Scheduling Double Round-Robin Sports Tournaments

Carlos Lamas-Fernandez · Antonio Martinez-Sykora · Chris N Potts

Abstract This paper considers the problem of assigning matches to time slots in a double-round robin sports tournament. An integer linear programming (ILP) model is developed which includes a variety of hard and soft constraint that are likely to be encountered when scheduling professional football/soccer league fixtures. The solution methodology used is a matheuristic that fixes a large number of variables in the ILP model at each iteration to enable a solution to be generated relatively quickly. In this fix-and-relax approach, different methods are used to determine which variables are to be fixed. Computational results are given for the instances having 16, 18 and 20 teams that form an international timetabling competition on sports timetabling (ITC2021). The main findings are that the matheuristic finds solutions for most ITC2021 instances relatively quickly with all hard constraints satisfied, and generates many best-known solutions for these instances.

Keywords Sports Tournament Scheduling · Double Round-Robin · Matheuristic

Mathematics Subject Classification (2010) 90

Carlos Lamas-Fernandez
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
Southampton Business School
University of Southampton
E-mail: C.Lamas-Fernandez@soton.ac.uk

Antonio Martinez-Sykora
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
Southampton Business School
University of Southampton
E-mail: A.Martinez-Sykora@soton.ac.uk

Chris N Potts
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
School of Mathematical Sciences
University of Southampton
E-mail: C.N.Potts@soton.ac.uk

1 Introduction

Research on designing algorithms for scheduling sports competitions has taken place for almost fifty years, although interest in this research topic has increased during the last twenty years. For an overview of the major contributions, we refer to Knust [5]. In this paper, we address the scheduling of double round-robin tournaments of the type used in many European and South American football/soccer leagues where each team in the competition plays one home game and one away game against every other team. Goossens and Spieksma [4] provide an overview of the structure of the main football leagues in Europe. As pointed out by Van Bulck et al. [6], most publications focus on designing an algorithm that is suited to the constraints imposed by a particular league. As a consequence, computational work assessing the relative performance of such algorithms is scarce. This has motivated a unified data format for round-robin sports timetabling by Van Bulck et al. [7] that facilitates the comparison of algorithmic approaches. It has also motivated the International Timetabling Competition on Sports Timetabling, named ITC2021 [8], that "aims to stimulate the development of solvers for the construction of round-robin timetables". This paper reports on the solver developed by the authors for the double round-robin sports scheduling problem (DRRSSP), and provides computational results in the form of objective function values for the 45 instances on which the result of ITC2021 is based.

The double round-robin tournaments of ITC2021 are *compact*, meaning that in each time slot every team has exactly one match. We distinguish between *phased* and *unphased* double round-robin tournaments. In a phased tournament, the matches played in the first half of the slots comprise a single round-robin tournament, and similarly for the matches in the second half of the slots, whereas an unphased tournament has no such constraints on the matches. A *break* occurs if a team plays at home in some slot having played at home in its previous slot (*home break*), or if a team plays away in some slot having played away in its previous slot (*away break*).

There are different classes of constraints on the DRRSSP. These arise due to the interests of multiple stakeholders. The football clubs competing in a tournament are concerned about maximizing revenue by having their home matches being played in slots when more fans are likely to be able to watch the game. They also prefer schedules in which home and away matches alternate, thereby reducing the number of breaks. Teams that use the same stadium clearly create constraints, and other events occurring within the vicinity of the stadium in certain time slots may prevent a home match being assigned to these slots. The police are responsible for safety outside of the ground, which may impose constraints on the number of teams playing in the same city in the same slot. TV companies invest significantly in gain broadcasting rights to the matches, and may prefer schedules having the most high-profile matches spread throughout the season.

This aim of our study is to propose a new algorithm for creating schedules for the DRRSSP, and to evaluate this approach on the 45 instances of

the ITC2021 competition. In Section 2, we provide a formal description of the DRRSSP that we are addressing, and Section 3 contains an integer programming formulation of the problem. Section 4 describes our proposed matheuristic, which takes the form of a fix-and-relax procedure. Computational results obtained by applying our matheuristic to the 45 instances provided within ITC2021 are presented and discussed in Section 5. Lastly, Section 6 contains some concluding remarks.

2 Problem Description

For the DRRSSP, it is required to design a round-robin tournament that allocates matches to time slots. However, Van Bulck et al. [6] observe from various studies reported in the literature that there can be various constraints that affect how the matches are scheduled.

Let n denote the number of teams competing in the double round-robin tournament, where n is even. Further, let $T = \{1, \ldots, n\}$ be the set of all teams. For each pair of teams $i, j \in T$, where $i < j$, there is match (i, j) for which team i plays a home game against team j and a match (j, i) for which team i plays an away match against team j. Thus, each team plays $n - 1$ home games at its own venue and $n - 1$ away games at its opponent's venue. The tournament has a set S of time slots for the matches. We assume that the minimum number of time slots is used so that $S = \{1, \ldots, 2n - 2\}$, which produces a *compact* tournament. If the matches that are played in slots $1, \ldots, n - 1$ define a single round-robin tournament, then the matches in slots $n, \ldots, 2n-2$ also define a single round-robin tournament, and such a structure is *phased* tournament. For a phased tournament, we define $S' = \{1, \ldots, n-1\}$ to be the set of slots used for the first phase.

There are structural constraints that ensure that the matches assigned to time slots satisfy the conditions of a double round-robin tournament, with additional constraints added if the tournament is phased. However, there are many other types of constraints within ITC2021, as listed below, which can either be hard or soft.

Capacity constraints: within a given set of time slots, a team is forced to play at home or away, and the total number of matches played by a team or by a set of teams has an upper limit.

Game constraints: given a set of time slots and a set of matches, the number of matches assigned to these time slots has an upper limit and a lower limit.

Break constraints: within a given set of time slots, there is an upper limit on the total number of breaks for a given set of teams.

Fairness constraints: within each of a given set of slots, there is an upper limit on the largest difference in the number of home games played between each pair of teams within a given set.

Separation constraints: for a given set of teams, there is a lower limit on the gap between home and away matches between each pair of teams in this set.

A solution of the DRRSSP has, for each soft constraint, a non-negative deviation that specifies the number of units of violation of the constraint. Also, each soft constraint has an associated weight that represents the penalty per unit violation of the constraint. The objective of the problem is to design a double round-robin tournament in which all hard constraints are satisfied so that the sum of weighted deviations for all soft constraints is minimized.

3 Integer Linear Programming Model

Our integer linear programming (ILP) model has variables and structural constraints that are widely used in round-robin sports scheduling, such as in the study of Durán et al. [3]. The key parameters used in our ILP are the set of teams T and the set of slots S, as defined in Section 2, with S' representing the set containing the first half of the slots.

The variables in our integer programming model that define the structure of the tournament are

$$x_{ijs} = \begin{cases} 1 & \text{if match } (i,j) \text{ is played in slot } s, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i,j \in T,\ \forall s \in S$$

However, some tournament specifications impose constraints on numbers of breaks that are allowed. Thus, we introduce additional variables

$$b_{is}^{\mathrm{HA}} = \begin{cases} 1 & \text{if } i \text{ has a home or away break in slot } s, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i \in T,\ \forall s \in S \setminus \{1\}$$

In some cases, it is necessary to differentiate between a home break and an away break. Thus, if $b_{is}^{\mathrm{HA}} = 1$, then $b_{is}^{\mathrm{H}} = 1$ or $b_{is}^{\mathrm{A}} = 1$ depending on whether a home break or an away break occurs for team i in slot s. Also, there is a relationship $b_{is}^{\mathrm{HA}} = b_{is}^{\mathrm{H}} + b_{is}^{\mathrm{A}}$ for all $i \in T$ and $s \in S \setminus \{1\}$. Also, when there are constraints on the separation in terms of the number of slots between the two matches played by a pair of teams i and j, it is useful to introduce the variables

$$y_{ij} = \begin{cases} 1 & \text{if match } (i,j) \text{ occurs before match } (j,i), \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i,j \in T$$

The following subsections provide the classes of constraints that are included in the model. There are some structural constraints given below in Section 3.1 which must be satisfied in a double round-robin tournament. The remaining constraints are non-structural.

Each non-structural constraint has an index by which it is identified. Associated with most constraints c is a threshold value t_c, which is the maximum value that some linear combination of the x_{ijs}, b_{is}^{H}, b_{is}^{A}, b_{is}^{HA} and y_{ij} variables can achieve without incurring a penalty. In such cases, the right-hand side of

the constraint is set to $t_c + d_c$, which d_c is an integer deviation variable for constraint c. The remaining constraints c have a threshold value t_c, which is the minimum value that some linear combination of the x_{ijs}, b^{H}_{is}, b^{A}_{is}, b^{HA}_{is} and t_{ij} variables can achieve without incurring a penalty. For these constraints, the right-hand side of the constraint is set to $t_c - d_c$, which d_c is an integer deviation variable for constraint c. If a constraint c of either type is hard, we introduce $d_c = 0$ as a further constraint.

3.1 Structural constraints

The structural constraints on a double round-robin tournament ensure that the variables are assigned values that create a valid tournament. The structural constraints are all hard. The first set of constraints below ensure that within each slot s team i either has a home game or an away game against some other team j. The second constraints impose the condition that a match (i, j) for each pair of teams i and j appears in exactly one slot. The third set of constraints, which are applied only when the DRRSSP is phased, force all pairs of teams i and j to play exactly one match in the first half of the time slots S', and consequently exactly one match in the second half of the time slots $S \setminus S'$.

$$\sum_{j \in T \setminus \{i\}} (x_{ijs} + x_{jis}) = 1 \qquad \forall i \in T,\ \forall s \in S \quad (1)$$

$$\sum_{s \in S} x_{ijs} = 1 \qquad \forall i, j \in T \quad (2)$$

$$\sum_{s \in S'} (x_{ijs} + x_{jis}) = 1 \qquad \forall i, j \in T \quad (3)$$

The constraints linking the b^{H}_{is} and b^{A}_{is} variables with the x_{ijs} variables are

$$\sum_{j \in T} (x_{ijs} + x_{i,j,s-1)}) \leq b^{\mathrm{H}}_{is} + 1 \qquad \forall i \in T, s \in S \setminus \{1\} \quad (4)$$

$$\sum_{j \in T} (x_{jis} + x_{j,i,s-1}) \leq b^{\mathrm{A}}_{is} + 1 \qquad \forall i \in T, s \in S \setminus \{1\} \quad (5)$$

Further, the constraints linking y_{ij} with the x_{ijs} variables are

$$\sum_{s \in S} s(x_{jis} - x_{ijs}) \leq M y_{ij} \qquad \forall i, j \in T \quad (6)$$

$$\sum_{s \in S} s(x_{ijs} - x_{jis}) \leq M(1 - y_{ij}) \qquad \forall i, j \in T \quad (7)$$

where M is a constant that satisfies the condition $M \geq |S| - 1$.

3.2 Capacity constraints

The DRRSSP has four types of capacity constraints. These sets of constraints are denoted by CA_1, CA_2, CA_3 and CA_4.

There is a set $\mathrm{CA}_1 = \mathrm{CA}_1^{\mathrm{H}} \cup \mathrm{CA}_1^{\mathrm{A}}$ of home and away capacity constraints of type 1. Each constraint c of CA_1 is specified by a team i_c, a set of slots S_c and a threshold value t_c. The home capacity constraint is

$$\sum_{j \in T} \sum_{s \in S_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_1^{\mathrm{H}},\ i = i_c \qquad (8)$$

while the corresponding constraints for $c \in \mathrm{CA}_1^{\mathrm{A}}$ are identical except that x_{ijs} is replaced by x_{jis}.

The capacity constraints c of types 2 and 3 are each specified by a team i_c, a set of teams T_c and a set of slots S_c for type 2 constraints. Also, there are constraints that refer to home games, away games, and home-away games that are index by H, A and HA. Thus, the constraint sets are $\mathrm{CA}_t = \mathrm{CA}_t^{\mathrm{H}} \cup \mathrm{CA}_t^{\mathrm{A}} \cup \mathrm{CA}_t^{\mathrm{HA}}$ for types $t = 2$ and $t = 3$. The home constraints for type 2 and type 3 are

$$\sum_{j \in T_c} \sum_{s \in S_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_2^{\mathrm{H}},\ i = i_c \qquad (9)$$

$$\sum_{j \in T_c} \sum_{s=k+1}^{k+I_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_3^{\mathrm{H}},\ i = i_c,\ \forall k \in K \qquad (10)$$

where $K = \{0, \ldots, |S| - I_c\}$ and I_c is the length of an interval defining the slots to be considered for type 3 constraints. The corresponding constraints for $c \in \mathrm{CA}_t^{\mathrm{A}}$ and $c \in \mathrm{CA}_t^{\mathrm{HA}}$ for $t = 2$ and $t = 3$ are identical except that x_{ijs} is replaced by x_{jis} for away constraints and are x_{ijs} is replaced by $x_{ijs} + x_{jis}$ for home-away constraints.

Type 4 capacity constraints are specified by two sets of teams T_{c1} and T_{c2}, and a set of slots S_c. The constraint set is $\mathrm{CA}_4 = \mathrm{CA}_4^{\mathrm{H}} \cup \mathrm{CA}_4^{\mathrm{A}} \cup \mathrm{CA}_4^{\mathrm{HA}}$. Moreover, the home constraint set comprises$\mathrm{CA}_4^{\mathrm{H}} = \mathrm{CA}_{4a}^{\mathrm{H}} \cup \mathrm{CA}_{4b}^{\mathrm{H}}$ where $\mathrm{CA}_{4a}^{\mathrm{H}}$ and $\mathrm{CA}_{4b}^{\mathrm{H}}$ are defined by

$$\sum_{i \in T_{c1}} \sum_{j \in T_{c2}} \sum_{s \in S_c} x_{ijs} \leq t_c + d_c \qquad \forall c \in \mathrm{CA}_{4a}^{\mathrm{H}} \qquad (11)$$

$$\sum_{i \in T_{c1}} \sum_{j \in T_{c2}} x_{ijs} \leq t_c + d_c \qquad \forall c \in \mathrm{CA}_{4b}^{\mathrm{H}},\ \forall s \in S_c \qquad (12)$$

The away, and home-away constraints are similarly partitioned, with the replacement of x_{ijs} by x_{jis} and $x_{ijs}+x_{jis}$, respectively, providing the constraints.

3.3 Game constraints

The set of game constraints is denoted by GA. For each constraint $c \in \text{GA}$, a set S_c of slots and a set G_c of games are specified. A game in which team i plays at home against team j is denoted by (i, j). The constraints are

$$\sum_{(i,j)\in G_c} \sum_{s\in S_c} x_{ijs} \leq t_c + d_c \qquad \forall c \in \text{GA} \qquad (13)$$

$$\sum_{(i,j)\in G_c} \sum_{s\in S_c} x_{ijs} \geq t'_c - d_c \qquad \forall c \in \text{GA} \qquad (14)$$

where t'_c is a lower limit on the number of games from G_c that are played in the slots of S_c. For each $c \in \text{GA}$, there are lower and upper bound constraints, although a positive d_c value for one of these constraints implies that the other constraint is satisfied as a strict inequality.

3.4 Break constraints

There are two types of constraints that limit numbers of breaks. The set of type 1 breaks is denoted by $\text{BR}_1 = \text{BR}_1^{\text{H}} \cup \text{BR}_1^{\text{A}} \cup \text{BR}_1^{\text{HA}}$, where home breaks, away breaks and home-away breaks are considered. Each type 1 break constraint c specifies a team i_c and a set of slots S_c. The home break constraints of type 1 are

$$\sum_{s\in S_c} b_{is}^{\text{H}} \leq y_c + d_c \qquad \forall c \in \text{BR}_1^{\text{H}}, i = i_c \qquad (15)$$

while the away and home-away break constraints of BR_2^{A} and BR_3^{HA} are of a similar form.

A set BR_2 specifies the type 2 break constraints. Each constraint $c \in \text{BR}_2$ specifies sets S_c of slots T_c of teams. These type 2 break constraints are

$$\sum_{i\in T_c} \sum_{s\in S_c} b_{is}^{\text{HA}} \leq t_c + d_c \qquad \forall c \in \text{BR}_2 \qquad (16)$$

3.5 Fairness constraints

Fairness constraints aim to ensure that pairs of teams play approximately the same number of home games at the end of selected slots. The set of fairness constraints is denoted by FA. Each constraint $c \in \text{FA}$ specifies a pair of teams i_c and i'_c and a set of slots S_c. The constraints are

$$\sum_{j\in T} \sum_{s=1}^{\hat{s}} (x_{ijs} - x_{i'js}) \leq t_c + d_c \qquad \forall c \in FA,\ i = i_c,\ i' = i'_c,\ \forall \hat{s} \in S_c \qquad (17)$$

3.6 Separation constraints

A set SE of separation constraints aims at avoiding the two matches between a pair of teams being too close together. Each constraint $c \in \mathrm{SE}$ specifies a set T_c from which the pairs of teams are selected. The constraints are

$$\sum_{s \in S} s(x_{ijs} - x_{jis}) \geq t_c + 1 - d_c - My_{ij} \qquad \forall c \in SE,\ \forall i, j \in T_c \qquad (18)$$

3.7 Objective function

Let $C = \mathrm{CA}_1 \cup \mathrm{CA}_2 \cup \mathrm{CA}_3 \cup \mathrm{CA}_4 \cup \mathrm{GA} \cup \mathrm{BR}_1 \cup \mathrm{BR}_2 \cup \mathrm{FA} \cup \mathrm{SE}$ be the set of all the constraints in the problem, excluding the structural constraints defined in Section 3.1. Let $C = \tilde{C} \cup \bar{C}$ where $\tilde{C}$ is the set of soft and $\bar{C}$ is the set of hard constraints. Let w_c be the given unit penalty for constraint $c \in \tilde{C}$. Then the objective function can be written as

$$\text{Min} \sum_{c \in \tilde{C}} w_c d_c \qquad (19)$$

Recall that we set $d_c = 0$ for all $c \in \bar{C}$ to guarantee feasibility.

4 Matheuristic

Our Matheuristic algorithm relies on the ILP model described in Section 3. The instances created for ITC2021 are very challenging, and the ILP model is unable to solve them with Gurobi (version 9.0) using reasonable computational effort, mainly because of the size in terms of numbers of teams, soft constraints and hard constraints. Therefore, the rationale in this study is to solve smaller problems so that we can expect a competitive behaviour from the solvers that are currently available.

The fix-and-relax matheuristic approach (also known as relax-and-fix) provides a framework for producing solutions for ILPs by solving a series of smaller problems. These smaller problems are created by fixing many of the variables and then solving the ILP for the variables that are not fixed. The early research on fix-and-relax concentrated on lot-sizing problems and was initiated by Dillenberger et al. [2]. More recently, fix-and-relax has been applied in sports scheduling to the traveling umpire problem (TUP) by de Oliveira et al. [1]. In the traveling umpire problem, matches in a round-robin tournament are specified as an input, and it is required to allocate an umpire to each match so that the total distance traveled by the umpires is minimized. In the study of de Oliveira et al., there are binary assignment variables that define the allocation of umpires. The high-quality of the solutions obtained for the TUP suggests that a fix-and-relax approach could be successful for our DRRSSP.

One obvious strategy is to consider as a first step only the hard constraints and check whether the ILP in which the soft constraints are ignored is solvable with a reasonable amount of computational effort. However, we have identified that this relaxation is not sufficient for the instances provided in the ITC2021 competition because there are no instances for which a feasible solution can be obtained. Consequently, our proposed strategy is based on fixing a subset of the variables to specific values, where the method of variable fixing takes into account the features of the current solution. The other variables will remain as decision variables, thereby producing a fix-and-relax approach. Based on this idea, we introduce five different neighbourhoods to be used at the two stages of our algorithm. In the first stage, we aim to find a feasible solution ignoring the soft constraints, while the second stage considers the full model. In the first stage we propose a Variable Neighbourhood Search (VNS), and in the second stage we propose a Variable Neighbourhood Descent (VND) using the same neighbouring structure but adding a multi-start feature.

We define ILP_c to the constrained new ILP model in which we fixed some of the variables to predefined values. We also denote by $\hat{x}_{ijs}$ by the value of each variable x_{ijs} in the current solution. In Section 4.1 we define the neighbourhoods, while Sections 4.2 and 4.3 then explain the VNS and VND approaches that are used in the corresponding stages of the algorithm.

4.1 Neighbourhoods

We use the following neighbourhoods (N1-N5). All of these neighbourhoods are based on a fix-and-relax approach using the ILP, where several ILP models are solved by Gurobi with a time limit imposed (in our experiments, we ran tests with 30, 60 and 600 second per model).

- N_1 **Slots**. In this neighbourhood we select a subset of slots $\bar{S} \subseteq S$ and then we fix all the other slots as in the current solution, i.e, $x_{ijs} = \hat{x}_{ijs}$ for all $i, j \in T$ and $s \in S \setminus \bar{S}$. This neighbourhood requires a new parameter for the algorithm, n_1, which is the number of slots to be selected in S. We perform as many iterations as constraints we have violated in the model, and for each constraint violated we select the slots where there is any match scheduled that contributes to the Left Hand Side (LHS) of the constraint. We then randomly add other slots until n_1 slots are chosen.
- N_2 **Teams**. In this neighbourhood, we select a subset of teams $\bar{T} \subseteq T$ and then we fix all the matches for all pairs of teams, except the pairs $i, j \in \bar{T}$, which remain as variables. Therefore, $x_{ijs} = \hat{x}_{ijs}$ and $x_{jis} = \hat{x}_{jis}$, $\forall i \in T, j \in T \setminus \bar{T},\ s \in S$. As in N1, this neighbourhood also requires a new parameter, n_{2s}, which is the number of teams that will be considered in $\bar{T}$. Similarly to N1, for each constraint we select in $\bar{T}$ the teams that are contributing to the left-hand side of the inequalities violated in the current iteration. We then randomly add other teams until n_2 teams are chosen.
- N_3 **Rows and Columns**. We select a subset of slots $\bar{S} \subseteq S$ and a subset of teams $\bar{T} \subseteq T$, and then we fix all the matches that are not scheduled on

any slot in $\bar{S}$ and both teams are not in $\bar{T}$, i.e, $x_{ijs} = \hat{x}_{ijs}, \forall i, j \in T \setminus \bar{T},\ s \in S \setminus \bar{S}$. Slots in $\bar{S}$ are selected in a similar way as in N1, but we select only $n_1/2$ slots, and also we always select 2 teams in $\bar{T}$, and these two teams are always the ones contribute more towards the violation of the current violated constraint being analysed.

N_4 **Phased** (only for phased tournaments). We fix one half of the competition and we optimise the other half, i.e, we solve first the model where $x_{ijs} = \hat{x}_{ijs}$, $\forall s \in \{1, \ldots, |S|/2\}$, and then the model in which $x_{ijs} = \hat{x}_{ijs}$, $\forall s \in \{|S|/2+1, \ldots, |S|\}$. We solve only two models every time we use this neighbourhood.

N_5 **Home and away**. In this neighbourhood, we allow home and away matches between the same pair of teams to be swapped. Specifically, we fix $x_{ijs} = 0$, for all $i, j \in T$ and $s \in S$ such that $\hat{x}_{ijs} + \hat{x}_{jis} = 0$. This neighbourhood has no random selection, but the resulting model is generally challenging. Thus, in our computer experiments, Gurobi rarely proves optimality, and generally stops due to the time limit condition.

4.2 Checking feasibility - VNS (Hard only)

In this stage we consider the ILP where all the soft constraints except the BR2 are ignored. If a feasible solution is found, it is then used for the second stage (Section 4.3). A high-level pseudo-code is presented in Algorithm 1.

An initial solution where many of the constraints are violated is found by solving the ILP model only with the structural constraints (see Section 3.1) and ignoring all the other constraints. We refer to that solution as Sol_0, which generally has many other constraints that are violated. We then compute the value of total deviation that corresponds to solution Sol_0, and set F_0 be that value (line 2 Algorithm 1).

In line 7 we apply all the neighbourhoods until no improvement is found, where we regard the solution as being a local optimum. In line 11, if the solution is still not feasible, we then increase the coefficients in the objective function of all the d_c variables, with the exception of the BR2 (break 2 constraints), where we increase the coefficients of the b_{is} variables in the objective function if $\hat{b}_{is} = 1$. In lines 12-13 we check whether the solution is feasible.

For any d_c variables that take a value of 0 but have a positive coefficient in the objective function due to the previous iterations, we decrease their weight by taking into account the slack of the corresponding constraint in the model. If there is a positive slack then we decrease the current weight by one unit.

4.3 Optimising with soft constraints

In this stage we consider the full ILP model (with both hard and soft constraints), and the input is the final solution obtained by algorithm described in Section 4.2. A pseudo-code is provided in Algorithm 2. We first reset all the

Algorithm 1 VNS (Hard only)

```
1: procedure VNS(n1, n2)                    ▷ n1 and n2 do not change in this part
2:     Sol0 ← Compute initial solution (with objective function F0)
3:     i ← 0
4:     while Improvement do
5:         i ← i + 1
6:         Soli ← Sol0
7:         Apply N1-N5 in Soli
8:         if Improvement found then        ▷ with the current objective function being used
9:             Update Soli
10:        else
11:            Change weights on the objective function
12:        if Solution is feasible then
13:            Stop and return feasible solution Soli
```

weights on the objective function as follows. The weight on the d_c variables, where $c \in \tilde{C}$ is given by the penalty of the constraints, and the weight of all the other d_c variables related to hard constraints is set to 100. We identified that this value is sufficiently large to converge to better solutions without violating too many hard constraints, which may lead to difficulties on recovering feasibility. However, this approach indeed allows to visit infeasible solutions during the search process. Therefore, it is important to check that the solution we check in line 10 of Algorithm 2 is indeed feasible.

We start with the objective function defined by the solution obtained by the VNS (hard-only) algorithm, which corresponds to the actual penalty of the solution. Next we apply the N1-N5 neighbourhoods until we obtain a local optimum, at which stage we increase the two parameters of the algorithm, n_1 and n_2 (see Section 4.1) by one, until Gurobi cannot solve the model efficiently. In some instances where there are not many constraints, the resulting ILP models are easier to solve and, therefore, higher values for n_1 and n_2 are explored (up to $n_1 = 30$ and $n_2 = 14$ in the best cases). For the larger or more complex instances resulting in more challenging ILPs, the model becomes intractable with $n_1 = 18$ and $n_2 = 8$.

After performing some computational experiments and exploring other strategies, we found that developing a multi-start algorithm is able to produce the best quality solutions. We execute multiple runs (n_s) with the same initial solution (by introducing randomness in N1-N3) or by different initial solutions obtained by the VNS (hard-only) algorithm.

5 Computational experiments

For the computational experiments we have used 2.6GHz Intel Sandybridge processors, and each run was performed by 4 CPUs with 16GB of memory. We have used Gurobi (version 9.0) to solve the ILP models and we run the algorithm three times with 30, 60 and 600 second per model. In stage 1 of the algorithm (VNS), we run the algorithm until a feasible solution is found.

Algorithm 2 VND (hard + soft) - multistart

```
1: procedure VND(n_s)                          ▷ n_s is the number of muti-start runs
2:     for i = 1..., n_s do
3:         Sol_i ← One random solution obtained from VNS (Hard only) (Algorithm 1).
4:         n_1 = 5 and n_2 = 4
5:         exploring = True
6:         while exploring do
7:             Apply N1-N5 in Sol_i
8:             if Improvement found then          ▷ Objective function does not change
9:                 Update Sol_i
10:                if best solution improved then
11:                    Update best solution
12:            n_1 = n_1 + 1
13:            if Gurobi cannot find feasible solution of the resulting ILP then
14:                n_1 = 5
15:                n_2 = n_2 + 1
16:                if Gurobi cannot find feasible solution of the resulting ILP then
17:                    exploring = False
18:    Return best solution
```

At this stage we also considered adding the soft BR2 (break 2) constraints in order to begin stage 2 with a solution that already has a low number of breaks.

The computation time needed to find a feasible solution is shown in the first three columns of Table 1. It is worth highlighting that for almost all instances it is relatively quick to find a feasible solution, although in some instances it is more challenging and many iterations of the VNS are required to find a feasible solution, especially the instance Middle 2.

In stage 2 of the algorithm (Section 4.3), for each run of the algorithm we set $n_s = 60$ (multi-start runs), and we initially set $n_1 = 8$ and $n_2 = 6$. The computation time of one single run of the algorithm strongly depends on the instance that we are solving, but the largest amount of time on a single run was up to 6 days. Our best solutions obtained are reported in Table 1. The instances solved are divided into three sets of 15 instances each (Early, Middle and Late), which were released at different times during the ITC2021 competition.

Table 1 Best results obtained by the proposed algorithm (VNS+VND) for the ITC2021 instances. The second column (TTF) represents the time to achieve feasibility (in hours) and the third column (ITC2021) presents the best solution known for the instance at the end of the competition. The last column presents the gap from the best VNS+VND solution to the best known.

Instance		TTF (h)	VNS+VND	ITC2021	Gap
Early					
	1	< 1	362	362	0%
	2	< 1	222	160	28%
	3	< 1	1052	1012	4%
	4	< 24	536	512	4%
	5	< 24	3127	3127	0%
	6	< 1	3714	3352	10%
	7	< 1	4763	4763	0%
	8	< 1	1114	1114	0%
	9	< 1	108	108	0%
	10	< 72	3400	3400	0%
	11	< 1	4436	4436	0%
	12	< 1	510	380	25%
	13	< 1	121	121	0%
	14	< 1	47	4	91%
	15	< 1	3368	3368	0%
Middle					
	1	< 24	5177	5177	0%
	2	> 100	7381	7381	0%
	3	< 24	9800	9701	1%
	4	< 1	7	7	0%
	5	< 1	494	413	16%
	6	< 1	1275	1125	12%
	7	< 1	2049	1784	13%
	8	< 1	129	129	0%
	9	< 1	450	450	0%
	10	< 1	1250	1250	0%
	11	< 1	2608	2511	4%
	12	< 1	923	911	1%
	13	< 1	282	253	10%
	14	< 1	1323	1172	11%
	15	< 1	965	495	49%
Late					
	1	< 1	1969	1969	0%
	2	< 1	5400	5400	0%
	3	< 1	2369	2369	0%
	4	< 1	0	0	-
	5	< 3	2218	1939	13%
	6	< 1	923	923	0%
	7	< 1	1652	1558	6%
	8	< 1	934	934	0%
	9	< 1	563	563	0%
	10	< 3	2031	1988	2%
	11	< 1	226	207	8%
	12	< 1	3912	3689	6%
	13	< 1	2110	1820	14%
	14	< 1	1363	1206	12%
	15	< 1	40	20	50%

The proposed algorithm obtained the best results in 22 out of 45 instances by the end of competition, and in further 7 instances the best known result is within 6% of the solution obtained by the proposed algorithm.

6 Concluding remarks

In this study, we propose a novel matheuristic algorithm having two stages to solve the DRRSSP. In the first stage, we address the feasibility problem by using a VNS framework, where a second stage we optimizes the soft constraint violations by using a multi-start algorithm with a VND. Both the VNS and the VND use the same neighbourhood structure that combines five different neighbourhoods. The results obtained shows that both stages of the algorithm perform well, being able to prove optimality in one instance (Late 4, with an objective value of 0) and finding a feasible solution in all 45 instances of the ITC2021 challenge competition.

Acknowledgements The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work. Useful comments by two anonymous referees on an earlier draft of our paper have helped in creating this improved version.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. de Oliveira L, de Souza CC, Yunes T: Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. *European Journal of Operational Research* **236** 592–60 (2014)
2. Dillenberger C, Escudero LF, Wollensak A, Zhang W: On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research* **75** 275-286 (1994)
3. Durán G, Guajardo M, Miranda J, Sauré D, Souyris S, Weintraub A, Wolf R: Scheduling the Chilean soccer league by integer programming. *Interfaces* **37(6) 539–552 (2007)**
4. **Goossens DR, Spieksma FCR: Soccer schedules in Europe: An overview.** *Journal of Scheduling* **15 641-651 (2011)**
5. **Knust, S: Classification of sports scheduling literature. http://www2.informatik.uni-osnabrueck.de/knust/sportssched/sportlit_class/. Accessed 07/06/2021.**
6. **Van Bulck D, Goossens D, Schönberger J, Guajardo M: RobinX: A three-field classification and unified data format for round-robin sports timetabling.** *European Journal of Operational Research* **280(2), 568-580 (2020)**
7. **Van Bulck D, Goossens D, Beliën, J, Davari, M: ITC—Sports Timetabling Problem Description and File Format. https://www.sportscheduling.ugent.be/ITC2021/ (2021)**
8. **Van Bulck D, Goossens D, Beliën, J, Davari, M: The Fifth International Timetabling Competition (ITC 2021): Sports Timetabling.** *Proceedings of MathSport International 2021 Conference, MathSport* **pp. 117-122. (2021)**