Journal of Scheduling

A heuristic for minimizing weighted carry-over effects in round robin tournaments

Allison C. B. Guedes* · **Celso C. Ribeiro†**

August 15, 2010

Abstract The carry-over effects value is one of the various measures one can consider to assess the quality of a round robin tournament schedule. We introduce and discuss a new, weighted variant of the minimum carry-over effects value problem. The problem is formulated by integer programming and an algorithm based on the hybridization of the Iterated Local Search metaheuristic with a multistart strategy is proposed. Numerical results are presented.

Keywords: Carry-over; round robin; tournaments; sports scheduling; timetabling; heuristics; iterated local search

1 Motivation

Sports optimization has been attracting the attention of an increasing number of researchers and practitioners in multidisciplinary areas such as operations research, scheduling theory, constraint programming, graph theory, combinatorial optimization, and applied mathematics. Kendall et al. [1] surveyed state-of-the-art applications and methods for solving optimization problems in sports scheduling. Rasmussen and Trick [2] reviewed scheduling problems in round robin tournaments, which are of special importance due to their practical relevance and interesting mathematical structure.

There are many relevant aspects to be considered in the determination of the best fixture for a tournament. In some situations, one seeks for a schedule minimizing the total traveled distance, as in the case of the traveling tournament problem [3] and in that of its mirrored variant [4], which is common to many tournaments in South America [5]. Other problems attempt to minimize the number

* Supported by the FAPERJ grant E-26/100.497/2008.

† Supported by CNPq grants 301.694/2007-9 and 485.328/2007-0, as well as by FAPERJ grant E-26/102.805/2008.

Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
E-mail: {aguedes,celso}@ic.uff.br

of breaks, i.e., the number of pairs of consecutive home games or consecutive away games. Ribeiro and Urrutia [6, 7] tackled the scheduling of the Brazilian national football tournament, a bi-criteria optimization problem in which one of the objectives consists in maximizing the number of games that may be broadcast by open TV channels (to increase the revenues from broadcast rights) and the other consists in finding a balanced schedule with a minimum number of home breaks and away breaks (for sake of fairness).

The minimization of the *carry-over effects value* [8] is another fairness criterion leading to an even distribution of the sequence of games along the schedule.

A major issue in the strategy of a team (or an athlete), in particular in long competitions, consists of balancing their efforts over the competition. If a team plays against a weak opponent, it is likely to be in better shape to play in the next round than if it had played against a hard opponent before. Teams that play against strong opponents will very likely be more tired for their next game. Therefore, it is likely that a team (or an athlete) makes much less effort playing against an opponent that played before against a very strong contestant, than it would make against an opponent that faced an easy contestant.

The above situation is particularly true in the case of sports which require a great amount of physical effort (such as wrestling, rugby, and martial arts). In this sort of sports, it is not uncommon that a team (or an athlete) plays several games in a row, making a sequence of very tired (resp. well reposed) opponents very attractive (resp. unattractive). Some fixtures may contain several of such sequences of easier or harder games assigned to one or more teams. This situation does not characterize a fair schedule and is highly undesirable in any tournament. To illustrate this effect, suppose a Karate-Do or Judo competition, for which there is no weight division in open-weight categories: a physically weak athlete may fight a strong one. A contestant that has just fought a very strong opponent will possibly be very tired (and even wounded) in his/her next fight. This would deteriorate his/her performance, giving to the next opponent a strong advantage that otherwise he/she would not have.

Although some authors advocate that carry-over effects do not play a major role in collective sports [9], Flatberg et al. [10] have recently shown a real-life application to a football league in Norway in which carry-overs determined by one specific team and player strongly affected the final results of the competition. Furthermore, they have also shown that the minimization of such effects lead to a more fair fixture and to a much better schedule of games. Another interesting real-life application is illustrated by problems in US college football, whereby a team (Alabama) was repeatedly scheduled against teams with byes the week before. The sequence of games was very unattractive for Alabama, because it was supposed to often meet a restful team that has not played in the previous round [11].

Suppose that team (or athlete) A plays team C (or athlete) right after playing team (or athlete) B. If B is much stronger than the other competitors, then C will possibly take some advantage over A in their game. This is due to the great effort A has made in its previous game. This type of situation in which one of the teams (or athletes) may be benefited should be avoided or, at least, minimized.

We say that C receives a carry-over effect due to B if there is a team (or an athlete) A that plays C just after its game against B. We consider a single round robin tournament played by n (even) teams, in which every team plays each other exactly once. Figure 1 (a) displays a matrix describing a hypothetical tournament

fixture, whose entry in row i and column j informs the team playing against team j in round i. One may count the number of carry-over effects each team gives to every other in the fixture of a round robin tournament and then build the *carry-over effects matrix*. Each entry in row i and column j of this matrix indicates the number of carry-over effects team i gives to team j. Figure 1 (b) presents the underlying carry-over effects matrix associated with the fixture represented in Figure 1 (a).

	A	B	C	D	E	F	G	H
1	H	C	B	E	D	G	F	A
2	C	D	A	B	G	H	E	F
3	D	E	F	A	B	C	H	G
4	E	F	H	G	A	B	D	C
5	F	G	E	H	C	A	B	D
6	G	H	D	C	F	E	A	B
7	B	A	G	F	H	D	C	E

(a)

	A	B	C	D	E	F	G	H
A	0	0	3	0	1	2	1	0
B	5	0	0	0	1	0	0	1
C	0	1	0	3	0	3	0	0
D	0	2	0	0	2	0	3	0
E	1	1	0	2	0	2	0	1
F	0	0	0	0	2	0	3	2
G	0	3	1	0	0	0	0	3
H	1	0	3	2	1	0	0	0

(b)

Fig. 1 (a) Fixture of a tournament with eight teams and (b) its carry-over effects matrix.

A fair tournament regarding carry-over effects is one in which the latter are evenly distributed over the cells of the carry-over effects matrix and the total carry-over effects value is minimized.

The problem of minimizing the carry-over effects value was originally proposed by Russell [8]. In this work, we extend this problem and propose its variant consisting in the minimization of the *weighted carry-over effects value*. This new, weighted problem is introduced and formulated in the next section. Section 3 reviews previous solution approaches for the unweighted original problem and describes a heuristic based on the hybridization of the Iterated Local Search metaheuristic with a multistart strategy for approximately solving the weighted variant. Numerical results are reported and discussed in Section 4. Concluding remarks are drawn in the last section.

2 Weighted formulation

A *single round robin tournament* is one in which each team plays every other exactly once. The games take place along a predefined period of time organized into rounds or slots. A *compact tournament* has a minimum number of rounds. A *compact single round robin tournament* with n (even) teams has $n-1$ rounds.

For a given compact single round robin schedule with n teams, one says that team i gives a carry-over effect to team j if some other team plays consecutively against teams i and j, i.e., some team plays against team i in round t and against team j in round $t+1$ for some $t \in \{1, \ldots, n-1\}$, where the rounds are considered cyclically (i.e., round $n-1$ is followed by the first round). If team i is a very strong or very weak team and several teams play consecutively against teams i and j in this order, then team j may be, respectively, handicapped or favored when compared with other teams. In an "ideal" schedule with respect to carry-

over effects, no teams i, j, x, y should exist such that teams x and y both play against team j immediately after playing against team i.

The carry-over effects matrix $C = (c_{ij})$, with $i, j = 1, \ldots, n$, has been introduced to measure the balance of a tournament with respect to this criterion. Each entry c_{ij} of this matrix counts the number of carry-over effects given by team i to team j. It can be seen that $c_{ii} = 0$ and $\sum_{j=1}^{n} c_{ij} = \sum_{i=1}^{n} c_{ij} = n - 1$ for all rows and columns $i, j = 1, \ldots, n$. The quality of a schedule with respect to carry-over effects is measured by the carry-over effects value $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^2$. An ideal or balanced schedule is one in which the lower bound value $n(n-1)$ is achieved, i.e., all non-diagonal elements of the corresponding matrix C are equal to one.

The original (unweighted) carry-over effects value minimization problem does not consider any information with respect to the relative strengths of the teams: all carry-over effects have the same unit weight. However, in real sports competitions it is very likely that the organizers do have an initial estimate of how well a specific team should perform in the competition, considering e.g. its ranking in last year's competition, its tradition, the quality of its players, and the support of its fans in home games. Noronha et al. [5] provided a concrete example of a competition (the Chilean national football tournament) in which the teams are classified a priori according to some of these criteria in the effort to build a fair schedule.

Therefore, one may conclude that the minimization of the carry-over effects value does not guarantee a fair schedule. In fact, a broader approach consists of assigning a weight w_{ij} to every ordered pair (i, j) of teams, based on their relative strengths or handicaps, and minimizing the total weighted carry-over effects value.

For every pair of teams $i, j = 1, \ldots, n$ (with $i \neq j$) and for every round $k = 1, \ldots, n$ (with rounds cyclically represented, such as that the round $n-1$ is followed by the first round), we define the binary variable $y_{kij} = 1$ if and only if team i plays against team j in round k, with $y_{kij} = 0$ otherwise. We also define the number of carry-over effects given by team i to team j as

$$z_{ij} = \sum_{\ell=1}^{n} \sum_{k=1}^{n-1} y_{k\ell i} \cdot y_{(k+1)\ell j}, \tag{1}$$

for $i \neq j$; $z_{ij} = 0$ if $i = j$. An integer programming formulation of the weighted carry-over effects value minimization problem is presented below:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot z_{ij}^2 \tag{2}$$

$$y_{kij} = y_{kji}, \qquad i, j, k = 1, \ldots, n \tag{3}$$

$$\sum_{i=1}^{n} y_{kij} = 1, \qquad j, k = 1, \ldots, n \tag{4}$$

$$\sum_{k=1}^{n-1} y_{kij} = 1, \qquad i, j = 1, \ldots, n : i \neq j \tag{5}$$

$$y_{kii} = 0, \qquad i, k = 1, \ldots, n \tag{6}$$

$$y_{nij} = y_{1ij}, \qquad i, j = 1, \ldots, n \tag{7}$$

$$y_{kij} \in \{0, 1\}, \qquad i, j, k = 1, \ldots, n. \tag{8}$$

The objective function (2) minimizes the weighted sum of carry-over effects. Constraints (3) ensure that variables $y_{kij} = y_{kji}$ are the same or, alternatively, that the game between teams i and j is the same as that between teams j and i (there is a unique game between teams i and j in a single round robin tournament). Constraints (4) enforce that every team plays exactly once in each round of a compact schedule. Constraints (5) and (6) guarantee that each team plays exactly once against every other team. Constraints (7) enforce that the n-th round is equivalent to the first. Constraints (8) impose the binary requirements on the variables.

This formulation has $O(n^3)$ variables and defines a quadratic minimization problem. The linearization of the objective function (2) leads to a new formulation with $O(n^4)$ variables. The commercial solver CPLEX 11.0 was applied to the solution of unweighted (i.e., $w_{ij} = 1, \forall i, j = 1, \ldots, n : i \neq j$) instances of this reformulation. Although the solver has been able to find the optimal solution for $n = 4$ in a couple of seconds and for $n = 6$ in a few minutes, it took about three days of computations time to come up with the optimal solution for $n = 8$. CPLEX was not able to solve instances with ten or more teams in reasonable times, leading to the need of heuristics for approximately solving larger problems.

3 Hybrid heuristic for weighted minimization

Russell [8] proposed a construction algorithm for the unweighted problem that generates fixtures matching the lower bound to the carry-over effects value when n is a power of two. The method proposed by Anderson [12] obtained solutions that are still the best known to date for unweighted instances. It makes use of algebraic structures called *starters* [13] to generate schedules. However, the approach presumes that a suitable starter is known beforehand, which may imply in huge computation times.

Trick [14] developed a constraint programming method that made it possible to prove the optimality of Russell's method for $n = 6$. Henz, Müller, and Thiel [15] improved the solution obtained by the previous approach for $n = 12$, also using constraint programming. Miyashiro and Matsui [16] developed a time-consuming heuristic based on random permutations of the rounds of fixtures created by the polygon method [17]. They reported more than two days of computation times for $n \geq 18$. However, their heuristic does not explore different strategies to generate the initial fixtures, which limits the quality of the solutions it can provide. Furthermore some of the above algorithms are computationally expensive and are not appropriate for the weighted variant of the carry-over effects value minimization problem.

We propose a tailored heuristic for the minimization of weighted carry-over effects. This heuristic is based on the hybridization of the Iterated Local Search (ILS) metaheuristic [18, 19] with a multistart strategy. It has two main steps: a multistart phase and an ILS phase. A complete run comprises a number of independent sequences of these two steps and returns the best solution found. The multistart phase generates 100 initial solutions, each of them obtained by a constructive method followed by a local search procedure. The best solution found during the multistart phase is used as the starting point for the ILS phase. The

main steps of the pseudo-code of the hybrid heuristic corresponding to Algorithm 1 are described in detail in the next sections.

Algorithm 1 Hybrid heuristic

```
 1: for iteration = 1 to 10 do
 2:     Build a starting fixture;
 3:     repeat
 4:         Generate a new initial solution by applying a constructive method;
 5:         Improve the initial solution by local search;
 6:         Update the best initial solution;
 7:     until 100 initial solutions are generated;
 8:     Set S as the best initial solution;
 9:     repeat
10:         Obtain a new solution S' by applying a perturbation to S;
11:         Apply local search to solution S';
12:         Replace the current solution S by S' using an acceptance criterion;
13:         Update the best known solution S*;
14:     until a stopping criterion is reached
15: end for
16: return S*;
```

3.1 Construction method to build initial solutions

A *factor* of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G, with $E' \subseteq E$. A factor G' is a *1-factor* if all its nodes have their degrees equal to one. A *factorization* F of G is a set of edge-disjoint factors of G, such that the union of their edge-sets is equal to E. A factorization of G formed exclusively by 1-factors is said to be a *1-factorization*. In an *ordered 1-factorization* of G, its 1-factors are taken in a fixed order.

There is a one-to-one correspondence between 1-factorizations and round robin schedules. If each team is assigned to a vertex of G, then each edge of the latter corresponds to a game. The games in each round correspond to the edges of a 1-factor, making the entire ordered 1-factorization equivalent to the complete tournament fixture.

The rounds of a tournament schedule can be freely permuted without violating any of its properties. Therefore, new schedules can be generated by picking the rounds of a given schedule in any possible order.

The construction method first generates one starting fixture in line 2 of Algorithm 1, using any of the two approaches described below. The first approach is the well known *polygon method* [17], which gives the so-called canonical 1-factorization [20]. It can be used for any value of n. Assuming the teams are numbered from 1 to n, the edge-set of the 1-factor corresponding to round k is given by $\{(k, n)\} \cup \{(a(k, \ell), b(k, \ell)) : \ell = 1, \ldots, n/2-1\}$, for $k = 1, \ldots, n-1$, with

$$a(k, \ell) = \begin{cases} k + \ell, & \text{if } (k + \ell) < n, \\ k + \ell - n + 1, & \text{if } (k + \ell) \geq n, \end{cases} \tag{9}$$

and

$$b(k, \ell) = \begin{cases} k - \ell, & \text{if } (k - \ell) > 0, \\ k - \ell + n - 1, & \text{if } (k - \ell) \leq 0. \end{cases} \tag{10}$$

The second approach [20,21] can be applied whenever n is a multiple of four. It first separates the teams in two sets $V_1 = \{1, \ldots, n/2\}$ and $V_2 = \{n/2+1, \ldots, n\}$. The first $n/2$ rounds are made up only by games with one team in V_1 and the other in V_2. The games scheduled for round k correspond to the edges of the factor whose edge-set is defined as $\{(\ell, c(k,\ell)) : \ell = 1, \ldots, n/2\}$, for $k = 1, \ldots, n/2$, with

$$c(k,\ell) = (k + \ell - 2) \bmod (n/2) + n/2 + 1. \tag{11}$$

Each of the last $n/2 - 1$ remaining rounds is formed by picking and putting together one 1-factor from a 1-factorization of the complete graph associated with V_1 and one 1-factor from a 1-factorization of the complete graph associated with V_2.

The loop in lines 3 to 7 builds each of the 100 initial solutions in line 4 as follows. First, two rounds of the starting fixture are randomly selected and used as the two first rounds of the new solution. In each subsequent step, one still unused round of the starting fixture is selected to be used in the current, incomplete fixture. This process continues, until a complete solution is obtained. The rules used for selecting and placing rounds of the starting fixture in the incomplete solution under construction mimic those in the well known *nearest neighbor* and *arbitrary insertion* heuristics for the traveling salesman problem [22, 23]:

- *nearest neighbor heuristic*: select an unused round which causes the minimum increment in the weighted carry-over effects value and place it as the last in the solution under construction.
- *arbitrary insertion heuristic*: randomly select any unused round and insert it between the two rounds which minimize the increase in the weighted carry-over effects value.

One of these strategies is randomly chosen with equal probability to generate a new initial solution at each iteration of the multistart phase.

3.2 Local search

The initial solutions built by the construction method described in the previous section are tentatively improved by local search in line 5 of Algorithm 1. We use a local search procedure following the Variable Neighborhood Descent (VND) [24,25] strategy. The best improving move in each neighborhood is applied to the current solution. The following neighborhood structures described by Costa et al. [26] (see also [4]) are used in this order:

- *Team swap* (TS): a move in this neighborhood corresponds to swapping all opponents of a given pair of teams over all rounds, which is the same as swapping two columns of the matrix representing the fixture. Figure 2 presents an example of a move using the TS neighborhood.
- *Round swap* (RS): a move in this neighborhood consists of swapping all games of a given pair of rounds, which is the same as swapping two rows of the matrix representing the fixture. Figure 3 presents an example of a move using the RS neighborhood.

– *Partial team swap* (PTS): for any round r and for any two teams t_1 and t_2, let S be a minimum cardinality subset of rounds including round r in which the opponents of teams t_1 and t_2 are the same. A move in this neighborhood corresponds to swapping the opponents of teams t_1 and t_2 over all rounds in S. Figure 4 presents an example of a move using the PTS neighborhood.
– *Partial round swap* (PRS): for any team t and for any two rounds r_1 and r_2, let U be a minimum cardinality subset of teams including team t in which the opponents of the teams in U in rounds r_1 and r_2 are the same. A move in this neighborhood consists of swapping the opponents of all teams in U in rounds r_1 and r_2. Figure 5 presents an example of a move using the PRS neighborhood.

If the locally optimal solution (with respect to neighborhoods TS, RS, PTS, and PRS) obtained by the VND local search strategy improves the best initial solution, then the latter is updated in line 6.

	A	B	C	D	E	F	G	H
1	H	C	B	E	D	G	F	A
2	C	D	A	B	G	H	E	F
3	D	E	F	A	B	C	H	G
4	E	F	H	G	A	B	D	C
5	F	G	E	H	C	A	B	D
6	G	H	D	C	F	E	A	B
7	B	A	G	F	H	D	C	E

(a)

	A	B	C	D	E	F	G	H
1	H	G	F	E	D	C	B	A
2	G	D	E	B	C	H	A	F
3	D	E	H	A	B	G	F	C
4	E	F	D	C	A	B	H	G
5	F	C	B	H	G	A	E	D
6	C	H	A	G	F	E	D	B
7	B	A	G	F	H	D	C	E

(b)

Fig. 2 Example of a move using the Team Swap neighborhood: (a) original fixture and (b) fixture with the games of teams C and G swapped.

	A	B	C	D	E	F	G	H
1	H	C	B	E	D	G	F	A
2	C	D	A	B	G	H	E	F
3	D	E	F	A	B	C	H	G
4	E	F	H	G	A	B	D	C
5	F	G	E	H	C	A	B	D
6	G	H	D	C	F	E	A	B
7	B	A	G	F	H	D	C	E

(a)

	A	B	C	D	E	F	G	H
1	H	C	B	E	D	G	F	A
2	F	G	E	H	C	A	B	D
3	D	E	F	A	B	C	H	G
4	E	F	H	G	A	B	D	C
5	C	D	A	B	G	H	E	F
6	G	H	D	C	F	E	A	B
7	B	A	G	F	H	D	C	E

(b)

Fig. 3 Example of a move using the Round Swap neighborhood: (a) original fixture and (b) fixture with rounds 2 and 5 swapped.

3.3 Iterated local search

The best initial solution obtained at the end of the multistart phase is copied in line 8 of Algorithm 1 to be used as the starting solution S by the ILS phase in lines 9-14, until some stopping criterion is met.

Each ILS iteration starts in line 10 by a perturbation of the current solution S. The perturbation consists of a sequence of random moves within the *game rotation* (GR) neighborhood introduced by Ribeiro and Urrutia [4]. For each move,

	t_1	t_2						
	A	B	C	D	E	F	G	H
1	D	F	E	A	C	B	H	G
$r = 2$	H	E	D	C	B	G	F	A
3	F	D	G	B	H	A	C	E
4	B	A	F	H	G	C	E	D
* 5	C	H	A	G	F	E	D	B
* 6	E	G	H	F	A	D	B	C
* 7	G	C	B	E	D	H	A	F

(a)

	t_1	t_2						
	A	B	C	D	E	F	G	H
1	D	F	E	A	C	B	H	G
$r = 2$	E	H	D	C	A	G	F	B
3	F	D	G	B	H	A	C	E
4	B	A	F	H	G	C	E	D
* 5	H	C	B	G	F	E	D	A
* 6	G	E	H	F	B	D	A	C
* 7	C	G	A	E	D	H	B	F

(b)

Fig. 4 Example of a move using the Partial Team Swap neighboorhood with $r = 2$, $t_1 = A$ and $t_2 = B$.

	t		*	*		*	*	*		
	A	B	C	D	E	F	G	H	I	J
$r_1 = 1$	C	I	A	F	J	D	H	G	B	E
2	J	H	D	C	G	I	E	B	F	A
3	G	F	I	J	H	B	A	E	C	D
$r_2 = 4$	H	E	F	G	B	C	D	A	J	I
5	E	C	B	H	A	J	I	D	G	F
6	B	A	J	E	D	G	F	I	H	C
7	D	J	G	A	I	H	C	F	E	B
8	I	D	H	B	F	E	J	C	A	G
9	F	G	E	I	C	A	B	J	D	H

(a)

	t		*	*		*	*	*		
	A	B	C	D	E	F	G	H	I	J
$r_1 = 1$	H	I	F	G	J	C	D	A	B	E
2	J	H	D	C	G	I	E	B	F	A
3	G	F	I	J	H	B	A	E	C	D
$r_2 = 4$	C	E	A	F	B	D	H	G	J	I
5	E	C	B	H	A	J	I	D	G	F
6	B	A	J	E	D	G	F	I	H	C
7	D	J	G	A	I	H	C	F	E	B
8	I	D	H	B	F	E	J	C	A	G
9	F	G	E	I	C	A	B	J	D	H

(b)

Fig. 5 Example of a move using the Partial Round Swap neighborhood with $t = A$, $r_1 = 1$ and $r_2 = 4$.

a randomly generated game is enforced to be changed from the round where it is currently played to a different, randomly selected round. This first change is followed by a sequence of modifications recovering a feasible solution S', according to the ejection chain mechanism proposed by Glover [27].

A bias is introduced in the perturbation procedure, to drive the move away from pure randomness. Although one cannot know beforehand the effect of a complete ejection chain move on the weighted carry-over effects value of a solution, the increment (or decrement) implied by the first step of the chain can be easily evaluated. The bias consists simply of enforcing the game with the smallest first increment.

The same local search procedure described in Section 3.2 is applied to the perturbed solution S' in line 11. The current solution S is replaced by S' in line 12 if the carry-over effects value of the latter is less than or equal to $(1 + b)$ times the carry-over effects value of the former. The value of parameter b doubles after every $2n$ iterations without the modification of the current solution. It is reset to its initial value whenever the current solution is updated, following the same strategy already used in [4]. Finally, the best known solution S^* is updated in line 13. The ILS phase stops if line 14 detects that a maximum number of deteriorating moves have been accepted since the last time the best solution was updated.

3.4 Iterated local search with destructive perturbations

Another perturbation strategy is based on the use of destructive neighborhoods [28, 29] to generate new solutions. Given the current solution, the main underlying idea consists in destroying randomly selected parts of the fixture, while keeping all the rest untouched. The partial solution made up of the untouched parts is given as input to an integer programming solver in order to obtain a new solution as the outcome of the perturbation, different from the current one. The subproblem solved consists simply in filling up the fixture to obtain a different feasible solution satisfying constraints (3)-(8) subject to the variable fixations.

Two rules for selecting the parts to be destroyed were devised and considered in the experiments, leading to two different types of neighborhoods that are described in the following sections.

3.4.1 Rows destruction

Considering the matricial representation of a round-robin tournament as illustrated in Figure 1 (a), each of its rows corresponds to a tournament slot. The games occurring in a given slot are indicated in the corresponding row. A destructive perturbation move in this neighboorhood is obtained by erasing a number of rows and replacing them by different ones. Given the current fixture, we enforce that some selected slots (rows) of the current solution will not occur in the new solution. Figure 6 illustrates a perturbation move in this neighboorhood. The shaded rows correspond to the slots selected to be destroyed and modified. While all other slots remain unchanged, the set of games of any destroyed slot cannot appear together in any of the slots of the new solution.

Let $\bar{y}$ be the current solution. We recall that $\bar{y}_{kij} = 1$ if and only if team i plays against team j in round k of the current fixture, $\bar{y}_{kij} = 0$ otherwise, for

every pair of teams $i, j = 1, \ldots, n$ (with $i \neq j$) and for every round $k = 1, \ldots, n$. Furthermore, let $D \subseteq \{1, \ldots, n\}$ be the set of rows to be destroyed in the current solution. Constraints (12) are added to the model to fix all rows not in D, while constraints (13) enforce the destruction of all rows in D and avoid their repetition in any other slot of the new solution:

$$y_{kij} = \bar{y}_{kij}, \qquad k \in \{1, \ldots, n\} \setminus D, \quad i, j = 1, \ldots, n : i \neq j \tag{12}$$

$$\sum_{\substack{i,j=1 \\ \bar{y}_{kij}=1}}^{n} y_{\ell ij} \leq n - 2, \qquad k, \ell \in D. \tag{13}$$

A different solution cannot be generated if one single row is destroyed. If two rows are destroyed, then only one different feasible fixture could be generated, corresponding to swapping these two rows. Experimental results showed that destroying three rows often makes it impossible to rebuild a feasible fixture. Therefore, we considered in the computational experiments only rows destruction perturbation moves involving exactly four rows, since the computation times increase enormously if more broader moves are permitted.

	A	B	C	D	E	F	G	H	I	J
1	B	A	I	H	G	J	E	D	C	F
2	G	F	E	J	C	B	A	I	H	D
3	I	H	G	F	J	D	C	B	A	E
4	J	C	B	G	I	H	D	F	E	A
5	D	I	J	A	F	E	H	G	B	C
6	E	D	H	B	A	I	J	C	F	G
7	H	G	F	E	D	C	B	A	J	I
8	F	E	D	C	B	A	I	J	G	H
9	C	J	A	I	H	G	F	E	D	B

(a) Current solution

	A	B	C	D	E	F	G	H	I	J
1	J	C	B	H	I	G	F	D	E	A
2	G	F	E	J	C	B	A	I	H	D
3	I	H	G	F	J	D	C	B	A	E
4	B	A	I	G	H	J	D	E	C	F
5	C	J	A	I	F	E	H	G	D	B
6	E	D	H	B	A	I	J	C	F	G
7	H	G	F	E	D	C	B	A	J	I
8	F	E	D	C	B	A	I	J	G	H
9	D	I	J	A	G	H	E	F	B	C

(b) Perturbed solution

Fig. 6 Perturbation move using the rows destruction neighborhood applied to slots 1, 4, 5, and 9.

3.4.2 Columns destruction

Considering the matricial representation of a round-robin tournament as illustrated in Figure 1 (a), each of its columns corresponds to the sequence of games

played by a team. A destructive perturbation move in this neighboorhood is obtained by erasing a number of columns and replacing them by different ones. Given the current fixture, we enforce that some selected teams (columns) of the current solution will not have the same sequence of opponents in the new solution. Figure 7 illustrates a perturbation move in this neighboorhood. The dark shaded columns correspond to the teams selected to have their game sequences destroyed and modified. All other columns remains unchanged, except the light shaded cells corresponding to games involving the teams associated with the columns selected for destruction.

Let again $\bar{y}$ be the current solution, with $\bar{y}_{kij} = 1$ if and only if team i plays against team j in round k of the current fixture, $\bar{y}_{kij} = 0$ otherwise, for every pair of teams $i, j = 1, \ldots, n$ (with $i \neq j$) and for every round $k = 1, \ldots, n$. Furthermore, let $D \subseteq \{1, \ldots, n\}$ be the set of columns to be destroyed in the current solution. Constraints (14) are added to the model to fix all the games of the columns not in D and whose associated opponent is also not in D, while constraints (15) are added to the model to enforce the destruction of all columns in D:

$$y_{kij} = \bar{y}_{kij}, \qquad k = 1, \ldots, n, \quad i, j \in \{1, \ldots, n\} \setminus D : j \neq i \tag{14}$$

$$y_{kij} \leq 1 - \bar{y}_{kij}, \qquad k = 1, \ldots, n, \quad i \in D, \quad j = 1, \ldots, n : j \neq i. \tag{15}$$

As for the previous case, the computational experiments addressed columns destruction perturbation moves involving the destruction of exclusively four columns.

	A	B	C	D	E	F	G	H	I	J
1	F	E	D	C	B	A	J	I	H	G
2	B	A	I	H	G	J	E	D	C	F
3	I	D	J	B	F	E	H	G	A	C
4	E	J	H	G	A	I	D	C	F	B
5	G	F	E	I	C	B	A	J	D	H
6	J	H	G	F	I	D	C	B	E	A
7	C	I	A	J	H	G	F	E	B	D
8	D	C	B	A	J	H	I	F	G	E
9	H	G	F	E	D	C	B	A	J	I

(a)

	A	B	C	D	E	F	G	H	I	J
1	J	E	H	I	B	G	F	C	D	A
2	B	A	I	F	G	D	E	J	C	H
3	I	F	D	C	H	B	J	E	A	G
4	E	D	F	B	A	C	H	G	J	I
5	G	H	E	J	C	I	A	B	F	D
6	F	J	G	H	I	A	C	D	E	B
7	C	I	A	G	J	H	D	F	B	E
8	H	C	B	E	D	J	I	A	G	F
9	D	G	J	A	F	E	B	I	H	C

(b)

Fig. 7 Perturbation move using the columns destruction neighborhood applied to columns D, F, H, and J.

4 Computational results

Each instance is defined by the number of teams and by a weighted matrix of carry-over effects. Four classes of weighted instances have been generated for the computational experiments:

- random instances: weights are randomly generated in the interval $[1, 2n]$. Three matrices identified by letters A, B and C have been generated for each value of n.
- linear instances: a strength in the interval $[1, n]$ is assigned to each team. For simplicity, the strength of team i is made equal to i. Each weight w_{ij} is defined as the absolute value of the difference between the strengths of teams i and j.
- perturbed linear instances: each weight of a linear instance is increased by an individual perturbation, randomly generated in the interval $[-n/2, n/2]$. Three different instances identified by letters A, B and C are generated for each value of n. Absolute values are taken whenever the perturbation leads to a negative weight.
- real-life inspired instances: these six instances are derived from the last six issues of the Brazilian football championship. The strength of each team is given by the number of points it obtained in the previous year. As for the linear instances, each weight w_{ij} is defined as the absolute value of the difference between the strengths of teams i and j.

Data of the weighted instances is available from the authors for benchmarking at `http://www.ic.uff.br/~celso/grupo/Weighted_carry-over_instances.zip`. We also considered instances made up only of unit costs, which are equivalent to those of the original, unweighted problem.

The computational experiments were performed on an AMD Athlon 64 X2 machine with 2.3 GHz and one GB of RAM memory. The code was implemented in C++ and compiled with the GNU C/C++ compiler (GCC) version 4.2.4 under Ubuntu Linux 8.04.

4.1 Tuning

The strategy used to generate the starting fixtures sensibly changes the quality of the solutions found. We tested two different alternatives to obtain the starting fixture whenever n is a multiple of four. In the first, only the polygon method is used and canonical factorizations are produced. In the second, the two methods described in Section 3.1 are indistinctly employed to obtain 1-factorizations.

We observed from preliminary results that the main parameters influencing the behavior of the ILS phase are the maximum number of deteriorating moves accepted before termination of this phase, the number of ejection chain (game rotation) moves applied in each perturbation, and the initial value for b. We also observed that reasonable values for these parameters are:

$$\begin{aligned} \text{maximum number of deteriorating moves: } & \{10, 100, 200\} \\ \text{number of ejection chain moves in a perturbation: } & \{1, 5, 10\} \\ \text{initial value for } b\text{: } & \{0.01, 0.05, 0.10\} \end{aligned} \tag{16}$$

To assess the behavior of the algorithm with respect to parameter values and implementation strategies, we ran the algorithm on three unweighted instances for

which n is a multiple of four – namely, for $n = 12, 16$, and 20. For each of these instances, we tested every possible combination of the above parameter values with the two alternatives for generating the initial fixture. For each configuration, we took average results over three independent runs. Only game rotation perturbation moves were used for tuning.

We first discuss the best strategy to generate the initial fixture. Table 1 displays the results obtained by each of the two alternatives for each problem size: average solution value, best solution value, average computation time (in seconds), and longest computation time (in seconds) over the three runs for each combination of the parameter values. The upper part of this table gives the results for the first alternative, based exclusively on canonical factorizations. The lower part reports results for the second alternative, which makes use of both approaches. Although the first alternative (upper part) is less time consuming than the second, the latter (lower part) seems to find better (or comparable) solutions. In consequence, we selected the second alternative to generate the initial fixture.

Strategy	n	COEV (avg.)	COEV (best)	Time (avg.)	Time (max.)
Canonical	12	192.000	192.000	3.519	4.000
factorizations	16	306.914	304.444	178.765	183.296
	20	489.852	488.444	20.790	21.296
Both	12	167.630	164.370	64.790	69.111
factorizations	16	267.630	258.519	244.827	263.926
	20	492.000	490.370	54.481	62.481

Table 1 COEV values and computation times in seconds for two different strategies for the generation of initial fixtures.

To choose the most appropriate configuration of parameter values, we ran the hybrid heuristic (using the second alternative to generate the initial fixture) with all of the 27 possible combinations of parameter values. We collected minimum and maximum results obtained over three runs of each instance tested with $n = 12, 16$, and 20. For each of them, the results found with each combination were normalized to the interval $[0, 1]$. Finally, we computed average normalized results over all instances considered. The plot in Figure 8 presents the average normalized carry-over effects value and the average normalized running time for each combination. Similarly, the plot in Figure 9 displays the average normalized best carry-over effects value and the average normalized longest computation time for each combination.

Each point in these plots represents a unique combination of parameter values. The five winning combinations closest to the ideal point $(0, 0)$ are the same for both plots. They are identified and labeled with the corresponding maximum accepted number of deteriorating moves, number of ejection chain moves in a perturbation, and initial value of parameter b (in this order).

Since four out of the five best combinations accept a maximum of 200 deteriorating moves before termination of the ILS phase, this value was selected for the first parameter. The number of ejection chain moves in a perturbation is set to one, since three out of the five best points plotted correspond to this value. The best choice for the initial value of parameter b is less evident from these results. For sake of robustness, we selected $b = 0.01$.

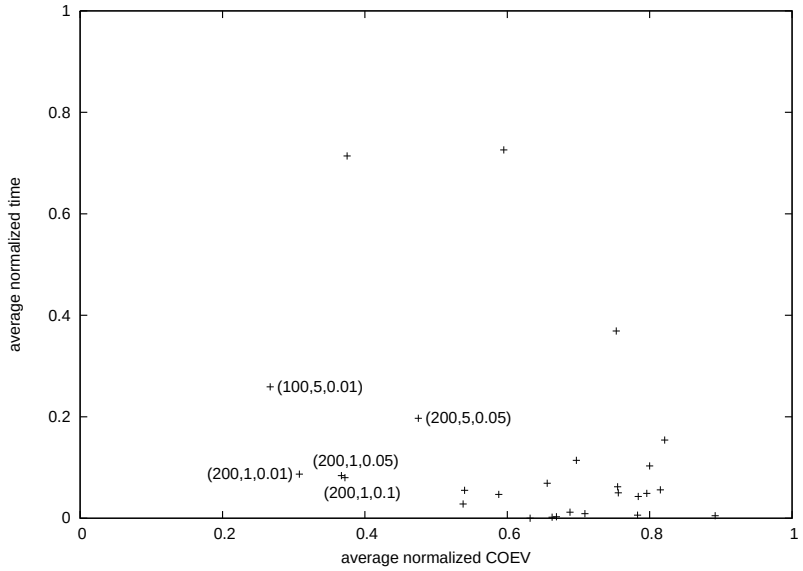

Fig. 8 Average normalized results obtained with different combinations of parameters.

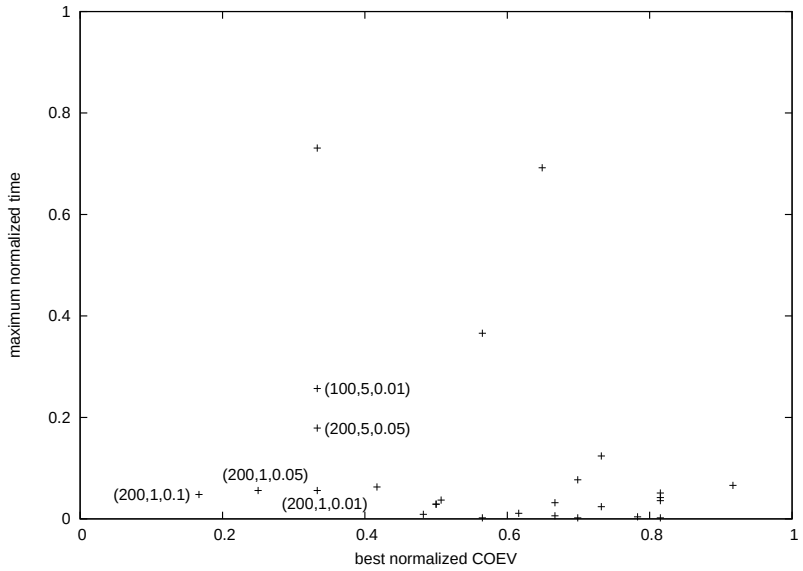

Fig. 9 Extremal normalized results obtained with different combinations of parameters.

4.2 Weighted instances

The hybrid heuristic was run five times for each instance, with the parameter values and implementation strategies as selected above. Average and best carry-over effects values and average and worst computation times in seconds over

five runs are reported for each class of weighted test instances, together with the average deviation from the best solution found.

Results for the random instances are shown in the upper part of Table 2. These instances present the smallest computation times over all classes. The ILS phase was always able to improve the solutions found in the multistart phase, except for the three smallest instances.

Instance	Avg. COEV	Dev. (%)	Best COEV	Avg. time (s)	Worst time (s)
inst4randomA	63.0	0.0	63	0.2	1.0
inst4randomB	53.0	0.0	53	0.2	1.0
inst4randomC	45.0	0.0	45	0.0	0.0
inst6randomA	233.0	0.0	233	0.8	1.0
inst6randomB	274.0	0.0	274	0.8	1.0
inst6randomC	235.0	0.0	235	1.2	2.0
inst8randomA	505.0	0.0	505	7.2	8.0
inst8randomB	495.0	0.0	495	18.6	20.0
inst8randomC	470.0	0.0	470	13.8	14.0
inst10randomA	912.6	2.0	895	139.4	150.0
inst10randomB	793.0	1.5	781	151.0	173.0
inst10randomC	744.6	1.0	737	149.0	172.0
inst12randomA	1549.2	1.9	1521	453.2	560.0
inst12randomB	1520.2	2.1	1489	402.4	451.0
inst12randomC	1594.4	1.4	1572	455.8	546.0
inst14randomA	2620.0	0.5	2608	39.8	44.0
inst14randomB	2903.8	1.2	2870	38.0	45.0
inst14randomC	2777.6	0.6	2760	36.8	42.0
inst16randomA	3812.2	1.5	3756	2605.8	2878.0
inst16randomB	3846.6	1.3	3797	2848.2	3294.0
inst16randomC	3773.6	0.5	3755	3138.0	3483.0
inst18randomA	5574.4	1.1	5515	1381.2	2442.0
inst18randomB	5816.2	1.2	5745	792.4	1271.0
inst18randomC	5598.0	0.9	5548	1547.2	3157.0
inst20randomA	7764.4	0.1	7760	136.8	145.0
inst20randomB	7928.0	0.5	7888	147.8	174.0
inst20randomC	7700.2	0.8	7636	136.8	144.0
Average		0.7			
inst4linear	20.0	0.0	20	0.2	1.0
inst6linear	114.0	0.0	114	0.6	1.0
inst8linear	168.0	0.0	168	8.4	9.0
inst10linear	318.0	0.0	318	64.6	70.0
inst12linear	504.0	1.6	496	271.4	309.0
inst14linear	960.0	0.2	958	18.6	20.0
inst16linear	1088.0	1.1	1076	2007.2	2313.0
inst18linear	1698.0	2.3	1660	4102.0	4729.0
inst20linear	2257.6	2.1	2212	5213.0	5624.0
Average		0.8			

Table 2 Results for random (upper part) and linear (lower part) instances.

We consider next the results for linear instances, reported in the lower part of Table 2. The ILS phase improved the solutions found in the multistart phase for all instances with $n \geq 8$. The computation times are much greater than those observed for the previous class of instances. This seems to be particularly true for the instance with 20 teams.

Results for perturbed linear instances are shown in the upper part of Table 3. As for the two previous classes, the ILS phase always improved the best solution found in the multistart phase, except for the three smallest instances. The computational times are greater than those observed for the random and linear classes. For both classes of random and linear instances, the heuristic was able to find very good solutions that are off the best values obtained by less than 1% on average, illustrating its robustness.

Instance	Avg. COEV	Dev. (%)	Best COEV	Avg. time (s)	Worst time (s)
inst4perturbedlinearA	16.0	0.0	16	0.2	1.0
inst4perturbedlinearB	17.0	0.0	17	0.2	1.0
inst4perturbedlinearC	22.0	0.0	22	0.0	0.0
inst6perturbedlinearA	68.0	0.0	68	1.4	2.0
inst6perturbedlinearB	73.0	0.0	73	0.6	1.0
inst6perturbedlinearC	60.0	0.0	60	0.8	1.0
inst8perturbedlinearA	137.0	0.0	137	31.0	32.0
inst8perturbedlinearB	141.0	0.0	141	22.4	23.0
inst8perturbedlinearC	162.0	0.0	162	26.8	28.0
inst10perturbedlinearA	329.0	0.9	326	136.0	162.0
inst10perturbedlinearB	277.0	1.1	274	134.0	163.0
inst10perturbedlinearC	291.8	2.7	284	128.6	152.0
inst12perturbedlinearA	601.2	2.4	587	484.2	638.0
inst12perturbedlinearB	528.6	0.7	525	494.6	580.0
inst12perturbedlinearC	486.6	1.8	478	546.6	648.0
inst14perturbedlinearA	940.0	2.2	920	42.8	45.0
inst14perturbedlinearB	947.4	1.7	932	39.8	43.0
inst14perturbedlinearC	993.2	0.3	990	36.2	40.0
inst16perturbedlinearA	1403.0	2.0	1376	3432.8	3905.0
inst16perturbedlinearB	1360.2	0.9	1348	3343.8	3668.0
inst16perturbedlinearC	1118.0	1.8	1098	3961.0	4683.0
inst18perturbedlinearA	2063.4	2.9	2005	5162.4	6983.0
inst18perturbedlinearB	1965.4	2.3	1921	4038.0	5343.0
inst18perturbedlinearC	1712.0	8.0	1585	4833.2	5805.0
inst20perturbedlinearA	3092.2	0.9	3065	482.0	1304.0
inst20perturbedlinearB	2866.8	2.4	2800	2923.4	4719.0
inst20perturbedlinearC	2837.2	1.7	2791	1479.6	3509.0
Averages		1.4			
inst24brasileirao2003	7730.4	2.5	7542	13897.8	15755.0
inst24brasileirao2004	7179.6	1.3	7088	12992.8	13468.0
inst22brasileirao2005	5228.8	1.4	5158	10599.0	13959.0
inst20brasileirao2006	5310.0	1.4	5236	5705.4	6358.0
inst20brasileirao2007	4876.0	0.9	4834	2715.8	3655.0
inst20brasileirao2008	4045.6	2.6	3944	6805.6	8308.0
Averages		1.7			

Table 3 Results for perturbed linear (upper part) and real-life inspired (lower part) instances.

Finally, the results obtained for the set of real-life inspired instances are shown in the lower part of Table 3. This class presents the largest computation times for the instances with $n = 20$. This seems to be due to the fact that the weights in this class are more diverse than in any other. As for the previous classes of instances, for both the perturbed linear and real-life-inspired instances, the heuristic was

able to find very good solutions that are off the best values obtained by less than 2% on average.

All instances with $n \leq 8$ teams have been exactly solved with CPLEX 11.0. For all these solved instances, the proposed heuristic always found an optimal solution in all runs, further illustrating its robustness. No larger instances, with $n \geq 10$ teams, could be solved within three entire days of computations. The lower bounds provided by the LP relaxation are very weak and do not help the solver.

4.3 Validation: unweighted instances

To validate the proposed hybrid heuristic, we also applied it to instances of the original unweighted problem. These instances are equivalent to weighted instances with unit weights.

Table 4 displays the best known solution values known to date and that obtained by the hybrid ILS heuristic for $n = 4, \ldots, 16$. The heuristic matched the best known results to date for $n = 4, 6, 8, 10$, and 16. Although it was not able to match the best known solution for $n = 14$, we observe that it improved the best known upper bound to date for $n = 12$ by almost 10%, producing a new, previously unknown solution whose carry-over effects value is only 160.

n	Best to date	Hybrid	
4	12	12	
6	60	60	
8	56	56	
10	108	108	
12	176	160	(new best)
14	234	254	
16	240	240	

Table 4 Results for the unweighted instances.

The ILS phase of the heuristic terminates once 200 deteriorating moves have been accepted without improvement in the best solution found. For all classes of weighted instances, we observed that this counter very rarely reached values greater than 50. However, a very different behavior was noticed for the unweighted instances, for which many improvements obtained during the ILS phase were achieved after more than 50 deteriorating moves.

4.4 Destructive neighboorhoods

In this section, we report on the computational results obtained with the use of destructive perturbations and compare them with those presented in the previous section.

Table 5 displays the results obtained with the three types of perturbations for the unweighted instances. It also displays the results obtained when one of the three neighborhoods is randomly selected with uniform probability to be used at each iteration. For each value of $n = 10, 12, 14$, and 16 we report the average and

best carry-over effects values, as well as the average and worst computation times (in seconds) over five runs of the heuristic.

Perturbation	n	Avg. COEV	Best COEV	Avg. time	Longest time
Game rotation	10	109.2	108.0	7.2	8.0
	12	164.4	160.0	15.2	19.0
	14	254.0	254.0	5.6	6.0
	16	244.8	240.0	71.2	83.0
	Averages	193.1	190.5	24.8	29.0
Rows destruction	10	108.0	108.0	311.8	330.0
	12	160.0	160.0	894.6	1068.0
	14	246.0	244.0	1093.2	1220.0
	16	240.0	240.0	1700.2	1913.0
	Averages	188.5	188.0	1000.0	1132.8
Columns destruction	10	108.0	108.0	60.6	68.0
	12	161.6	160.0	144.2	158.0
	14	248.0	246.0	269.0	298.0
	16	240.0	240.0	566.6	785.0
	Averages	189.4	188.5	260.1	327.3
Random selection	10	108.0	108.0	31.8	35.0
	12	162.4	160.0	72.8	78.0
	14	248.8	246.0	96.6	113.0
	16	243.2	240.0	309.6	386.0
	Averages	190.6	188.5	127.7	153.0

Table 5 Results for game rotation, rows destruction, columns destruction, and random neighborhood selection.

Game rotation perturbations provided the fastest (by far) algorithm. However, the use of destructive perturbations improved the carry-over effects value for the instance with $n = 14$ and matched the values obtained for the other instances. The algorithm using rows destruction perturbations always found all the best values over all variants, although at the cost of a big computational effort. The use of columns destruction perturbations, on the other hand, demanded much less computational effort, while still providing better results than game rotation perturbations. Randomly selecting the neighborhood to be used at each iteration appears to be a good compromise.

We observe that the use of columns destruction neighborhoods is much less time consuming than employing rows destructions, because the first fixes fewer elements of the fixture than the second, making it easier for the solver to find a feasible solution to replace the current.

5 Concluding remarks

We discussed possible applications of the minimum carry-over effects value minimization problem as a fairness criterion to build good fixtures for round robin tournaments. A new, weighted version of the problem was introduced and formulated by integer programming, in which a weight is assigned to each pair of teams.

A hybrid heuristic based on the combination of the Iterated Local Search metaheuristic with a multistart strategy was proposed and applied to four classes of

weighted problem instances with up to 20 teams. The weighted test instances are available from the authors for benchmarking purposes. Three perturbation strategies were proposed and compared. Numerical results obtained for the original, unweighted instances contributed to validate the effectiveness of the hybrid heuristic, which has improved the best known solution to date for $n = 12$.

These results confirm previous successful cases of the hybridization of the Iterated Local Search metaheuristic with multistart strategies (see e.g. [4, 30, 31]), in particular in the context of scheduling problems in sports.

References

1. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers and Operations Research **37**, 1–19 (2010)
2. Rasmussen, R.V., Trick, M.A.: Round robin scheduling – A survey. European Journal of Operational Research **188**, 617–636 (2008)
3. Easton, K., Nemhauser, G., Trick, M.A.: The travelling tournament problem: Description and benchmarks. In: T. Walsh (ed.) Principles and Practice of Constraint Programming, *Lecture Notes in Computer Science*, vol. 2239, pp. 580–585. Springer (2001)
4. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research **179**, 775–787 (2007)
5. Noronha, T.F., Ribeiro, C.C., Duran, G., Souyris, S., Weintraub, A.: A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament. In: Practice and Theory of Automated Timetabling VI, *Lecture Notes in Computer Science*, vol. 3867, pp. 174–186. Springer (2007)
6. Ribeiro, C.C., Urrutia, S.: Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. In: Practice and Theory of Automated Timetabling VI, *Lecture Notes in Computer Science*, vol. 3867, pp. 147–157. Springer, Berlin (2007)
7. Ribeiro, C., Urrutia, S.: Bicriteria integer programming approach for scheduling the Brazilian national soccer tournament. In: Proceedings of the Third International Conference on Management Science and Engineering Management, pp. 46–49. Bangkok (2009)
8. Russell, K.G.: Balancing carry-over effects in round robin tournaments. Biometrika **67**, 127–131 (1980)
9. Goossens, D., Spieksma, F.: Does the carry-over effect exist? In: Book of Abstracts of the 23rd European Conference on Operational Research, p. 288. Bonn (2009)
10. Flatberg, T., Nilssen, E., Stlevik, M.: Scheduling the topmost fotball leagues of Norway. In: Book of Abstracts of the 23rd European Conference on Operational Research, p. 240. Bonn (2009)
11. Goodbread, C.: SEC aims to fix schedule problem by month's end (2010). Online reference at `http://www.tidesports.com/article/20100409/NEWS/100409591`, last visited on August 14, 2010
12. Anderson, I.: Balancing carry-over effects in tournaments. In: F. Holroyd, K. Quinn, C. Rowley, B. Webb (eds.) Combinatorial Designs and Their Applications, CRC Research Notes in Mathematics, pp. 1–16. Chapman & Hall (1999)
13. Dinitz, J.H.: Starters. In: C.J. Colbourn, J.H. Dinitz (eds.) The CRC Handbook of Combinatorial Designs, The CRC Press Series on Discrete Mathematics and its applications, pp. 467–473. CRC Press, Boca Raton (1996)
14. Trick, M.A.: A schedule-then-break approach to sports timetabling. In: E. Burke, W. Erben (eds.) Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, *Lecture Notes in Computer Sciences*, vol. 2079, pp. 242–253. Springer-Verlag (2000)
15. Henz, M., Müller, T., Thiel, S.: Global constraints for round robin tournament scheduling. European Journal of Operational Research **153**, 92–101 (2004)
16. Miyashiro, R., Matsui, T.: Minimizing the carry-over effects value in a round robin tournament. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, pp. 460–463. Brno (2006)
17. Kirkman, T.: On a problem in combinations. Cambridge Dublin Math Journal **2**, 191–204 (1847)

18. Martin, O.C., Otto, S.W., Felten, E.W.: Large-step markov chains for the traveling salesman problem. Complex Systems **5**, 299–326 (1991)
19. Lourenço, H.R., Martin, O.C., Stutzle, T.: Iterated local search. In: F. Glover, G.A. Kochenberger (eds.) Handbook of Metaheuristics, pp. 321–353. Kluwer (2003)
20. de Werra, D.: Scheduling in sports. In: P. Hansen (ed.) Studies on Graphs and Discrete Programming, *Annals of Discrete Mathematics*, vol. 11, pp. 381–395. North Holland (1981)
21. de Werra, D.: Geography, games and graphs. Discrete Applied Mathematics **2**, 327–337 (1980)
22. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley (1985)
23. Gutin, G., Punnen, P. (eds.): The Traveling Salesman Problem and Its Variations. Kluwer (2002)
24. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **34**, 1097–1100 (1997)
25. Hansen, P., Mladenovic, N.: Variable neighborhood search. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer (2002)
26. Costa, F.N., Urrutia, S., Ribeiro, C.C.: An ILS heuristic for the traveling tournament problem with fixed venues. In: E.K. Burke, M. Gendreau (eds.) Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
27. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics **65**, 223–253 (1996)
28. Glover, F., Laguna, M.: Tabu Search. Kluwer (1997)
29. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics **29**(3), 653–684 (2000)
30. Duarte, A.R., Ribeiro, C.C., Urrutia, S., Haeusler, E.: Referee assignment in sports leagues. In: Practice and Theory of Automated Timetabling VI, *Lecture Notes in Computer Science*, vol. 3867, pp. 158–173. Springer (2007)
31. Lucena, A.P., Ribeiro, C.C., Santos, A.C.: A hybrid heuristic for the diameter constrained minimum spanning tree problem. Journal of Global Optimization (to appear)