Minimization of the Number of Breaks in Sports Scheduling Problems using Constraint Programming.

Jean-Charles Régin
ILOG
Les Taissounières HB2
06560 Valbonne, FRANCE
e-mail : regin@ilog.fr

Abstract

This paper aims to show the interest of constraint programming for minimizing the number of breaks in sports scheduling problems. We consider single round-robin problems with an even number of teams. In such a problem we are given n teams and $n-1$ periods and for each period each team has to play either at home or away game against another team such that every team plays every other team exactly once during all the periods. A break for a team is defined to be two consecutive home matches or two consecutive away matches. For the considered problem, it has been proven by Schreuder that the minimal number of breaks is $n-2$. We propose a model using constraint programming that has the capability to efficiently prove this result. For 20 teams this takes a mere 0.61s and for 60 teams it still takes less than 1 minute. The main reason for this is the use of several global constraints with which powerful filtering algorithms are associated.

Moreover, this model is well adapted to solve some variations of the initial problem in which new constraints are added such as: for each team the number of away and home matches has to be balanced, it is forbidden to have two consecutive breaks, etc. We are also able to find and prove the minimal number of breaks for some given timetables of teams.

1 Introduction

Sport scheduling problems is an area of increasing interest as amateur and professional sports leagues proliferate and grow in size and complexity. Organizers are increasingly turning to computer assisted scheduling. The scientific literature in this area is also growing. Some systems dedicated to the resolution of sports scheduling problems are also proposed.

Constraint programming has already been shown to be a good way to attack these problems, while Integer programming methods do not perform very well, because when n increases the number of 0-1 variables and the number of constraints grow too much [MTW97].

In this paper we consider only a restriction of real world problem. We study only one part of the problem, which is the core of most of the real-life applications.

The rest of the paper is organized as follows. First we present the sports scheduling problem that we will consider. Then we introduce constraint programming and try to define what is a good model in constraint programming. In section 3, we study the minimization of the number of breaks of a round-robin with an even number of teams in which no schedule are predefined. We will show two models, a simple one and a more complex one, with which it is possible to solve this problem with an almost polynomial complexity. In section 4, we study several variations of the initial problem in which the schedule is precomputed. The problem is to determine the location of the games. We propose several new constraints that leads to an efficient resolution of this problem. Six models will be successively studied. At last, we introduce some real world constraints in order to understand their impact on our model.

1.1 Sports Scheduling Problems

We consider single round-robin problems with an even number of teams. In such a problem we are given n teams and $n-1$ periods and for each period each team has to play either at home or away game against another team such that every team plays every other team exactly once during all the periods.

This problem models half of the season, the second part is usually obtained by mirroring the first part.

A break for a team is defined to be two consecutive home matches or two consecutive away matches. The following table present a solution for 8 teams. Home games are represented by the sign +, while away game are represented by the sign −. Breaks appear in bold.

1	+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2	- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3	- 8	**+ 1**	**+ 5**	- 7	+ 2	- 4	+ 6
4	+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5	- 6	+ 8	- 3	**+ 1**	**+ 7**	- 2	+ 4
6	+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7	- 4	+ 6	- 8	+ 3	- 5	**+ 1**	**+ 2**
8	+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

In real-life application a lot of other constraints are added. We will not consider them, and some information about this subject can be found in [NT98, Sch92].

We concentrate our study to the minimization of the number of breaks, because this problem is one of the main problems to solve when we want to find a solution to a real-life application, and because no constraints can be violated in this problem.

When the teams are not already assigned this problem can be solved in polynomial time as it is explained in [Sch92]. It has been proven that the minimal number of breaks is $n-2$. Nethertheless, when the teams are already assigned the problem is not easy to solved, because the optimal bound is not true for all the timetables. In real-life applications some constraints restrict the possible timetables. Thus, the determination of the minimal number of breaks becomes an important problem.

Our study can be divided into two parts.

In the first part we propose a model that has the capability to prove that the minimal number of breaks is $n-2$ even for large n. This result has no interest in pratice, because an algorithm was already known. However, this thought process has several advantages: it proves that constraint programming, which is a technique to solve some hard problems is also an efficient way to solve easy problems and it permits to identify where are the bottlenecks of the sports scheduling problems and to model and solve hard variations of the initial problem.

In the second part we introduce some modifications of the initial problem, and show where are the difficulties and what constraint programming is able to solve. For instance, we consider the problem where the teams are already assigned but not the place where the matches are played. We also introduce some real-life constraints: the first match is different from the last match, there is no two consecutive breaks...

1.2 Constraint Programming (CP)

Constraint programming forms a simple formal frame to represent and solve certain problems. They involve finding values for problem variables subject to constraints on which combinations are acceptable. Constraint programming is based on the constraint network theory.

Constraint network A finite *constraint network* $\mathcal{N}$ is defined as a set of n *variables* $X = \{x_1, \ldots, x_n\}$, a set of current *domains* $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible *values* for variable x_i, and a set $\mathcal{C}$ of *constraints* between variables. We introduce the particular notation $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$. Indeed, we consider that any constraint network $\mathcal{N}$ can be associated to an initial domain $\mathcal{D}_0$ (containing $\mathcal{D}$), on which constraint definitions were stated.

A total ordering $<_d$ can be defined on $D(x_i), \forall x_i \in X$, without loss of generality. $max(D(x_i))$ and $min(D(x_i))$ denotes respectivly the maximum and the minimum value of the domain of the variable x_i.

Constraints. Then, a constraint C on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ that specifies the *allowed* combinations of values for the variables $x_{i_1} \times \ldots \times x_{i_r}$. An element of $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ is called a *tuple on* $X(C)$. $|X(C)|$ is the *arity* of C.

A constraint C involving the subset of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ can be described by the set of allowed tuples (resp. the set of forbidden tuples) given in extension when the constraint is tight (resp. is loose), or by an arithmetic relation. More generally, it can be represented by any boolean function defined on $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$.

Solutions. The *search space* consists of the Cartesian product of the domains of the variables of the problem.

A *solution* of a constraint network is an instantiation of the variables such that all the constraints are satisfied.

Notations. A value a for a variable x is often denoted by (x, a). $\mathbf{index}(C, x)$ is the position of variable x in $X(C)$. $\tau[k]$ denotes the k^{th} value of τ.

Consistency. Let $\mathcal{N} = (X, \mathcal{D}, \mathcal{C})$ be a constraint network, C a constraint in $\mathcal{C}$ defined on $X(C) = (x_{i_1}, \ldots, x_{i_k})$.
A tuple τ of $X(C)$ is *valid* if $\forall (x, a) \in \tau, a \in D(x)$; otherwise it is *rejected*.
C is *consistent* if there exists a tuple of $T(C)$ which is valid. A value $a \in D(x)$ is *consistent with* C iff $x \notin X(C)$, or $\exists \tau \in T(C)$, such that $a = \tau[\mathbf{index}(C, x)]$ and τ is valid.
C is *arc consistent* iff $\forall x_i \in X(C), \forall a \in D(x_i), a$ is consistent with C

Constraint programming In constraint programming, each constraint is associated with a *filtering algorithm* which aims to remove some values that are not consistent with the constraint. Sometimes a filtering algorithm achieves arc consistency, sometimes not.

We will denote by:

- $f(C)$ the filtering algorithm associated with the constraint C.
- $R(C)$ the set of values deleted from the domain of their variable when $f(C)$ is called.

In this paper, we will consider that a variation of a backtrack algorithm is used to search for a solution. In this variation each time the domain of a variable is modified, the filtering algorithm associated with each constraint involving the variable is called. This process is called *propagation*. And, the variables and the values assigned to the variables are choosen by following what we call a *variable-value ordering*. These orderings can be complex and dynamic (that is the next variable or value to try corresponds to the variable or the value which satisfies a given criteria).

Constraint programming offers a wide range of predefined constraints. These predefined constraints greatly clarify the problem description, the problem representation, and the problem solution.

Constraint programming also allows the definition of new constraints and the filtering algorithm associated with them. This is particularly usefull to represent a particular statement of the problem.

We have used ILOG Solver 4.3 on Windows NT in all of our experiements. The used computer is based on a Pentium Pro at 200Mhz.

Optimization problem Usually, in constraint programming, branch-and-bound algorithm is used to solve optimization problem. The solver we used, performs a depth-first search branch and bound algorithm.

2 What is a good model in CP?

Obviously a good model is a model that leads to an efficient resolution of a given problem. However, we can give some basic ideas involved in a good model.

A good model deals with four important notions of constraint programming:

- symmetries
- implicit constraints
- global constraints
- pertinent and redundant constraints

2.1 symmetries

The complexity of a problem can often be reduced by detecting intrinsic symmetries. Parts of the search space can then be safely ignored. When two or more variables have identical characteristics, it is pointless to differentiate them artificially. Identical characteristics can be viewed as the satisfaction of the following conditions:

- the initial domains of these variables are identical;
- these variables are subject to the same constraints;

- the variable can be permuted without changing the statement of the problem

then there is really no point in examining all the possible solutions for these variables and their values. In fact, the permutations give rise to sets of solutions that are identical as far as the physical reality of the problem is concerned. We can exploit this idea to minimize the size of the search space.

If we reduce the domains of theses variables by introducing a supplementary constraint, such as order, or by imposing a special feature on each of these variables, then we can markedly reduce the size of the search space.

For instance, suppose that you want to solve $x + y = z$ where x and y have the same initial domain. In this case, we can add the constraint $x \leq y$ and solve the new problem. From the solutions of this problem we can build all the solutions of the initial problem.

Detecting some hidden symmetries, and finding a way to avoid them is certainly one of the main problem of the modelisation.

In sports scheduling problems, we will see that it is possible to remove some non obvious symmetries in a efficient way when we want to determine the minimum number of breaks.

2.2 implicit constraints

An *implicit* constraint makes explicit a property that satisfies any solution implicitly.

Since constraint programming decreases the search space by reducing the domains of variables, it is obviously important to express all necessary constraints. In some cases, it is even a good idea to introduce implicit constraint to reduce the size of the search space by supplementary domain reductions.

The introduction of implicit constraints does not change the nature of the solution but can slow down the execution of the propagation process. However, this slowing down may be negligible in certain problems when it is compared with the efficiency gained from reducing the size of the search space obtained by the filtering algorithms.

Here is a good example of the interest of an implied constraint:
Suppose you want to solve a constraint network involving 100 variables with the same domain $\{0, 1, 2, 3, 4\}$. The constraints are: all the values but 0 must be taken at least 5 times. The filtering algorithm associated with each

of these four constraints acheives the arc consistency. In this case, this algorithm corresponds to the idea: "if a is assigned to $k < 5$ variables and if there are only $5-k$ variables that are not instantiated, then assigned a to all these variables". Suppose that the variable-value ordering consists of choosing the variable with the domain of the minimum size and the first value in the domain (the classical domain-min ordering). Then, 95 variables will be instantiated to 0 before detecting an inconsistency. Because when 94 variables are instantiated to 0 then all the filtering algorithms remove no value. Even so, it is clear that at most $100-4\times 5 = 80$ variables can be instantiated with 0. This remark is an implicit constraint. If we add this implicit constraint we will avoid $5^{14} = 6,103,515,625$ instantiations! Moreover, we can add a more sophisticated implied constraint which states that the number of uninstatiated variables must be sufficient to satisfy simultaneously the four initial constraints.

In sports scheduling we will see that the number of break is always an even number. The introduction of such an implied constraint speeds up the search for the minimal solution.

2.3 global constraints

A *global constraint* C is a constraint involving a set of other constraints, such that the filtering algorithm associated with C is more powerful than the conjunction of the filtering algorithm of each constraint involved in C taken separately.

From a mathematical point of view, if C is a global constraint involving the constraints $c_1, ..., c_p$ then $R(c_1) \cup ... \cup R(c_p) \subseteq R(C)$.

Such a global constraint is particularly interesting if the complexity of $f(C)$ is comparable to the sum of the complexity of $f(c_1), ..., f(c_p)$.

A well known example of global constraint is the alldiff constraint. This constraint states that the involved variables must take different values from each other when they are instantiated. For instance, consider three variables x_1, x_2 and x_3 such that $D_{x_1} = \{a, b\}$, $D_{x_2} = \{a, b\}$ and $D_{x_3} = \{a, b, c\}$ and the constraints $x_1 \neq x_2$, $x_1 \neq x_3$ and $x_2 \neq x_3$. If the filtering algorithm associated with each of these constraints acheives the arc consistency, then no value will be deleted. While, if we use an alldiff constraints involving x_1, x_2 and x_3 and if the filtering algorithm associated with the alldiff constraint acheives arc consistency then x_3 will be instantiated with c. Such a filtering algorithm has

been proposed by [Rég94]. Its complexity is low ($(\sum_{x_i} |D(x_i)|)$). In the rest of the paper when will use this constraint and without particular mention, we will consider that the filtering algorithm associated with it is this one.

The introduction of global constraints will not change the set of solutions and can be essential for solving some problems as noted by [CGL93]: "The problem is that efficient resolution of a timetable problem requires a global computation on the set of min/max constraints, and not the efficient implementation of each of them separately."

Sometimes, a global constraint is simply the conjunction of a set of constraints and an implied constraint deduced from the simultaneous presence of this set of constraints, and the filtering algorithm associated with this global constraint is the conjunction of all the filtering algorithms, that is no new algorithm is defined. Such a constraint can be obtained for the example given in the paragraph about implied constraints.

Often, the introduction of global constraints leads to the discovery of some new variable-value orderings that improve the search for solution, because global constraints lead to more domain reductions after each choice of variable or value.

2.4 pertinent and redundant constraints

At first glance it seems that adding a constraint which removes some symmetries, or an implicit constraint, or a global constraint always improves the current model.

This is not true. For instance, the removal of some symmetries can increase the time needed to find a solution. This can be generally explained by the fact that the adequation between the current variable-value ordering and the constraints introduced to remove the symmetries becomes poor. The filtering algorithms associated with these constraints can remove a lot of values that can change the next chosen variable. Note that with a static ordering such a behaviour cannot arise. Similarly, if we add an implicit constraint to a model, sometimes the number of backtracks and the time needed to solve the problem do not decrease, because the necessarily condition introduced by this constraint is already taken into account by the model.

So, we insist on the fact that the reduction of the size of the search space from a mathematical point of view does not necessarily lead to better performance for finding a solution in practice. This is another major problem

of the modelisation.

From these remarks, we can define a the notion of pertinent constraint. A constraint is *pertinent* w.r.t. a model if the introduction of this constraint:

- is needed by the definition of the problem;
- or if it permits to remove some symmetries, or it is an implied or a global constraint, and the introduction of the constraint improves the search for the solution in term of performance.

It is important to note that the pertinence of a constraint is defined w.r.t. a given model, because sometimes the introduction of a constraint does not improve the resolution for a given variable-value ordering, but leads to the discovery of another ordering with better performance only when the constraint is added.

When a constraint is not pertinent it is called *redundant* constraint.

3 Minimisation of the number of break

In such a problem we are given n teams and $n-1$ periods and for each period each team has to play either at home or away game against another team such that every team plays every other team exactly once during all the periods. A break for a team is defined to be two consecutive home matches or two consecutive away matches.

3.1 A first model

From the definition of the problem, some variables are defined:

• *For each period each team has to play game against another team.*

So, for a team i, $n-1$ variables are defined. O_{ij} is the variables corresponding to the opponent of the team i at the period j. Then, $O_{ij} = k$ means that team i plays versus team k at the period j (this is j^{th} match of team i). Initially, the domain of any variable O_{ij} contains all the teams except i. These variables are called *opponent variables.*

• *For each period each team has to play either at home or away game.*

For each team i and for each period j a 0-1 variable P_{ij} is defined. By convention 0 corresponds to a away game and 1 to an home game. Then

$P_{ij} = 0$ means that the team i will have an away game at the period j. These variables are called *place variables.*

• *A break for a team is defined to be two consecutive home matches or two consecutive away matches.*
For each team and for each pair of consecutive period, a 0-1 variable B_{ij} is defined. $B_{ij} = 1$ means that the team i has a break involving the games play at period j and at period $j + 1$. These variables are called *break variables.* For each team there are $n - 2$ break variables. A break due to the presence of two consecutive home games is called an *home break.* A break due to the presence of two consecutive away games is called an *away break.*

The objective variable, that is the variable to minimize, is $\#B$. This is the variable that counts the total number of breaks for the schedule.

The constraints of the problem can be represented as follows:

• *every team plays every other team exactly once during all the periods.*
For each team i, an alldiff constraint involving the O_{ij} variables for $j = 1..n-1$ is defined.

• *for the period j, if team i plays against team k, then team k plays against team i and the place where the game is played is different for i and k*
A constraint ($O_{ij} = k \Leftrightarrow O_{kj} = i$ and $P_{ij} \neq P_{kj}$) is defined for each period j and each pair of teams i and k. For a given period j these constraints are called *place-opponent contraint.*

• *A break for a team is defined to be two consecutive home matches or two consecutive away matches.*
For each team i and each period j, $j = 1..n-2$, the constraint ($B_{ij} = 1 \Leftrightarrow P_{ij} = P_{i(j+1)}$) is defined.

For counting the number of breaks, the constraint ($\#B = \sum_{i=1..n} \sum_{j=1..n-2} B_{ij}$) is introduced in the model

The filtering algorithm associated with each of the above constraints acheives the arc consistency. For the alldiff constraints, such a filtering algorithm can be found in [Rég94]. For the two other kinds of constraint, specific algorithms have to be designed.

These algorithms are not complex, and corresponds to the enumeration

and checking of all the possible cases. For instance, the filtering algorithm associated with a place-opponent constraint ($O_{ij} = k \Leftrightarrow O_{kj} = i$ and $P_{ij} \neq P_{kj}$), will be implemented as follows:

1) if $O_{ij} = k$ then $O_{kj} \leftarrow i$

2) if P_{ij} is instantiated then remove from O_{ij} all the values k such that P_{kj} is instantiated with the same value as P_{ij}.

3) if k is removed from O_{ij} then remove i from O_{kj}.

We can easily prove that this algorithm will acheive arc consistency.

A depth-first branch and bound algorithm is used to find the minimal value and to prove that it is minimal.

It is difficult to find a variable-value ordering that leads to solutions with this simple model. However, if the break variables are instantiated first, the place variable second and the opponent variables third, the following results are obtained:

#teams	4	6
#bk	73	62,082
time (s)	0.1	17.7

3.2 Introduction of pertinent constraints

The previous results can be greatly improved by removing some symmetries and changing the variable-value ordering in order to take into account efficiently the fact that some symmetries have been removed. Some new variables are needed:

for each team i, $\#Bt_i$ is a variable that counts the number of breaks for i.

Then $\#B$ is redefined by ($\#B = \sum_{i=1..n} \#Bt_i$).

The initial formulation contains several symmetries. Two kind of symmetries can be identified:

- All the home games can be replaced by away games and conversly
- The teams can be permuted

The first kind of symmetry can be easily removed by deciding that the first game of the first team will be an home game.

The second kind of symmetry can be removed by defining the schedule of the first team. That is, for instance, by choosing the opponent w.r.t an increasing order. Moreover, the first team can be any team since the teams are interchangeable. So, we decide that the first team will have the lowest number of breaks.

So, in order to break the symmetries the following constraints are added to the model:

- $P_{11} \leftarrow 1$
- $\forall j = 1..n-1 : O_{1j} \leftarrow j+1$
- $\forall i = 2..n : Bt_1 \leq Bt_i$

The variable-value ordering used is the following:

1. the $\#Bt_i$ variables are instantiated first, by dynamically choosing the variable that has the minimum value in its domain and by choosing this value.
2. the break variables are then instantiated by trying first the value 1.
3. the place variables are then instantiated
4. the opponent variables are then instantiated.

3.3 Results

Here are the results with this new model:

#teams	4	6	8	10	12	14	16	18	20	22	40	60
#bk	3	22	45	76	115	162	217	280	351	430	1,501	3,451
time (s)	0.0	0.0	0.0	0.1	0.1	0.2	0.3	0.5	0.7	1.0	10.7	58.0
$\frac{\#bk}{(\#teams)^2}$	0.19	0.61	0.70	0.76	0.80	0.83	0.85	0.86	0.88	0.89	0.94	0.97

The last row of this table indicates that the proposed model leads to an almost linear algorithm, because the square of number of teams corresponds to the size of the problem.

If arc consistency is not acheived by the filtering algorithm associated with the alldiff constraint then for 4 teams the results are identical , for 6 teams $1,713$ backtracks and 0.31s are needed and for 8 teams the problem cannot be solved in a reasonable amount of time.

4 Other problems

First, we will consider, that the schedule of the teams is given. In this problem the places where the games are played have to be determined. The goal is to minimize the number of breaks.

The initial assignements of the teams are precomputed. Here are the problems we have considered:

6 teams : optimal solution : 4 breaks

1	+ 2 - 3 + 4 - 5 + 6
2	- 1 - 4 + 5 - 6 + 3
3	- 5 + 1 - 6 + 4 - 2
4	- 6 + 2 - 1 - 3 + 5
5	+ 3 + 6 - 2 + 1 - 4
6	+ 4 - 5 + 3 + 2 - 1

8 teams : optimal solution : 8 breaks

1	+ 2 - 3 + 4 - 5 + 6 - 7 + 8
2	- 1 - 4 + 3 - 6 + 5 - 8 + 7
3	- 4 + 1 - 2 + 7 - 8 + 5 - 6
4	+ 3 + 2 - 1 + 8 - 7 + 6 - 5
5	- 6 + 7 - 8 + 1 - 2 - 3 + 4
6	+ 5 + 8 - 7 + 2 - 1 - 4 + 3
7	+ 8 - 5 + 6 - 3 + 4 + 1 - 2
8	- 7 - 6 + 5 - 4 + 3 + 2 - 1

10 teams : optimal solution : 12 breaks

1	+ 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10
2	- 1 - 4 + 3 - 6 + 5 - 8 + 7 - 10 + 9
3	- 4 + 1 - 2 + 7 - 8 + 9 - 10 + 5 - 6
4	+ 3 + 2 - 1 + 8 - 9 + 10 - 5 + 6 - 7
5	- 7 + 9 - 10 + 1 - 2 - 6 + 4 - 3 + 8
6	- 10 + 7 - 8 + 2 - 1 + 5 + 9 - 4 + 3
7	+ 5 - 6 + 9 - 3 - 10 + 1 - 2 - 8 + 4
8	- 9 + 10 + 6 - 4 + 3 + 2 - 1 + 7 - 5
9	+ 8 - 5 - 7 + 10 + 4 - 3 - 6 + 1 - 2
10	+ 6 - 8 + 5 - 9 + 7 - 4 + 3 + 2 - 1

12 teams : optimal solution : 14 breaks

1	+ 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12
2	- 1 - 4 + 3 - 6 + 5 - 8 + 7 - 10 + 9 - 12 + 11
3	- 4 + 1 - 2 - 7 + 8 - 5 + 6 - 11 + 12 - 9 + 10
4	+ 3 + 2 - 1 - 8 + 7 - 9 + 10 - 12 + 11 - 5 + 6
5	- 9 + 10 - 11 + 1 - 2 + 3 - 12 + 6 - 7 + 4 - 8
6	- 10 + 9 - 12 + 2 - 1 + 11 - 3 - 5 - 8 + 7 - 4
7	- 11 + 12 - 10 + 3 - 4 + 1 - 2 - 8 + 5 - 6 + 9
8	- 12 + 11 - 9 + 4 - 3 + 2 - 1 + 7 + 6 - 10 + 5
9	+ 5 - 6 + 8 + 10 - 12 + 4 - 11 + 1 - 2 + 3 - 7
10	+ 6 - 5 + 7 - 9 - 11 + 12 - 4 + 2 - 1 + 8 - 3
11	+ 7 - 8 + 5 - 12 + 10 - 6 + 9 + 3 - 4 + 1 - 2
12	+ 8 - 7 + 6 + 11 + 9 - 10 + 5 + 4 - 3 + 2 - 1

14 teams : optimal solution : 20 breaks

1	+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8	- 9	+ 10	- 11	+ 12	- 13	+ 14
2	- 1	- 4	+ 3	- 6	+ 5	- 8	+ 7	- 10	+ 9	- 12	+ 11	- 14	+ 13
3	- 4	+ 1	- 2	- 7	+ 8	- 5	+ 6	- 11	+ 12	- 13	+ 14	- 9	+ 10
4	+ 3	+ 2	- 1	- 8	+ 7	- 6	+ 5	- 12	+ 11	- 14	+ 13	- 10	+ 9
5	- 9	+ 10	- 11	+ 1	- 2	+ 3	- 4	+ 13	- 14	+ 6	- 7	+ 8	- 12
6	- 10	+ 11	- 13	+ 2	- 1	+ 4	- 3	+ 14	+ 8	- 5	+ 9	- 12	+ 7
7	- 12	+ 9	- 14	+ 3	- 4	+ 1	- 2	- 8	+ 13	- 10	+ 5	- 11	- 6
8	- 13	+ 14	- 12	+ 4	- 3	+ 2	- 1	+ 7	- 6	- 9	+ 10	- 5	+ 11
9	+ 5	- 7	- 10	+ 11	- 12	+ 13	- 14	+ 1	- 2	+ 8	- 6	+ 3	- 4
10	+ 6	- 5	+ 9	+ 12	- 14	+ 11	- 13	+ 2	- 1	+ 7	- 8	+ 4	- 3
11	+ 14	- 6	+ 5	- 9	+ 13	- 10	+ 12	+ 3	- 4	+ 1	- 2	+ 7	- 8
12	+ 7	- 13	+ 8	- 10	+ 9	+ 14	- 11	+ 4	- 3	+ 2	- 1	+ 6	+ 5
13	+ 8	+ 12	+ 6	+ 14	- 11	- 9	+ 10	- 5	- 7	+ 3	- 4	+ 1	- 2
14	- 11	- 8	+ 7	- 13	+ 10	- 12	+ 9	- 6	+ 5	+ 4	- 3	+ 2	- 1

In the previous studied problem, the minimal solution can be found only if some symmetries have been removed. In this new problem, it is difficult to identify some symmetries. Only one symmetry appears: the home games and the away games can be all exchanged. So the constraint $P_{11} \leftarrow 1$ remains in the model. All the others constraints that was introduced in order to remove some symmetries must be deleted.

We can also use the result proved by Schreuder: the minimal value is at least $n-2$. Then, the constraint $\#B \geq n-2$ is introduced in the model. This information is also interesting because another algorithm than branch-and-bound can be used. If the minimal value is close to $n-2$ then it is more interesting to try successively all the values from $n-2$ w.r.t an increasing order than finding a first solution and then trying to reduce the objective value.

For small numbers of teams there is no real difference as it is shown in the two following tables. In the first one, a depth-first search branch-and-bound algorithm is used, while in the second one the objective value is first instantiated.

#teams	6	8	10
#bk	16	3,899	352,701
time (s)	0.0	0.7	73

#teams	6	8	10
#bk	5	3,843	349,159
time (s)	0.0	0.7	71.5

When n increases the time needed to find the first solution becomes too long to use a branch and bound algorithm. This is also due to the fact that the $\#Bt_i$ variables are first instantiated. Neithertheless, we do not find better variable-value ordering.

From these experiements we can see that we are not able to solve instances

for more than 10 teams. So, the model has to be changed.

4.1 Introduction of other pertinent constraints

The first pertinent constraint that can be added comes from the observation that the number of breaks is always an even number.

Property 1 *For each pair of consecutive periods j and $j+1$, let $\#Bp_j$ be the variable that counts the number of breaks for the pair of consecutive periods. Then, $\forall j = 1..n-2 : \#Bp_j$ is even*

proof:

> Consider two consecutive periods j and $j+1$. Let p be the number of home breaks of these two consecutive periods and q be the number of away breaks. Suppose that $p \neq q$. Without loss of generality we can consider that $p > q$. When a team plays a home game its opponent plays a away game, so in each period there are $n/2$ home games and $n/2$ away games. Then, in period j, $n/2-q$ away games have to be defined such that no new breaks appears. This result can be obtained if $n/2 - q$ home games are available in period $j+1$, but $p > q$ this means $n/2-p < n/2-q$. Thus, the number of remaining home games in period $j+1$ is not sufficient. Hence $p = q$ and $\#Bp_j = p + q = 2p$ is even.

Property 2 *$\#B$ is even*

proof:

> $\#B = \sum_{j=1..n-2} \#Bp_j$, each $\#Bp_j$ is even so $\#B$ is even.

After the introduction of the constraint $\#B$ is even, the number of backtracks and the time needed to solve the problems are divided by a factor of 3:

#teams	6	8	10
#bk	5	970	101,844
time (s)	0.0	0.2	20.8

However, only a small factor is gained. This is not sufficient to solve instances with more than 10 teams.

By a careful study of the problem a non obvious constraint can be added. This constraint deals with the parity of the $\#Bt_i$ variables. Suppose that for a team i the first game is an home game and the third game is an away game. The other locations are not known for this team. Suppose now that only one break is required for this team. Such a requierement arises often because the $\#Bt_i$ variables are instantiated first. It is obvious that there will be a break either for the first and the second games or for the second and the third games. So, if only one break is required, it is impossible that there is a break involving a game succeeding to the third game. That means that the fourth game will be an home game, the fifth an away game, the sixth an home game etc...

The following property is a generalization of this idea.

Property 3 *Let j be any period, and k be any other period such that $k = j + p$, $p \geq 0$. Let $\#Bt_i(j,k)$ be the number of breaks for the team i between the period j and the period k. Then,*
$P_{ij} = P_{ik} \Leftrightarrow \#Bt_i(j,k)$ has the parity of p.

proof:

By induction. This is true for $p = 0, 1, 2$.
Let $\overline{P_{ik}}$ be the opposite value of P_{ik}.

First, suppose that $P_{ij} = P_{ik}$. For any assignement of the P_{ix} for $x = j+1, ..., k-1$ there exists a period l, such that $j \leq l < k$ and $P_{il} = P_{ik}$ and $P_{iy} = \overline{P_{ik}}$ for $z = l+1, ..., k-1$. Moreover, $\#Bt_i(j,k) = \#Bt_i(j,l) + \#Bt_i(l+1, k-1)$, because $P_{il} \neq P_{i(l+1)}$ and $P_{i(k-1)} \neq P_{ik}$. And $P_{ij} = P_{il}$ so $\#Bt_i(j,l)$ has the parity of $l - j$, because $l < k$. All the P_{iy} values for $z = l+1, ..., k-1$ are equals, then $\#Bt_i(l+1, k-1) = k - l - 2 = j + p - l - 2$. So $\#Bt_i(j,k)$ has the parity of $(l-j) + (j+p-l-2) = p-2$ which has the parity of p.

Now, suppose that $P_{ij} \neq P_{ik}$. For any assignement of the P_{ix} for $x = j+1, ..., k-1$ there exists a period l, such that $j < l \leq k$ and $P_{il} = P_{ij}$ and $P_{iy} = P_{ik}$ for $z = l+1, ..., k$. Moreover, $\#Bt_i(j,k) = \#Bt_i(j,l) + \#Bt_i(l+1,k)$, because $P_{il} \neq P_{i(l+1)}$.

Since $P_{il} = P_{ij}$ and $P_{i(l+1)} = P_{ik}$, $\#Bt_i(j,l)$ has the parity of $l-j$ and $\#Bt_i(l+1,k)$ has the parity of $k-l-1 = j+p-l-1$. So $\#Bt_i(j,k)$ has the parity of $(l-j)+(j+p-l-1) = p-1$ which has not the parity of p.

As we have already mentionned it, this property is very usefull to deduce a lot of things for the place where the games are played for each team. A filtering algorithm can be deduced from this property.

Notation 1 *Let i be a team and*

• *$OddUSeq_i(j,k)$ a sequence of periods from the period j to the period k such that $k > j+1$, P_{ij} and P_{ik} are instantiated and $\forall z = j+1, ..., k-1, P_{iz}$ is not instantiated and the number of breaks involved in the sequence is odd.*

• *$EvenUSeq_i(j,k)$ a sequence of periods from the period j to the period k such that $k > j+1$, P_{ij} and P_{ik} are instantiated and $\forall z = j+1, ..., k-1, P_{iz}$ is not instantiated and the number of breaks involved in the sequence is even.*

• *$ExtUSeq_i(j)$ a sequence of periods either from the period 1 to the period j such that $j > 1$, P_{ij} is instantiated and $\forall z = 1, ..., j-1, P_{iz}$ is not instantiated, or from the period j to the period $n-1$ such that $j < n-1$, P_{ij} is instantiated and $\forall z = j+1, ..., n-1, P_{iz}$ is not instantiated.*

• *$\#oddUSeq_i$ is the number of sequences $OddUSeq_i(j,k)$*

• *$\#evenUSeq_i$ is the number of sequences $EvenUSeq_i(j,k)$.*

• *$\#extUSeq_i$ is the number of sequences $ExtUSeq_i(j)$.*

• *$\#Binst_i$ is the number of breaks already instantiated for the teams i.*

• *$maxOddUSeq_i$ the maximal number of breaks that can be generated by instantiate the sequences $OddUSeq_i(j,k)$.*

• *$maxEvenUSeq_i$ the maximal number of breaks that can be generated by instantiate the sequences $EvenUSeq_i(j,k)$.*

• *$maxExtUSeq_i$ the maximal number of breaks that can be generated by instantiate the sequences $ExtUSeq_i(j)$.*

Proposition 1 *Let i be a team, then the following properties hold:*

a) $min(\#Bt_i) \geq \#Binst_i + \#oddUSeq_i$

b) $max(\#Bt_i) \leq \#Binst_i + maxOddUSeq_i + maxEvenUSeq_i + maxExtUSeq_i$

c) $\#extUSeq_i = 0 \Leftrightarrow \#Bt_i$ *has the parity of* $(\#Binst_i + \#oddUSeq_i)$

d) $\#Bt_i$ *is odd* $\Leftrightarrow P_{i1} \neq P_{i(n-1)}$

e) $\#Bt_i$ *is even* $\Leftrightarrow P_{i1} = P_{i(n-1)}$

f) $\#Bt_i = nb$ *and* $\#oddUSeq_i = nb - \#Binst_i \Leftrightarrow$ *no break are generated by all the* $EvenUSeq_i(j,k)$ *and by all the* $ExtUSeq_i(j)$

g) $\#Bt_i = nb$ *and* $\#oddUSeq_i + 1 = nb - \#Binst_i \Leftrightarrow$ *no break are generated by all the* $EvenUSeq_i(j,k)$

h) $\#Bt_i = nb$ *and* $maxOddUSeq_i + maxEvenUSeq_i + maxExtUSeq_i + \#Binst_i = nd \Leftrightarrow$ *all the* $EvenUSeq_i(j,k)$ *and all the* $ExtUSeq_i(j)$ *generate their maximum of breaks.*

This proposition can be easily proved.

Note that if $\#Bt_i$ is instantiated then properties d) and e) prevents $\#extUSeq_i$ from being equal to 1.

For each team i a constraint is added and the filtering algorithm associated with this constraint corresponds to the application of the above proposition each time any $\#Bt$ variable or place variable is modified.

The results obtained by introducing these new constraints are given by the following table:

#teams	6	8	10	12
#bk	2	226	11,542	135,129
time (s)	0.0	0.1	4.0	55.3

For the first time the problem can be solved for 12 teams. The new added constraints can be considered as essential to solve the problem.

However, the resolution of the problem for 12 teams is not sufficient.

The model can be improved by taking into account a new conjecture:

Conjecture 1 *There are at most two teams* i *and* j *such that* $\#Bt_i = \#Bt_j = 0$.

proof:

> This conjecture has been "proved" by a computer until 60 teams (see the appendix).

The introduction of a constraint based on this conjecture greatly helps to solve the problem, because the $\#Bt$ variables are instantiated first, so the number of combinaisons to study is reduced.

Here are the new results:

#teams	6	8	10	12	14
#bk	2	41	846	2,435	1,716,513
time (s)	0.0	0.1	0.4	1.37	904.4

The problem for 14 teams can now be solved.
Finally we propose to slightly modify the variable-value ordering.

1. the $\#Bt_i$ variables are instantiated first, by dynamically choosing the variable that has the minimum value in its domain and by choosing this value.

2. the place variables for the first period are then instantiated.

3. the break variables are then instantiated by trying first the value 1.

4. the place variables are then instantiated

The results we obtained are:

#teams	6	8	10	12	14
#bk	2	41	846	2,209	711,408
time (s)	0.0	0.1	0.4	1.18	397.1

The performance of this new variable-value ordering can be explained by the correlation between the constraint obtained from proposition 1 and the place variables of the first period.

The problem can also be solved for 16 teams but in more than 5 hours and 20 millions of bakctracks.

We do not find any more constraints to improve the model. So, the problem becomes clearly solvable until 12 teams, and sometimes solvable for more than 12 teams.

4.2 Restrictions of the problem

If further information is known the problem can be solved more quickly.

Suppose that it is known that $\#Bt_1 > 0$ and $\#Bt_1 = 0$.

In this case the problem can be solved for 16 teams in one minute as shown in the following table:

#teams	6	8	10	12	14	16
#bk	1	6	67	128	4,533	108,563
time (s)	0.0	0.1	0.2	0.2	2.6	64.2

This means that such an information is quite usefull to solve the problem.

Suppose that no two consecutive breaks are allowed

This is a request that arises in real-life problem. This request slightly improves the resolution of the problem.

Suppose that for each team the first and the last game must have to be played at a different place

This is a request that arises in real-life problem.
By proposition 1, this means that all the $\#Bt$ variables are odd. The optimal value is changed. The time needed to solve the problem changes also.

#teams	6	8	10	12	14
opt val	6	12	16	20	24
#bk	2	61	464	4,740	158,784
time (s)	0.0	0.1	0.3	2.4	118.5

Suppose that for each team the number of away and home games must be almost equal

This is a request that arises in real-life problem. There is no modification in time and in number of backtracks to solve the problem.

5 Discussion

In real-life applications, it is often impossible to have a solution that respects all the constraints. In this case, a certain function must be minimized. Often, this function includes the number of breaks. So our study can be used to compute a minimal bound of this function. Moreover, we have presented a lot of constraints that cannot be violated for any sports scheduling problem. Thus, these constraints should be usefull in practice. Some new constraints should also be defined when some other information are available. For instance, if no two consecutive breaks are allowed then proposition 1 can be extended to deal with this new condition.

When the teams are not instantiated, we advice to introduce the following hidden constraint:

- *for each period every team plays.*

For each period j, an alldiff constraint involving the O_{ij} variables for $i = 1..n$ is defined.
This constraint is often redundant. (This is the case for the problem we have studied). However, we think that it can greatly help the resolution of certain instances where only some games are already defined.

6 Conclusion

In this paper, we have shown the interest of constraint programming for solving a large scale optimization problem: the minimization of the number of breaks in sports scheduling problems. First we introduced a model that solves a polynomial problem in a polynomial time with constraint programming. This shows that constraint programming is also an efficient technique to solve easy instances. Then, we have studied a modification of the original problem which is hard to solve. The introduction of ad-hoc usefull constraints leads to results that are quite promising. These new constraints should be used to improve the resolution of complex real-life applications.

References

[CGL93] Y. Caseau, P-Y. Guillo, and E. Levenez. A deductive and object-oriented approach to a complex scheduling problem. In *Proceed-*

ings of DOOD'93, 1993.

[MTW97] K. McAloon, C. Tretkoff, and G. Wetzel. Sports league scheduling. In *Proceedings of ILOG user's conference*, Paris, July 1997.

[NT98] G. Nemhauser and M. Trick. Scheduling a major college basketball conference. *Operations Research*, 46(1):1–8, 1998.

[Rég94] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI-94, proceedings of the Twelth National Conference on Artificial Intelligence*, pages 362–367, Seattle, Washington, 1994.

[Sch92] J. Schreuder. Combinatorial aspects of construction of competition dutch professional football leagues. *Discrete Applied Mathematics*, 35:301–312, 1992.

A Appendix

The source code in Solver 4.3 of the different models and algorithms presented in this paper is available upon request by sending a message to regin@ilog.fr.