minimizing the total number of breaks, we advise practitioners to use the memetic algorithm.

6.4. Memetic algorithm: component analysis

In order to analyze the contribution of the local search operators, we disabled the local search component by fixing n_{iter} to zero and retrained the memetic algorithm using 5,000 evaluations. The results of the retrained configuration without local search are summarized in the 'No local' column of Table 9. When comparing the results with the default MA|PM strategy, we observe that the local search operators drastically improve the performance of the memetic algorithm: the mean absolute ARTP gap decreases with more than a factor 5, the GPDI gap almost halves, and the mean break gap decreases with more than 9 breaks. Not surprisingly, the p-values show that the mean gap reduction is significant for all three objectives. In Section 5.2.2 we proposed a cycle crossover that, in contrast to the row- and column-wise operators, allows for altering the cells within the same row of the parent solutions. With the local search components enabled, the value of the cycle crossover is questionable as `irace` sampled the column-wise operators more often (see Figure 11). Contrarily, when the local search components are disabled, `irace` sampled the cycle crossover operator much more frequently than the other crossover operators for all three objectives (due to space limitations, the new sampling distributions are not shown in this paper). This may be explained by the fact that the local search operators focus on reassigning all home games of a team, hence recombining information within rows may become less valuable.

Table 9 also analyzes the contribution of the population management strategy. We compare the performance with two more traditional strategies that generate at each iteration λ offspring. In the (μ, λ) population model, $\lambda \geq \mu$, we retain the $e\%$ fittest candidate solutions from the old population and replace the remainder by the fittest offspring. Alternatively, the $(\mu + \lambda)$ model merges the old generation with the newly generated offspring

		$\|T\| = 10$			$\|T\| = 14$			$\|T\| = 18$			$\|T\| = 22$		
h	a / $\|S\|$	100	175	250	100	175	250	100	175	250	100	175	250
Gurobi 3h													
0	0-20	0	0	1	0	10	38	4	62	205	0	52	406
	30-50	0	0	1	0	7	22	3	39	111	2	62	313
5	0-20	15	234	688	11	131	467	65	323	554	18	420	1,737
	30-50	3	176	547	7	88	297	28	277	520	15	326	1,613
10	0-20	43	414	1,210	7	281	764	82	563	972	23	465	1,900
	30-50	15	346	1,170	15	223	761	32	349	740	-	404	1,850
15	0-20	42	467	1,049	3	209	554	91	453	724	12	566	2,100
	30-50	25	490	1,196	12	242	612	23	439	781	-	522	2,050
ALNS													
0	0-20	2	34	238	1	29	117	20	63	135	1	69	238
	30-50	2	46	338	3	33	124	21	68	146	11	78	206
5	0-20	20	154	422	11	79	225	50	141	275	20	179	543
	30-50	11	154	465	13	79	217	35	126	276	23	152	493
10	0-20	25	175	347	5	92	247	63	214	356	15	220	660
	30-50	20	175	371	14	91	247	35	155	326	-	190	589
15	0-20	21	180	312	1	85	228	67	221	354	19	266	662
	30-50	22	181	328	9	92	240	31	207	366	-	233	680
Memetic													
0	0-20	4	29	128	2	23	65	15	45	93	2	45	178
	30-50	2	27	180	6	23	63	19	49	90	11	50	145
5	0-20	17	136	342	10	61	172	41	115	224	19	142	436
	30-50	10	120	362	15	59	163	33	93	213	23	119	420
10	0-20	25	166	322	3	74	210	52	180	285	14	175	487
	30-50	20	161	322	17	74	209	39	128	271	-	157	482
15	0-20	21	167	275	0	64	181	61	175	280	23	205	512
	30-50	23	180	327	13	77	200	39	175	297	-	200	526

Table 5: Mean ARTP absolute gap of the randomly generated test instances for the different solution methods.

and retains the fittest overall candidate solutions. Note that the $(\mu + \lambda)$ strategy is used in [47]. For each of the two population management strategies, we included the two additional parameters λ with range $1 - 250$ and e with range $0 - 100$ and retrained the memetic algorithm using 5,000 evaluations. Table 9 shows that the added value of the specialized population management strategy when optimizing the ARTP measure is statistically significant. With regard to the GPDI measure, the MA|PM and (μ, λ) strategy perform equally well, whereas MA|PM performs statistically better than $(\mu + \lambda)$. Finally, when considering breaks, the (μ, λ) strategy performs better than the MA|PM model. However, the p-value is weaker and both models produce near-optimal solutions. Interestingly, the (μ, λ) strategy generally performs better than the $(\mu + \lambda)$ strategy.

A final question of interest is how the performance of the MA|PM heuristic improves when given more computational resources. To answer this question, we can either rerun the algorithm with more computation time or we can parallelize the algorithm such that more iterations can be performed within the same amount of time. The last column of Table 9 shows the results for a multithreaded implementation of the MA|PM heuristic using OpenMP and 8 threads (without retraining). The results hint that the solution quality converged fairly well within two minutes. Hence, it may be more useful to exploit the additional threads to reduce computation times.

7. Conclusion

In this study we have investigated how the presence of availability constraints impacts the computational complexity of time-relaxed timetabling. In particular, we showed that timetabling becomes difficult as soon as player or venue availability is considered. Moreover, we employed list-edge coloring techniques to derive sufficient conditions on the availability of teams and venues for a feasible timetable to exist. Besides, we proposed several mathematical models and an ALNS and a memetic algorithm to solve fairness issues inherently related to time-relaxed timetables. The memetic algorithm can be seen as a collection of algorithmic modules that can be adapted to solve a wide range of different time-relaxed sports timetabling problems. As an example, the structure of the transportation network can be enhanced to deal with forbidden game assignments or to optimize the costs related to the assignments. Similarly, the idea of beam search can be used to optimize rest differences. It is remarkable that our heuristics outperform Gurobi for several real-life instances, despite being given 90 times less computational resources. These computational savings are important since organizers often need to timetable dozens of tournaments, e.g. different skill or age divisions, at the same time. Moreover, the ease with which timetables can be constructed enables organizers to perform a what-if analysis or to revise timetables based on participants' feedback.

		$\|T\| = 10$			$\|T\| = 14$			$\|T\| = 18$			$\|T\| = 22$		
h	a / $\|S\|$	100	175	250	100	175	250	100	175	250	100	175	250
Gurobi 3h													
0	0-20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.4
	30-50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.3	0.5
5	0-20	0.2	0.6	0.8	1.0	1.0	1.0	0.9	1.3	1.6	1.1	1.4	2.5
	30-50	0.7	0.9	0.8	0.6	1.0	1.0	0.4	1.0	1.8	0.0	1.5	2.0
10	0-20	0.2	0.3	0.7	0.9	1.0	1.0	1.0	1.1	1.3	1.1	1.6	2.3
	30-50	0.4	0.7	0.9	0.7	1.0	1.0	0.4	1.0	1.3	-	1.6	1.8
15	0-20	0.0	0.2	0.4	0.6	1.0	1.0	0.9	1.0	1.2	1.1	1.7	2.5
	30-50	0.2	0.6	0.7	0.7	1.0	1.0	0.3	1.1	1.4	-	1.7	1.9
ALNS													
0	0-20	0.0	0.0	0.0	0.0	0.0	0.1	0.7	0.9	0.8	0.8	0.9	1.5
	30-50	0.0	0.0	0.0	0.2	0.1	0.2	0.6	0.9	0.8	0.6	0.9	1.3
5	0-20	0.3	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.2	1.4	2.0	2.1
	30-50	0.4	0.6	1.0	0.6	1.0	1.0	0.7	1.1	1.1	0.6	1.9	2.0
10	0-20	0.2	0.9	0.9	1.0	1.0	1.0	1.0	1.0	1.2	1.2	2.0	2.5
	30-50	0.1	0.7	1.0	0.8	1.0	1.0	0.6	1.0	1.1	-	2.0	2.2
15	0-20	0.2	0.8	0.9	0.8	1.0	1.0	1.0	1.1	1.4	1.3	2.2	2.9
	30-50	0.0	0.8	0.9	0.8	1.0	1.0	0.5	1.0	1.1	-	2.1	2.3
Memetic													
0	0-20	0.0	0.0	0.0	0.6	0.6	0.6	0.9	0.9	0.8	2.1	1.4	1.7
	30-50	0.2	0.0	0.0	0.7	0.8	0.7	1.0	1.0	0.8	3.5	1.5	1.9
5	0-20	1.0	1.0	1.0	1.0	1.0	1.0	1.8	2.0	2.0	2.0	2.0	2.0
	30-50	1.0	1.0	1.0	1.1	1.1	1.0	1.4	2.0	2.0	1.1	2.0	2.0
10	0-20	1.0	1.0	1.0	1.0	1.0	1.0	1.2	1.4	1.8	2.0	2.0	2.0
	30-50	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.8	1.9	-	2.0	2.0
15	0-20	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.2	1.6	2.0	2.0	2.0
	30-50	1.0	1.0	1.0	0.9	1.0	1.0	1.2	1.5	1.8	-	2.0	2.0

Table 6: Mean GPDI absolute gap of the randomly generated test instances for the different solution methods.

Acknowledgements

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government – department EWI.

References

[1] I. Anderson. *Combinatorial designs and tournaments*, volume 6. Oxford University Press, 1997.

[2] T. Atan and B. Çavdaroğlu. Minimization of rest mismatches in round robin tournaments. *Comput. Oper. Res.*, 99:78 – 89, 2018.

[3] T. Atan and B. Çavdaroğlu. Rest differences among teams in European football leagues. In D. Karlis, I. Ntzoufras, and S. Drikos, editors, *Proceedings of MathSport International 2019 Conference*, volume 4, pages 10–15. Athens University of Economics and Business, 2019.

[4] M. O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16:21 – 38, 2011.

[5] R. Bao and M. Trick. The relaxed traveling tournament problem. In B. McCollum, E. Burke, and G. White, editors, *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, pages 472–476, Belfast, 2010. PATAT.

[6] J. C. Bean and J. R. Birge. Reducing travelling costs and player fatigue in the National Basketball Association. *Interfaces*, 10:98–102, 1980.

[7] H. Bengtsson, J. Ekstrand, and M. Hägglund. Muscle injury rates in professional football increase with fixture congestion: an 11-year follow-up of the UEFA Champions League injury study. *Br. J. Sports. Med.*, 47: 743–747, 2013.

[8] O.V. Borodin, Kostochka A.V., and D.R. Woodall. List edge and list total colourings of multigraphs. *Journal of Combinatorial Theory, Series B*, 71:184 – 204, 1997.

[9] F. Bueno. *Mathematical modeling and optimization approaches for scheduling the regular-season games of the National Hockey League*. PhD thesis, École Polytechnique de Montréal, 2014.

[10] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems*. Springer, 2009.

[11] E. Burke, D. de Werra, and J. Kingston. Applications to timetabling. In L. Gross, J. Yellen, and P. Zhang, editors, *Handbook of graph theory*, pages 530–562. CRC Press, 2 edition, 2014.

[12] D. Costa. An evolutionary tabu search algorithm and the NHL scheduling problem. *Infor*, 33:161–178, 1994.

[13] M. C. Costa, D. de Werra, and C. Picouleau. Using graphs for some discrete tomography problems. *Discrete Appl. Math.*, 154:35 – 46, 2006.

[14] D. de Werra. Scheduling in sports. In P. Hansen, editor, *Studies on graphs and discrete programming*, pages 381–395, Amsterdam, 1981. North-Holland.

[15] Á. P. Dorneles, O. C. B. de Araújo, and L. S. Buriol. A fix-and-optimize heuristic for the high school timetabling problem. *Comput. Oper. Res.*, 52:29–38, 2014.

[16] G. Dupont, M. Nedelec, A. McCall, D. McCormack, S. Berthoin, and U. Wislff. Effect of 2 soccer matches in a week on physical performance and injury rate. *Am. J. Sport. Med.*, 38:1752–1758, 2010.

[17] G. Durán, M. Guajardo, and D. Sauré. Scheduling the South American qualifiers to the 2018 FIFA World Cup by integer programming. *Eur. J. Oper. Res.*, 262:1109 – 1115, 2017.

[18] G. Durán, S. Durán, J. Marenco, F. Mascialino, and P. A. Rey. Scheduling Argentina's professional basketball leagues: A variation on the travelling tournament problem. *Eur. J. Oper. Res.*, 2019.

[19] T. Easton and R. G. Parker. On completing latin squares. *Discrete Appl. Math.*, 113:167 – 181, 2001.

[20] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[21] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 184–193, 1975.

[22] D. Forrest and R. Simmons. New issues in attendance demand: The case of the English football league. *J. Sports Econ.*, 7:247–266, 2006.

[23] D. Frončеk and M. Meszka. Round robin tournaments with one bye and no breaks in home-away patterns are unique. In G. Kendall, E. K. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling:*

		$\lvert T\rvert = 10$			$\lvert T\rvert = 14$			$\lvert T\rvert = 18$			$\lvert T\rvert = 22$		
h	a / $\lvert S\rvert$	100	175	250	100	175	250	100	175	250	100	175	250
Gurobi 3h													
0	0-20	6.6	3.4	2.2	18.9	7.7	5.8	41.4	20.4	16.2	67.4	42.2	28.1
	30-50	3.0	5.4	3.2	15.1	11.4	7.4	12.1	28.2	19.9	8.0	59.3	34.0
5	0-20	0.0	0.0	0.0	0.7	0.1	0.4	5.7	4.1	17.7	32.6	22.1	59.6
	30-50	0.1	0.0	0.1	6.9	0.3	0.2	24.5	6.8	7.4	59.1	17.1	35.1
10	0-20	0.0	0.0	0.0	1.0	0.6	2.4	6.7	26.7	37.2	29.1	121.8	433.3^{8}
	30-50	0.0	0.0	0.0	5.8	0.1	1.3	27.2	11.2	35.5	-	74.8	371.5
15	0-20	0.0	0.0	0.0	0.4	8.1^{2}	4.4	6.3	62.9	269.5^{9}	52.6	310.3^{2}	428.9^{26}
	30-50	0.0	0.0	0.0	6.8	1.1	2.5	20.4	30.7	164.9^{2}	-	136.7	440.8^{14}
ALNS													
0	0-20	7.6	5.9	3.7	21.8	25.2	40.4	47.5	52.7	71.3	110.3	71.9	112.4
	30-50	4.3	7.2	5.6	18.7	33.3	36.4	28.7	53.2	69.5	57.7	88.6	106.7
5	0-20	0.4	0.6	1.2	17.4	14.1	22.4	85.4	87.6	76.2	190.0	211.4	278.7
	30-50	2.5	0.9	1.4	25.9	16.8	21.0	64.7	75.7	85.6	93.5	179.7	184.5
10	0-20	0.6	1.1	1.4	22.5	16.5	49.4	101.5	213.0	265.8	225.4	433.6	432.8
	30-50	0.5	0.6	1.0	24.3	19.2	20.3	85.7	94.8	191.1	-	313.3	438.0
15	0-20	1.7	2.2	7.6	22.7	96.9	140.0	97.3	285.1	285.9	287.4	438.1	622.1
	30-50	0.6	0.2	2.1	43.9	24.9	78.1	84.0	219.3	274.3	-	438.2	433.8
Memetic													
0	0-20	7.1	3.8	2.7	18.8	8.8	6.2	37.0	17.0	13.0	57.6	34.9	21.9
	30-50	3.8	6.0	4.0	15.7	12.2	8.4	11.0	25.2	16.6	8.3	49.6	27.0
5	0-20	0.0	0.0	0.0	1.3	0.0	0.0	7.6	2.1	0.8	25.6	9.4	5.1
	30-50	1.4	0.0	0.1	10.8	0.6	0.2	25.7	5.1	1.4	49.0	16.8	8.2
10	0-20	0.0	0.0	0.0	0.1	0.0	0.0	2.3	0.3	0.1	15.0	3.8	2.2
	30-50	0.1	0.0	0.0	3.1	0.0	0.0	22.2	1.2	0.4	-	8.2	3.7
15	0-20	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.1	0.0	13.1	1.9	1.2
	30-50	0.0	0.0	0.0	0.9	0.0	0.0	10.9	0.5	0.2	-	4.4	1.7

Table 7: Mean break value of the randomly generated test instances for the different solution methods. Superscripts indicate the total number of instances for which the IP solver was stopped due to an out-of-memory error.

	LB	No obj.	Gur. 1h	Gur. 3h	ALNS	Memetic
ARTP	2,297	4,407,468	2,951	1,951	2,127	1,661
GPDI	1	5.7	0.9	0.8	1	1
Breaks	0	86.2	0	0	5.1	0

Table 8: Mean absolute gaps for the different solution methods on the artificial single round-robin test instances. Gaps are based on the best lower bound found by Gurobi run with 3 hours of computation time (see column 'LB'). The 'No obj.' column represents the mean gap when solving the IP model without considering any objective.

Theory and Applications, pages 331–340, Boston, MA, 2005. Springer.

[24] F. Galvin. The list chromatic index of a bipartite multigraph. *Journal of Combinatorial Theory, Series B*, 63:153–158, 1995.

[25] M. Grabau. Softball scheduling as easy as 1-2-3 (strikes you're out). *Interfaces*, 42:310–319, 2012.

[26] R. Häggkvist and J. Janssen. New bounds on the list-chromatic index of the complete graph and other simple graphs. *Comb. Probab. Comput.*, 6: 295–313, 1997.

[27] R. Hoshino and K. Kawarabayashi. Scheduling bipartite tournaments to minimize total travel distance. *J. Artif. Intell. Res.*, 42:91–124, 2011.

[28] T. Januario, S. Urrutia, C. R. Celso, and D. de Werra. Edge coloring: A natural model for sports scheduling. *Eur. J. Oper. Res.*, 254:1 – 8, 2016.

[29] G. Kendall, S. Knust, C. C. Ribeiro, and S. Urrutia. Scheduling in sports: An annotated bibliography. *Comput. Oper. Res.*, 37:1–19, 2010.

[30] D. E. King. Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res.*, 10:1755–1758, 2009.

[31] S. Knust. Scheduling non-professional table-tennis leagues. *Eur. J. Oper. Res.*, 200:358–367, 2010.

[32] S. Knust and M. von Thaden. Balanced home–away assignments. *Discrete Optim.*, 3:354–365, 2006.

[33] K. J. Kostuk and K. A. Willoughby. A decision support system for scheduling the Canadian Football League. *Interfaces*, 42:286–295, 2012.

[34] J. Kyngäs, K. Nurmi, N. Kyngäs, G. Lilley, T. Salter, and D. Goossens. Scheduling the Australian Football League. *J. Oper. Res. Soc.*, 68:973–982, 2017.

[35] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.*, 3:43–58, 2016.

[36] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997.

[37] P. Moscato and M. G Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel computing and transputer applications*, 1:177–186, 1992.

[38] G. L. Nemhauser and M. A. Trick. Scheduling a major college basketball conference. *Oper. Res.*, 46:1–8, 1998.

[39] Gurobi Optimization. Inc., Gurobi optimizer reference manual. `www.gurobi.com`, 2019.

[40] D. Pisinger and S. Ropke. *Large Neighborhood Search*, pages 399–419. Springer, Boston, MA, 2010. ISBN 978-1-4419-1665-5.

[41] R. Pollard and M.A. Gómez. Components of home advantage in 157 national soccer leagues worldwide. *International Journal of Sport and Exercise Psychology*, 12:218–233, 2014.

[42] R. Pollard, J. Prieto, and M.A. Gómez. Global differences in home advantage by country, sport and sex. *International Journal of Performance Analysis in Sport*, 17:586–599, 2017.

[43] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transport. Sci.*, 40:455–472, 2006.

	MA|PM	No local	(μ, λ)	$(\mu + \lambda)$	Parallel
ARTP	286	1,753 (<2e-16)	304 (<2e-16)	315 (<2e-16)	266 (< 2e-16)
GPDI	1.11	1.95 (<2e-16)	1.08 (0.71)	1.17 (9.7e-05)	1.01 (1.1e-08)
Breaks	0.14	9.71 (<2e-16)	0.08 (0.00063)	0.10 (0.31244)	0.09 (0.06541)

Table 9: Contribution of local search operators and population management strategy. Numbers outside brackets represent the mean absolute gap of a solution method on the real-life double round-robin problem instances. Numbers in brackets represent the p-values resulting from a pairwise Wilcoxon rank sum test comparing the mean absolute gap of an alternative with the MA|PM setting.

[44] U. Schauz. Proof of the list edge coloring conjecture for complete graphs of prime degree. *The Electronic Journal of Combinatorics*, 21:3–43, 2014.

[45] U. Schauz. The tournament scheduling problem with absences. *Eur. J. Oper. Res.*, 254:746 – 754, 2016.

[46] J. Schönberger, D. C. Mattfeld, and H. Kopfer. Automated timetable generation for rounds of a table-tennis league. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 277–284. IEEE, 2000.

[47] J. Schönberger, D. C. Mattfeld, and H. Kopfer. Memetic algorithm timetabling for non-commercial sport leagues. *Eur. J. Oper. Res.*, 153: 102–116, 2004.

[48] J. A. M. Schreuder. Combinatorial aspects of construction of competition Dutch professional football leagues. *Discrete Appl. Math.*, 35:301–312, 1992.

[49] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *J. Comput. Phys.*, 159:139–171, 2000.

[50] V. Scoppa. Fatigue and team performance in soccer: Evidence from the FIFA World Cup and the UEFA European Championship. *J. Sport. Econ.*, 16:482–507, 2015.

[51] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer.

[52] K. Sörensen and M. Sevaux. MA|PM: memetic algorithms with population management. *Comput. Oper. Res.*, 33:1214 – 1225, 2006.

[53] W. Suksompong. Scheduling asynchronous round-robin tournaments. *Oper. Res. Lett.*, 44:96–100, 2016.

[54] T. A. M. Toffolo, J. Christiaens, F. C. R. Spieksma, and G. Vanden Berghe. The sport teams grouping problem. *Ann. Oper. Res.*, 2017.

[55] M. A. Trick. Integer and constraint programming approaches for round-robin tournament scheduling. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, pages 63–77, Berlin, Heidelberg, 2003. Springer.

[56] S. Urrutia and C. C. Ribeiro. Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Appl. Math.*, 154: 1932–1938, 2006.

[57] D. Van Bulck and D. Goossens. Time-relaxed round-robin problem instances with availability constraints – website. `www.sportscheduling.ugent.be/research.php`, 2019.

[58] D. Van Bulck, D. Goossens, J. Schönberger, and M. Guajardo. Robinx: A three-field classification and unified data format for round-robin sports timetabling. *Eur. J. Oper. Res.*, 280(2):568 – 580, 2019.

[59] D. Van Bulck, D. R. Goossens, and F. C. R. Spieksma. Scheduling a non-professional indoor football league: a tabu search based approach. *Ann. Oper. Res.*, 275:715–730, 2019.

[60] V. G Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.