Journal of Scheduling (2022) 25:301–319
https://doi.org/10.1007/s10951-022-00740-y

Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021

Roberto Maria Rosati[1] · Matteo Petris[2] · Luca Di Gaspero[1] · Andrea Schaerf[1]

Accepted: 3 May 2022 / Published online: 28 June 2022

Abstract
We describe the solver that we developed for the Sports Timetabling Competition ITC2021, a three-stage simulated annealing approach, that makes use of a portfolio of six different neighborhoods. Five of these neighborhoods are taken from the literature on round-robin tournament scheduling, whereas the last one, denoted as PartialSwapTeamsPhased, is a novel contribution and it is specifically designed for the phased version of the problem. We perform a comprehensive and statistically principled tuning procedure to find the best combination of parameters for the competition instances. We dedicate specific focus to evaluate the contribution given by the new neighborhood PartialSwapTeamsPhased, which yielded better results on most phased instances. Overall, the final outcome is that the three-stage simulated annealing solver is able to find a feasible solution on 44 out of 45 instances and ranked second in both the first competition milestone and the final round. We also propose an Integer Linear Programming model implemented in CPLEX, which, unfortunately, did not produce significant results on the instances of the competition.

Keywords Sport Scheduling · Simulated Annealing · ITC2021 · Timetabling

1 Introduction

Sports timetabling is an active research field, mainly due to the commercial interest in the maximization of fan attendance (in person or remotely) to sport events. Among the various possible structures for sport competitions, the round-robin tournament, where each team plays against each other, is the most frequently used for most team sports.

Many variants of the round-robin tournament problem have been discussed in the literature. We consider here the version proposed for the International Timetabling Competition ITC2021 (Van Bulck et al., 2021): a double round-robin tournament (all teams play with each team twice), which takes into account a very rich set of constraints and objectives collected from real-world cases.

All versions of this problem have in common the fact of being generally difficult to solve in practice. In fact, it is often hard to find optimal (or near-optimal) solutions already for instances of relatively small sizes, i.e., 16–20 teams, which is indeed the typical size of national championships.

As mentioned above, the ITC2021 problem considers a large set of constraints and objectives, also known as *hard* and *soft* constraints, respectively. This formulation has the peculiarity that every single specific constraint can be stated as either hard or soft. Another characteristic of the ITC2021 formulation is that it has abandoned the classical *mirrored* structure in which the second leg is identical to the first one, with home and away positions swapped. That is, the structure of ITC2021 instances is either completely free or *phased*. The latter imposes that each team meets all other teams in each leg, but not necessarily in the same order.

In this paper, we describe the search method employed in our participation in the ITC2021. It is a three-stage simulated annealing approach that uses a portfolio of six different neighborhood structures. Five of them are classical ones,

✉ Roberto Maria Rosati
robertomaria.rosati@uniud.it

Matteo Petris
matteo.petris@inria.fr

Luca Di Gaspero
luca.digaspero@uniud.it

Andrea Schaerf
andrea.schaerf@uniud.it

[1] DPIA, University of Udine, via delle Scienze 206, 33100 Udine, Italy

[2] Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, University of Lille, 59000 Lille, France

already proposed in the literature, whereas the sixth one, named *PartialSwapTeamsPhased*, is a variant of one of them that we specifically designed to deal with phased instances. Simulated annealing has been used also by other authors for sports timetabling with good results, suggesting that it is particularly suitable for this type of problem (see Sect. 2 on related work). In addition, we have experienced promising results with multi-neighborhood simulated annealing also on problems that show some similarities, like for example Examination Timetabling (Bellio et al., 2021) or Minimum Interference Frequency Assignment (Ceschia et al., 2021).

Our solver has many parameters, and it has been tuned using the F-RACE procedure (Birattari et al., 2010), upon a set of experimental configurations designed using the Hammersley point set (Hammersley & Handscomb, 1964).

We also propose an Integer Linear Programming (ILP) model for the problem. We implemented it in CPLEX, but, unfortunately, it was able to solve systematically only small artificially generated instances, and it did not produce significant results on the instances of the competition even after long running times.

The paper is organized as follows. Section 2 is dedicated to the discussion of previous work. Section 3 introduces a mathematical model for the problem. Section 4 describes in detail our approach, and its experimental results are illustrated in Sect. 5. Finally, conclusions and future work are discussed in Sect. 6.

2 Related work

Interest in Sports Timetabling dates back to the 1970s. Initial research by Gelling 1973, Russell 1980, Wallis 1983, and de Werra et al. 1990 focused on the relationship between 1-factorizations of a complete graph and the Sports Timetabling Problem. In sports timetabling, 1-factorizations take the name of *patterns* and de Werra 1981 proposed an easy way to generate a 1-factorization that has been named *canonical pattern*. Nevertheless, Rosa and Wallis 1982 and Dinitz et al. 1994 warned about the complexity in the generation of non-isomorphic 1-factorizations. Due to its complexity, applications of metaheuristics to the Sports Timetabling Problem date back to the 1990s, with contributions from Costa 1995, Della Croce et al. 1999 and Hamiez and Hao 2000. In the 2000s Ribeiro and Urrutia 2004, Anagnostopoulos et al. 2006, and Di Gaspero and Schaerf 2007 proposed a set of new neighborhoods for local search-based metaheuristics. They have been employed either with Tabu search or simulated annealing and were particularly effective for the solution of the Traveling Tournament Problem (TTP), proposed by Easton et al. 2001.

In the last decade, Lewis and Thompson 2011, Costa et al. 2012, and Januario and Urrutia 2016 worked on further heuristics and new neighborhoods for the solution of the Sports Timetabling Problem. More recently, Van Bulck et al. 2020b introduced a unified data format for the round-robin sports timetabling, named RobinX, that synthesize 18 different constraints belonging to five different constraint groups, and they published a large set of instances in the proposed format. The RobinX format is employed in the Sports Timetabling Competition ITC2021 (Van Bulck et al., 2021).

More complete bibliographic revisions for sports timetabling can be found in Rasmussen and Trick 2008, and Kendall et al. 2010. An up-to-date bibliography is also available online and maintained by Knust 2010.

3 Problem formulation

We introduce here the ITC2021 problem through its Integer Linear Programming (ILP) model, and we refer to Van Bulck et al. 2021 for a comprehensive presentation.

Let n be an even number and $\mathcal{T} = \{1, \ldots, n\}$ be the set of teams of a sport league. In a double round-robin tournament, each team $i \in \mathcal{T}$ plays a game against each other team $j \in \mathcal{T}$, $j \neq i$, twice, once at home and once away. We identify the home and away games of team i against team j, respectively, with the pairs (i, j) and (j, i). Hence, the set of games that have to be scheduled in the league is $\mathcal{G} = \{(i, j) \in \mathcal{T} \times \mathcal{T} : i \neq j\}$. In addition, in a time-constrained tournament the number of rounds available to schedule the games of $\mathcal{G}$ has to be minimal. Then, $\mathcal{R} = \{1, \ldots, 2(n-1)\}$ is the set of the available rounds in a double round-robin tournament and, at every round $r \in \mathcal{R}$, each team plays exactly once, either at home or away. A timetable is an assignment of exactly one round of $\mathcal{R}$ to each game in $\mathcal{G}$. We say that a timetable is *phased* if the season is split in two legs, where each team plays against all the other teams exactly once: team i and j cannot play both their mutual games ((i, j) and (j, i)) in the same leg. We suppose that the first leg occurs in rounds $1, \ldots, |\mathcal{R}|/2$, whereas the second one in rounds $|\mathcal{R}|/2 + 1, \ldots, |\mathcal{R}|$.

Sport timetables usually consider several additional constraints. Specifically, we consider five groups of constraints. *Capacity constraints* regulate the number of home games, away games or games that a team or a subset of teams can play in a given subset of rounds. *Game constraints* fix or forbid specific assignments of games to rounds. *Break constraints* are used to limit the number of *breaks*, that is the number of consecutive home or away games for a team. Breaks are mostly undesired in a fair timetable. *Fairness constraints* limit the difference of home games played by two teams after each round. Finally, *separation constraints* ensure that the mutual games of two teams are separated by a given number of rounds. We call $\mathcal{C}$ the set of these constraints. Set $\mathcal{C}$ contains *hard* and *soft* constraints: the former express fundamental properties of the timetable and must be satisfied, whereas the latter express preferences and can be violated. We denote by $\mathcal{C}_{\text{hard}}$ and $\mathcal{C}_{\text{soft}}$ the subsets of $\mathcal{C}$ containing, respectively, the hard and soft constraints. For each soft constraint $c \in \mathcal{C}_{soft}$, we denote by w_c the weight associated with its violation.

For each game $(i, j) \in \mathcal{G}$ and each round $r \in \mathcal{R}$, we introduce a binary variable x_{ijr} defined as follows

$$x_{\text{ijr}} = \begin{cases} 1 & \text{if game } (i, j) \text{ is played in round } r \\ 0 & \text{otherwise.} \end{cases}$$

For each soft constraint $c \in \mathcal{C}_{\text{soft}}$, we include a non-negative continuous variable d_c representing the deviation triggered if the constraint is violated.

The model, denoted by $\mathcal{M}$, reads as follows.

$$\min \sum_{c \in \mathcal{C}_{\text{soft}}} w_c d_c \tag{1}$$

$$\sum_{r \in \mathcal{R}} x_{\text{ijr}} = 1 \quad \forall (i, j) \in \mathcal{G} \tag{2}$$

$$\sum_{j \in \mathcal{T}, j \neq i} x_{\text{ijr}} + x_{\text{jir}} \leq 1 \quad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \tag{3}$$

$$\sum_{r \in \mathcal{R}, r \leq |\mathcal{R}|/2} x_{\text{ijr}} + x_{\text{jir}} = 1 \quad \forall (i, j) \in \mathcal{G}, i < j. \tag{4}$$

Objective function (1) minimizes the weighted violation of the soft constraints. Constraints (2) and (3) define a timetable for the games in $\mathcal{G}$ in the rounds of $\mathcal{R}$. Specifically, Constraint (2) imposes that every game is played, i.e., it is assigned to exactly one round, and Constraint (3) ensures that each team plays at most one time per round. Finally, Constraint (4) guarantees that the timetable is phased, if required.

In the following, we list the constraint types considered in set $\mathcal{C}$. We first discuss the hard version of these constraints. Some additional notation, such as subsets of teams or rounds and parameters, may be required for each constraint $c \in \mathcal{C}$. For example, if constraint c is identified by a team $i \in \mathcal{T}$, we denote by $\mathcal{T}(i)$ and/or $\mathcal{R}(i)$, respectively, the subsets of teams and rounds considered by the constraint itself: the dependency on c is dropped to lighten the notation. Furthermore, we explicit the correspondence between the constraints in set $\mathcal{C}$ and those considered in Van Bulck et al. 2021 to avoid ambiguities.

- **Capacity Constraints (CA)**

$$\underline{k}(i) \leq \sum_{j \in \mathcal{T}(i), j \neq i} \sum_{r \in \mathcal{R}(i)} x_{\text{ijr}} \leq \bar{k}(i) \quad \forall i \in \mathcal{T} \tag{5}$$

$$\underline{k}(i) \leq \sum_{j \in \mathcal{T}(i), j \neq i} \sum_{r'=r}^{r+\bar{r}(i)} x_{\text{ijr}'} \leq \bar{k}(i) \quad \forall i \in \mathcal{T}', \forall r = 1, \ldots, |\mathcal{R}| - \bar{r}(i) + 1 \tag{6}$$

$$\underline{k} \leq \sum_{i \in \mathcal{T}'} \sum_{j \in \mathcal{T}'', j \neq i} \sum_{r \in \mathcal{R}'} x_{\text{ijr}} \leq \bar{k} \quad \forall \mathcal{T}', \mathcal{T}'' \subseteq \mathcal{T}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{7}$$

Constraints (5) (CA1 and CA2 in Van Bulck et al., 2021) impose that team i plays at least $\underline{k}(i)$ and at most $\bar{k}(i)$ home games against the teams in subset $\mathcal{T}(i) \subseteq \mathcal{T}$ in the rounds of set $\mathcal{R}(i) \subseteq \mathcal{R}$. These constraints can be used to model the so-called *place constraints* that forbid a team to play at home in a given round and the so-called *top team and bottom team constraints* which avoid bottom teams to play all the initial games against top teams. Then, Constraint (5) (CA3 in Van Bulck et al., 2021) forces team i to play at least $\underline{k}(i)$ and at most $\bar{k}(i)$ home games against the teams in subset $\mathcal{T}(i) \subseteq \mathcal{T}$ in each sequence of $\bar{r}(i)$ rounds. Finally, Constraint (7) (CA4 in Van Bulck et al., 2021) imposes that the number of home games of teams in $\mathcal{T}'$ against teams in $\mathcal{T}''$ in the rounds of $\mathcal{R}'$ has to be between $\underline{k}$ and $\bar{k}$. These constraints are used, for example, to limit the total number of home games per round between teams that share the same venue. Similar constraints can be imposed in case of away games or games.

- **Game Constraints (GA)**

$$\underline{k} \leq \sum_{(i,j) \in \mathcal{G}'} \sum_{r \in \mathcal{R}'} x_{\text{ijr}} \leq \bar{k} \quad \forall \mathcal{G}' \subseteq \mathcal{G}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{8}$$

Given a subset of games $\mathcal{G}' \subseteq \mathcal{G}$ and a subset of rounds $\mathcal{R}' \subseteq \mathcal{R}$, Constraint (8) (GA1 in Van Bulck et al., 2021) imposes a lower bound $\underline{k}$ and an upper bound $\bar{k}$ on the number of games of $\mathcal{G}'$ that can be played in the rounds of $\mathcal{R}'$.

- **Break Constraints (BR)** A team $i \in \mathcal{T}$ has a *home/away break* in round $r \in \mathcal{R} \setminus \{0\}$ if i has a home/away game in rounds $r - 1$ and r. To model these constraints, we introduce two binary variables y^h_{ir} and y^a_{ir} for each team $i \in \mathcal{T}$ and each round $r \in \mathcal{R} \setminus \{0\}$:

$$y^h_{\text{ir}} = \begin{cases} 1 & \text{if team } i \text{ has a home break in round } r \\ 0 & \text{otherwise} \end{cases}$$

and

$$y^a_{\text{ir}} = \begin{cases} 1 & \text{if team } i \text{ has an away break in round } r \\ 0 & \text{otherwise.} \end{cases}$$

The break constraints read as follows.

$$y^h_{\text{ir}} \geq \sum_{j \in \mathcal{T}(i), j \neq i} x_{\text{ijr}} + x_{\text{ijr}-1} \quad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \setminus \{0\} \tag{9}$$

$$y^a_{\mathrm{ir}} \geq \sum_{j \in \mathcal{T}(i), j \neq i} x_{\mathrm{jir}} + x_{\mathrm{jir}-1} \quad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \setminus \{0\} \tag{10}$$

$$\sum_{r \in \mathcal{R}(i)} y^h_{\mathrm{ir}} \leq \bar{k}(i) \quad \forall i \in \mathcal{T} \tag{11}$$

$$\sum_{r \in \mathcal{R}(i)} y^a_{\mathrm{ir}} \leq \bar{k}(i) \quad \forall i \in \mathcal{T} \tag{12}$$

$$\sum_{r \in \mathcal{R}(i)} y^h_{\mathrm{ir}} + y^a_{ir} \leq \bar{k}(i) \quad \forall i \in \mathcal{T} \tag{13}$$

$$\sum_{i \in \mathcal{T}'} \sum_{r \in \mathcal{R}'} y^h_{\mathrm{ir}} + y^a_{ir} \leq \bar{k} \quad \forall \mathcal{T}' \subseteq \mathcal{T}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{14}$$

Constraints (9) and (10) define binary variables y^h_{ir} and y^a_{ir}, respectively. For each team $i \in \mathcal{T}$, Constraint (11) (BR1 in Van Bulck et al., 2021) imposes an upper bound on the number of home breaks of i in the rounds of $\mathcal{R}(i) \subseteq \mathcal{R}$. The same is imposed by Constraints (12) for the away breaks and by Constraints (13) for the total breaks. Finally, given a subset of teams $\mathcal{T}'$ and a subset of rounds $\mathcal{R}'$, Constraint (14) (BR2 in Van Bulck et al., 2021) fixes the overall number of home and away breaks of the teams in $\mathcal{T}'$ in the rounds of $\mathcal{R}'$ to be at most $\bar{k}$.

– **Fairness Constraints (FA)**

$$-\bar{k}(i,j) \leq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{\mathrm{ilr}'} - x_{\mathrm{jlr}'} \leq \bar{k}(i,j)$$
$$\forall i,j \in \mathcal{T}, i \neq j, \forall r \in \mathcal{R}. \tag{15}$$

For all pair of teams $i, j \in \mathcal{T}, i \neq j$ and all rounds $r \in \mathcal{R}$, Constraint (15) (FA2 in Van Bulck et al., 2021) ensures that the difference between the home games played by i and those played by j is at most $\bar{k}(i,j)$ after round r. Analogous constraints can be applied for the away games or games.

– **Separation Constraints (SE)**

$$\sum_{r' \in \mathcal{R}(i,j)} x_{\mathrm{ijr}'} + x_{\mathrm{jir}'} \leq 1 - (x_{\mathrm{ijr}} + x_{\mathrm{jir}}) \quad \forall (i,j) \in \mathcal{G},$$
$$i < j, \forall r \in \mathcal{R}, \tag{16}$$

where $\mathcal{R}(i,j) = \{r' \in \mathcal{R} : r - \underline{k}(i,j) \leq r' \leq r + \underline{k}(i,j)\} \cup \{r' \in \mathcal{R} : r' \leq r - \bar{k}(i,j) \vee r' \geq r + \bar{k}(i,j)\}$. Constraint (16) (SE1 in Van Bulck et al., 2021) ensures that if one of the two mutual games (i,j) or (j,i) of two teams $i, j \in \mathcal{T}$ is assigned to round r, then the other one cannot be assigned to the rounds of $\mathcal{R}(i,j)$: games (i,j) and (j,i) are separated by at least $\underline{k}(i,j)$ and at most $\bar{k}(i,j)$ rounds.

All constraints presented so far can be handled also in their soft version. Here, we discuss in general terms how their deviation is computed (see Van Bulck et al., 2020b, for a detailed description). We remark that all constraints $c \in \mathcal{C}$ have the same structure, i.e., they impose a lower and/or an upper bound on a linear expression:

$$lb_c \leq f_c(x, y^h, y^a) \leq ub_c,$$

where $f_c(x, y^h, y^a)$ is a linear expression in the variables x_{ijr}, y^h_{ir} and y^a_{ir} and lb_c and ub_c are, respectively, a lower and upper bound imposed on $f_c(x, y^h, y^a)$. Except for the fairness and separation constraints, the deviation triggered if one of these constraints is violated is given by the following two constraints

$$d_c \geq lb_c - f_c(x, y^h, y^a) \tag{17}$$
$$d_c \geq f_c(x, y^h, y^a) - ub_c. \tag{18}$$

For example, if c is a capacity constraint which imposes a lower and an upper bound, respectively, $\underline{k}(i)$ and $\bar{k}(i)$, on the number of home games that a team $i \in \mathcal{T}$ can play in the rounds of set $\mathcal{R}(i)$ (Constraint (5)), then the deviation triggered by c is equal to the number of home games of i in the rounds of $\mathcal{R}(i)$ less than $\underline{k}(i)$ or more than $\bar{k}(i)$.

Finally, let us discuss how the deviation of the fairness and separation constraints is computed. A fairness constraint limits the difference of home (away or any) games between teams $i \in \mathcal{T}$ and $j \in \mathcal{T}$ after each round $r \in \mathcal{R}$. However, the deviation triggered when it is violated is equal to the maximal difference in home (away or any) games more than $\bar{k}(i,j)$ played by i and j over all the rounds of $\mathcal{R}$ (see Van Bulck et al., 2021). Hence, to express such deviation, we need to include a non-negative continuous variable z_{ij} storing the maximal difference in home (away or any) games between i and j. For the case of home games, the deviation is computed by including the following constraints.

$$z_{\mathrm{ij}} \geq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{\mathrm{ilr}'} - x_{\mathrm{jlr}'} \tag{19}$$

$$z_{\mathrm{ij}} \geq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{\mathrm{jlr}'} - x_{\mathrm{ilr}'} \tag{20}$$

$$d_c \geq z_{\mathrm{ij}} - \bar{k}(i,j). \tag{21}$$

Now, let c be a soft version of a separation constraint, which require that the two mutual games of teams $i, j \in \mathcal{T}, i < j$ are separated by at least $\underline{k}(i,j)$. The deviation triggered if c is violated has to be equal to the difference between $\underline{k}(i, j) + 1$ and the number of rounds between games (i, j) and (j, i):

$$d_c \geq \sum_{r' \in \mathcal{R}(i,j)} |r' - r|(x_{\mathrm{ijr}} + x_{\mathrm{jir}} + x_{\mathrm{ijr'}} + x_{\mathrm{jir'}} - 1) \qquad (22)$$

The case where the two mutual games have to be separated by at most $\bar{k}(i, j)$ rounds can be treated similarly.

4 Solution method

We designed a three-stage multi-neighborhood simulated annealing for the solution of the problem. The multi-neighborhood is a hexamodal neighborhood made up by a portfolio of six different local search neighborhoods, which are specifically tailored for the sports timetabling problems. The metaheuristic method employed for the search of the solution is a slightly modified version of the classical simulated annealing defined by Kirkpatrick et al. 1983. The search is executed in three distinct sequential stages, characterized by different parameter values of the metaheuristic and different restriction of the search space. In this section, we explain first of all the general features of the search space and the method employed for the generation of the initial solution. Next, we discuss thoroughly the multi-neighborhood and the simulated annealing metaheuristic. Finally, we illustrate the characteristics of the three stages of execution of the algorithm.

4.1 Search space

Given the structure of the problem described in Sect. 3, as search space we consider the set of all two-leg round-robin timetables. This means that every possible round-robin timetable, though not necessarily feasible, is a solution in the search space. Thus, in every solution, each team plays with every other team twice (home and away), and all teams play exactly one match at every round.

For the instances that require a phased timetable, we allow the algorithm to visit states that break the phase structure. Since the formulation of the problem does not provide an explicit phase constraint, we added an artificial tenth cost component that measures the number of matches that violate the phase requirement. This number is then multiplied for a suitable weight, and the resulting value is assigned to the new cost component. This mechanism, which is applied only to phased instances, makes phased violations possible but penalized in the cost function.

A solution is internally described as a matrix of size $|\mathcal{T}| \times |\mathcal{R}|$. Each cell (t, r) contains the index of the opponent of t at the match r. The value is positive if t plays at home at the match r, negative otherwise. Figure 1b provides an example of this encoding, which is used also in the figures in Sect. 4.3 for the explanation of the multi-neighborhood.

4.2 Initial solution generation

The initial state can be generated either randomly or through a greedy algorithm. The random procedure consists in different permutations of teams and rounds on the *canonical pattern* (see, e.g., de Werra, 1981). It produces a double round-robin tournament, but it does not provide any feasibility guarantee regarding the hard constraints of the problem, which is then restored by the simulated annealing procedure.

Given an input instance with $|\mathcal{T}|$ teams and $|\mathcal{R}|$ rounds, the random initial solution is generated performing the following steps:

1. A single-leg canonical pattern for the $|\mathcal{T}|$ teams with $|\mathcal{R}|/2$ rounds is generated. Each team meets every other team exactly once.
2. A random permutation is performed on the $|\mathcal{T}|$ teams.
3. The timetable is mirrored in order to obtain a two-leg tournament. At this moment, the second leg is identical to the first one, except for the home-away order that is inverted.
4. If the instance is not phased, a random permutation is executed on the $|\mathcal{R}|$ rounds. Otherwise, if the instance is phased, two random permutations are executed. The first one involves the rounds $\{0, \ldots, |\mathcal{R}|/2 - 1\}$, and the second one is performed on the rounds $\{|\mathcal{R}|/2, \ldots, |\mathcal{R}| - 1\}$. Hence, the initial random solution does not violate the phase constraint.

Also the greedy algorithm is based on the canonical pattern, which is used as a reference for the constructive steps. The idea behind the greedy algorithm is to generate and test the addition of candidate rounds that are constructed on the basis of a reference tournament of *template rounds* obtained as in the random procedure. These rounds are templates instead of actual ones since all their possible perturbations, according to some of the symmetries that are inherent in round-robin tournaments, are produced in the generation process. In detail, the symmetries used are those among rounds (i.e., permuting the order of the rounds does not violate the round-robin tournament property) and those among the venues of each game.

Starting from an empty initial solution, the greedy process selects, at each step, the best candidate to be added to the current solution according to its contribution to constraint violations. Since the solution is incomplete, the check is restricted to only those constraints that can be (at least partially) evaluated in the current partial solution once it has been extended with any of the candidate rounds.

Fig. 1 Example of the internal solution representation of a timetable for a round-robin tournament with 6 teams and 10 games: the upper part is the listing of the actual games in the tournament, while the lower part reports its encoding

round_0	round_1	round_2	round_3	round_4	round_5	round_6	round_7	round_8	round_9
2 - 0	4 - 0	3 - 0	0 - 5	0 - 1	0 - 3	0 - 4	0 - 2	5 - 0	1 - 0
1 - 3	2 - 1	5 - 1	1 - 4	5 - 2	1 - 5	1 - 2	3 - 1	4 - 1	2 - 5
4 - 5	3 - 5	4 - 2	3 - 2	3 - 4	2 - 4	3 - 5	5 - 4	2 - 3	4 - 3

(a) An example tournament with $|T| = 6$

	round_0	round_1	round_2	round_3	round_4	round_5	round_6	round_7	round_8	round_9
team_0	-2	-4	-3	+5	+1	+3	+4	+2	-5	-1
team_1	+3	-2	-5	+4	-0	+5	+2	-3	-4	+0
team_2	+0	+1	-4	-3	-5	+4	-1	-0	+3	+5
team_3	-1	-5	+0	+2	+4	-0	+5	+1	-2	-4
team_4	+5	+0	+2	-1	-3	-2	-0	-5	+1	+3
team_5	-4	+3	+1	-0	+2	-1	-3	+4	+0	-2

(b) Internal representation of the previous tournament

In Fig. 2 we exemplify a step of the greedy process in case of $|T| = 6$ teams in a non-phased setting. In the example, the current solution consists of four assigned rounds and six remaining template rounds (denoted by χ_i), which are still available from the initial canonical pattern. Different situations arise, for instance, in the generation of round candidates for the χ_1 and χ_2 templates.

As for the template χ_1, which has not been included yet in the solution, there is full freedom in deciding the home-away status of the games; therefore, all possible venues permutations can be generated and evaluated. Conversely, since one copy of the χ_2 template has been already included in the solution (namely in round 1), among all the possible venues permutations only the one that mirrors the already included copy of the template is possible. This is however inherent in the fact that the reference tournament for the round templates is created through a concatenation of two single round-robin canonical patterns.

Each generated round candidate is tried for the completion of the current solution, and the partial cost of this addition is computed. For example, in the figure, the constraint `BR1`[1] reported above the current solution requires that no more than 2 *breaks* occur for team 5 during periods 1–5. This constraint can be partially checked for its cost (which is zero, since there are no more than 2 breaks) for the periods 1–3 already added to the solution, whereas it cannot be checked yet for period 4 and the possible candidate addition.

Among all the possible candidates, the one that achieves the minimum (partial) cost is selected and the corresponding template is removed from the set of available ones. Possible ties in the cost value are randomly broken.

In the case of a phased tournament the greedy procedure is adapted to ensure that the two legs of the tournament are separated. In order to achieve this goal, the tournament used as a reference for the first leg consists of a single round-robin tournament template and after the first leg is completed the second leg is constructed with another (possibly different, in terms of team permutations) single round-robin tournament using the full generation of combinations but pruning those that overlap with the games already included in the first leg.

Finally, to obtain numerous diverse initial solutions also with the greedy procedure, the indexes of the teams are randomly permuted. That is, before starting the process, a random permutation of the indexes is drawn and the mapping between the teams in the candidate round (i.e., the logical indexes) and the actual teams is computed by applying this permutation. To enhance the randomness, in the case of the phased tournament two distinct permutations are computed for the first leg and the second leg assignments.

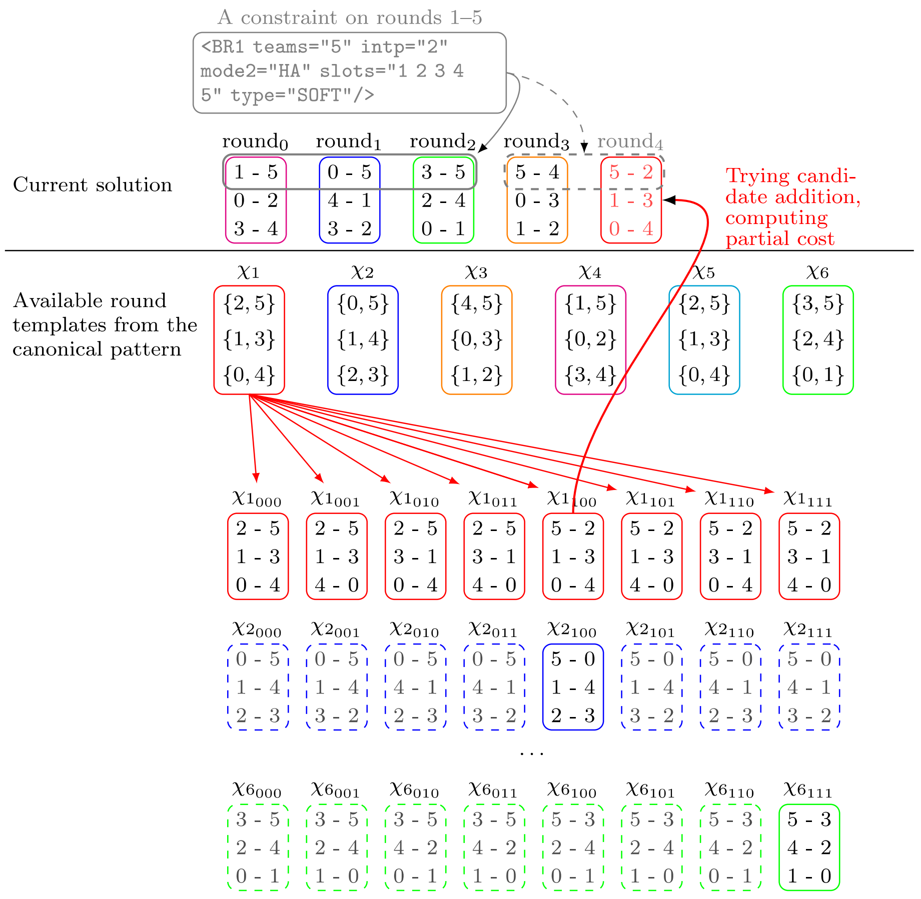
As discussed further in Sect. 5.2, this choice seems to mildly outperform the random initial solution. Nevertheless, the improvement margin is not considerably large, so we decided to keep both possibilities in our algorithm, leaving to the user the possibility to choose the start method through an input parameter.

4.3 Multi-neighborhood relations

We propose a multi-neighborhood composed by the union of different neighborhoods. Five of them, called *SwapHomes*, *SwapTeams*, *SwapRounds*, *PartialSwapTeams*, and *PartialSwapRounds*, are adaptations of classical ones from Ribeiro and Urrutia 2004, Anagnostopoulos et al. 2006, and Di Gaspero and Schaerf 2007. In addition, we introduce a novel neighborhood called *PartialSwapTeamsPhased*, specifically designed to deal with phased instances. Experimental results highlight that the usage of the novel neighborhood allows us to reach better solutions in terms of objective function and to achieve feasibility on certain large phased instances, which would be, otherwise, very hard to tackle. All the neighborhoods ensure that the double round-robin structure of the tournament is conserved, but they do not provide any guarantees on the feasibility of the solution.

The multi-neighborhood is designed to be employed by the simulated annealing metaheuristic, described in

[1] Refer to (Van Bulck et al., 2020a) for the comprehensive explanation of all constraints employed in the competition.



```
<BR1 teams="5" intp="2"
mode2="HA" slots="1 2 3 4
5" type="SOFT"/>
```

Fig. 2 One step of the greedy constructive procedure in the case of a non-phased tournament: the procedure tries to complete the partial solution with the best possible combination of template assignment and game venues

Sect. 4.4, that randomly draws a move from the multi-neighborhood at every iteration. A desirable feature of the multi-neighborhood is to give higher frequency of execution to those moves that belong to neighborhoods that, on average, lead to the most significant improvements of the solution. So, an essential property of the multi-neighborhood is that each neighborhood is associated with a probability. The draw of the move in the simulated annealing is then executed into two steps. The first step is the random selection of one of the six neighborhoods, according to the given probabilities. The second step is the random selection of a move inside the neighborhood. The probability of the move inside the neighborhoods is shaped as a uniform random variable.

The values of the probabilities are defined through a tuning procedure discussed in Sect. 5.2, while the six neighborhoods and their specifications are illustrated hereafter.

4.3.1 SwapHomes

The move *SwapHomes* takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and it is denoted as $\mathcal{SH}\langle t_i, t_j\rangle$. It swaps the home/away position of the two games between t_i and t_j. Figure 3 shows the execution of the move.

4.3.2 SwapTeams

The move *SwapTeams* takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and it is denoted as $\mathcal{ST}\langle t_i, t_j\rangle$. It swaps the positions of t_i and t_j throughout the whole timetable. Figure 4 shows the execution of the move.

4.3.3 SwapRounds

The move *SwapRounds* takes as attributes two rounds $r_i, r_j \in \mathcal{R}$, $r_i \neq r_j$, and it is denoted as $\mathcal{SR}\langle r_i, r_j\rangle$. It swaps the two rounds in the timetable. That is to say, all the matches assigned to r_i are moved to r_j, and vice versa. Figure 5 shows the execution of the move.

4.3.4 PartialSwapTeams

The move *PartialSwapTeams* takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and a set of rounds $\mathcal{R}_s = \{r_1, \ldots, r_s\}$,

-2	-4	-3	+5	+1	+3	+4	+2	-5	-1
+3	-2	-5	+4	-0	+5	+2	-3	-4	+0
+0	+1	-4	-3	-5	+4	-1	-0	+3	+5
-1	-5	+0	+2	**-4**	-0	+5	+1	-2	**+4**
+5	+0	+2	-1	**+3**	-2	-0	-5	+1	**-3**
-4	+3	+1	-0	+2	-1	-3	+4	+0	-2

$\longrightarrow$

-2	-4	-3	+5	+1	+3	+4	+2	-5	-1
+3	-2	-5	+4	-0	+5	+2	-3	-4	+0
+0	+1	-4	-3	-5	+4	-1	-0	+3	+5
-1	-5	+0	+2	**+4**	-0	+5	+1	-2	**-4**
+5	+0	+2	-1	**-3**	-2	-0	-5	+1	**+3**
-4	+3	+1	-0	+2	-1	-3	+4	+0	-2

Fig. 3 Execution of the *SwapHomes* move $\mathcal{SH}\langle 3, 4\rangle$. The figure on the left shows the state before the move, and figure on the right represents the new state. Changes are marked in bold and colored

-2	**-4**	**-3**	+5	+1	**+3**	**+4**	+2	-5	-1
+3	-2	-5	**+4**	-0	+5	+2	**-3**	**-4**	+0
+0	+1	**-4**	**-3**	-5	**+4**	-1	-0	**+3**	+5
-1	-5	+0	+2	**+4**	-0	+5	+1	-2	**-4**
+5	+0	+2	-1	**-3**	-2	-0	-5	+1	**+3**
-4	**+3**	+1	-0	+2	-1	**-3**	**+4**	+0	-2

$\longrightarrow$

-2	**-3**	**-4**	+5	+1	**+4**	**+3**	+2	-5	-1
+4	-2	-5	**+3**	-0	+5	+2	**-4**	**-3**	+0
+0	+1	**-3**	**-4**	-5	**+3**	-1	-0	**+4**	+5
+5	+0	+2	-1	**-4**	-2	-0	-5	+1	**+4**
-1	-5	+0	+2	**+3**	-0	+5	+1	-2	**-3**
-3	**+4**	+1	-0	+2	-1	**-4**	**+3**	+0	-2

Fig. 4 Execution of the *SwapTeams* move $\mathcal{ST}\langle 3, 4\rangle$. Figure on the left shows the state before the move, and figure on the right represents the new state. Changes are marked in bold and colored

-2	-4	**-3**	+5	+1	+3	+4	+2	**-5**	-1
+3	-2	**-5**	+4	-0	+5	+2	-3	**-4**	+0
+0	+1	**-4**	-3	-5	+4	-1	-0	**+3**	+5
-1	-5	**+0**	+2	+4	-0	+5	+1	**-2**	-4
+5	+0	**+2**	-1	-3	-2	-0	-5	**+1**	+3
-4	+3	**+1**	-0	+2	-1	-3	+4	**+0**	-2

$\longrightarrow$

-2	-4	**-5**	+5	+1	+3	+4	+2	**-3**	-1
+3	-2	**-4**	+4	-0	+5	+2	-3	**-5**	+0
+0	+1	**+3**	-3	-5	+4	-1	-0	**-4**	+5
-1	-5	**-2**	+2	+4	-0	+5	+1	**+0**	-4
+5	+0	**+1**	-1	-3	-2	-0	-5	**+2**	+3
-4	+3	**+0**	-0	+2	-1	-3	+4	**+1**	-2

Fig. 5 Execution of the *SwapRounds* move $\mathcal{SR}\langle 2, 8\rangle$. Figure on the left shows the state before the move, and figure on the right represents the new state. Changes are marked in bold and colored

$\mathcal{R}_s \subset \mathcal{R}$. The move is denoted as $\mathcal{PST}\langle t_i, t_j, \mathcal{R}_s\rangle$. It swaps the positions of t_i and t_j on the set of rounds in $\mathcal{R}_s$. As the name suggests, it works similarly to *SwapTeams*, with the main difference that the move is not executed on the whole timetable, but only on a subset of rounds.

A fundamental requirement for the construction of $\mathcal{R}_s$ is that each team t_k, playing against teams t_i, t_j in the rounds in $\mathcal{R}_s$, must play against both t_i and t_j exactly the same amount of times. If $\mathcal{R}_s$ satisfies the precondition, the swap of t_i and t_j in the rounds in $\mathcal{R}_s$ leads to another correct round-robin timetable. In practice though, large subsets $\mathcal{R}_s$ are not particularly desirable, because they considerably slow down the generation of the move with a negative impact on the overall time performance of the algorithm. For this reason, we impose a limitation on the maximal size of the set $\mathcal{R}_s$ during the move generation procedure. Figure 6 shows the execution of the move.

4.3.5 PartialSwapRounds

The move *PartialSwapRounds* takes as attributes two rounds $r_i, r_j \in \mathcal{R}, r_i \neq r_j$, and a set of teams $\mathcal{T}_s = \{t_1, \ldots, t_s\}, \mathcal{T}_s \subset \mathcal{T}$. The move is denoted as $\mathcal{PSR}\langle \mathcal{T}_s, r_i, r_j\rangle$. It produces the swap between r_i and r_j of the matches including teams in $\mathcal{T}_s$. As the name suggests, it works similarly to *PartialRounds*, with the main difference that the move is not executed on the whole set of matches in the two rounds, but only on a subset of matches.

A fundamental requirement for the construction of $\mathcal{T}_s$ is that every team $t_k \in \mathcal{T}_s$ plays only with teams from $\mathcal{T}_s$ in the two rounds r_i and r_j. In practice though, large subsets $\mathcal{T}_s$ are not particularly desirable, because they considerably slow down the generation of the move with a negative impact on the overall time performance of the algorithm. For this reason, we impose a limitation on the maximal size of the set $\mathcal{T}_s$ during the generation of the move. Figure 7 shows the execution of the move.

4.3.6 PartialSwapTeamsPhased

The move *PartialSwapTeamsPhased* is a novel neighborhood that we designed with the main motivation to deal with the phased version of the problem. The five neighborhoods discussed so far, indeed, work well on the non-phased instances, but turned out to be insufficient for obtaining good results on the phased ones. The neighborhood *PartialSwapTeams*, in particular, has quite disruptive side effects on the phase structure of the timetable that are only sporadically beneficial to the search. *PartialSwapTeamsPhased*, on the other hand, allows to reach new solutions through partial swaps of teams without variations of the current state of the phase. We discuss in this section the fundamentals of the neighborhood, and we forward the reader to Sect. 5.4 for an analysis of the experimental data.

As the name suggests, *PartialSwapTeamsPhased* takes inspiration from the above-mentioned *PartialSwapTeams*.

$$\boxed{\begin{array}{rrrrrrrrrr}
-2 & -4 & +3 & -1 & +5 & -3 & +1 & +2 & -5 & +4 \\
+4 & -3 & -5 & +0 & \mathbf{+2} & -2 & -0 & +5 & -4 & \mathbf{+3} \\
+0 & +5 & -4 & +3 & \mathbf{-1} & +1 & \mathbf{+4} & -0 & -3 & \mathbf{-5} \\
+5 & +1 & -0 & -2 & \mathbf{+4} & +0 & \mathbf{-5} & -4 & +2 & \mathbf{-1} \\
-1 & +0 & +2 & -5 & \mathbf{-3} & +5 & \mathbf{-2} & +3 & +1 & -0 \\
-3 & -2 & +1 & +4 & -0 & -4 & \mathbf{+3} & -1 & +0 & \mathbf{+2}
\end{array}} \longrightarrow \boxed{\begin{array}{rrrrrrrrrr}
-2 & -4 & +3 & -1 & +5 & -3 & +1 & +2 & -5 & +4 \\
+4 & -3 & -5 & +0 & \mathbf{+3} & -2 & -0 & +5 & -4 & \mathbf{+2} \\
+0 & +5 & -4 & +3 & \mathbf{+4} & +1 & \mathbf{-5} & -0 & -3 & \mathbf{-1} \\
+5 & +1 & -0 & -2 & \mathbf{-1} & +0 & \mathbf{+4} & -4 & +2 & \mathbf{-5} \\
-1 & +0 & +2 & -5 & \mathbf{-2} & +5 & \mathbf{-3} & +3 & +1 & -0 \\
-3 & -2 & +1 & +4 & -0 & -4 & \mathbf{+2} & -1 & +0 & \mathbf{+3}
\end{array}}$$

Fig. 6 Execution of the *PartialSwapTeams* move $\mathcal{PST}\langle 3, 4, \{4, 6, 9\}\rangle$. Figure on the left shows the state before the move, and figure on the right shows the new state. Changes are marked in bold and colored

$$\boxed{\begin{array}{rrrrrrrrrr}
-2 & -4 & +3 & -1 & +5 & -3 & +1 & +2 & -5 & +4 \\
\mathbf{+4} & -3 & -5 & +0 & +2 & -2 & -0 & \mathbf{+5} & -4 & +3 \\
+0 & +5 & -4 & +3 & -1 & +1 & +4 & -0 & -3 & -5 \\
\mathbf{+5} & +1 & -0 & -2 & +4 & +0 & -5 & \mathbf{-4} & +2 & -1 \\
\mathbf{-1} & +0 & +2 & -5 & -3 & +5 & -2 & \mathbf{+3} & +1 & -0 \\
\mathbf{-3} & -2 & +1 & +4 & -0 & -4 & +3 & \mathbf{-1} & +0 & +2
\end{array}} \longrightarrow \boxed{\begin{array}{rrrrrrrrrr}
-2 & -4 & +3 & -1 & +5 & -3 & +1 & +2 & -5 & +4 \\
\mathbf{+5} & -3 & -5 & +0 & +2 & -2 & -0 & \mathbf{+4} & -4 & +3 \\
+0 & +5 & -4 & +3 & -1 & +1 & +4 & -0 & -3 & -5 \\
\mathbf{-4} & +1 & -0 & -2 & +4 & +0 & -5 & \mathbf{+5} & +2 & -1 \\
\mathbf{+3} & +0 & +2 & -5 & -3 & +5 & -2 & \mathbf{-1} & +1 & -0 \\
\mathbf{-1} & -2 & +1 & +4 & -0 & -4 & +3 & \mathbf{-3} & +0 & +2
\end{array}}$$

Fig. 7 Execution of the *PartialSwapRounds* move $\mathcal{PSR}\langle\{1, 5, 3, 4\}, 0, 7\rangle$. Figure on the left shows the state before the move, and figure on the right shows the new state. Changes are marked in bold and colored

An example of the disruptive effect of *PartialSwapTeams* on the phase structure is given in Fig. 6. Before the execution of the move, the phase structure is respected. After the move is applied, matches between teams 2 and 4 are both executed during the first phase of the tournament, and matches between teams 3 and 4 are both executed in the second phase of the tournament, which constitute two violations of the phase constraint. To overcome this issue, the new neighborhood *PartialSwapTeamsPhased* makes use of a new concept of *mixed phase* that allows the new move to be invariant with respect to the phase.

We define as mixed phase of a two-leg round-robin tournament the partition of the timetable in two subsets, named mixed legs, where each couple of teams play together, respectively, for the first and for the second time. Hence, the first mixed leg is the set of all the matches where teams meet with each other for the first time, and the second mixed leg is the set of all the matches where teams meet for the second time. This definition is independent from the current satisfaction of the phase constraint. It might happen that mixed phase and actual phase correspond: this is the case when the phased constraint is respected.

Figures 8 and 9 visually explain the concept. Both figures contain two boxes: the box on the left is the representation of the current tournament timetable, and the box on the right is the corresponding mixed phase. The first and second mixed legs are denoted, respectively, by means of zeros and ones. Figure 8 describes the situation of a timetable that satisfies the phase constraint, and Fig. 9 represents a timetable where the phase constraint is violated. Matches where teams meet each other for the first time belong to the first mixed leg and are denoted by zeros in the box on the right, and matches where teams meet each other for the second time belong to the second mixed leg and are denoted by ones.

Therefore, the move *PartialSwapTeamsPhased* takes as attribute two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and a set of rounds $\mathcal{R}_s = \{r_1, \ldots, r_s\}$, $\mathcal{R}_s \subset \mathcal{R}$. The essential prerequisite is that the set of matches involved in the move must all belong to the same mixed leg, according to the definition provided. The move is denoted as $\mathcal{PSTP}\langle t_i, t_j, \mathcal{R}_s\rangle$, and, similarly to the move *PartialSwapTeams*, it produces the swap of the positions of t_i and t_j on the set of rounds in $\mathcal{R}_s$. Consequently, the maximum size of the set $\mathcal{R}_s$ corresponds to the size of a mixed leg, which is $|\mathcal{R}_s| \leq |\mathcal{R}|/2$.

4.4 Metaheuristic

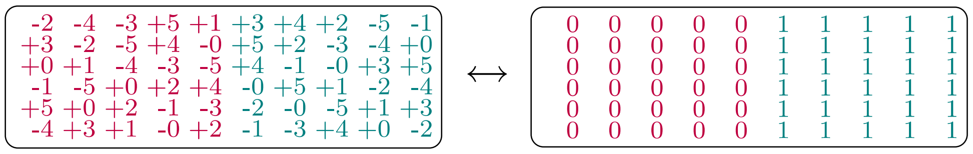
The metaheuristic employed is basically the traditional simulated annealing defined in Kirkpatrick et al. 1983.

At each iteration, a random move is drawn as explained in Sect. 4.3, and its corresponding variation of cost Δ is computed. The move is always accepted if it is improving or sideways, i.e., $\Delta \leq 0$, and it is accepted with probability $e^{-\Delta/T}$ if $\Delta > 0$ (the Metropolis acceptance criterion).

The *temperature* T starts from an initial value T_0 and decreases according to a geometric cooling scheme after a given number of iterations m, until it reaches the final temperature $T_{\min}$. That is, every m iterations we set $T := \alpha \cdot T$, where α is the cooling rate.

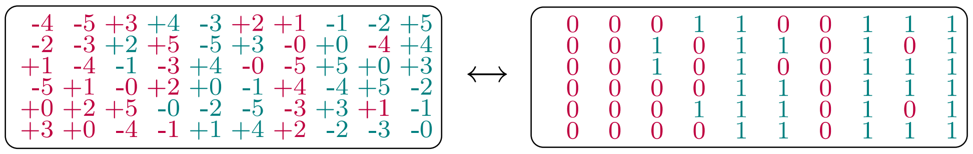
We make use of a *cutoff* mechanism to limit the number of evaluated moves at each temperature level (Johnson et al., 1989). Given a current temperature T and a cutoff value of γ, when an amount of $\gamma \cdot m$ solutions have already been accepted at temperature T, the algorithm stops evaluating moves at the current temperature and passes to the next temperature. The purpose of the cutoff is to reduce the computational time spent on chaotic states, commonly at high temperatures, when most moves are accepted.

In our version of the simulated annealing we do not use a timeout, but we fix *a priori* the number M of total moves evaluated. To keep the total number of iterations fixed independently of the values of the parameters, we do not fix m,



$$\boxed{\begin{array}{cccccccccc} -2 & -4 & -3 & +5 & +1 & +3 & +4 & +2 & -5 & -1 \\ +3 & -2 & -5 & +4 & -0 & +5 & +2 & -3 & -4 & +0 \\ +0 & +1 & -4 & -3 & -5 & +4 & -1 & -0 & +3 & +5 \\ -1 & -5 & +0 & +2 & +4 & -0 & +5 & +1 & -2 & -4 \\ +5 & +0 & +2 & -1 & -3 & -2 & -0 & -5 & +1 & +3 \\ -4 & +3 & +1 & -0 & +2 & -1 & -3 & +4 & +0 & -2 \end{array}} \longleftrightarrow \boxed{\begin{array}{cccccccccc} 0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1&1&1 \end{array}}$$

Fig. 8 Mixed phase and actual phase correspond. The box on the left shows in different colors the mixed legs on the timetable, while the box on the right shows the internal representation where mixed legs are denoted, respectively, by zeros and ones



$$\boxed{\begin{array}{cccccccccc} -4 & -5 & +3 & +4 & -3 & +2 & +1 & -1 & -2 & +5 \\ -2 & -3 & +2 & +5 & -5 & +3 & -0 & +0 & -4 & +4 \\ +1 & -4 & -1 & -3 & +4 & -0 & -5 & +5 & +0 & +3 \\ -5 & +1 & -0 & +2 & +0 & -1 & +4 & -4 & +5 & -2 \\ +0 & +2 & +5 & -0 & -2 & -5 & -3 & +3 & +1 & -1 \\ +3 & +0 & -4 & -1 & +1 & +4 & +2 & -2 & -3 & -0 \end{array}} \longleftrightarrow \boxed{\begin{array}{cccccccccc} 0&0&0&1&1&0&0&1&1&1 \\ 0&0&1&0&1&1&0&1&0&1 \\ 0&0&1&0&1&0&0&1&1&1 \\ 0&0&0&0&1&1&0&1&1&1 \\ 0&0&0&1&1&1&0&1&0&1 \\ 0&0&0&0&1&1&0&1&1&1 \end{array}}$$

Fig. 9 Mixed phase and actual phase differ, the phase constraint is not respected. The box on the left shows in different colors the mixed legs on the timetable, while the box on the right shows the internal representation where mixed legs are denoted, respectively, by zeros and ones

and instead, we compute it based on the values of T_0, $T_{\min}$ and α.

Given the cutoff mechanism, the temperature might drop below T_{min}, as the iterations saved in the early stages are got back at the end.

Summarizing, the parameters of the metaheuristics are: the start temperature T_0, the final temperature $T_{\min}$, the cooling rate α, and the cutoff rate γ.

4.5 Stages of the algorithm

The algorithm is structured into three distinct stages, that consist in three independent runs of the Simulated Annealing procedure described in Section 4.4. The first stage starts its search either from a random or from a greedy solution, the second and the third stages are warm-started with the output of the previous stage as initial solution. The differences between the stages consist in the restrictions applied to the search space, and in the exclusion or inclusion of certain constraints (see Table 1).

Stage 1: Hard constraints are included in the cost function, and soft constraints are not considered. The objective of Stage 1 is to rapidly find a feasible solution. In our experiments, this stage shows its effectiveness specifically on large phased instances, where the metaheuristic faces most problems in finding a feasible timetable.

Stage 2: All constraints are applied and both feasible and infeasible regions are explored. The costs associated with hard constraints and phased cost component are multiplied by suitable weights. The goal of Stage 2 is to find a good solution in terms of objective function.

Stage 3: All constraints are applied, but moves that violate hard constraints are not allowed and only the feasible region is explored. The goal of Stage 3 is to reach the best possible solution that can be achieved from the best state found in Stage 2, through the exploration of the feasible region only. If the outcome of the previous stages is not a feasible solution, Stage 3 is not performed (Table 1).

Table 1 Description of the features of the three stages

	Constraints	
	Hard	Soft
Stage 1	As cost component	Not used
Stage 2	As cost component	As cost component
Stage 3	Violation not allowed	As cost component

5 Experimental results

In this section, we discuss the experimental setting of the simulated annealing algorithm and the results obtained by applying it to the instances proposed for the ITC2021 competition. We also assess the performances of Model $\mathcal{M}$ by running it on CPLEX within a time limit. For this latter experiments, we consider the instances of the competition and some others of smaller size obtained by performing reductions on the competition instances.

Our code was developed in C++ and compiled with GNU g++ version 9.3.0 on Ubuntu 20.04.2 LTS. The tuning phase was partially performed on a cluster of virtual machines provided by the CINECA consortium. All the other experiments presented in this section were run on a machine equipped with AMD Ryzen Threadripper PRO 3975WX processor with 32 cores, hyper-threaded to 64 virtual cores, with base clock frequency of 3.5 GHz, and 64 GB of RAM. In both settings, one single virtual core is used for each experiment.

Table 2 List of instances from the ITC2021 competition with indication of some general features and overall number of hard and soft constraints

Instance	Features				Instance	Features			
	Phased	Teams	Hard	Soft		Phased	Teams	Hard	Soft
Early_1	Yes	16	83	113	Middle_9	No	18	94	201
Early_2	Yes	16	53	114	Middle_10	Yes	20	198	714
Early_3	Yes	16	148	186	Middle_11	Yes	20	176	1048
Early_4	Yes	18	164	268	Middle_12	Yes	20	63	241
Early_5	Yes	18	207	587	Middle_13	No	20	219	350
Early_6	Yes	18	192	797	Middle_14	No	20	63	817
Early_7	No	18	175	1159	Middle_15	No	20	95	133
Early_8	No	18	70	582	Late_1	No	16	235	542
Early_9	No	18	90	102	Late_2	No	16	246	1077
Early_10	Yes	20	246	1015	Late_3	No	16	127	439
Early_11	No	20	246	1108	Late_4	Yes	18	96	34
Early_12	Yes	20	179	35	Late_5	Yes	18	176	747
Early_13	No	20	100	432	Late_6	Yes	18	163	159
Early_14	No	20	56	56	Late_7	No	18	126	738
Early_15	No	20	187	1224	Late_8	Yes	18	110	195
Middle_1	Yes	16	144	993	Late_9	No	18	102	402
Middle_2	Yes	16	246	1231	Late_10	Yes	20	233	694
Middle_3	No	16	237	1212	Late_11	Yes	20	52	366
Middle_4	Yes	18	97	168	Late_12	No	20	244	1009
Middle_5	Yes	18	151	197	Late_13	No	20	169	134
Middle_6	Yes	18	162	154	Late_14	No	20	116	993
Middle_7	No	18	141	476	Late_15	No	20	51	41
Middle_8	No	18	62	224					

Table 3 Details of the number of constraints in each instance

Name	CA1		CA2		CA3		CA4		BR1		BR2		GA1		SE1		FA2	
	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S
E1	25	17	10	0	0	0	0	84	35	0	1	0	12	10	0	1	0	1
E2	38	30	0	0	2	82	0	0	12	0	1	0	0	1	0	0	0	1
E3	24	0	72	21	0	112	0	0	18	0	0	1	34	51	0	0	0	1
E4	0	32	0	235	0	0	85	0	44	0	1	0	34	0	0	1	0	0
E5	41	27	36	331	2	111	81	117	23	0	1	0	23	0	0	1	0	0
E6	38	31	71	591	2	54	81	115	0	0	0	1	0	3	0	1	0	1
E7	42	31	30	620	1	112	84	340	12	0	1	0	5	55	0	1	0	0
E8	19	0	8	57	0	112	0	339	39	0	0	0	4	73	0	0	0	1
E9	39	0	14	0	0	88	0	0	23	10	0	1	14	2	0	0	0	1
E10	42	32	72	620	2	23	85	339	44	0	1	0	0	0	0	1	0	0
E11	42	32	72	620	2	112	85	340	44	0	1	0	0	3	0	1	0	0
E12	37	0	72	0	2	20	31	0	20	13	0	1	17	1	0	0	0	0
E13	41	31	27	257	1	110	0	0	20	10	1	0	10	24	0	0	0	0
E14	5	30	0	0	0	0	0	0	17	24	0	1	34	0	0	0	0	1
E15	42	0	72	620	2	112	71	340	0	24	0	1	0	126	0	0	0	1
M1	0	32	14	620	0	0	85	340	44	0	1	0	0	0	0	1	0	0
M2	42	32	72	620	2	112	85	340	44	0	1	0	0	126	0	1	0	0
M3	42	0	72	617	2	107	85	338	36	21	0	1	0	126	0	1	0	1
M4	31	18	17	0	1	0	0	41	23	24	0	0	25	85	0	0	0	0
M5	41	24	40	33	0	12	0	0	44	0	0	1	26	126	0	0	0	1

Table 3 continued

Name	CA1		CA2		CA3		CA4		BR1		BR2		GA1		SE1		FA2	
	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S
M6	39	0	41	0	2	111	30	27	44	13	0	1	6	1	0	1	0	0
M7	0	30	0	355	1	0	78	51	28	21	0	1	34	17	0	1	0	0
M8	16	0	0	27	2	108	0	34	32	14	0	0	12	41	0	0	0	0
M9	42	0	0	37	1	100	19	39	28	23	0	1	4	0	0	0	0	1
M10	41	15	71	363	0	0	46	262	39	0	1	0	0	74	0	0	0	0
M11	7	0	71	612	2	88	84	340	12	0	0	0	0	7	0	0	0	1
M12	0	32	28	168	1	13	0	0	29	21	0	1	5	4	0	1	0	1
M13	42	29	72	242	1	0	85	76	12	0	0	0	7	2	0	1	0	0
M14	5	18	11	319	2	112	0	338	38	10	1	0	6	19	0	0	0	1
M15	12	0	23	0	0	77	0	0	37	10	0	1	23	44	0	1	0	0
L1	42	32	72	198	1	13	82	283	38	0	0	0	0	15	0	0	0	1
L2	42	0	72	620	2	112	85	340	44	0	1	0	0	5	0	0	0	0
L3	42	0	72	326	1	43	0	60	12	0	0	1	0	7	0	1	0	1
L4	0	32	0	0	0	0	18	0	44	0	0	0	34	1	0	1	0	0
L5	0	19	69	614	2	0	81	109	0	0	1	0	23	4	0	0	0	1
L6	0	32	0	125	0	0	85	0	44	0	0	1	34	0	0	1	0	0
L7	42	32	40	601	1	61	0	0	37	0	1	0	5	43	0	1	0	0
L8	37	15	0	14	0	111	0	0	41	24	0	1	32	29	0	1	0	0
L9	40	0	20	250	2	112	0	0	40	20	0	1	0	18	0	0	0	1
L10	0	31	67	447	2	0	85	205	44	0	1	0	34	10	0	1	0	0
L11	6	0	16	274	0	88	0	0	12	0	1	0	17	3	0	0	0	1
L12	40	32	72	620	2	16	85	340	44	0	1	0	0	0	0	1	0	0
L13	14	32	72	15	2	0	81	71	0	0	0	1	0	13	0	1	0	1
L14	42	0	72	390	2	112	0	340	0	24	0	0	0	126	0	0	0	1
L15	5	0	0	0	0	15	0	0	12	24	0	1	34	0	0	0	0	1

5.1 Instances

The algorithm was run on the 45 instances of the competition that are available for download on the website of the competition (Van Bulck et al., 2020a). We performed a general analysis of their main features. The size, expressed in number of teams, is always comprised between 16 and 20 teams, and 22 instances in total have a phased requirement. As we can observe in Table 2, the total number of hard and soft constraints fluctuates considerably among instances. The range for hard constraints goes from a minimum of 51 in instance Late_11 to a maximum of 246 in instances Early_10, Early_11, Middle_2, and Late_2. The range for soft constraints goes from a minimum of 34 in instance Late_4 to a maximum of 1231 in instance Middle_2. Incidentally, instance Middle_2, which has the highest overall number of constraints, 1477, is also the only instance in which our solver was not able to determine any feasible solution. More in general, from our experimental results we observed that most instances where our solver shows the most deficiencies are also among those characterized by many hard constraints, but only when also the phase requirement is present. For this reason, the solver redistributes the total number of iterations in favor of Stage 1 when it recognizes a phased instance with a quantity of hard constraints above a certain threshold. Table 3 provides additional details on the cardinality of constraints of each type in their hard and soft versions. It is possible to notice that constraint types BR2, FA2, and SE1 are always expressed uniquely, if present, so only six constraint types out of nine are actually declined into multiple constraints. Finally, it is noteworthy to mention that each individual constraint involves different quantities of teams or slots, so that also the individual contribution to the instance complexity may differ substantially.

5.2 Parameters and tuning

For the three-stage multi-neighborhood algorithm to be effective on the given instances, several parameters need to be tuned. In this work, we tuned the probabilities of the six neighborhoods separately on the phased and on the non-phased instances. These probabilities are stage-independent.

Table 4 Parameter tuning

Parameter	Description	Tuning range	Assigned values	
			Not Phased	Phased
p_{sh}	SwapHomes	[0.0, 1.0]	0.154	0.130
p_{st}	SwapTeams	[0.0, 1.0]	0.070	0.020
p_{sr}	SwapRounds	[0.0, 1.0]	0.025	0.080
p_{pst}	PartialSwapTeams	[0.0, 1.0]	0.319	0.120
p_{pstp}	PartialSwapTeamsPhased	[0.0, 1.0]	0.070	0.130
p_{psr}	PartialSwapRounds	[0.0, 1.0]	0.350	0.520

			Stage 1	Stage 2	Stage 3
T_0	Start Temperature	[0, 2000]	179	600	17.9
$T_{\min}$	End Temperature	[0, 20]	2.1	3.52	0.21
w_{h,ca_1}	Weight of CA1 hard	[1, 10]	7	7	–
w_{h,ca_2}	Weight of CA2 hard	[1, 10]	8	8	–
w_{h,ca_3}	Weight of CA3 hard	[1, 10]	2	2	–
w_{h,ca_4}	Weight of CA4 hard	[1, 10]	8	8	–
w_{h,ga_1}	Weight of GA1 hard	[1, 10]	10	10	–
w_{h,br_1}	Weight of BR1 hard	[1, 10]	1	1	–
w_{h,br_2}	Weight of BR2 hard	[1, 10]	6	6	–
w_{h,fa_2}	Weight of FA2 hard	[1, 10]	1	1	–
w_{h,se_1}	Weight of SE1 hard	[1, 10]	1	1	–
	Initial solution	$\{random, greedy\}$	*greedy*	–	–
w_h	Feature-dependent w_h	$\{yes, no\}$	*no*	*yes*	–
w_p	Feature-dependent w_p	$\{yes, no\}$	*no*	*yes*	–
k	Correlation factor	[0.1, 1000.0]	–	0.5	–
w_h	Hard weight (fixed)	[1, 1000]	10	–	–
w_p	Phased weight (fixed)	[1, 1000]	117	–	–

The specific parameters of the simulated annealing meta-heuristics, on the other hand, were tuned separately for each stage. For the simulated annealing we decided to tune only the start temperature and the final temperature, which turned out to be the most critical parameters from previous research work (see, e.g., Bellio et al., 2016, 2021). Conversely, we fixed the sample acceptance ratio and the cooling rate values to consolidated and robust values found in previous work. In our algorithm, we assigned weights to the different hard cost components and these weights also underwent a tuning procedure. They were employed in Stage 1 and Stage 2, since moves that violate satisfiability are not considered in Stage 3. Finally, the decision whether to use a random or a greedy start for Stage 1 was also subject to a tuning procedure.

As introduced in Sect. 4.5, we allow in Stage 1 and Stage 2 the exploration of the infeasible region and, for phased instances, also the break of the phase structure. Thus, the weights assigned to hard violations and phased violations also require tuning. In this case, we did not only search for possible values, but we compared two different approaches: feature-based or fixed values. Specifically, the only feature that we take into account for this problem is the number of hard constraints in the given instance. Let N_h be the number of hard constraints, w_h and w_p, respectively, the hard cost component weight and the phased cost component weight, and k a generic constant. Equations 23 and 24 describe how we can obtain the weights for the hard cost component and the phased cost component from the number of hard constraints, in the feature-based scenario. The value of k is also determined through a tuning procedure. Please note that k is float-valued and w_h and w_p are integer-valued, so a rounding is applied. In our tuning procedure, the feature-dependent approach resulted to be the most effective for Stage 2, while for Stage 1 fixed values resulted to be more suitable.

$$w_h = N_h \cdot k \tag{23}$$

$$w_p = N_h \cdot k \cdot w_h \tag{24}$$

The whole tuning procedure was performed with the aid of the tool json2run (Urli, 2013), which implements the F-RACE procedure. The parameter space was sampled using an Hammersley point set (Hammersley & Handscomb, 1964).

Table 4 contains the list of the parameters and related values. The column *Tuning range* contains the lower and

Table 5 Comparison of the results obtained by the random start and the greedy start on 30 short runs of Stage 1

Instance	Random		Greedy		Instance	Random		Greedy	
	Feas.	Time (s)	Feas.	Time (s)		Feas.	Time (s)	Feas.	Time(s)
Early_1	1.00	4.44	1.00	5.03	Middle_9	1.00	10.49	1.00	12.96
Early_2	0.87	26.50	0.97	24.40	Middle_10	0.00	20.41	0.00	25.90
Early_3	1.00	2.24	1.00	2.72	Middle_11	0.83	64.05	0.90	69.47
Early_4	0.00	22.90	0.00	25.03	Middle_12	1.00	14.39	1.00	18.98
Early_5	0.00	73.41	0.00	76.77	Middle_13	1.00	25.04	1.00	36.09
Early_6	0.20	68.10	0.20	70.42	Middle_14	1.00	26.85	1.00	34.05
Early_7	0.33	40.14	0.30	44.24	Middle_15	1.00	1.67	1.00	6.91
Early_8	1.00	1.23	1.00	2.87	Late_1	0.97	21.75	1.00	20.89
Early_9	1.00	1.29	1.00	3.08	Late_2	0.00	58.60	0.00	60.31
Early_10	0.00	92.14	0.00	102.42	Late_3	1.00	7.49	1.00	8.07
Early_11	0.37	83.88	0.60	90.42	Late_4	1.00	3.11	1.00	4.50
Early_12	0.47	73.86	0.43	78.61	Late_5	0.00	73.04	0.00	76.02
Early_13	1.00	16.09	1.00	22.23	Late_6	1.00	7.15	1.00	9.64
Early_14	1.00	0.94	1.00	5.73	Late_7	1.00	13.66	1.00	16.13
Early_15	1.00	46.00	1.00	56.23	Late_8	1.00	2.78	1.00	3.91
Middle_1	0.00	23.55	0.00	24.47	Late_9	1.00	24.39	1.00	26.07
Middle_2	0.00	60.07	0.00	61.46	Late_10	0.00	90.38	0.00	99.97
Middle_3	0.00	58.68	0.00	60.33	Late_11	1.00	4.16	1.00	7.99
Middle_4	1.00	12.18	1.00	14.45	Late_12	0.87	66.50	0.87	80.68
Middle_5	1.00	3.26	1.00	4.55	Late_13	1.00	46.21	1.00	56.76
Middle_6	0.27	59.18	0.40	59.57	Late_14	1.00	31.55	1.00	39.43
Middle_7	1.00	19.80	1.00	23.17	Late_15	1.00	0.80	1.00	5.71
Middle_8	1.00	20.99	1.00	23.68					

upper bounds of the ranges taken into account by the tuning procedure, which do not constitute a boundary in our algorithm. The dual comparisons aimed to determine whether to use a random or a greedy start and whether to use fixed or feature-dependent w_h and w_p do not require a Hammersley sampling. The outcome of the tuning procedure is shown under the columns *Assigned values*.

Finally, Table 5 presents the outcome of a dual comparison between the random start and the greedy start. We limited the execution of the algorithm to one million iterations, which we consider a short run, of Stage 1, exclusively. The results are given in terms of feasibility ratio, because in this stage of the algorithm we are not focused yet in optimizing the objective function. In general, we can observe that the greedy start ensures a higher probability of finding a feasible solution, at a price of a slightly longer execution time.

5.3 Analysis of the results

We report in this section an overview of the experimental results.

First, we discuss the results obtained on Model $\mathcal{M}$. We used the solver CPLEX 20.1 and we imposed different time limits, ranging from one hour to 24 hours. We employed one single CPU per run.

Solving the model on the competition instances did not yield good results: within a time limit of one hour, a feasible solution was found only for instance Late_4, and the other 44 instances were left unsolved. Longer time limits provided very limited improvements. Within 24 hours, feasible solutions were found only for six instances (E8, M4, M8, M15, L4, L8), with values of the objective function far from those obtained by the simulated annealing within analogous running times. Hence, to assess the performances and the limits of Model $\mathcal{M}$, we run tests on two clusters of instances obtained by reducing the size of the competition ones. In the first cluster, we removed constraint types and pairs of constraint types from each competition instance which contains them. The columns of Table 6 (left) report, respectively: the removed constraint types; the number of reduced instances; the number of instances for which a feasible, but not optimal, solution is found; the number of instances solved to optimality. From Table 6 (left), we infer that the solver still struggles to produce feasible solutions when only one constraint type is removed. Removing pairs of constraints seems to bring benefits only when the capacity and break constraints are both removed: the solver manages to provide 26 feasible solutions

Table 6 Results obtained solving Model $\mathcal{M}$ on the first (left) and second cluster (right) of reduced instances

Removed constraints	# Inst.	# Feas.	# Opt.
CA	45	3	3
GA	42	1	0
BR	45	3	0
FA	23	0	0
SE	24	0	1
CA, GA	42	8	3
CA, BR	45	20	6
CA, FA	23	6	4
CA, SE	24	4	3
GA, BR	42	7	0
GA, FA	22	0	0
GA, SE	24	1	0
BR, FA	23	4	3
BR, SE	24	2	2
FA, SE	3	0	0

$\lvert\mathcal{T}\rvert$	# Inst.	# Feas.	# Opt.
6	20	7	13
8	20	14	6
10	20	15	2
12	20	4	1
14	20	2	0

Table 7 Iterations per stage, feature-based

Instance type	Max iterations (M) per stage		
	M_1	M_2	M_3
$Phased \wedge N_h > 200$	500000000	50000000	40000
$\neg Phased \vee N_h \leq 200$	20000000	250000000	40000

for the considered instances, six of which are proven to be optimal in an average time of 30 seconds. This is in line with the structure of the instances; indeed, several of them consider many capacity and break constraints in their hard version (see Table 3).

In the second cluster of instances, we reduced the size of the competition instances in terms of number of teams. Specifically, we restrict the cardinality of team set $\mathcal{T}$ to $|\mathcal{T}| = 6, 8, 10, 12$ and for each cardinality we consider 20 reduced instances. These instances are obtained from the competition ones by randomly selecting the teams to remove and by deleting them from all the constraints in which they appear. Table 6 (right) reports the same data as in Table 6 (left), except for the first column which, in this case, reports the size of set $\mathcal{T}$. As expected, the larger the size of $\mathcal{T}$, the less feasible (and optimal) solutions the solver is able to provide. Moreover, we observe that the model is solved consistently on instances up to size ten. Starting from $|\mathcal{T}| = 12$, the performances of the model drop drastically: only four feasible solutions and an optimal one are found with $|\mathcal{T}| = 12$ and only two feasible solutions are found with $|\mathcal{T}| = 14$.

In what follows, we discuss the results obtained by the simulated annealing algorithm on the competition instances. Specifically, we report both the best overall solution that we obtained for each instance and data on the average behavior of the algorithm. In order to assess the average performances of the simulated annealing in terms of cost, time, and feasibility, we report the results of a dedicated experiment batch. All the experiments were run without a time limit, but rather with a fixed number of maximum iterations per stage (see Sect. 4.4). Depending on the number of hard constraints in the instance, two different configurations of iterations per stage were applied, as shown in Table 7.

Table 8 reports the results obtained by the solver. The column *Best solution found* reports the best solution that our solver was able to find in all experiments. Some of these values are those that we submitted to the ITC2021 competition, and others have been found in later experiments. When

Table 8 Best and average results

Instance	Best solution found	Average values			Best known cost	CPLEX best bound
		Cost	Time (s)	Feasible		
Early_1	423	540.7	5667	1.00	362	1.0
Early_2	318	384.6	14844	1.00	145	0.0
Early_3	1068	1176.5	12195	1.00	992	48.9
Early_4	556	1007.8	8760	0.56	507	0.0
Early_5	4117	–	28517	0.00	3127	247.2
Early_6	3927	4543.0	35162	1.00	3325	587.3
Early_7	5205	6721.7	37487	1.00	4763	1233.1
Early_8	**1051**	1151.9	21394	1.00	1051	212.1
Early_9	132	228.7	10324	1.00	108	0.0
Early_10	4986	–	35856	0.00	3400	308.2
Early_11	4526	5784.5	43692	1.00	4426	309.5
Early_12	1010	1200.2	14726	1.00	380	0.0
Early_13	173	233.8	19675	1.00	121	2.0
Early_14	63	82.3	5616	1.00	4	1.0
Early_15	3556	3945.8	46715	1.00	3362	484.5
Middle_1	5657	6075.0	26291	0.06	5177	2857.5
Middle_2	5H	–	26891	0.00	7381	2909.8
Middle_3	**9542**	11403.1	44749	0.23	9542	3266.8
Middle_4	16	33.0	5660	1.00	7	7.0
Middle_5	510	624.4	6223	1.00	413	46.8
Middle_6	1701	2186.3	21350	1.00	1120	23.0
Middle_7	2203	2452.7	16303	1.00	1783	23.6
Middle_8	136	196.6	19718	1.00	129	2.0
Middle_9	640	772.1	17611	1.00	450	0.0
Middle_10	1357	1687.5	14433	1.00	1250	3.8
Middle_11	2696	2996.5	43877	1.00	2446	345.0
Middle_12	950	1054.2	14599	1.00	911	1.0
Middle_13	362	479.3	15687	1.00	252	0.0
Middle_14	**1172**	1304.6	37484	1.00	1172	0.0
Middle_15	985	1099.7	8705	1.00	485	0.7
Late_1	2021	2372.7	20242	1.00	1922	1102.4
Late_2	5715	6085.5	41433	0.49	5400	2817.7
Late_3	2457	2718.0	18328	1.00	2369	347.9
Late_4	**0**	0.0	2355	1.00	0	0.0
Late_5	2341	–	9191	0.00	1923	397.5
Late_6	930	1121.3	7122	1.00	923	5.4
Late_7	1765	2226.5	22959	1.00	1558	3.1
Late_8	997	1155.3	11286	1.00	934	77.9
Late_9	715	881.2	25963	1.00	563	2.9
Late_10	2571	3527.3	32511	0.05	1945	1.0
Late_11	207	289.3	15892	1.00	202	0.0
Late_12	3944	4830.6	35514	1.00	3428	156.2
Late_13	1868	2285.5	21007	1.00	1820	6.1
Late_14	**1202**	1326.3	39161	1.00	1202	6.5
Late_15	60	82.8	6435	1.00	20	0.0

Table 9 Comparison of the results obtained on phased instances with and without the neighborhood PartialSwapTeamsPhased

Instance	With PSTP		Without PSTP		gap
	avg	feasible (%)	avg	feasible (%)	(%)
Early_1	540.7	1.00	563.5	1.00	+4.22%
Early_2	384.6	1.00	388.3	1.00	+0.96%
Early_3	1176.5	1.00	1204.4	1.00	+2.37%
Early_4	1007.8	0.56	1125.8	0.38	+11.71%
Early_5	–	0.00	–	0.00	–
Early_6	4543.0	1.00	4553.2	1.00	+0.22%
Early_10	–	0.00	–	0.00	–
Early_12	1200.2	1.00	1326.4	1.00	+10.51%
Middle_1	6075.0	0.06	–	0.00	$+\infty$
Middle_2	–	0.00	–	0.00	–
Middle_4	33.0	1.00	33.3	1.00	+0.91%
Middle_5	624.4	1.00	656.8	1.00	+5.19%
Middle_6	2186.3	1.00	2224.7	1.00	+1.76%
Middle_10	1687.5	1.00	1766.8	1.00	+4.70%
Middle_11	2996.5	1.00	3131.8	1.00	+4.52%
Middle_12	1054.2	1.00	1137.0	1.00	+7.85%
Late_4	0.0	1.00	0.0	1.00	+0.00%
Late_5	–	0.00	–	0.00	–
Late_6	1121.3	1.00	1141.7	1.00	+1.82%
Late_8	1155.3	1.00	1186.1	1.00	+2.67%
Late_10	3527.3	0.05	3590.0	0.05	+1.58%
Late_11	289.3	1.00	321.6	1.00	+11.16%

the current known best was determined by our solver, the corresponding value in the first column is marked in bold. When no feasible solution has been found, the number of hard violations followed by a letter H is reported. Next columns, labeled *Average values*, report the data obtained in a set of experiments that we run independently from the competition, in order to extract information on the average behavior of the algorithm in its final configuration. At least 48 runs per instance were performed to collect this data. Columns *Cost* and *Time* report, respectively, the average values of the objective function and the average time needed for a complete run of the three stages. Regarding the average cost, the value is computed only on feasible solutions. Column *Feasible* reports the ratio between feasible solutions and total runs. Finally, column *Best known cost* contains the best known results at the moment this article is written, according to data published on the website of the competition (Van Bulck et al., 2020a), and column *CPLEX best bound* contains the best lower bound that CPLEX was able to determine on Model $\mathcal{M}$. Overall, our three-stage multi-neighborhood Simulated Annealing solver could find at least one feasible solution on 44 out of 45 instances. According to data, in its final configuration it manages to determine very easily a feasible solution on 36 instances, which are characterized by a feasibility ratio of 1.00, as it can be observed in column *Feasible* of the above-discussed table. The other instances appear to be less easy to solve for the algorithm. In particular, instances Early_5, Early_10, Middle_2, and Late_5 result to be considerably challenging, as feasible solutions are found just sporadically.

5.4 Analysis of the neighborhood PartialSwapTeamsPhased

One of the main contributions of the presented work is the new neighborhood *PartialSwapTeamsPhased* that was introduced with the main purpose to solve the phased version of the problem. In order to test the effectiveness of the novel neighborhood, we run an additional set of experiments on phased instances without making use of *PartialSwapTeamsPhased*. To do so, we associated a probability of 0.00 to *PartialSwapTeamsPhased* in the multi-neighborhood and rescaled the probabilities associated with the other neighborhoods proportionally, in order to keep the same mutual ratios among them (according to the values in Table 4).

In Table 9 we report the average costs and the feasibility ratios obtained by the standard configuration and those obtained by the configuration that does not employ *PartialSwapTeamsPhased*. At least 20 runs per instance were executed. The last column reports, where possible, the percentage gap between the average cost obtained without *PartialSwapTeamsPhased* and the average cost obtained by the full configuration. It is possible to observe that instance Middle_1 is solved to feasibility only in the configuration that employs *PartialSwapTeamsPhased*. Sixteen instances, solved by both configurations, have worse results when solved without *PartialSwapTeamsPhased*. For just one instance, Late_4, the configurations obtain the same average cost. Finally, the remaining four instances, Early_5, Early_10, Late_5, and Middle_2, are not solved to feasibility by any of the two configurations, in the given number of runs. According to these data, the neighborhood *PartialSwapTeamsPhased* appears to bring a tangible improvement in 17 out of 22 phased instances.

6 Conclusions

In this study, we considered the version of the Sports Timetabling Problem proposed for the ITC2021 competition. We presented an ILP model for the problem, which did not yield significant results when solved by a commercial solver. Then, we tackled the problem employing a three-stage multi-neighborhood simulated annealing approach that makes use of six different neighborhoods. In particular, the neighborhood that we named PartialSwapTeamsPhased is a novel contribution. Finally, we performed a parameter tuning for the solver using the F-RACE procedure that allowed us to find a set of parameter values for this problem.

This approach managed to find a feasible solution for 44 out of the 45 instances proposed by the competition. Feasible solutions were found rather easily for most of the instances; however, the metaheuristic struggled to produce feasible solutions for certain instances, even in long execution times. The results obtained by the simulated annealing approach allowed us to rank second out of 13 participants in the final ranking of the competition.

Future work will be devoted to improve the results and performances of the simulated annealing algorithm both on the considered instances and on other benchmark instances for round-robin tournament. Specifically, we think that relevant advancements can be achieved through a wider study and application of the PartialSwapTeamsPhased neighborhood on a larger set of instances. Possible research directions may also include the definition and integration of new neighborhoods in the simulated annealing algorithm to better deal with feasibility in heavily constrained instances. In addition, we will work on the implementation and evaluation of new greedy techniques to generate different initial solutions, not restricted to the canonical pattern. Further research may also be committed to develop a metaheuristic approach, such as Large Neighborhood Search (LNS), which embeds exact methods in our simulated annealing algorithm.

Acknowledgements The authors thank the CINECA consortium for the access to the Cloud Computing resources under the Italian Super Computing Resource Allocation Project Grant IA4ITC_C. The authors are also grateful to the ITC2021 organizers David Van Bulck, Dries Goossens, Jeroen Beliën, and Morteza Davari, and to the anonymous reviewers for their helpful and detailed comments.

Funding Open access funding provided by Università degli Studi di Udine within the CRUI-CARE Agreement.

References

Anagnostopoulos, A., Michel, L., Van Hentenryck, P., & Vergados, Y. (2006). A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling, 9*(2), 177–193.

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research, 65*, 83–92.

Bellio, R., Ceschia, S., Di Gaspero, L., & Schaerf, A. (2021). Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers and Operations Research, 132*, 105300.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: An overview. *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Berlin: Springer.

Ceschia, S., Di Gaspero, L., Rosati, R. M., & Schaerf, A. (2021). Multi-neighborhood simulated annealing for the minimum interference frequency assignment problem. *EURO Journal on Computational Optimization*, 1–32.

Costa, D. (1995). An evolutionary tabu search algorithm and the NHL scheduling problem. *INFOR, 33*(3), 161–178.

Costa, F. N., Urrutia, S., & Ribeiro, C. C. (2012). An ils heuristic for the traveling tournament problem with predefined venues. *Annals of Operations Research, 194*(1), 137–150.

de Werra, D. (1981). Scheduling in sports. In P. Hansen (Ed.), *Studies on Graphs and Discrete Programming* (pp. 381–395). North Holland.

de Werra, D., Jacot-Descombes, L., & Masson, P. (1990). A constrained sports scheduling problem. *Discrete Applied Mathematics, 26*, 41–49.

Della Croce, F., Tadei, R., & Asioli, P. (1999). Scheduling a round robin tennis tournamentunder courts and players availability constraints. *Annals of Operations Research, 92*, 349–361.

Di Gaspero, L., & Schaerf, A. (2007). A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics, 13*(2), 189–207.

Dinitz, J. H., Garnick, D. K., & McKay, B. D. (1994). There are 526,915,620 nonisomorphic one-factorizations of k_{12}. *Journal of Combinatorial Design, 2*, 273–285.

Easton, K., Nemhauser, G., & Trick, M. (2001). The traveling tournament problem description and benchmarks. In: *Seventh international conference on the principles and practice of constraint programming (CP 99)*, Springer-Verlag, LNCS, (vol. 2239, pp. 580–589).

Gelling, E. N. (1973). *On 1-factorizations of the complete graph and the relationship to round robin schedules*. PhD thesis.

Hamiez, J. P., & Hao, J. K. (2000). Solving the sports league scheduling problem with tabu search. In: *Workshop on local search for planning and scheduling*, Springer, (pp. 24–36).

Hammersley, J. M., & Handscomb, D. C. (1964). *Monte Carlo methods*. Chapman and Hall.

Januario, T., & Urrutia, S. (2016). A new neighborhood structure for round robin scheduling problems. *Computers & Operations Research, 70*, 127–139.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research, 37*(6), 865–892.

Kendall, G., Knust, S., Ribeiro, C., & Urrutia, S. (2010). Scheduling in sports: An annotated bibliography. *Computers and Operations Research, 37*(1), 1–19.

Kirkpatrick, S., Gelatt, D., & Vecchi, M. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Knust, S. (2010). Classification of literature on sports scheduling. http://www2.informatik.uni-osnabrueck.de/knust/sportssched/sportlit_class/, last accessed: 09/11/2021.

Lewis, R., & Thompson, J. (2011). On the application of graph colouring techniques in round-robin sports scheduling. *Computers & Operations Research, 38*(1), 190–204.

Rasmussen, R. V., & Trick, M. A. (2008). Round robin scheduling-a survey. *European Journal of Operational Research, 188*(3), 617–636.

Ribeiro, C. C., & Urrutia, S. (2004). Heuristics for the mirrored traveling tournament problem. In: *Proc. of the 5th international conference on the practice and theory of automated timetabling (PATAT-2004)*, (pp. 323–342).

Rosa, A., & Wallis, W. D. (1982). Premature sets of 1-factors or how not to schedule round robin tournaments. *Discrete Applied Mathematics, 4*, 291–297.

Russell, K. G. (1980). Balancing carry-over effects in round robin tournaments. *Biometrika, 67*(1), 127–131.

Urli, T. (2013). json2run: A tool for experiment design & analysis. CoRR abs/1305.1112.

Van Bulck, D., Goossens, D., Beliën, J., & Davari, M. (2020a). Website of the fifth international timetabling competition (itc 2021): Sports timetabling. https://www.sportscheduling.ugent.be/ITC2021/, last accessed: 16/11/2021.

Van Bulck, D., Goossens, D., Schönberger, J., & Guajardo, M. (2020b). Robinx: A three-field classification and unified data format for round-robin sports timetabling. *European Journal of Operational Research, 280*(2), 568–580.

Van Bulck, D., Goossens, D., Beliën, J., & Davari, M. (2021). The fifth international timetabling competition (itc 2021): Sports timetabling. In: *MathSport International 2021 Conference*, (pp. 117–122).

Wallis, W. D. (1983). A tournament problem. *Journal of the Australian Mathematical Society Series B, 24*, 289–291.