A linear-time algorithm to solve the Sports League Scheduling Problem (prob026 of CSPLib)

Jean-Philippe Hamiez

LGI2P – École des Mines d'Alès (EERIE), Parc Scientifique Georges Besse, 30035 Nîmes CEDEX 01, France

Jin-Kao Hao *

LERIA – Université d'Angers, 2 boulevard Lavoisier, 49045 Angers CEDEX 01, France

Revised version, 23 May, 2003

Abstract

In this paper, we present a repair-based linear-time algorithm to solve a version of the *Sports League Scheduling Problem* (SLSP) where the number T of teams is such that $(T-1) \bmod 3 \neq 0$. Starting with a conflicting schedule with particular properties, the algorithm removes iteratively the conflicts by exchanging matches. The properties of the initial schedule make it possible to take the optimal exchange at each iteration, leading to a linear-time algorithm. This algorithm can find thus valid schedules for several thousands of teams in less than one minute. It is still an open question whether the SLSP can be solved efficiently when $(T-1) \bmod 3 = 0$.

Key words: Sports League Scheduling, linear-time algorithm, repair techniques.

1 Introduction

Many sports leagues (e.g., soccer, hockey, basketball) must deal with scheduling problems for tournaments. These scheduling problems contain in general many conflicting constraints to satisfy and different objectives to optimize, like

* Corresponding author.
Email address: Jin-Kao.Hao@univ-angers.fr (Jin-Kao Hao).

minimization of traveling distance [1,3], only one match per team and per day, stadium unavailability at particular dates, minimum number of days between a home match and its corresponding away match, etc. Generating satisfactory schedules with respect to these conditions and objectives is therefore a very difficult problem to solve.

Many solution approaches have been proposed to solve these problems with varying degrees of success: integer linear programming [9,17], constraint programming [14,20], local search (simulated annealing [22], tabu search [26], hybrid approach [4]). Sports scheduling was also investigated in terms of edge colorings of graphs [6–8].

This paper deals with a specific Sports League Scheduling Problem (SLSP for short) described by K. McAloon et al. in [16]. It is "prob026" of CSPLib [5]:

- There are T teams (T even). All teams play each other exactly once (half competition);
- The season lasts $T - 1$ weeks;
- Every team plays one game in every week of the season;
- There are $T/2$ periods and, each week, one game is scheduled in every period;
- No team plays more than twice in the same period over the course of the season.

The problem then is to schedule a tournament with respect to all these constraints. Table 1 shows an example of a valid schedule for $T = 8$ teams labeled from 1 to 8; there are 7 weeks and 4 periods.

Table 1
Example of a valid schedule for 8 teams

		Weeks						
		1	2	3	4	5	6	7
Periods	1	5, 8	2, 3	3, 4	1, 8	5, 6	6, 7	1, 7
	2	3, 7	1, 4	2, 5	3, 6	2, 8	1, 5	4, 8
	3	4, 6	6, 8	7, 8	2, 7	1, 3	2, 4	3, 5
	4	1, 2	5, 7	1, 6	4, 5	4, 7	3, 8	2, 6

As shown in Table 1, a configuration (schedule) may be represented as a two-dimensional array with weeks in columns and periods in rows. Each column satisfies the cardinality constraint: each team appears exactly once, i.e., all the teams are different. In each row, no team appears more than twice. There is also a global constraint on the array: each match appears only once, i.e., all matches are different.

In this paper, we present E3S and L3S, two new algorithms for a special case of the SLSP when the number T of teams is such that $(T-1) \bmod 3 \neq 0$ [1]. E3S and L3S are repair-based algorithms. They start with a particular conflicting schedule with interesting properties and then eliminate the conflicts by exchanging matches. The way the exchanges are realized is based on the properties of the initial conflicting schedule. While E3S explores in an exhaustive way these exchanges (exponential-time complexity), L3S is able to take the best exchange at each step of its search (linear-time complexity). L3S can thus find valid schedules for several thousands of teams in less than one minute, going beyond the state-of-the-art solutions limited to 40 teams. It is still an open question whether the SLSP can be solved efficiently when $(T-1) \bmod 3 = 0$.

The paper begins with a survey of related work (section 2), followed by a formulation of the SLSP as a constraint satisfaction problem (CSP) [23] (section 3). We present then the different elements of the E3S algorithm and its simplified linear-time version L3S (sections 4 and 5). Computational results are shown in section 6, followed by a discussion on an alternative formulation in terms of colorings (section 7). Concluding remarks are given in the last section.

2 Related work

With integer programming, McAloon et al. [16] solved the problem with 12 teams. They also experimented with constraint programming (ILOG Solver©), leading to slightly better results since solutions were found for 14 teams within 45 minutes. Finally, with a basic local search algorithm, they produced the same results as ILOG Solver© does, but in less computing time (10 minutes).

C.P. Gomes et al. [11] obtained better results than those of McAloon et al. using constraint programming. With a randomized version of a deterministic complete search they solved problems involving up to 18 teams in approximately 22 hours.

R. Béjar and F. Manyà [2] transformed the SLSP into the satisfiability problem in propositional logic (SAT) and used a SAT solver. They obtained solutions for 18 teams in less than 2 hours. They also solved the SLSP with 20 teams in about 13 hours.

J.C. Régin proposed two approaches with constraint programming [18,19]. The first one, using powerful filtering algorithms, produced better results than

[1] "mod" is the modulo operator.

those of Béjar and Manyà in terms of solution time and robustness, since he solved problem instances with 24 teams in 12 hours. In the second approach, the SLSP is transformed into an equivalent problem by adding an implicit constraint. With a new heuristic and specific filtering algorithms, solutions for 40 teams were found for the first time.

J.P. Hamiez and J.K. Hao developed two local search algorithms based on tabu search [10]. The first algorithm [12] produced results comparable with those of [18], solving the problem for 22 teams within 28 minutes. The second algorithm [13] includes a search space reduction technique and a restricted neighborhood. It produced results which compared well with those of [19] (the best known results at that moment). Solutions were found for 40 teams.

Finally, let us mention the work reported by G. Wetzel and F. Zabatta [25]: using multiple threads on a 14 processor Sun system they obtained results for 28 teams.

3 Problem formulation

The SLSP can be conveniently formulated as a constraint satisfaction problem. An alternative formulation, in terms of colorings, is discussed later on in section 7.

3.1 Constraint Satisfaction Problem – CSP

A constraint satisfaction problem [23] is defined by a triplet (X, D, C) with:

- A finite set X of M variables: $X = \{x_1, \ldots, x_M\}$;
- A set D of M associated domains: $D = \{D_1, \ldots, D_M\}$. Each domain $D_m (1 \leq m \leq M)$ specifies the finite set of possible values of the variable x_m;
- A finite set C of N constraints: $C = \{c_1, \ldots, c_N\}$. Each constraint is defined for a set of variables and specifies which combinations of values are compatible for these variables.

Given such a triplet, the problem is to generate a complete assignment of the values to the variables, which satisfies all the constraints: such an assignment is said to be consistent. Since the set of all assignments (not necessarily consistent) is defined by the Cartesian product $D_1 \times \cdots \times D_M$ of the domains, solving a CSP means to determine a particular assignment among a potentially huge number of possible assignments.

The CSP is a powerful and general model. It can be used to conveniently model some well-known problems such as k-coloring and satisfiability, as well as many practical applications related to resource assignment, planning or timetabling.

3.2 Formulation of the SLSP as a CSP

We will use the following notations to represent the SLSP as a constraint satisfaction problem:

- T: number of teams. Teams are numbered from 1 to T;
- P: number of periods, $P = T/2$;
- W: number of weeks, $W = T - 1$;
- $m = \langle t_1, t_2 \rangle$: schedule of the match m opposing teams t_1 and $t_2, (t_1, t_2) \in \{1, \ldots, T\}^2 (t_1 \neq t_2)$. Values of this variable type are of (p, w) pattern, meaning that m is scheduled in period $p (1 \leq p \leq P)$ and week $w (1 \leq w \leq W)$, noted $m \mapsto (p, w)$.

The set M of variables (which are *matches*) is naturally $M = \{m = \langle t_1, t_2 \rangle, 1 \leq t_1 < t_2 \leq T\}$ and all domains D_m are equal to $D = \{(p, w), 1 \leq p \leq P, 1 \leq w \leq W\}; \forall m \in M, D_m = D$. The set C of constraints contains the following three types of constraints:

WEEK constraint: Uniqueness of all teams in each week. For each team $t \in \{1, \ldots, T\}$ and each week $w \in \{1, \ldots, W\}$, we impose the constraint: $\text{WEEK}(t, w) \Leftrightarrow |\{m = \langle t_1, t \rangle \mapsto (p, w), 1 \leq t_1 \leq T (t_1 \neq t), 1 \leq p \leq P\}| = 1$;

PERIOD constraint: No more than two matches for each team in each period. For each team $t (1 \leq t \leq T)$ and each period $p (1 \leq p \leq P)$, we impose the constraint: $\text{PERIOD}(t, p) \Leftrightarrow |\{m = \langle t_1, t \rangle \mapsto (p, w), 1 \leq t_1 \leq T (t_1 \neq t), 1 \leq w \leq W\}| \leq 2$;

ALLDIFF constraint: All matches are different. For each tuple $(p_1, p_2, w_1, w_2) \in \{1, \ldots, P\}^2 \times \{1, \ldots, W\}^2$ with $m_1 = \langle t_1, t_2 \rangle \mapsto (p_1, w_1), 1 \leq t_1 < t_2 \leq T$ and $m_2 = \langle t_3, t_4 \rangle \mapsto (p_2, w_2), 1 \leq t_3 < t_4 \leq T$, we impose the constraint: $\text{ALLDIFF}(p_1, p_2, w_1, w_2) \Leftrightarrow (t_3, t_4) \neq (t_1, t_2)$ and $(t_4, t_3) \neq (t_1, t_2)$.

The WEEK and ALLDIFF constraints are always satisfied in our algorithm.

3.3 Search space – Complexity of the SLSP

As shown in Table 1 (section 1), a configuration s is a complete assignment of $D = \{(p, w), 1 \leq p \leq P, 1 \leq w \leq W\}$ items to variables of $M = \{m =$

$\langle t_1, t_2 \rangle, 1 \leq t_1 < t_2 \leq T\}$. Thus, a configuration is a $W * P$ sized table, whose items are integer couples $(t_1, t_2), 1 \leq t_1 < t_2 \leq T$. For $T = 50$ teams, this represents a problem with 1 225 variables (matches) and 1 225 values per variable.

There are $|M| = T(T-1)/2$ matches to be scheduled. A schedule can be thought of as a permutation of these matches. So, for T teams, the search space size is $[T(T-1)/2]!$ In other words, the search space size grows as the factorial of the square of $T/2$.

4 Solving the SLSP using an exhaustive repair method

Traditional complete procedures usually start with an empty assignment (configuration) s_0. Then, they iteratively choose a free variable $x \in X$ in configuration $s_k (k \in \quad)$ and a value $v \in D_x$ for this variable which does not violate any constraint. Next, a *branch* of the search tree is built by assigning v to x. This step leads to a *partial* valid assignment s_{k+1}. If no value v remains for a free variable x, the process returns (*backtracks*) to a previous valid assignment and tries other values. A solution s_* is found when all variables are assigned a value. Recall that a CSP has no solution (it is said to be *unsatisfiable*) if the process backtracks until the root of the search tree and no value remains for the starting variable.

Our exhaustive repair approach (let us call it E3S, for *Exhaustive Sport Scheduling Solver*) is different since it starts with a complete conflicting assignment, like in local search based methods. The initial configuration s_0 is built in order to satisfy the WEEK and ALLDIFF constraints; at this stage, the remaining PERIOD constraint is not verified. The goal of E3S is then to satisfy the PERIOD constraint. With this statement, a branch of E3S corresponds to particular swaps of matches, with at least one involved in the PERIOD constraint. So, nodes of the search tree are complete assignments, and leaves are failures or solutions.

We detail hereafter the way we build the initial assignment and its properties, the repair rule and, finally, the overall E3S procedure.

4.1 Building the root of the search tree

The initial assignment is built (in linear-time complexity) using results of graph theory, namely *edge-coloring of complete graphs*. This construction has been studied in [7,21] for sports scheduling. We recall the geometrical construc-

tion scheme proposed in [21] followed by an equivalent formalism inspired from [24].

In the following descriptions, we use "mod" to refer to the *modulo* operator. Let $a = b * q + r$ (Euclidean division) with $(a, b, q, r) \in \ ^4$, then $a \bmod b = r$.

4.1.1 Geometrical construction

(1) Construct a complete undirected graph with the first $T - 1$ teams as vertices. Place the vertices in order to form a regular polygon, see left drawing in Fig. 1. Remark that each edge $\langle t_1, t_2 \rangle, 1 \leq t_1 < t_2 \leq T - 1$, represents the match opposing teams t_1 and t_2;

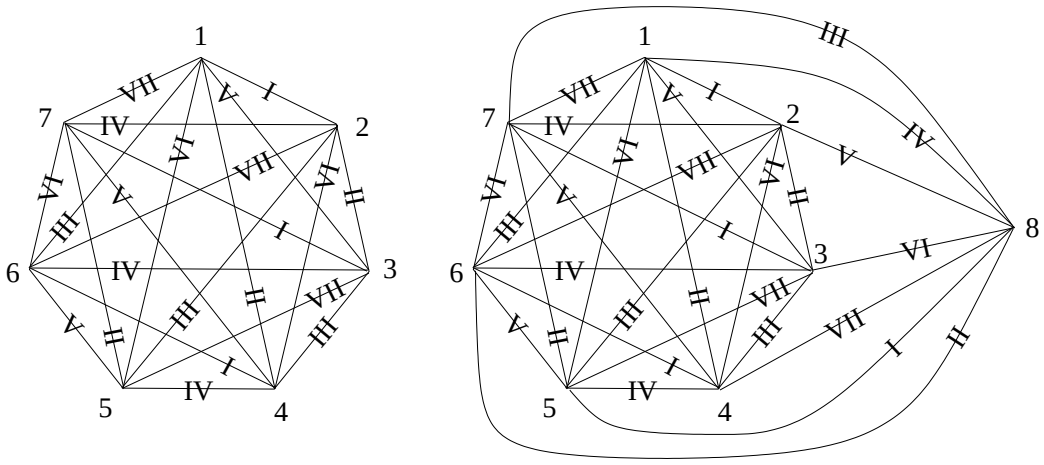

Fig. 1. Constructing initial configuration ($T = 8$)

(2) Color the W edges around the boundary using a different color $w = t$ for each edge $\langle t, [t \bmod (T - 1)] + 1 \rangle, 1 \leq t \leq T - 1$. These edges are used to initialize the first period of the tournament (see Table 2): $\forall m = \langle t, [t \bmod (T - 1)] + 1 \rangle, m \mapsto (1, t)$. Note that colors map weeks;
(3) Remaining edges are colored by assigning to each edge the same color as that used for the boundary edge parallel to it. At each vertex $t (1 \leq t \leq T - 1)$ there will be exactly one color $w_t (1 \leq w_t \leq W)$ missing and these missing colors are different;
(4) Add a new vertex T to the graph and link it with all the other vertices $t (1 \leq t \leq T - 1)$. The edges $\langle t, T \rangle$ of the new complete graph incident to the last vertex (i.e., the last team T) can be colored using the missing colors $w_t (1 \leq w_t \leq W)$ identified in the previous step, see right drawing in Fig. 1. These edges will be placed in the last period P of the schedule making the PERIOD constraint violated (see Table 2): $\forall m = \langle t, T \rangle, m \mapsto (P, w_t)$;
(5) Finally, fill in week $w (1 \leq w \leq W)$, in the initial configuration s_0, with edges colored w.

Table 2
Initial schedule for 8 teams

		Weeks						
		1	2	3	4	5	6	7
Periods	1	1, 2	2, 3	3, 4	4, 5	5, 6	6, 7	1, 7
	2	3, 7	1, 4	2, 5	3, 6	4, 7	1, 5	2, 6
	3	4, 6	5, 7	1, 6	2, 7	1, 3	2, 4	3, 5
	4	5, 8	6, 8	7, 8	1, 8	2, 8	3, 8	4, 8

4.1.2 Formal model

- Let the match m in week 1 of the first period be $\langle 1, 2\rangle : m \mapsto (1, 1)$;
- Fill in all the periods $p(2 \leq p \leq P-1)$ of the **first week** with the match $m = \langle p+1, T-p+1\rangle : m \mapsto (p, 1)$;
- Let t_1 and t_2 be the two teams $(1 \leq t_1 < t_2 \leq T-1)$ of the match m_1 scheduled on the period p in the week $w : m_1 = \langle t_1, t_2\rangle \mapsto (p, w)$. Initialize period $p(1 \leq p \leq P-1)$ and week $w+1$ with the match $m_2 = \langle t_3, t_4\rangle$ where $t_3 = t_1 + 1$ and $t_4 = (t_2 \bmod W) + 1 : m_2 \mapsto (p, w+1)$;
- Finally, fill in all the weeks $w(1 \leq w \leq W)$ of the **last period** $P = T/2$ with the match $m = \langle [(P + w - 1) \bmod W] + 1, T\rangle : m \mapsto (P, w)$.

4.2 Properties of the root configuration

The two previous equivalent construction schemes lead to a starting tournament which embodied several interesting properties. These properties are really important since the repair rule of the E3S algorithm (next section) extensively uses some of them. Proofs are not given here since they are almost evident and naturally result from the construction step.

Property 1 *The ALLDIFF constraint is satisfied.*

Property 2 *The WEEK constraint is verified.*

Property 3 *The last team T only appears in all the matches of the last period P, each time with a different team $t, 1 \leq t \leq T-1 : \forall m = \langle t, T\rangle, m \mapsto (P, w), 1 \leq w \leq W$. The PERIOD constraint is then violated in the last period.*

Property 4 *Each of the first $T-1$ teams appears exactly twice in each of the first $P-1$ periods.*

Remark that the set of matches in each of the first $P-1$ periods corresponds to an Hamiltonian circuit in the complete undirected graph, see Fig. 2 together

with Table 2 (section 4.1.1). This remark holds only for problem instances where $(T-1) \bmod 3 \neq 0$. When $(T-1) \bmod 3 = 0$, a particular "failing" period $p_{\mathcal{F}}(1 \leq p_{\mathcal{F}} \leq P-1)$ does not follow this topological observation. Indeed, the set of matches in period $p_{\mathcal{F}}$ corresponds to an union of circuits of length 3 (triangles).

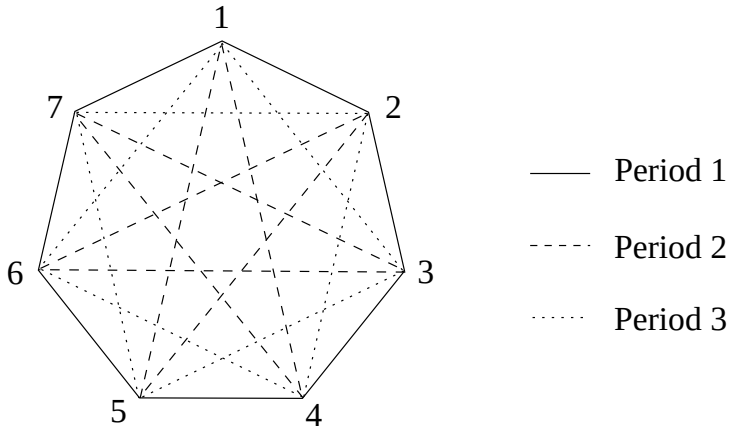

Fig. 2. Using Hamiltonian circuits to fill the first $P-1$ periods ($T=8$)

4.3 Repair rule

Our exhaustive repair algorithm starts with a **full assignment** s_0 which contains conflicts for the last team T in the last period P. So, we wish to remove these conflicts. Since the first $P-1$ periods do not contain the last team (Property 3), E3S will try to distribute the conflicting matches into these periods. This is done by swapping matches, **within the same week** to keep the WEEK constraint satisfied. Before describing how repair is done, we give hereafter a function which can be used to compute the number of conflicts at any node (configuration s) of the search tree $\mathcal{T}$. We finally give an illustration of the repair rule (section 4.3.3).

4.3.1 Evaluating nodes

Let $OccP_s(p,t)$ be the occurrence number, at node s, of team $t(1 \leq t \leq T)$ in period $p, 1 \leq p \leq P$. The number of conflicts $f(s)$ is naturally the total number of excess appearances of all teams in all periods, i.e., the minimum number of variables to be changed to satisfy the PERIOD constraint:

$$f(s) = \sum_{p=1}^{P} \sum_{t=1}^{T} \chi(s,t,p), \tag{1}$$

$$\text{with } \chi(s,t,p) = \begin{cases} 0 & \text{if } OccP_s(p,t) \leq 2, \\ OccP_s(p,t) - 2 & \text{otherwise.} \end{cases}$$

Solving the Sports League Scheduling Problem means finding a particular configuration s_* such as $f(s_*) = 0$. This can be done by minimizing f.

Property 5 *For the initial configuration* s_0, $f(s_0) = T - 3$.

PROOF.

$$\text{Property 3} \Leftrightarrow \begin{cases} \forall p \in \{1, \ldots, P-1\}, \chi(s_0, T, p) = 0; \\ \forall t \in \{1, \ldots, T-1\}, \chi(s_0, t, P) = 0. \end{cases}$$

$$\text{Property 4} \Leftrightarrow \forall t \in \{1, \ldots, T-1\} \text{ and } \forall p \in \{1, \ldots, P-1\}, \chi(s_0, t, p) = 0;$$

$$\begin{aligned} \text{So, } f(s_0) &= \chi(s_0, T, P) \\ &= OccP_{s_0}(P, T) - 2 \\ &= (T-1) - 2 \text{ (Property 3).} \end{aligned} \qquad \square$$

Note that this property means that at least $T - 3$ matches of the last period P must be rescheduled within other periods p where $1 \leq p \leq P - 1$.

4.3.2 Repairing by means of swaps

In configuration $s_i (i \in \quad)$, let a *primal match* $m_{\mathcal{P}} = \langle t_{\mathcal{P}}, T \rangle, 1 \leq t_{\mathcal{P}} \leq T - 1$, be a conflicting match of the last period P in week $w_{\mathcal{P}} (1 \leq w_{\mathcal{P}} \leq W)$; $OccP_{s_i}(P, T) > 2$. A *primal swap* is an exchange between a primal match $m_{\mathcal{P}} \mapsto (P, w_{\mathcal{P}})$ and a non-conflicting match $m_1 = \langle t_1, t_{\mathcal{D}} \rangle, (t_1, t_{\mathcal{D}}) \in \{1, \ldots, T-1\}^2 (t_1 \neq t_{\mathcal{D}})$, subject to:

- m_1 is located in week $w_{\mathcal{P}}$ and a period $p_{\mathcal{P}} (1 \leq p_{\mathcal{P}} \leq P-1) : m_1 \mapsto (p_{\mathcal{P}}, w_{\mathcal{P}})$;
- t_1 and $t_{\mathcal{D}}$ appeared at most once in period P before swapping: $OccP_{s_i}(P, t_1) \leq 1$ and $OccP_{s_i}(P, t_{\mathcal{D}}) \leq 1$.

Note that, according to Property 3, $W * (P - 1)$ primal swaps are available from the starting configuration s_0.

Due to Property 4, a primal swap will add a conflict in period $p_{\mathcal{P}}$ for the team $t_{\mathcal{P}}$ while removing a conflict in the last period P for the team T. The primal swap definition insures that this exchange does not add any conflict for the teams t_1 and $t_{\mathcal{D}}$ in period P.

From any configuration $s_i (i \in \quad)$ of the search tree, this first step leads then to a configuration s_{i+1} such that $f(s_{i+1}) = f(s_i)$. In other words, the new configuration s_{i+1} has exactly the same number of conflicts than s_i. The conflicts are now distributed not only into the last period P but also in a period $p_{\mathcal{P}}, 1 \leq p_{\mathcal{P}} \leq P-1$. So, this step is not enough to remove conflicts.

Similarly to the previous definition of a primal match, let us define a *dual match* $m_{\mathcal{D}}$ to be a conflicting match of the last period P in a week $w_{\mathcal{D}} (1 \leq w_{\mathcal{D}} \leq W) : m_{\mathcal{D}} \mapsto (P, w_{\mathcal{D}})$. Given a primal swap, in week $w_{\mathcal{P}} (1 \leq w_{\mathcal{P}} \leq W)$, between $m_{\mathcal{P}} = \langle t_{\mathcal{P}}, T \rangle (1 \leq t_{\mathcal{P}} \leq T-1)$ and a match $m_1 = \langle t_1, t_{\mathcal{D}} \rangle, (t_1, t_{\mathcal{D}}) \in \{1, \ldots, T-1\}^2 (t_1 \neq t_{\mathcal{D}})$ located in a period $p_{\mathcal{P}} (1 \leq p_{\mathcal{P}} \leq P-1)$, a *dual swap* is an exchange between a dual match $m_{\mathcal{D}}$ and a **conflicting** match m_2 subject to:

(1) $w_{\mathcal{D}} \neq w_{\mathcal{P}}$;
(2) $m_{\mathcal{D}} = \langle t_{\mathcal{D}}, T \rangle$;
(3) $m_2 = \langle t_2, t_{\mathcal{P}} \rangle (1 \leq t_2 \leq T-1, t_2 \neq t_{\mathcal{P}})$ is scheduled in week $w_{\mathcal{D}}$ and period $p_{\mathcal{P}} : m_2 \mapsto (p_{\mathcal{P}}, w_{\mathcal{D}})$;
(4) t_2 appeared at most once in period P before primal and dual swaps: $OccP_{s_i}(P, t_2) \leq 1$.

The first objective of the dual swap is to remove the conflict generated on period $p_{\mathcal{P}}$ for the team $t_{\mathcal{P}}$ (due to a primal swap). This is done by imposing condition 3. Its second objective is to remove another conflict in the last period P for the team T. Applying a dual swap (if available) after a primal swap to a configuration s_i insures then to reach a new configuration s_{i+1} with the following features:

- All teams $t (1 \leq t \leq T)$, except t_1 and t_2, occur exactly twice in period $p_{\mathcal{P}} : \forall t \in \{1, \ldots, T\} / t \neq t_1$ and $t \neq t_2, OccP_{s_{i+1}}(p_{\mathcal{P}}, t) = 2$;
- Teams t_1 and t_2 occur exactly once in period $p_{\mathcal{P}} : OccP_{s_{i+1}}(p_{\mathcal{P}}, t_1) = OccP_{s_{i+1}}(p_{\mathcal{P}}, t_2) = 1$;
- s_{i+1} have two conflicts less in period P (for team T) than $s_i : OccP_{s_{i+1}}(P, T) = OccP_{s_i}(P, T) - 2$.

The first two features mean that there is no more conflicts in period $p_{\mathcal{P}}$. The last feature is very interesting since it simplifies the formula 1 (previous section) to $f(s_{i+1}) = f(s_i) - 2$, with $f(s_0) = T-3$ (Property 5). The correctness of this new formula relies on the fact that periods $p (1 \leq p \leq P-1, p \neq p_{\mathcal{P}})$ are not modified when a dual swap immediately follows a primal swap. It also means that, from the root configuration s_0, **exactly** $T-2$ matches of the last period P will be swapped into the first $P-1$ periods. Hence, the depth of the search tree will be $P-1$.

E3S repair rule: At node s_i of depth d_i (assuming $d_0 = 0$), apply a primal swap followed by one of its corresponding dual swap, with $p_{\mathcal{P}} = d_i + 1$.

From a theoretical point of view, remark that the number B_i of possibilities to choose primal and dual matches ($m_{\mathcal{P}}$ and $m_{\mathcal{D}}$) at each node $s_i (i \in \quad)$ can be quite large since at most two dual matches correspond to each of the $W - 2d_i$ matches candidate to be primal ones (mainly due to Property 4). This suggests that $B_i \leq 2 * (W - 2d_i)$. This upper bound can easily be reduced using the natural symmetries of the SLSP, namely by replacing condition 1 with "$w_{\mathcal{D}} > w_{\mathcal{P}}$", leading thus to $B_i \leq W - 2d_i$.

4.3.3 Illustration of the repair rule

Table 3 shows an extract of the starting root s_0 for $T = 8$ teams ($f(s_0) = 5$) restricted to periods 1 and 4. The corresponding full configuration is available in Table 2 (section 4.1.1).

Table 3
Starting configuration for 8 teams (extract)

		Weeks						
		1 ($w_{\mathcal{P}}$)	2	3	4	5	6	7
Periods	1 ($p_{\mathcal{P}}$)	1, 2 (m_1)	2, 3	3, 4	4, 5	5, 6	6, 7	1, 7
	4	5, 8 ($m_{\mathcal{P}}$)	6, 8	7, 8	1, 8	2, 8	3, 8	4, 8

Since $d_0 = 0$, we try to identify a primal match $m_{\mathcal{P}}$ (in period $P = 4$) to be swapped with a match m_1 scheduled in period $p_{\mathcal{P}} = 1 (= d_0 + 1)$. Suppose we choose the primal match $m_{\mathcal{P}} = \langle 5, 8 \rangle \mapsto (4, 1) : w_{\mathcal{P}} = 1, t_{\mathcal{P}} = 5$. This choice forces us to swap it with the match m_1 scheduled in period $p_{\mathcal{P}} = 1$ and week $w_{\mathcal{P}} = 1; m_1 \mapsto (1, 1)$. That is $m_1 = \langle 1, 2 \rangle : t_{\mathcal{D}} = 1, t_1 = 2$.

Table 4
Intermediate configuration for 8 teams (extract)

		Weeks						
		1 ($w_{\mathcal{P}}$)	2	3	4 ($w_{\mathcal{D}}$)	5	6	7
Periods	1 ($p_{\mathcal{P}}$)	5, 8 ($m_{\mathcal{P}}$)	2, 3	3, 4	4, 5 (m_2)	5, 6	6, 7	1, 7
	4	1, 2 (m_1)	6, 8	7, 8	1, 8 ($m_{\mathcal{D}}$)	2, 8	3, 8	4, 8

The resulting intermediate configuration is given in Table 4. The primal swap generated a conflict for the team $t_{\mathcal{P}} = 5$ in period $p_{\mathcal{P}} = 1$ (it appears three times) while removing a conflict for team $T = 8$ in the last period $P = 4$. Observe also that teams $t_{\mathcal{D}} = 1$ and $t_1 = 2$ don't add any conflict in the last period $P = 4$. The number of conflicts of this intermediate configuration is the same as in s_0: 5.

The dual swap imposes then to remove the conflict generated by the primal swap in period $p_{\mathcal{P}} = 1$ for the team $t_{\mathcal{P}} = 5$. This can be achieved by swapping

either the matches $m_2 = \langle 4, 5\rangle (t_2 = 4)$ and its corresponding dual match $m_{\mathcal{D}} = \langle 1, 8\rangle (w_{\mathcal{D}} = 4)$, or matches $\langle 5, 6\rangle$ and $\langle 2, 8\rangle (w_{\mathcal{D}} = 5)$.

The first choice leads to the configuration reported in Table 5. Observe that $OccP_{s_1}(1, 2) = OccP_{s_1}(1, 4) = 1$, following the definition of primal and dual swaps. There is no conflict in periods 1 to 3 and the last period ($P = 4$) have now two conflicts less, namely 3 ($f(s_1) = 3$) instead of 5 in s_0.

Table 5
Result of the first branching (8 teams)

		Weeks						
		1	2	3	4	5	6	7
Periods	1	5, 8	2, 3	3, 4	1, 8	5, 6	6, 7	1, 7
	2	3, 7	1, 4	2, 5	3, 6	4, 7	1, 5	2, 6
	3	4, 6	5, 7	1, 6	2, 7	1, 3	2, 4	3, 5
	4	1, 2	6, 8	7, 8	4, 5	2, 8	3, 8	4, 8

4.4 E3S: overall procedure

Starting from the particular root configuration s_0 defined in section 4.1, the repair rule imposes to iteratively move conflicting matches from the last period P to periods 1, 2, ..., $P - 1$, respecting this order. Given T, this leads to the following E3S($p_{\mathcal{P}}, w$) procedure, which is naturally recursive. E3S is first called with $p_{\mathcal{P}} = 1$ and $w = 1$, meaning that we look for a primal match $m_{\mathcal{P}}$ in week 1 and a dual match $m_{\mathcal{D}}$ in week $w_{\mathcal{D}} > 1$ to be swapped with their corresponding matches in period $p_{\mathcal{P}} = 1$.

E3S($p_{\mathcal{P}}, w$)**:**

(1) **if** $p_{\mathcal{P}} = P$ **then return** True. {*A solution is obtained.*}
E3S has succeed to branch on all periods $p_{\mathcal{P}} (1 \leq p_{\mathcal{P}} \leq P - 1)$;
(2) $w_{\mathcal{P}} \leftarrow$ **PrimalWeek**($p_{\mathcal{P}}, w$).
Try to find the first week $w_{\mathcal{P}} (w \leq w_{\mathcal{P}} < W)$ such that $m \mapsto (P, w_{\mathcal{P}})$ is a primal match, with respect to $m_1 \mapsto (p_{\mathcal{P}}, w_{\mathcal{P}})$;
(3) **if** no $w_{\mathcal{P}}$ exists **then return** False.
No primal swap can be performed between periods P and $p_{\mathcal{P}}$;
(4) *A primal match* $m_{\mathcal{P}} \mapsto (P, w_{\mathcal{P}})$ *is available.*
Exchange($p_{\mathcal{P}}, w_{\mathcal{P}}$).
Primal swap between $m_{\mathcal{P}} \mapsto (P, w_{\mathcal{P}})$ and $m_1 \mapsto (p_{\mathcal{P}}, w_{\mathcal{P}})$;
(5) **if DualWeek**($p_{\mathcal{P}}, w_{\mathcal{P}}$) = False **then**
(a) **Exchange**($p_{\mathcal{P}}, w_{\mathcal{P}}$).
A *backtrack* is made since the primal swap done in step 4 always

leads to a failure;

(b) **return E3S**$(p_{\mathcal{P}}, w_{\mathcal{P}} + 1)$.
Try to find a primal match into the next week;

(6) **return** True.
The primal swap done in step 4 leads to a solution.

The **DualWeek()** function checks if one of the two possible dual swaps (if enable) leads to a solution. If so, it returns *True*, else *False*.

4.5 Complexity of E3S

One way to compute the E3S complexity is to estimate the size $|\mathcal{T}|$ (in number of nodes) of the tree $\mathcal{T}$ developed by E3S in the worst case.

From a theoretical point of view, this worst case occurs when, at node $s_i (i \in$), each possible primal swap (there are $W - 2d_i$) leads to two possible dual swaps. In other words, and due to the natural symmetries of the problem, E3S may generate, from each node s_i, a maximum of $B_i = W - 2d_i$ nodes $s_{i+1}^k (1 \leq k \leq W - 2d_i)$.

$$\text{So, } |\mathcal{T}| = \prod_{i=0}^{i \leq P-2} (W - 2d_i) \tag{2}$$

Roughly speaking, formula 2 suggests that $|\mathcal{T}|$ is in $O(\sqrt{T^T})$.

4.6 Limitations of E3S

The design of E3S is such that it cannot find solution for some particular T, precisely when $(T - 1) \bmod 3 = 0$. In that special failing case, the initial configuration has an extra property related to the specific "failing" period $p_{\mathcal{F}}$ introduced in section 4.2. $p_{\mathcal{F}}$ is such that, even if a primal swap is available for it, no dual swap at all can be performed. So, E3S cannot branch on this period to reduce (by 2) the number of conflicts. Then, the search always ends with remaining conflicts in the last period.

Table 6 gives an extract of the initial configuration s_0 built for $T = 10$ teams ($9 \bmod 3 = 0$), restricted to periods $p_{\mathcal{F}} = 2$ and the last one. It's easy to see that no dual swap is possible whatever any of the nine possible primal swaps is performed.

For instance, suppose we do a primal swap in week $w_{\mathcal{P}} = 3$ with $p_{\mathcal{P}} = p_{\mathcal{F}} =$

Table 6
The "failing" period (10 teams)

		Weeks								
		1	2	3 ($w_{\mathcal{P}}$)	4	5	6	7	8	9
Periods	2 ($p_{\mathcal{F}}$)	3,9	1,4	2,5 (m_1)	3,6	4,7	5,8	6,9	1,7	2,8
	5	6,10	7,10	8,10 ($m_{\mathcal{P}}$)	9,10	1,10	2,10	3,10	4,10	5,10

$2 : m_{\mathcal{P}} = \langle 8, 10\rangle (t_{\mathcal{P}} = 8)$ and $m_1 = \langle 2, 5\rangle \mapsto (2, 3)$. This generates a conflict for team $t_{\mathcal{P}} = 8$ in period $p_{\mathcal{P}} = 2$ and removes a conflict in period 5 for team 10. The resulting intermediate configuration is shown in Table 7.

Table 7
The "failing" period for $T = 10$ teams (intermediate configuration)

		Weeks								
		1	2	3 ($w_{\mathcal{P}}$)	4	5	6 (w_1)	7	8	9 (w_2)
Periods	2 ($p_{\mathcal{F}}$)	3,9	1,4	8,10 ($m_{\mathcal{P}}$)	3,6	4,7	5,8 (m_2)	6,9	1,7	2,8 (m'_2)
	5	6,10	7,10	2,5 (m_1)	9,10	1,10	2,10 (m)	3,10	4,10	5,10 (m')

Only two possible swaps are then "candidates" to end branching (in weeks $w_1 = 6$ and $w_2 = 9$) but they are not **dual swaps** as formally defined in section 4.3.2. Swapping $m = \langle 2, 10\rangle \mapsto (5, 6)$ and $m_2 = \langle 5, 8\rangle \mapsto (2, 6)$ in week $w_1 = 6$ (respectively $m' = \langle 5, 10\rangle$ and $m'_2 = \langle 2, 8\rangle$ in week $w_2 = 9$) will add a conflict for team 5 (resp. 2) in period 5 and remove a conflict for team 10 in the last period. In the two cases, the resulting node will have the same number of conflicts as the one of its parent's node. Note also that the conflicts are now distributed (in the last period $P = 5$) between the last team $T = 10$ and another one (2 or 5 in our sample).

Finally, when T is such that $(T - 1) \bmod 3 = 0$, empirical experiments suggested that $p_{\mathcal{F}} = \lceil (T - 1)/6 \rceil$, where $\lceil a \rceil$ refers to the minimal integer $b \geq a$.

5 From exponential to linear-time complexity

The previous E3S algorithm has one interesting quality: it is *exhaustive*. This means that, according to its repair rule, it will always find a solution to the SLSP for all T even, assuming $(T-1) \bmod 3 \neq 0$ (general case). But, its main drawback is that it runs in exponential time (see formula 2 in section 4.5). This is mainly due to the possible large number B_i of child nodes $s^k_{i+1} (1 \leq k \leq W - 2d_i)$, generated from a parent node $s_i (i \in \quad)$, which can lead to a failure.

5.1 L3S: a linear-time algorithm to solve the SLSP

The computational experiments of E3S have shown that solutions for some T, restricted to problem instances such that $(T-1) \bmod 3 \neq 0$, were found by developing a search tree $\mathcal{T}$ of size $P-1$ exactly. This means that, in this case, E3S directly follows a path to a solution. In other words, from any node $s_i (i \in \quad)$, E3S was able to branch to the right node $s_{i+1}^k (1 \leq k \leq W - 2d_i)$.

Furthermore, solutions found by E3S share common characteristics. They were obtained by doing primal and dual swaps in specific weeks $(w_\mathcal{P}, w_\mathcal{D})$ according to the value and the parity of $p_\mathcal{P}$. In other words, given $p_\mathcal{P}, w_\mathcal{P}$ and $w_\mathcal{D}$ are unique and can now be found in a deterministic way. This means that E3S can be simplified (assume $p_\mathcal{P}$ fixed):

- $w_\mathcal{P}$ is known means that a primal swap is always available in a week $w_\mathcal{P}$. So, steps 3, 5a and 5b can be removed from E3S. The **PrimalWeek** function is modified to return the right value for $w_\mathcal{P}$, which depends on $p_\mathcal{P}$ (and P):

  PrimalWeek$(p_\mathcal{P})$**:**
  if $p_\mathcal{P} \bmod 2 = 1$ {odd period}
  then return $(p_\mathcal{P} + 1)/2$
  return $P + p_\mathcal{P}/2$ {even period}

- $w_\mathcal{D}$ is known (dual swap available in week $w_\mathcal{D}$) and directly computed by this new version of the **DualWeek** function:

  DualWeek$(p_\mathcal{P})$**:**
  if $p_\mathcal{P} \bmod 2 = 1$ {odd period}
  then return $1 + (W - p_\mathcal{P})/2$
  return $T - p_\mathcal{P}/2$ {even period}

- Thanks to the new definition of **DualWeek**, step 5 of E3S will then be replaced with:

  $w_\mathcal{D} \leftarrow$ **DualWeek**$(p_\mathcal{P})$
  Exchange$(p_\mathcal{P}, w_\mathcal{D})$

- Finally, replace step 6 in E3S with "**return E3S**$(p_\mathcal{P}+1)$" to do primal and dual swaps within the next period.

The following iterative procedure summarizes the changes in E3S. Let us call it L3S for *Linear Sport Scheduling Solver* (L3S is the optimized version of E3S). Note that L3S naturally runs in linear time since, starting from the initial configuration described in section 4.1, it makes exactly $2 * (P-1)$ swaps.

L3S():
  for each $p_{\mathcal{P}} \in \{1, \ldots, P-1\}$
    $w_{\mathcal{P}} \leftarrow$ **PrimalWeek**$(p_{\mathcal{P}})$; $w_{\mathcal{D}} \leftarrow$ **DualWeek**$(p_{\mathcal{P}})$
    Exchange$(p_{\mathcal{P}}, w_{\mathcal{P}})$ {Primal swap}
    Exchange$(p_{\mathcal{P}}, w_{\mathcal{D}})$ {Dual swap}

Table 8 gives a geometrical interpretation of how L3S works for $T = 12$ teams. For a greater readability, we only represent here matches to be swapped. Recall that the other matches are not concerned by L3S. Their respective periods and weeks are fixed when building the starting configuration s_0 (section 4.1) and don't change anymore.

Table 8
Geometrical interpretation of L3S (12 teams)

		Weeks										
		1	2	3	4	5	6	7	8	9	10	11
Periods	1	1,2					6,7					
	2							6,9				2,10
	3		5,11			3,8						
	4								1,5		3,7	
	5			8,10	9,11							
	6	7,12	8,12	9,12	10,12	11,12	1,12	2,12	3,12		5,12	6,12

According to the parity of $p_{\mathcal{P}}$, one can observe in Table 8 that primal and dual swaps made by L3S with odd (respectively even) periods are performed into weeks $w_{\mathcal{P}}$ and $w_{\mathcal{D}}$ such that $(w_{\mathcal{P}}, w_{\mathcal{D}}) \in \{1, \ldots, P\}^2$ (resp. $\{P+1, \ldots, W\}^2$). Note also that there is no swap in exactly one week (9 in the example). This is evident since L3S only makes $2(P-1) = W - 1$ swaps (10 in our sample).

5.2 Limitations of L3S

Table 9
L3S: the resulting "failing" case (10 teams)

		Weeks								
		1	2	3	4	5	6	7	8	9
Periods	1	6,10	2,3	3,4	4,5	1,10	6,7	7,8	8,9	1,9
	2	3,9	1,4	2,5	3,6	4,7	2,10	6,9	1,7	5,10
	3	4,8	7,10	1,6	9,10	3,8	4,9	1,5	2,6	3,7
	4	5,7	6,8	7,9	1,8	2,9	1,3	3,10	4,10	4,6
	5	1,2	5,9	8,10	2,7	5,6	5,8	2,4	3,5	2,8

Recall that L3S is based on the E3S repair rule. So, since E3S cannot solve problem instances where $(T-1) \bmod 3 = 0$, L3S cannot solve such problems

either. See Table 9 for an example of the resulting configuration (not a solution) reached by L3S in that special case. The first $P-1$ periods do not violate the PERIOD constraint but the last one does: teams 2, 5 and 8 appear more than twice.

6 Computational results

In this section, we present computational results and contrast these results with those obtained by the two best known approaches: a constraint programming algorithm [19] (call it CP hereafter) and a tabu search algorithm [13] (called TS-SLSP). The tests were carried out on a Sun Sparc Ultra 1 (256 RAM, 143 MHz). E3S and L3S are implemented in C (CC compiler with -O5 option). Results are reported on problem instances having 6 to 50 teams for E3S and 6 to 4 464 teams for L3S (only those greater than 24 are shown).

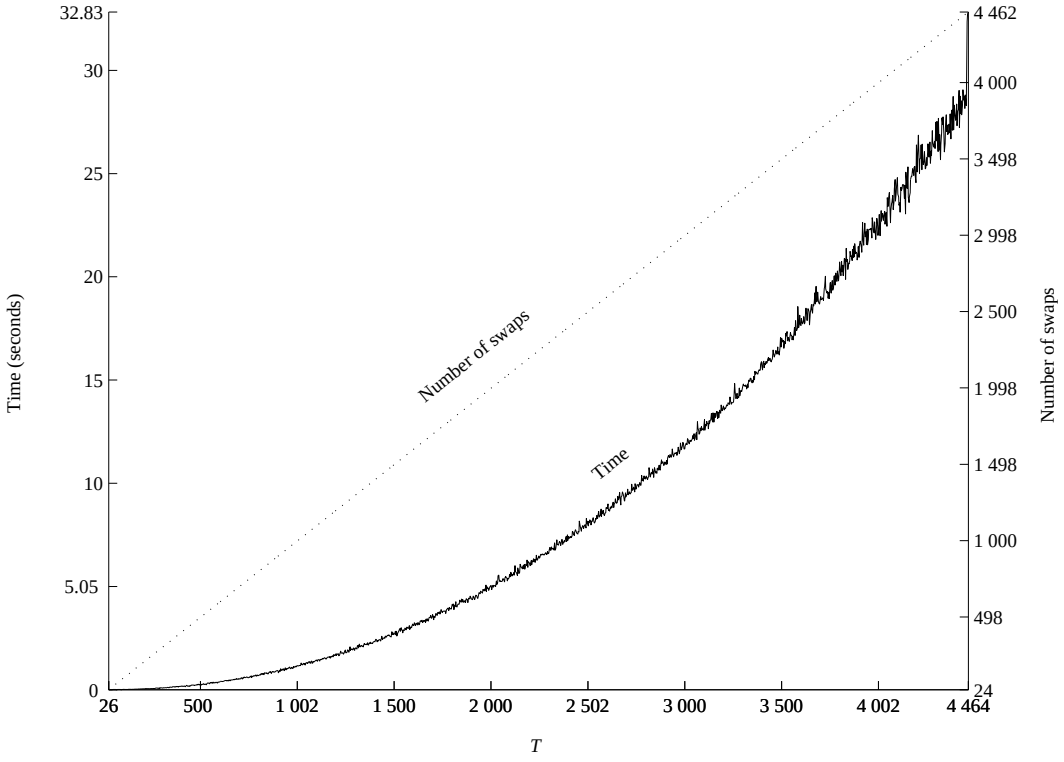

Fig. 3. L3S finds valid schedules for several thousands of matches in less than 60 seconds ($[T-1] \bmod 3 \neq 0$)

Given its linear-time complexity, L3S can be used to solve efficiently any SLSP instance with a number T of teams verifying $(T-1) \bmod 3 \neq 0$. Fig. 3 summarizes the computing times (in seconds) and numbers of swaps of L3S to

find a valid schedule for up to 4 464 teams. One observes that L3S needs less than 35 seconds to schedule 4 464 teams.

In order to get an idea about the performance of the exponential-time E3S algorithm, we show in Table 10 the results of E3S together with those of CP [19] and TS-SLSP [13]. Column 5 recalls the mean number of moves performed by TS-SLSP to find a solution. Columns 3 and 7 give numbers of backtracks for CP and E3S. Information about computing time (in minutes) is only for indicative purpose because results of CP were obtained on a different machine. A "-" sign means no result is available.

Table 10
Comparative results of E3S and best-known methods

T	CP [19]		TS-SLSP [13]		E3S	
	Time	Backtracks	Time	Moves	Time	Backtracks
26	0.4	6 683	10.7	2 219 711	< 0.01	3 333
28	5.3	32 324	12.5	2 043 353		-
30	2.3	11 895	22	3 034 073	0.4	2 202 643
32		-	49	6 387 470	< 0.1	325 929
34		-	25	2 917 279		-
36		-	91	9 307 524	2.3	10 540 201
38		-		-	12	53 224 412
40	360	2 834 754	1	68 746		-
42		-		-	40	138 880 823
44		-		-	141	471 475 040
46		-		-		-
48		-		-	78.6 hours	430 867 072
50		-		-	204.3 hours	-

From Table 10, one observes that even E3S largely outperforms CP and TS-SLSP, in terms of solution quality and speed. Indeed, E3S provides solutions for problems having up to 50 teams, while CP and TS-SLSP are limited to $T \leq 40$. Furthermore, the computing times of E3S are much shorter. Finally, remark that, for some T (28, 34, 40 and 46 teams), no results are reported for E3S since $(T-1) \bmod 3 = 0$.

7 Discussion

Section 3 described a formulation of the SLSP as a constraint satisfaction problem. We give here an alternative model expressed in terms of edge colorings [8] (also called "factorization" [15]).

Let K_T be a complete undirected graph with T vertices (teams). Solving the SLSP is equivalent to find a particular coloring of the edges of K_T. In such an approach, one has to assign two colors w and p to each edge, where w ($1 \leq w \leq W$) define a classical edge coloring (see section 4.1.1). The second color (p, $1 \leq p \leq P$) must satisfy the following condition: edges within the same color class w must receive different color p. Furthermore, at each node t ($1 \leq t \leq T$) the largest number of occurrences of a color p on edges incident to t (this number is $OccP(p,t)$) must be as small as possible.

More precisely, $OccP(p,t)$ must be lower than 3 due to the PERIOD constraint. Obviously, this last requirement can even be more precise since solutions to the SLSP satisfy another property (here again, the proof is not given since it is almost evident).

Property 6 *In a solution* s_*, $\forall\ 1 \leq p \leq P$, $\exists\ 1 \leq t' < t'' \leq T$ *such that:*

$$\begin{aligned} OccP_{s_*}(p,t') = OccP_{s_*}(p,t'') = 1 \\ and\ \forall\ 1 \leq t \leq T, t \notin \{t', t''\}, OccP_{s_*}(p,t) = 2 \end{aligned} \tag{3}$$

Furthermore, $\forall\ 1 \leq p_1 < p_2 \leq P$ *and* $\{t'_1, t''_1\}$ *(respectively* $\{t'_2, t''_2\}$*) satisfying equation 3 within period* p_1 *(resp.* p_2*),* $\{t'_1, t''_1\} \cap \{t'_2, t''_2\} = \emptyset$.

In terms of colorings, property 6 means that, at each node t in K_T ($1 \leq t \leq T$), the number of occurrences of a color p on edges incident to t must exactly be 2 except for one color which appears exactly once.

8 Conclusion

In this paper, we presented E3S, an exhaustive repair algorithm and its simplified linear-time algorithm L3S for a special case of the Sports League Scheduling Problem. These algorithms are applicable when the number T of teams is such that $(T-1) \bmod 3 \neq 0$.

Both E3S and L3S start with a conflicting schedule built using results from graph theory. Then, they iteratively remove the conflicts by doing particular

swaps of matches. The way matches are swapped uses extensively some properties of the initial conflicting schedule. E3S relies on backtracking during its search while L3S is totally backtrack-free.

Even though L3S, like E3S, can only solve problem instances when the number T of teams verifies the condition $(T-1) \bmod 3 \neq 0$, L3S is the first linear-time algorithm for this special case of the SLSP. So, L3S can find a valid schedule for several thousands of teams ($[T-1] \bmod 3 \neq 0$) in less than one minute while other state-of-the-art algorithms based on constraint programming and tabu search are limited to 40 teams. Let us mention also that even the exponential-time algorithm E3S competes favorably with these state-of-the-art algorithms.

Finally, it is still an open question whether the SLSP can be solved efficiently when the number T of teams verifies $(T-1) \bmod 3 = 0$.

Acknowledgements

We would like to thank gratefully the reviewer of the paper for having suggested to us the formulation in terms of colorings and complementary references. This work was partially supported by two grants from the Sino-French Joint Laboratory in Computer Science, Control and Applied Mathematics (LIAMA) and the Sino-French Advanced Research Programme (PRA).

References

[1] J.C. Bean and J.R. Birge, Reducing traveling costs and player fatigue in the National Basketball Association, *Interfaces* **10(3)** (1980) 98–102.

[2] R. Béjar and F. Manyà, Solving the round robin problem using propositional logic, in: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00, Austin, Texas, USA)* (AAAI Press / The MIT Press, 2000) 262–266.

[3] R.T. Campbell and D.S. Chen, A minimum distance basket ball scheduling problem, in R.E. Machol and S.P. Ladany (editors) *Management Science in Sports* (North Holland, New York, USA, 1976) 15–25.

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

[5] CSPLib: a problem library for constraints, maintained by I.P. Gent, T. Walsh and B. Selman, http://www-users.cs.york.ac.uk/~tw/csplib/index.html.

[6] D. de Werra, Geography, games and graphs, *Discrete Applied Mathematics* **2** (1980) 327–337.

[7] D. de Werra, Scheduling in sports, in P. Hansen (editor) *Studies on graphs and discrete programming* (North Holland, Amsterdam, 1981) 381–395.

[8] D. de Werra, On the multiplication of divisions; the use of graphs for sports scheduling, *Networks* **15** (1985) 128–138.

[9] J.A. Ferland and C. Fleurent, Allocating games for the NHL using integer programming, *Operations Research* **41(4)** (1993) 649–654.

[10] F. Glover and M. Laguna, *Tabu Search* (Kluwer Academic Publishers, 1997).

[11] C.P. Gomes, B. Selman and H.A. Kautz, Boosting combinatorial search through randomization, in: *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98, Madison, WI, USA)* (AAAI Press / The MIT Press, 1998) 431–437.

[12] J.P. Hamiez and J.K. Hao, Tabu search and sports league scheduling (in French), in: *Proceedings of the Twelfth French Congress on Pattern Recognition and Artificial Intelligence (RFIA'00, Paris, France)* Vol. 1 (2000) 357–366.

[13] J.P. Hamiez and J.K. Hao, Solving the sports league scheduling problem with tabu search, *Lecture Notes in Artificial Intelligence* **2148** (Springer, 2001) 24–37.

[14] M. Henz, Scheduling a major college basketball conference - Revisited, *Operations Research* **49(1)** (2001) 163–168.

[15] A. Kotzig, Decomposition of complete graphs into regular bichromatic factors, *Discrete Mathematics* **2** (1972) 383-387.

[16] K. McAloon, C. Tretkoff and G. Wetzel, Sports league scheduling, in: *Proceedings of the Third ILOG Optimization Suite International Users' Conference* (Paris, France, 1997).

[17] G.L. Nemhauser and M.A. Trick, Scheduling a major college basketball conference, *Operations Research* **46(1)** (1998) 1–8.

[18] J.C. Régin, Modeling and solving sports league scheduling with constraint programming, *INFORMS* (Montréal, Québec, 1998).

[19] J.C. Régin, Sports scheduling and constraint programming, *INFORMS* (Cincinnati, Ohio, USA, 1999).

[20] A. Schaerf, Scheduling sport tournaments using constraint logic programming, *Constraints* **4(1)** (1999) 43–65.

[21] J.A.M. Schreuder, Constructing timetables for sport competitions, *Mathematical Programming Study* **13** (1980) 58–67.

[22] B.J. Terril and R.J. Willis, Scheduling the Australian state cricket season using simulated annealing, *Journal of the Operational Research Society* **45(3)** (1994) 276–280.

[23] E.P.K. Tsang, *Foundations of constraint satisfaction* (Academic Press, London and San Diego, 1993).

[24] P. Van Hentenryck, L. Michel, L. Perron and J.C. Régin, Constraint programming in OPL, *Lecture Notes in Computer Science* **1702** (Springer, 1999) 98–116.

[25] G. Wetzel and F. Zabatta, Technical Report, CUNY Graduate Center CS (1998).

[26] M. Wright, Timetabling county cricket fixtures using a form of tabu search, *Journal of the Operational Research Society* **45(7)** (1994) 758–770.