Using solution properties within an enumerative search to solve a sports league scheduling problem

J.-P. Hamiez *, J.-K. Hao

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

Abstract

This paper presents an enumerative approach for a particular sports league scheduling problem known as "Prob026" in CSPLib. Despite its exponential-time complexity, this simple method can solve **all** instances involving a number T of teams up to 50 in a reasonable amount of time while the best known tabu search and constraint programming algorithms are limited to $T \leq 40$ and the direct construction methods available only solve instances where $(T-1) \bmod 3 \neq 0$ or $T/2$ is odd. Furthermore, solutions were also found for some T values up to 70. The proposed approach relies on discovering, by observation, interesting properties from solutions of small problem instances and then using these properties in the final algorithm to constraint the search process.

Key words: Sports league scheduling, Prob026 from CSPLib, enumerative search

1 Introduction

Many sports leagues deal with scheduling problems for tournaments. These problems contain in general many constraints to satisfy and different objectives to optimize like minimization of traveling distance or minimum number of days between a home match and its corresponding away match e.g. Sports league scheduling is therefore a very general and difficult search problem.

* Corresponding author. Tel.: (+33) 241 73 53 85. Fax: (+33) 241 73 50 73
Email addresses: Jean-Philippe.Hamiez@univ-angers.fr (J.-P. Hamiez), Jin-Kao.Hao@univ-angers.fr (J.-K. Hao).
URL: info.univ-angers.fr/pub/hao (J.-K. Hao).

Many solution approaches have been proposed to solve these problems with varying degrees of success: Integer linear programming [21], constraint programming [16], local search [5,25,28]. Sports scheduling was also investigated in terms of edge colorings of graphs [6, and references therein].

This paper deals with a specific sports league scheduling problem, namely "Prob026" from CSPLib [10]. It seems to be first introduced in [9]:

- There are $T = 2n$ teams (i.e. T even). The season lasts $W = T - 1$ weeks. Weeks are partitioned into $P = T/2$ slots called "periods" or "stadiums". Each week, one match is scheduled in every period;
- $c_{\mathcal{H}}$ constraint: All teams play each other exactly once ($\mathcal{H}$alf competition);
- $c_{\mathcal{W}}$ constraint: Every team plays exactly one game in every $\mathcal{W}$eek of the season, i.e. all teams are different in a week;
- $c_{\mathcal{P}}$ constraint: No team plays more than twice in a $\mathcal{P}$eriod. This constraint may be motivated by the equal distribution of stadiums to teams.

The problem then is to schedule a tournament with respect to these definitions and constraints. Table 1 shows an example of a valid schedule for $T = 6$.

Table 1
A valid schedule for 6 teams

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

Note that solutions exist for all $T \neq 4$ [24] (see also [17] for a simpler proof). Furthermore, direct construction methods have already been proposed when $(T-1) \bmod 3 \neq 0$ [14,15] or $T/2$ is odd [1,17,24]. While a construction for the case where $T/2$ is even is given in [1], this method uses pairs of orthogonal Latin squares of even order for which no systematic construction is known even though their existence has been shown. This leaves open the cases where $T \bmod 12 = 4$.

Prob026 is also known as the "balanced tournament design" problem (BTD) in combinatorial design theory, see [4, pages 238-241] for a brief survey. Here is its seminal definition: A BTD of order n, defined on a $2n$-set $\mathcal{T}$ (teams) is an arrangement of the $n(2n-1)$ distinct ordered pairs (matches) of the elements of $\mathcal{T}$ into an $n \times (2n-1)$ array such that:

(1) Every element of $\mathcal{T}$ is contained in precisely one cell of each column (week);
(2) Every element of $\mathcal{T}$ is contained in at most two cells in any row (period).

In this paper, we present EnASS, an Enumerative Algorithm for Sports Scheduling applied to Prob026. Given T, EnASS starts building a particular conflicting schedule (called $\overline{s}$) verifying a set $\mathcal{R}$ of properties (or "$\mathcal{R}$equirements"). The set S of solutions is generated using $\overline{s}$ in a simple exhaustive way with chronological backtracks and observed to identify new properties. $\mathcal{R}$ is then updated to solve Prob026 for larger T or to accelerate the resolution. Despite the exponential-time complexity of EnASS, we manage to build particular $\mathcal{R}$ sets that enable EnASS to find solutions to Prob026 for **all** T up to 50 in a reasonable amount of time going beyond the state-of-the-art approaches limited to $T \leq 40$ [13,23], $(T-1) \bmod 3 \neq 0$ [14,15] or $T/2$ odd [1,17,24]. Furthermore, solutions were also found for some T values up to 70.

The paper begins with a survey of related work followed by a first basic formulation as a Constraint Satisfaction Problem (CSP). The complexity of Prob026 and its symmetries are discussed in Section 4. Section 5 shows some ways to reduce the search space size and introduces a tighter CSP model. We present then the EnASS algorithm and preliminary computational results in Section 6. Section 7 describes how the $T = 70$ instance was solved. Concluding remarks are given in the last section.

2 Related work

With integer programming, McAloon et al. [19] solved the $T = 12$ case. 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).

Gomes et al. [11] obtained better results using constraint programming. They solved problems involving up to 18 teams in approximately 22 hours with a randomized version of a deterministic complete search. See also [12].

Béjar and Manyà [3] transformed Prob026 into SAT and used a SAT solver. They obtained solutions for 18 teams in less than 2 hours. They also solved Prob026 with 20 teams in about 13 hours. See also [2].

According to [11], results were also obtained for 26 and 28 teams using multiple threads on a 14 processor Sun system.

Régin [22,23] proposed two approaches with constraint programming. The first one, using powerful filtering algorithms, produced better results than those from Béjar and Manyà [3] since he solved the $T = 24$ case in 12 hours. In the second approach, Prob026 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[1]. See also [20,27] for approaches similar to those from [23].

Hamiez and Hao [13] developed first a tabu search algorithm including a search space reduction technique and a restricted neighborhood. The approach produced results which compared well with those from Régin [23] (the best known results at that moment). Solutions were found for all T up to 40 except 38.

Hamiez and Hao [14] presented then a repair-based algorithm that solves Prob026 in linear-time when T is such that $(T-1) \bmod 3 \neq 0$. Starting with $\overline{s}$, the algorithm removes the conflicts by exchanging matches. It finds valid schedules for several thousands of teams in less than a minute. Similar direct construction approaches for the BTD are given in [1,15,17,24] for particular T values.

Finally, let us mention the recent work from de Werra et al. [7] who consider a variant of Prob026: An additional dummy week is introduced and each team must appear exactly twice in every period. An inductive construction has been proposed for $T = 2^n$ (integer $n > 3$).

3 A first basic formulation of Prob026

Prob026 can be conveniently formulated as a Constraint Satisfaction Problem (CSP) [26]. Alternative models include linear programming [12,19], SAT [3] or edge colorings of graphs [14].

3.1 Constraint Satisfaction Problem

A CSP [26] 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 for the x_m variable;
- 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

[1] Surprisingly, the $T \in \{32, 34, 36, 38\}$ instances were not solved.

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 M domains, solving a CSP means to determine a particular assignment (or to prove that none exists) 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 SAT, as well as many practical applications related to resource assignment, planning or timetabling.

3.2 A first basic CSP formulation of Prob026

There are two ways to formulate the problem:

(1) Assign matches to slots. More precisely, assign each different (t, t') couple of teams ($1 \leq t < t' \leq T$) to a (p, w) couple of period and week ($1 \leq p \leq P$, $1 \leq w \leq W$). This corresponds to the primal form used in [19] with linear programming;
(2) Assign slots to matches. More precisely, assign each different (p, w) couple to a (t, t') couple. This is the dual form;

A CSP formulation in the dual form can be found in [14]. We describe here a basic CSP model in primal form. A tighter model will be presented in Section 5.3.

Let $x = \langle p, w \rangle$ be any assignment of a match in period p and week w. Values of this variable type are of (t, t') pattern, meaning that team t meets team t' in period p and week w, noted $x \mapsto (t, t')$. The set X of variables (which are slots) is $X = \{x = \langle p, w \rangle, 1 \leq p \leq P, 1 \leq w \leq W\}$ and all domains d_x are equal to $D = \{(t, t'), 1 \leq t < t' \leq T\} : \forall x \in X, d_x = D$. The constraints set $C = \{c_{\mathcal{W}}, c_{\mathcal{P}}, c_{\mathcal{H}}\}$ contains:

- $c_{\mathcal{W}}$ constraint: Uniqueness of all teams in each week. For each team t and each week w, we impose the constraint: $c_{\mathcal{W}}(t, w) \Leftrightarrow |\{x = \langle p, w \rangle \mapsto (t, t'), 1 \leq p \leq P\}| = 1$;
- $c_{\mathcal{P}}$ constraint: No more than two matches for each team in each period. For each team t and each period p, we impose the constraint: $c_{\mathcal{P}}(t, p) \Leftrightarrow |\{x = \langle p, w \rangle \mapsto (t, t'), 1 \leq w \leq W\}| \leq 2$;
- $c_{\mathcal{H}}$ constraint: All matches are different. For each tuple (p, p', w, w'), with $p \neq p'$ or $w \neq w'$, we impose the constraint: $c_{\mathcal{H}}(p, p', w, w') \Leftrightarrow \langle p, w \rangle \neq \langle p', w' \rangle$.

4 Complexity and symmetries

As shown in Table 1 (see Section 1), a solution is a complete assignment of $D = \{(t, t'), 1 \leq t < t' \leq T\}$ items to variables of $X = \{x = \langle p, w \rangle, 1 \leq p \leq P, 1 \leq w \leq W\}$ verifying the constraint set $C = \{c_{\mathcal{W}}, c_{\mathcal{P}}, c_{\mathcal{H}}\}$. Thus, a solution is a $W * P$ sized table, whose items are integer couples (t, t'). For $T = 70$ teams, this represents a problem with 2 415 variables and 2 415 values per variable.

There are $T * (T-1)/2$ matches to be scheduled. A valid schedule can be thought of as a particular 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 $T/2$.

Prob026 does have symmetries [8]. Equivalent solutions can be obtained from another one just by renumbering some or all teams, see Fig. 1 where teams 1 and 2 are exchanged (changes appear in bold face). Weeks and periods can also be permuted, see Fig. 2 where matches in the last two weeks are exchanged and Fig. 3 where periods 1 and 3 are simply swapped. Finally, symmetries can be combined: See Fig. 4 which includes the three previous symmetries (i.e. periods and weeks permutations with teams renumbering).

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

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

Fig. 1. Two equivalent tournaments up to a renumbering of the teams ($T = 8$)

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

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

Fig. 2. Two equivalent tournaments up to a permutation of weeks ($T = 8$)

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

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

Fig. 3. Two equivalent tournaments up to a permutation of periods ($T = 8$)

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

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

Fig. 4. Two equivalent tournaments up to periods and weeks permutations with teams renumbering ($T = 8$)

5 Reducing the search space size

Complete search procedures usually start with an empty assignment s_0. Then, they iteratively choose a free variable $x \in X$ in s_k (integer $k \geq 0$) and a value $v \in d_x$ for this variable which does not violate C. 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} which is locally consistent. If no value v remains for a free variable x, the process returns (or "backtracks") to a previous valid assignment and tries other values. A solution 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 enumerative approach (let us call it `EnASS`, for "`En`umerative `A`lgorithm for `S`ports `S`cheduling") is different since it starts with a complete $\overline{s}$ conflicting assignment. $\overline{s}$ is built in order to satisfy the $c_{\mathcal{W}}$ and $c_{\mathcal{H}}$ constraints: At this stage, the remaining $c_{\mathcal{P}}$ constraint is not verified in $\overline{s}$.

We detail hereafter an interesting property of all Prob026 solutions, the way we build $\overline{s}$ and, finally, a simplified CSP model based on the $\overline{s}$ properties that avoids some symmetries identified in the previous section.

5.1 An interesting property of all Prob026 solutions

Prob026 solutions verify the following property: In each period, two teams (a 2-set $\mathcal{D}$), called "$\mathcal{D}$eficient" [24], appear exactly once and the other teams are present exactly twice. Furthermore, if one considers two different periods p and p', then the deficient teams of period p appear twice in period p'. More formally, if $c_{\mathcal{D}}$ refers to this implicit constraint and $\mathcal{D}_p$ denotes the set of deficient teams in period p then: $c_{\mathcal{D}}(p) \Leftrightarrow \forall p' \neq p, \mathcal{D}_p \cap \mathcal{D}_{p'} = \emptyset$, see Table 2.

Table 2
A valid tournament with deficient teams ($T = 8$)

Periods	Weeks							$\mathcal{D}$
	1	2	3	4	5	6	7	
1	1,2	5,7	1,6	4,5	4,7	**3,8**	2,6	3,8
2	3,**7**	1,4	2,5	3,**6**	2,8	1,5	4,8	6,7
3	4,6	6,8	7,8	2,7	**1**,3	2,4	3,**5**	1,5
4	5,8	**2**,3	3,**4**	1,8	5,6	6,7	1,7	2,4

5.2 Starting conflicting tournament

Patterned one-factorization [4, page 662] can be used to verify $c_{\mathcal{H}}$ and $c_{\mathcal{W}}$, the goal of `EnASS` being then to satisfy the last constraint $c_{\mathcal{P}}$: Form a regular polygon with the first $T-1$ teams. Draw W sets of $P-1$ parallels connecting vertices in pairs starting with each w side. Each set, augmented with the pair of missing teams, corresponds to the matches to place in week w [18].

Let $\overline{s}$ be the tournament obtained (in linear-time complexity) with this technique, where $\overline{s}\langle p, w\rangle$ is the match scheduled in period p and week w in $\overline{s}$. See [14] for a full detailed description and the formal model used to build $\overline{s}$.

Table 3
Initial conflicting $\overline{s}$ schedule for 8 teams

Periods	Weeks						
	1	2	3	4	5	6	7
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**

Table 3 shows the starting schedule for $T = 8$. Observe that $c_{\mathcal{H}}$ and $c_{\mathcal{W}}$ are satisfied, $c_{\mathcal{P}}$ being violated in period 4 (team 8 appears more than twice).

5.3 A tighter CSP model for Prob026

The previous construction scheme leads to a conflicting starting tournament $\overline{s}$ which embodies an interesting property: $c_{\mathcal{H}}$ and $c_{\mathcal{W}}$ are satisfied. Proofs are not given here since they result from the construction step. The goal of `EnASS` is then to satisfy $c_{\mathcal{P}}$. $\overline{s}$ is really important since it is used to:

- Avoid some symmetries, see Section 4;
- Simplify the basic CSP model presented in Section 3;
- Reduce the size of the search space associated with the basic CSP formulation, see Section 4. This results from the two previous items.

Let $\overline{s}\langle p, w\rangle$ be the match scheduled in period p and week w in $\overline{s}$. An easy way to avoid the symmetries due to periods swaps is to force the first week to be the same as in $\overline{s}$: $\forall x = \langle p, 1\rangle \in X, d_x = \overline{s}\langle p, 1\rangle$. Symmetries due to weeks swaps and teams renumberings can also be eliminated in a similar easy way by restricting the domain of each $x = \langle p, w\rangle$ variable ($w > 1$) to the set of matches in $\overline{s}$ appearing in week w: $\forall x = \langle p, w\rangle \in X(w > 1), d_x = \{\overline{s}\langle \overline{p}, w\rangle, 1 \leq \overline{p} \leq P\}$ [2]. These domains reductions allow us to simplify the C constraints set since $c_{\mathcal{W}}$ and $c_{\mathcal{H}}$ are now always satisfied: $C = \{c_{\mathcal{P}}\}$.

6 `EnASS`: Overall procedure with preliminary computational results

The fundamentals of `EnASS` result from observations of Prob026 solutions: some share particular properties. Considering these properties as additional requirements (constraints) may help `EnASS` solving Prob026 faster or tackling larger instances as it will be shown later on. So, we first describe here the basic `EnASS` procedure, variants involving additional requirements are detailed in Section 7.

6.1 `EnASS`: Overall procedure

There are two basic ways to build a tournament using the previous table representation: Columns by columns or rows by rows. We choose the second

[2] `EnASS` does not handle some *combinations* of symmetries, e.g. teams renumberings plus weeks swaps.

option due to the evident $c_{\mathcal{D}}$ property of all Prob026 solutions previously described in Section 5.1. Indeed, when `EnASS` has entirely filled a period p and no valid $\mathcal{D}_p$ can be identified with respect to previous $\mathcal{D}_{p'}$ ($p' < p$), backtracks can be done.

Without loss of generality, let w_f and w_l be the first (respectively last) week that `EnASS` considers when filling any period, with $1 < w_f < w_l \leq W$. Note that $w_f > 1$ since the first week is invariant with respect to $\overline{s}$ (see Section 5.3). So, for the moment, assume that $w_f = 2, w_l = W$ and $\mathcal{R} = \mathcal{R}_0 = \{c_{\mathcal{P}}, c_{\mathcal{D}}\}$. The values of the global variables w_f and w_l together with the set $\mathcal{R}$ of requirements will be modified later on according to properties of some solutions found by `EnASS`.

`EnASS` is described here in a recursive form for simplicity reasons. It admits three integer parameters: p and w specify which $\langle p, w\rangle$ variable is considered, $\overline{p}$ specifies the value assignment tried (see step 5). The function returns TRUE if a solution has been found or FALSE otherwise. Backtracks are sometimes performed in the latter case. `EnASS` is called first, after the $\overline{s}$ initialization (see Section 5.2), with $p = 1, w = w_f$ and $\overline{p} = 1$ meaning that it tries to fill the slot in the first period of week w_f with the $\overline{s}\langle 1, w_f\rangle$ match. Note that we only give here the pseudo-code of `EnASS` for finding a first solution since it can easily be modified to return numerous solutions.

`EnASS`$(p, w, \overline{p})$:

(1) If $p = P + 1$ then return TRUE: A solution is obtained since all periods are filled and valid according to $\mathcal{R}$;
(2) If $w = w_l + 1$ then return `EnASS`$(p + 1, w_f, 1)$: Period p is filled and valid according to $\mathcal{R}$, try to fill next period;
(3) If $\overline{p} = P + 1$ then return FALSE: Backtrack since no match from week w in $\overline{s}$ can be scheduled in period p of week w without violating $\mathcal{R}$;
(4) If $\exists 1 \leq p' < p / \langle p', w\rangle = \overline{s}\langle\overline{p}, w\rangle$ then return `EnASS`$(p, w, \overline{p} + 1)$: The $\overline{s}\langle\overline{p}, w\rangle$ match is already scheduled, try next match;
(5) $\langle p, w\rangle \leftarrow \overline{s}\langle\overline{p}, w\rangle$: Schedule the $\overline{s}\langle\overline{p}, w\rangle$ match in period p of week w;
(6) If $\mathcal{R}$ is locally verified and `EnASS`$(p, w+1, 1)$ = TRUE then return TRUE: The previous assignment and next calls lead to a solution;
(7) Undo step 5 and return `EnASS`$(p, w, \overline{p} + 1)$: $\mathcal{R}$ is locally violated or next calls lead to a failure, backtrack and try next value.

Notice that the check for integrity of $\mathcal{R}$ (step 6) slightly differs according to the value of w. If $w \leq w_l$, $c_{\mathcal{P}}$ must be verified. Furthermore, when $w = w_l$, a valid $\mathcal{D}_p$ set (see Section 5) must be identified.

We will refer to this basic `EnASS` function with $\texttt{EnASS}_0$ since $\texttt{EnASS}_{i>0}$ variants will be considered later on according to updates of w_f, w_l and $\mathcal{R}$.

6.2 *EnASS: Preliminary computational results*

All $\texttt{EnASS}_{i\geq 0}$ were coded in `C` (`cc` BSD compiler). All computational results were obtained running $\texttt{EnASS}_{i\geq 0}$ on an Intel PIV processor (2 Ghz) Linux station with 2 Go RAM. A time limit of 3 hours was imposed, mainly because the solution found for $T = 70$ required a bit less than this duration.

Table 4 gives results obtained with $\texttt{EnASS}_0$ for $T \geq 6$ (the $T \in \{2, 4\}$ cases are trivial) [3]: Number T of teams, number $|S_0|$ of solutions, time (in seconds for all results tables) and total number $|\mathrm{BT}|$ of backtracks to generate S_0, time and number of backtracks to reach a first solution. "-" marks mean that either $\texttt{EnASS}_0$ found no solution within the time limit (see the $T = 24$ case) or $|\mathrm{BT}|$ is larger than the maximal integer value authorized by the compiler/system (i.e. 4 294 967 295) or $\texttt{EnASS}_0$ was halted since it reached the time limit. $|S_0|$ entries like "$\geq n$" indicate that $\texttt{EnASS}_0$ found n solutions when reaching the time limit. In this case, $|\mathrm{BT}|$ is the total number of backtracks performed to generate all the n solutions.

Table 4
$\texttt{EnASS}_0$ solves Prob026 for T up to 32 in less than 3 minutes ($T \neq 24$)

T	All solutions			First solution	
	$\lvert S_0\rvert$	Time	$\lvert\mathrm{BT}\rvert$	Time	$\lvert\mathrm{BT}\rvert$
6	5	< 1	110	< 1	3
8	112	< 1	11 996	< 1	129
10	63 504	3.83	25 247 598	< 1	167
12	$\geq$ 1 332 823	-	2 174 095 859	< 1	3 585
14	$\geq$ 957 035	-	-	< 1	1 036
16	$\geq$ 2 215 884	-	-	< 1	11 050
18	$\geq$ 381 753	-	-	29.04	137 847 769
20	$\geq$ 378 646	-	-	< 1	33 509
22	$\geq$ 138 256	-	-	< 1	493 952
24	-	-	-	-	-
26	$\geq$ 41 921	-	-	5.97	20 209 362
28	$\geq$ 19 056	-	-	4.21	13 441 627
30	$\geq$ 4 222	-	-	24.66	73 226 314
32	$\geq$ 5 859	-	-	137.22	400 878 192

[3] $\texttt{EnASS}_0$ found no solution within the allowed time limit for 34 and 36 teams.

$\texttt{EnASS}_0$ is extremely fast to find a first solution for $6 \leq T \leq 32$ except for the $T = 24$ case where it failed to complete within the allowed time limit [4]. This suggests that some instances may be harder to solve than others. These preliminary computational results clearly outperform those reported in [2,3,11,12,19,20,22,27] and compete well with those in [13,23] [5].

7 Invariants in Prob026

$\texttt{EnASS}_0$ is clearly limited for Prob026 since it can only solve instances where $T \leq 32$ (except $T = 24$) within a reasonable amount of time. Nevertheless, $\texttt{EnASS}_0$ has one advantage: Given sufficient time, it can be used to generate many solutions. Indeed, this was our first objective. So, subsets of solutions were generated and observed to try to identify some invariants. While we managed to find more than 20 basic properties, only those used to solve the $T = 70$ case are described here.

One general way to solve larger instances or to speed up any complete algorithm is to fix more than one variable when exploring a new branch in the search tree. This is possible for Prob026 since some solutions verify the following $r_{\Rightarrow}$ property: assume that a match x has been scheduled in period p and week w, with $w_f \leq w \leq P$, then x and the match scheduled in period p and week $T - w + 1$ appear in the same period in $\overline{s}$. More formally, $\forall\, w_f \leq w \leq P, r_{\Rightarrow}(p, w) \Leftrightarrow \langle p, w\rangle = \overline{s}\langle\overline{p}, w\rangle \Rightarrow \langle p, T-w+1\rangle = \overline{s}\langle\overline{p}, T-w+1\rangle$.

This leads to $\texttt{EnASS}_1$ which comes from $\texttt{EnASS}_0$ by adding the $r_{\Rightarrow}$ requirement to $\mathcal{R}_0 : \mathcal{R}_1 = \{c_{\mathcal{P}},\, c_{\mathcal{D}}, r_{\Rightarrow}\}$. Step 5 in the EnASS description may be adapted since one additional variable has now to be fixed. Furthermore, w_l has to be set to P before running $\texttt{EnASS}_1$. Note that the EnASS variants considered in this section work now on a subset of the $\texttt{EnASS}_0$ solutions space.

Figure 5 shows an example of a solution found by $\texttt{EnASS}_1$ (the meaning of entries in bold or italic typefaces will be explained later): For instance, scheduling the $(5, 6)$ match from week 2 in period 1 forces the $(3, 6)$ match from week 5 $(5 = 6 - 2 + 1)$ to be in period 1.

[4] $\texttt{EnASS}_0$ failed to find a solution for $T = 24$ even after a long run of six days. A deep analysis showed that for $T = 24$, $\texttt{EnASS}_0$ makes a wrong assignment for a match at the beginning of the search (in week 7 of the first period). This bad choice at the top of the search tree can only be discovered after a huge number of irrelevant backtracks. This explains the failure of $\texttt{EnASS}_0$ for this particular case. Indeed, replacing this wrong assignment with a correct one allowed $\texttt{EnASS}_0$ to solve the $T = 24$ case in less than a second.

[5] Note that, without considering the implicit $c_{\mathcal{D}}$ constraint (see Section 5.1), EnASS only solved Prob026 for $T \leq 12$.

Periods	Weeks				
	1	**2**	3	4	**5**
1	1,2	**2,3**	3,4	4,5	**1,5**
2	3,5	**1,4**	2,5	1,3	**2,4**
3	4,6	**5,6**	1,6	2,6	**3,6**

Periods	Weeks				
	1	**2**	3	4	**5**
1	1,2	***5,6***	3,4	4,5	***3,6***
2	3,5	**1,4**	*1,6*	*2,6*	**2,4**
3	4,6	**2,3**	2,5	1,3	**1,5**

Fig. 5. $T = 6$: Initial schedule (left) and the solution obtained with $\texttt{EnASS}_{i>0}$ (right)

Table 5 gives results obtained with $\texttt{EnASS}_1$ (it failed to find a solution within the time limit for 52 and 54 teams). First of all, note that $\forall\, T \leq 12, |S_1| < |S_0|$. This is evident since some solutions found by $\texttt{EnASS}_0$ do not verify the $r_{\Rightarrow}$ requirement imposed in $\texttt{EnASS}_1$.

One can observe that $\texttt{EnASS}_1$ clearly improves the results from $\texttt{EnASS}_0$ for $T \geq 12$: |BT| and execution times are smaller when searching for a first solution. $\texttt{EnASS}_1$ found more solutions than $\texttt{EnASS}_0$ for $T \geq 18$ within the time limit. Furthermore, $\texttt{EnASS}_1$ solves larger instances. Indeed, it finds a first solution for **all** T up to 50. Within the time limit, $\texttt{EnASS}_1$ finds easily a solution for $T = 24$ while $\texttt{EnASS}_0$ failed in this case.

These results are thus better than those reported in [13,23]. They also compete well with [1,14,15,17,24] for $T \leq 50$: The direct construction methods proposed there cannot solve Prob026 instances where $(T-1) \bmod 3 = 0$ or $T/2$ even while $\texttt{EnASS}_1$ found solutions for **all** these special cases.

Table 5
$\texttt{EnASS}_1$ solves **all** Prob026 instances for T up to 50 in less than 2 hours

T	All solutions			First solution	
	$\lvert S_1\rvert$	Time	|BT|	Time	|BT|
6	1	< 1	13	< 1	6
8	4	< 1	155	< 1	16
10	36	< 1	6 541	< 1	715
12	17 162	< 1	3 020 195	< 1	86
14	38 031 026	2 454.82	3 073 872 699	< 1	451
16	$\geq$ 720 637	-	-	< 1	557
18	$\geq$ 51 255 702	-	-	< 1	1 099
20	$\geq$ 12 985 829	-	-	< 1	2 811
22	$\geq$ 25 617 500	-	-	< 1	11 615
24	$\geq$ 16 240 114	-	-	< 1	12 623

Table 6
$\texttt{EnASS}_1$ solves **all** Prob026 instances for T up to 50 in less than 2 hours (continued)

T	All solutions			First solution	
	$\lvert S_1\rvert$	Time	|BT|	Time	|BT|
26	$\geq$ 11 396 160	-	-	< 1	37 708
28	$\geq$ 8 727 287	-	-	< 1	35 530
30	$\geq$ 4 296 365	-	-	< 1	650 811
32	$\geq$ 3 657 013	-	-	< 1	332 306
34	$\geq$ 2 173 500	-	-	< 1	1 342 216
36	$\geq$ 1 122 145	-	-	< 1	2 160 102
38	$\geq$ 692 284	-	-	5.34	13 469 359
40	$\geq$ 523 804	-	-	6.25	16 393 039
42	$\geq$ 339 383	-	-	107.69	256 686 929
44	$\geq$ 236 614	-	-	876.91	1 944 525 360
46	$\geq$ 119 383	-	-	1 573.31	3 565 703 651
48	$\geq$ 90 009	-	-	542.79	1 231 902 706
50	$\geq$ 19 717	-	-	6 418.52	-

Despite the $\texttt{EnASS}_1$ excellent results, other invariants are needed to tackle larger instances within the time limit. The two additional properties used to solve some of the $T > 50$ cases are illustrated in Fig. 5:

(1) r_I (see entries in bold typeface): Inverse weeks w_f and W. More formally, $\forall\, w \in \{w_f, W\}, r_I(w) \Leftrightarrow \forall\, 1 \leq p \leq P, \langle p, w\rangle = \overline{s}\langle P - p + 1, w\rangle$;
(2) r_V (italic): matches (t, T) form a "V" like pattern. More formally, $\forall\, 1 \leq p < P, r_V(p) \Leftrightarrow \langle p, p+1\rangle = \overline{s}\langle P, p+1\rangle$ and $\langle p, T-p\rangle = \overline{s}\langle P, T-p\rangle$.

This leads to $\texttt{EnASS}_2$ which comes from $\texttt{EnASS}_1$ by adding the previous requirements to $\mathcal{R}_1$: $\mathcal{R}_2 = \{c_\mathcal{P}, c_\mathcal{D}, r_\Rightarrow, r_I, r_V\}$. An additional step must be added in the EnASS description (between steps 1 and 2) due to r_V. Furthermore, w_f has to be set to 3 before running $\texttt{EnASS}_2$.

Table 7 gives results obtained with $\texttt{EnASS}_2$ (it failed to find a first solution within the time limit for $72 \leq T \leq 82$). Note that, when no solution is found for some T (i.e. $|S_2| = 0$), the columns labeled "Time" and "|BT|" give, respectively, the time and numbers of backtracks required to prove $|S_2| = 0$.

No result is reported for $T > 16$ when T is a multiple of four since $\texttt{EnASS}_2$ finds no solution within the time limit in this special case. This is clearly shown for

small T values: $|S_2| = 0 \; \forall T \in \{8, 12, 16\}$. So, one may wonder if $|S_2| = 0$ for all $T > 16$ such that T is a multiple of four. This suggests that the r_I and r_V requirements may be too restrictive when used together.

Note also that $\texttt{EnASS}_2$ found a single solution within the time limit when $T/2$ is odd. Here again, this is clearly proved for small T values: $|S_2| = 1$ $\forall T \in \{6, 10, 14\}$. One may wonder if $|S_2| = 1$ for all $T > 16$ when $T/2$ is odd.

Table 7
$\texttt{EnASS}_2$ solves Prob026 for T up to 70 in less than 3 hours ($T \bmod 4 \neq 0$)

T	All solutions			First solution	
	$\|S_2\|$	Time	|BT|	Time	|BT|
6	1	< 1	2	< 1	0
8	0	< 1	5	-	-
10	1	< 1	39	< 1	11
12	0	< 1	641	-	-
14	1	< 1	34 465	< 1	3 488
16	0	14.95	84 720 639	-	-
18	≥ 1	-	13 280	< 1	13 280
22	≥ 1	-	16 300	< 1	16 300
26	≥ 1	-	5 786	< 1	5 786
30	≥ 1	-	1 031 313	< 1	1 031 313
34	≥ 1	-	130 149	< 1	130 149
38	≥ 1	-	2 829 421	< 1	2 829 421
42	≥ 1	-	7 836 823	2.11	7 836 823
46	≥ 1	-	1 323 929	< 1	1 323 929
50	≥ 1	-	47 370 701	13.75	47 370 701
54	≥ 1	-	29 767 940	10.59	29 767 940
58	≥ 1	-	827 655 311	269.88	827 655 311
62	≥ 1	-	494 071 117	279.38	494 071 117
66	≥ 1	-	1 614 038 658	7 508.51	1 614 038 658
70	≥ 1	-	-	8 985.05	-

8 Conclusion

We presented `EnASS`, an `En`umerative `A`lgorithm for `S`ports `S`cheduling, applied to a particular problem known as Prob026 in CSPLib and "balanced tournament design" in combinatorial design theory.

Based on this basic procedure, we derived two effective enumerative algorithms to constraint the search process by integrating solutions properties: the $r_{\Rightarrow}$ implied requirement, the r_I inversion property and the r_V pattern requirement.

Computational results showed that these algorithms clearly outperform the best known tabu search [13] and constraint programming [23] approaches limited to $T \leq 40$. Indeed, `EnASS` finds solutions to Prob026 in a reasonable amount of time for most T up to 70. Furthermore, even if direct construction methods exist when $(T-1) \bmod 3 \neq 0$ [14,15] or $T/2$ is odd [1,17,24], `EnASS` is the first approach solving **all** problem instances for T up to 50.

`EnASS` is a simple enumerative algorithm with chronological backtracks. One possible way to solve Prob026 for larger T or to speed up `EnASS` could be to handle the combination of symmetries or to use other elaborated techniques such as no-good learning or non-chronological backtracking. Also, the interest of forward checking from constraint programming could be investigated in particular when a team appears twice in a period.

Acknowledgments

We would like to thank greatly the referees of the paper for their very helpful suggestions and questions. Special thanks go to Professor J.H. Dinitz from the University of Vermont (Department of Mathematics and Statistics, Burlington, VT, USA) for having suggested to us the mathematical origins of Prob026 and some important previous works.

References

[1] S. Bauman, The existence of balanced tournament designs and partitioned balanced tournament designs, Master thesis of Mathematics in Combinatorics and Optimization. University of Waterloo, Ontario, Canada, available from `http://etd.uwaterloo.ca/etd/sbbauman2001.pdf` (2001).

[2] R. Béjar, F. Manyà, Solving combinatorial problems with regular local search algorithms, in: Proceedings of the Sixth International Conference on Logic

Programming and Automated Reasoning, vol. 1705 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin/Heidelberg, Germany, 1999, pp. 33–43.

[3] R. Béjar, F. Manyà, Solving the round robin problem using propositional logic, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park (CA) USA, 2000, pp. 262–266.

[4] C. Colbourn, J. Dinitz (eds.), The CRC Handbook of Combinatorial Designs, vol. 4 of Discrete Mathematics and Its Applications, CRC Press, Boca Raton (FL) USA, 1996.

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

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

[7] D. de Werra, T. Ekim, C. Raess, Construction of sports schedules with multiple venues, Discrete Applied Mathematics 154 (1) (2006) 47–58.

[8] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh, Breaking row and column symmetries in matrix models, in: Proceedings of the Eight International Conference on Principles and Practice of Constraint Programming, vol. 2470 of Lecture Notes in Computer Science, Springer-Verlag, Berlin/Heidelberg, Germany, 2002, pp. 462–476.

[9] E. Gelling, R. Odeh, On 1-factorizations of the complete graph and the relationship to round-robin schedules, Congressus Numerantium 9 (1974) 213–221.

[10] I. Gent, T. Walsh, CSPLib: A benchmark library for constraints, in: Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming, vol. 1713 of Lecture Notes in Computer Science, Springer-Verlag, Berlin/Heidelberg, Germany, 1999, pp. 480–481, see also `http://4c.ucc.ie/~tw/csplib/`.

[11] C. Gomes, B. Selman, H. Kautz, Boosting combinatorial search through randomization, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park (CA) USA, 1998, pp. 431–437.

[12] C. Gomes, B. Selman, K. McAloon, C. Tretkoff, Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems, in: Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, AAAI Press, Menlo Park (CA) USA, 1998, pp. 208–213.

[13] J.-P. Hamiez, J.-K. Hao, Solving the sports league scheduling problem with tabu search, in: Local Search for Planning and Scheduling, vol. 2148 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin/Heidelberg, Germany, 2001, pp. 24–36.

[14] J.-P. Hamiez, J.-K. Hao, A linear-time algorithm to solve the Sports League Scheduling Problem (prob026 of CSPLib), Discrete Applied Mathematics 143 (1-3) (2004) 252–265.

[15] J. Haselgrove, J. Leech, A tournament design problem, American Mathematical Monthly 84 (3) (1977) 198–201.

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

[17] E. Lamken, S. Vanstone, The existence of factored balanced tournament designs, Ars Combinatoria 19 (1985) 157–160.

[18] E. Lockwood, American tournaments, The Mathematical Gazette 20 (1936) 333.

[19] K. McAloon, C. Tretkoff, G. Wetzel, Sports league scheduling, Communication at the Third ILOG Optimization Suite International Users' Conference (Paris, France, July 1997), available from `http://www.ilog.com/products/optimization/tech/custpapers/brooklin.pdf` (1997).

[20] L. Michel, P. van Hentenryck, OPL++: A modeling layer for constraint programming libraries, Tech. Rep. CS-00-07, Brown University, Providence (RI) USA, available from `ftp://ftp.cs.brown.edu/pub/techreports/00/cs00-07.pdf` (2000).

[21] G. Nemhauser, M. Trick, Scheduling a major college basketball conference, Operations Research 46 (1) (1998) 1–8.

[22] J.-C. Régin, Modeling and solving sports league scheduling with constraint programming, Communication at INFORMS'98 (Montreal, Quebec, Canada, April 26–29) (1998).

[23] J.-C. Régin, Constraint programming and sports scheduling problems, Communication at INFORMS'99 (Cincinnati, Ohio, USA, May 2–5) (1999).

[24] P. Schellenberg, G. van Rees, S. Vanstone, The existence of balanced tournament designs, Ars Combinatoria 3 (1977) 303–318.

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

[26] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, London (England) and San Diego (California, USA), 1993.

[27] P. van Hentenryck, L. Michel, L. Perron, J.-C. Régin, Constraint programming in OPL, in: Proceedings of the International Conference on Principles and Practice of Declarative Programming, vol. 1702 of Lecture Notes in Computer Science, Springer-Verlag, Berlin/Heidelberg, Germany, 1999, pp. 98–116.

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