Global Constraints for Round Robin Tournament Scheduling

Martin Henz [a] Tobias Müller [b] Sven Thiel [c]

[a] *School of Computing, National University of Singapore, Singapore 117543*

[b] *Programming Systems Lab, Saarland University, 66041 Saarbrücken, Germany*

[c] *Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany*

Abstract

In the presence of side-constraints and optimization criteria, round robin tournament problems are hard combinatorial problems, commonly tackled with tree search and branch-and-bound optimization. Recent results indicate that constraint-based tree search has crucial advantages over integer programming-based tree search for this problem domain by exploiting global constraint propagation algorithms during search. In this paper, we analyze arc-consistent propagation algorithms for the global constraints "all-different" and "one-factor" in the domain of round robin tournaments. The best propagation algorithms allow us to compute all feasible perfectly mirrored pattern sets with minimal breaks for intermural tournaments of realistic size, and to improve known lower bounds for intramural tournaments balanced with respect to carry-over effects.

Key words: timetabling, constraints satisfaction, graph theory

1 Introduction

In round robin sport competitions, each team plays each other team a fixed number of times and the matches are organized in rounds. Round robin schedules can be characterized as one-factorizations of complete graphs and are studied in graph theory and combinatorial design. Numerous results have been obtained on variants of the round robin scheduling problem, including intermural tournaments, facility-sharing tournaments and bipartite tournaments; extensive references are given in [2,23]. The techniques in this field are constructive in a sense that interesting properties of tournaments are identified and then—by employing graph-theoretical and combinatorial arguments—methods to construct corresponding tournaments are described.

This works well for highly regular tournaments. However, in the presence of irregular constraints which occur in tournament planning practice and which are difficult to capture as properties of graphs, constructive methods fail and the problem degenerates to a combinatorial search problem. Techniques that have been used to solve such problems include integer programming [20,15], local search [24] and constraint programming [19,11,7]. Constraint programming has been shown recently to outperform integer programming on practical problems [8,7]. Constraint programming allows to systematically exploit the round robin and other constraints, often leading to relatively small search trees for medium-sized tournaments.

To solve large problems, stronger propagation algorithms for pruning the search trees become important. In finite domain programming systems such as Ilog Solver [10] and Mozart [13], where constraints are encoded as propagators that operate on a constraint store which stores domains of variables, arc-consistency is the strongest kind of propagation that can be achieved for a given constraint. The most important constraints for round robin tournaments are

- the all-different constraint, which expresses that the rows in a tournament contain every team only once, and
- the one-factor constraint, called symmetric all-different by Régin [17], which expresses that every column groups the teams into matches.

Régin gives arc-consistent propagation algorithms for both problems, which we will review in Section 5, after introducing graph-theoretical notation in Section 2, presenting the basic ideas of constraint programming in Section 3, and giving a formal description of the two constraints in Section 4.

A practical consideration in modeling of combinatorial search problems using constraint programming is the trade-off between the strength of propagation at each node of the search tree and its computation time. For example, a naive non-arc-consistent propagation algorithm for all-different constraint sometimes outperforms the arc-consistent one; in such cases, the decrease in the size of the search trees does not outweigh the increase of time that is spent at each of their nodes. This situation is common in constraint programming and necessitates an experimental evaluation for a given application domain.

An extensive experimental evaluation of propagation algorithms for the round robin tournaments is given in Section 6. Using the empirically best propagation algorithms, we are able to compute all feasible perfectly mirrored pattern sets with minimal breaks for intermural tournaments of realistic size, and to improve a known lower bound for intramural tournaments balanced with respect to carry-over effects.

2 Round Robin Tournaments and Graphs

In round robin sport competitions, each team plays each other team a fixed number of times g during the competition. Let us first assume $g = 1$, thus we are dealing with single round robin tournaments (SRR). A temporally dense single round robin (DSRR) for n teams distributes the $n(n-1)$ matches over a minimal number of rounds such that every team plays at most one match per round. If n is even, the number of rounds is $n-1$. A DSRR with an odd number of teams consists of n rounds in each of which $n-1$ teams play and one team does not. This team is said to have a bye. In the following, we are limiting ourselves to an even number of teams, since the problem for an odd number of teams $n-1$ can be reduced to the n case by introducing an additional team that always "plays" against the team with a bye. When $g = 2$, we speak of a dense double round robin (DDRR).

The planning of a DSRR consists of assigning for each round an opponent team to each team. Often other decisions have to be taken as well, such as the place in which the matches are carried out. For intermural tournaments, this amounts to the question whether a team plays home or away. For intramural tournaments, a court may need to be selected. We first concentrate on opponent team assignment and discuss intermural tournaments in Section 6.

The single round robin schedule on the right shows a valid assignment of opponent teams for $n = 6$ and $g = 1$ in each round. The value in row t and column r tells the team against which team t plays in round r.

		rounds				
		1	2	3	4	5
teams	1	2	4	6	3	5
	2	1	3	5	6	4
	3	5	2	4	1	6
	4	6	1	3	5	2
	5	3	6	2	4	1
	6	4	5	1	2	3

In order to characterize the mathematical properties of round robin schedules, we need to introduce some terminology on graphs.

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E where $(x, x) \notin E$ (no self-loops). The degree of G is the maximal number of edges incident to some vertex in G. G is called *complete*, if there is an edge from any vertex to any other. A *factor* of G is a subgraph of G with vertex set V. A *factorization* of G is a set of factors of G which are pairwise edge-disjoint and whose union of edges is E. A set $M \subset E$ is called a *matching* in G, if no two distinct edges in M share a common endpoint. We call a vertex v *matched* by M if it is incident to some edge in M, and *free* otherwise. A matching M is called *perfect* if it covers all vertices of G, i.e. there are no free vertices. A perfect matching is also called a *one-factor*, because it is a factor with degree 1. A one-factorization of G is a factorization of G consisting of one-factors.

A one-factorization of the complete graph with n nodes (n being even) corresponds to a DSRR for n teams as follows. Every node i represents a team, every one-factor represents a round, and an edge (i, j) in a one-factor r fixes a match between teams i and j in round r. The properties of factorizations guarantee that every team plays every other team exactly once. This fact is employed in constructive methods for tournament planning, see references in [23].

3 Constraint Programming

We represent round robin tournament problems as a constraint satisfaction problem (CSP). A CSP is a triple $\mathcal{P} = (X, D, C)$, where X is a finite set of variables, and D assigns to each variable $x \in X$ a finite domain D_x of possible values. Each element c of C expresses a constraint on some variables $x_1, \ldots, x_k$, and thus $c \subseteq D_{x_1} \times \cdots \times D_{x_k}$.

A solution s to the constraint problem $\mathcal{P}$ assigns to each variable x a value $s_x \in D_x$ such that each constraint is satisfied. This means that for every constraint c on variables $x_1, \ldots, x_k$, $(s_{x_1}, \ldots, s_{x_k}) \in c$ holds. The set of all solutions to a constraint problem $\mathcal{P}$ is denoted by $\mathit{sol}(\mathcal{P})$.

The constraint programming approach to solving combinatorial search problems such as round robin scheduling problems works as follows. Encode the problem as a constraint satisfaction problem $\mathcal{P}$, find a new problem $\mathcal{P}'$ that has the same set of solutions by applying so-called consistency techniques. Now augment $\mathcal{P}'$ in two ways, by adding a new constraint c' and its negation, respectively, to $\mathcal{P}'$. To the resulting problems $\mathcal{P}' + c'$ and $\mathcal{P}' + \neg c'$, apply again consistency techniques, find new constraints c'' for each of the problems, and so on. This process leads to a binary search tree at whose leaves are either problems that have no solution, or problems where D contains only singletons, which directly correspond to solutions.

There are many degrees of freedom in this process, including the original encoding of the problem, the consistency techniques to be applied, the choice of new constraints at each step and the order in which the resulting search tree is explored. The success of constraint programming relies on good choices for all these components. However, we are here mainly concerned with consistency techniques. For the other aspects of constraint-based round robin scheduling, see [7,8].

A propagation technique is a function that maps constraint problems $\mathcal{P} = (X, D, C)$ to new constraint problems $\mathcal{P}' = (X, D', C)$, where for every $x \in X$, ${D_x}' \subseteq D_x$, and where $\mathit{sol}(\mathcal{P}) = \mathit{sol}(\mathcal{P}')$.

A CSP is arc-consistent with respect to the constraint c on variables $x_1, \ldots, x_k$, if for each index $i \in \{1, \ldots, k\}$, and each value $v \in D_{x_i}$, there exists an element $(v_{x_1}, \ldots, v_{x_{i-1}}, v, v_{x_{i+1}}, \ldots, v_{x_k}) \in c$. A CSP is arc-consistent, if it is arc-consistent with respect to all of its constraints. Arc-consistent propagation is a propagation technique that turns a given CSP into an arc-consistent CSP.

4 Constraints for Round Robin Scheduling

The canonical constraint satisfaction problem for opponent team assignment in DSRR represents the target timetable by an $n \times (n-1)$ matrix o of variables, whose variables $o_{t,r}$ tell the opponent team against which team t plays in round r. More formally, we define a DSRR problem as $\mathcal{P}_{DSRR} = (X, D, C)$, where X contains all variables in o and $D_{o_{t,r}} = \{1, \ldots, n\}$ for every team t and round r. The set C contains the following constraints.

$$\textit{all-different}(o_{t,1}, \ldots, o_{t,n-1}), \text{ for every } t \in \{1, \ldots, n\}, \text{ and} \tag{1}$$
$$\textit{one-factor}(x_{1,r}, \ldots, x_{n,r}), \text{ for every } r \in \{1, \ldots, n-1\}. \tag{2}$$

where the constraints *all-different* and *one-factor* are defined as follows:

$$\begin{aligned} \textit{all-different}(x_1, \ldots, x_m) = \{&(v_1, \ldots, v_m) \in D_{x_1} \times \cdots \times D_{x_m} \mid \\ &\forall_{i,j,i \neq j}.v_i \neq v_j\}, \end{aligned} \tag{3}$$
$$\begin{aligned} \textit{one-factor}(x_1, \ldots, x_m) = \{&(v_1, \ldots, v_m) \in D_{x_1} \times \cdots \times D_{x_m} \mid \\ &\forall_{i,j}.v_i \neq i \wedge v_i = j \leftrightarrow v_j = i\} \end{aligned} \tag{4}$$

Propagation algorithms for all-different and one-factor vary in strength. For the all-different constraint, we consider the following two propagation algorithms:

(1) splitting the constraint up into $m(m-1)$ inequality constraints of the form $x_i \neq x_j$ according to its definition in formula (3) (arc-consistency with respect to such inequalities is trivial), and
(2) arc-consistent propagation with respect to the all-different constraint itself.

For the one-factor constraint, we consider three propagation algorithms:

(1) after introducing a half-matrix of auxiliary variables $r_{i,j}$, where $1 \leq i < j \leq m$, whose domains are $D_{r_{i,j}} = \{0, 1\}$, arc-consistent propagation corresponding to the constraints

$$x_i \neq i \text{ for } 1 \leq i \leq m \tag{5}$$
$$\textit{eq}(x_i, j, r_{i,j}) \text{ for } 1 \leq i < j \leq m \tag{6}$$
$$\textit{eq}(x_i, j, r_{j,i}) \text{ for } 1 \leq j < i \leq m \tag{7}$$

(the constraint *eq* reflects the equality of the first two arguments into the third argument),

(2) arc-consistent propagation with respect to these constraints, plus arc-consistent propagation with respect to the redundant constraint *all-different*$(x_1, \ldots, x_m)$, and
(3) arc-consistent propagation with respect to the one-factor constraint itself.

The propagation behavior of (1) is strictly weaker than arc-consistent propagation for the one-factor constraint.

Example 1 *For* $m = 6$, *let* $D_{x_1} = \{2, 3, 4, 5, 6\}$, $D_{x_2} = \{1, 2, 3, 6\}$, $D_{x_3} = \{1, 2, 4, 5, 6\}$, $D_{x_4} = \{1, 3, 6\}$, $D_{x_5} = \{1, 2, 3, 5\}$, $D_{x_6} = \{1, 2, 3, 5\}$. *Arc-consistency with respect to the neq constraints removes* 2 *from* D_{x_2} *and* 5 *from* D_{x_5}, *and arc-consistency with respect to the eq constraints removes* 6 *from* D_{x_4}, 2 *from* D_{x_5} *and* 5 *from* D_{x_6}. *Arc-consistency with respect to the neq and eq constraints fails to reach arc-consistency with respect to the one-factor constraint, which further removes* 2, 3 *and* 6 *from* D_{x_1}, 1 *and* 3 *from* D_{x_2}, 1, 2 *and* 6 *from* D_{x_3}, *and* 1 *and* 3 *from* D_{x_6}.

Adding the redundant constraint *all-different*$(x_1, \ldots, x_m)$ with arc-consistent propagation, as done in propagation algorithm (2), improves the propagation behavior in some cases. In Example 1, arc-consistent propagation with respect to this constraint and the *neq* and *eq* constraints achieves arc-consistency with respect to the *one-factor* constraint. The next example shows that this is not always the case.

Example 2 *For* $m = 6$, *let* $D_{x_1} = \{2, 3, 6\}$, $D_{x_2} = \{1, 6\}$, $D_{x_3} = \{1, 4, 5\}$, $D_{x_4} = \{3, 5\}$, $D_{x_5} = \{3, 4\}$, $D_{x_6} = \{1, 2\}$. *Here, arc-consistency with respect to the neq and eq constraints and the all-different constraint is not able to remove any values from any domain, whereas arc-consistency with respect to the one-factor constraint removes* 2 *and* 6 *from* D_{x_1}, 1 *from* D_{x_2}, 4 *and* 5 *from* D_{x_3}, 3 *from* D_{x_4}, 3 *from* D_{x_5}, *and* 1 *from* D_{x_6}, *thus fixing the one-factor* $1-3$, $2-6$ *and* $4-5$.

We conclude from these examples that arc-consistent propagation for the one-factor constraint deserves consideration and—assuming that there is an efficient algorithm for it—has the potential for improving round robin scheduling beyond the addition of an arc-consistent redundant all-different constraint.

Most previous work on constraint-based tournament planning [19,11,8,7] used only algorithm 1 for the all-different constraint and algorithm 1 for the one-factor constraint. Trick [22] suggests to use algorithm 2 for the all-different constraint and algorithm 2 for the one-factor constraint. Our goal is to evaluate the propagation algorithms to achieve guidelines for using propagation algorithms in round robin scheduling.

5 Propagation Algorithms

Consider the constraint *all-different*$(x_1,\ldots,x_m)$. The *value graph* of this constraint is the bipartite graph $G=(V,E)$ with $V=\{x_1,\ldots,x_m\}\cup\bigcup_i D_{x_i}$ and $E=\{\{x_i,v\}\mid v\in D_{x_i}\}$.

Lemma 1 *(Régin [16]) The following algorithm is a propagation technique for arc-consistent propagation with respect to the constraint all-different*$(x_1,\ldots,x_m)$. *Construct the value graph of the constraint. For every variable x and value v, such that $\{x,v\}$ is not a matchable edge in G, remove v from D_x.*

Régin gives an algorithm for identifying non-matchable edges in a value graph with m variables and k values with complexity $O(km^{3/2})$. For round robin tournaments, both k and m are bounded by the number of teams n, resulting in a complexity of $O(n^2\sqrt{n})$.

Now, consider the constraint *one-factor*$(x_1,\ldots,x_m)$. The *variable graph* of this constraint is the graph $G=(V,E)$ with the vertex set $V=\{x_1,\ldots,x_m\}$ and the edge set $E=\{\{x_i,x_j\}\mid j\in D_{x_i}\}$.

Looking at the definition of the one-factor constraint, it is easy to derive a one-to-one correspondence between the solutions of the constraint and the perfect matchings in its variable graph G:

- Let $(v_1,\ldots,v_m)\in$ *one-factor*$(x_1,\ldots,x_m)$ denote a solution. The set $M=\{\{x_i,x_{v_i}\}\mid 1\le i\le m\}$ is well-defined, a subset of E and a perfect matching in G.
- The matching M' corresponds to the solution $(v'_1,\ldots,v'_m)$ of the constraint where v'_i is the index of the mate of the node x_i in the matching M'.

This observation motivates the following definitions: We call an edge e of G *matchable* if there is a perfect matching M in G containing e and *unmatchable* otherwise.

Lemma 2 *(Régin [17]) The following algorithm is a propagation technique for arc-consistent propagation with respect to the constraint one-factor*$(x_1,\ldots,x_m)$. *Iterate over the domains $D_{x_1},\ldots,D_{x_m}$ to enforce constraints*

$$i\notin D_{x_i}\ \text{for}\ 1\le i\le m\quad\text{and}\quad i\in D_{x_j}\leftrightarrow j\in D_{x_i}\ \text{for}\ 1\le i<j\le m \tag{8}$$

and construct the variable graph G for the constraint. For every pair of values i and j, such that $\{x_i,x_j\}$ is not a matchable edge in G, remove i from D_{x_j} and remove j from D_{x_i}.

The problem of finding the unmatchable edges was presented in [1, Exercise 12.42]. It was stated that the problem can be solved with a variation of Edmonds' blossom-shrink algorithm [5,6]. Régin's algorithm follows these ideas. The running time of the resulting algorithms is $O(ms)$, where m denotes the number of nodes and s the number of edges of the variable graph. In round robin tournaments, m is bounded by the number of teams n and s is bounded by n^2, resulting in a complexity of the propagation algorithm of $O(n^3)$.

6 Experimental Evaluation

For the experimental evaluation, we use the programming system Mozart 1.1.0 [13] for the concurrent constraint language Oz [21], which provides extensive support for finite domain constraint programming. We implemented the propagation algorithms *all-different* and *one-factor* described in the previous sections using the LEDA library [12] and made them available to the Mozart system through Mozart's Constraint Propagator Interface [14]. The run times given in this section are always the average user time of five runs on a 256 MB 400MHz Pentium II PC running Linux. The coefficient of deviation (standard deviation / arithmetic mean) was always below 3 %.

The C++ and Oz source code for generating and running these benchmarks is available at [9]. The goals of the experimental evaluation are as follows:

- evaluate the usefulness of the arc-consistent one-factor constraint for round robin applications by comparing the sizes of search trees and run times resulting from searches that use the new constraint with searches that use arc-consistent all-different or simply the encoding using *neq* and *eq*,
- investigate the range of round robin problems to which arc-consistent one-factor provides advantages over other techniques, and
- evaluate the efficiency of the arc-consistent one-factor constraint for practical applications.

We observe from all experiments that the use of the arc-consistent all-different constraint is crucial. It typically leads to a reduction of the size of the search tree and the runtime by one or two orders of magnitude. Large tournament problems can only be solved using arc-consistent propagation for all-different, and thus we fix this propagation algorithm in the rest of this section.

Unconstrained Single Round Robin Tournaments

Here, we compare the performance of arc-consistent one-factor versus the encoding using *neq* and *eq* and the redundant all-different constraint. In this and all following benchmarks, we use constraint-based tree search by using constraints of the form $o_{t_1,r} = t_2$ as branching constraints c (see Section 3), which means that we enumer-

Table 1 Benchmarks on unconstrained DSRR; n: number of teams; F: number of failures in the search tree; UT: user time; the ? symbols indicate that 30 minutes of runtime were exceeded

	neq / eq		*all-diff*		*one-factor*	
n	F	UT	F	UT	F	UT
6	0	0.030	0	0.048	0	0.020
10	3	0.428	1	0.558	0	0.220
14	62	3.18	0	4.35	0	1.13
18	20	5.82	0	8.86	0	4.37
22	675	81.3	2	25.0	0	12.7
26	?	?	2	57.1	0	32.0
30	?	?	0	122.	0	71.2
34	?	?	17	241.	0	145.
38	?	?	5	429.	0	272.
42	?	?	4	742.	0	484.

ate the opponent variables. We order the variables round-wise. At each node, we enumerate the first variable that has a non-singleton domain. The value t_2 is always the smallest element of this domain. More sophisticated enumeration techniques such as first-fail do not improve the search. Table 1 compares the encodings using *neq / eq* from Section 4, Formula (4), the addition of a redundant arc-consistent all-different constraint as discussed in Example 1, and the algorithm *one-factor*, for finding the first DSRR.

We observe that for these kinds of benchmarks, arc-consistent one-factor achieves optimal propagation in a sense that there are no failures in the search trees, whereas the encoding using *neq / eq* requires search. From 26 teams onward, this method fails to produce solutions within reasonable time. Arc-consistent all-different occasionally requires a bit of search, but the performance difference to arc-consistent one-factor is not dramatic here.

Tightly Constrained Round Robin Tournaments

The next set of benchmarks looks at tightly constrained problems. We constrain the problems by randomly forbidding opponent teams, until there are very few or no solutions to the problem. Table 2 compares the three remaining propagation algorithms for finding all solutions, or proving unsatisfiability. The problems `s_*` are single round robin problems as described throughout the paper, whereas the problems `d_*` are double round robin problems. For the latter, we have instead of an all-different constraint per team (requiring that each team plays each other team once) constraints that force the number of occurrences of all other teams in each row of the o matrix to be 2. There is no restriction on the distance between first leg and return match.

Table 2 Benchmarks on tightly constrained DSRR and DDRR; n: number of teams; S: number of solutions; F: number of failures in the search tree; UT: user time

			neq / eq		*all-diff*		*one-factor 2*	
file	n	S	F	UT	F	UT	F	UT
`s_6_yes`	6	4	7	0.088	5	0.124	4	0.060
`s_8_yes`	8	5	47	0.540	24	0.606	10	0.200
`s_10_yes`	10	1	37	0.704	17	0.686	1	0.106
`s_12_yes`	12	1	3216	77.9	1452	64.1	179	6.26
`s_14_yes`	14	1	8407	242.	1328	75.3	527	20.4
`s_6_no`	6	0	4	0.048	4	0.066	4	0.022
`s_8_no`	8	0	13	0.172	12	0.246	6	0.086
`s_10_no`	10	0	20	0.530	13	0.646	6	0.168
`s_12_no`	12	0	241	6.64	111	5.37	25	0.794
`s_14_no`	14	0	537	16.7	182	10.9	69	2.54
`s_16_no`	16	0	1467	64.5	213	18.0	86	5.37
`s_18_no`	18	0	593	38.6	95	9.57	30	2.29
`s_20_no`	20	0	?	?	2755	314.	254	23.0
`d_6_yes`	6	4	21	0.066	4	0.084	2	0.020
`d_8_yes`	8	32	5776	19.2	2736	28.1	226	0.93
`d_10_yes`	10	2	76646	409.	38251	677.	5956	35.6

Although the results vary considerably, we note that in these tightly constrained problems, arc-consistent one-factor results in a reduction of the size of search trees by a factor of up to 10, and of the runtime by even more. The difference between the two techniques increases with the problem size. Note that for the double round robin problems, the reduction of the search tree afforded by the redundant all-different constraint does not justify its computational effort.

Minimizing Carry-Over

We consider sports, in which a match between two teams a and b has an impact on the performance of these teams in the next round, an effect called carry-over. In such a sport, each sequence of two teams should appear at most once in such a schedule, leading to a schedule balanced with respect to carry-over effects. For example, this is not the case for the schedule on page 3; the sequence $(3, 5)$ appears three times. Since this ideal is not always achievable,

		rounds								
		1	2	3	4	5	6	7	8	9
teams	1	6	5	10	9	4	8	7	2	3
	2	7	4	9	10	6	3	8	1	5
	3	8	9	7	6	5	2	4	10	1
	4	9	2	8	7	1	10	3	5	6
	5	10	1	6	8	3	7	9	4	2
	6	1	7	5	3	2	9	10	8	4
	7	2	6	3	4	10	5	1	9	8
	8	3	10	4	5	9	1	2	6	7
	9	4	3	2	1	8	6	5	7	10
	10	5	8	1	2	7	4	6	3	9

small carry-over schedule for $n = 10$

the goal is to minimize the carry-over effect using a cost function. Russell [18] gives a constructive method for generating tournaments with no carry-over effect where n is a power of two, and conjectures the non-existence of such tournaments in all other cases. He gives a constructive method to generate tournaments with small carry-over effects leading to a carry-over effect of 60 for $n = 6$, 138 for $n = 10$ and 196 for $n = 12$ according to a canonical cost measure. With constraint-based branch-and-bound using arc-consistent propagation for one-factor constraints, we are able to prove the optimality of his schedule for $n = 6$. However, for $n = 10$, we obtain a schedule with a better cost (136) after 26.2 seconds. The best values that we obtain is 128 for $n = 10$ shown on the previous page, which is obtained using a randomized search strategy (the runtime of about 30 minutes is therefore not very informative). Trick [22] reports an even better cost (122) after 1 day of runtime, also using constraint programming. For $n = 12$, we improve the best known schedule (cost 196) given in [18] and achieve a schedule with a cost of 188, which is given in [9].

In this study, the use of arc-consistent propagation for the one-factor constraint is again crucial.

Feasible Pattern Sets

Here, we consider *intermural* dense double round robins (DDRR) ($g = 2$), where the second part of the double round robin repeats the first part with opposite venues. A team is said to have a *break*, if it either plays two consecutive matches home or away. We consider the problem of finding all schedules that minimize the overall number of breaks. A widely accepted method of searching for intermural tournaments is to generate pattern sets first [20,15,7,8]. These are sets of home/away patterns that satisfy simple row and column constraints [7]. However, not all such pattern sets lead to schedules even if there are no other side constraints. De Werra [4] gives a construction of feasible pattern sets with $3n-6$ breaks, proves their optimality and states that no constructive method is known to enumerate all feasible pattern sets. In this situation, it is useful to enumerate *feasible* pattern sets, which—surprisingly—has to our knowledge not been tackled so far. For this task we use the constraint model given in [7] and add the model for opponents given in Section 4. The table on the right gives the number of pattern sets for $n \leq 18$; the pattern sets are listed in [9]. The column P gives the number of pattern sets that fulfill the pattern set constraints, the column F gives the number of pattern sets that fulfill the model for opponents. For all $n \leq 16$, there exists a schedule for every computed pattern set. For $n = 18$, there are 4 cases, for which we could neither prove infeasibility nor generate a schedule.

n	P	F
4	0	0
6	1	1
8	4	2
10	15	4
12	56	10
14	210	17
16	792	46
18	3003	84

For $n \leq 16$ the use of the arc-consistent one-factor constraint or the redundant arc-consistent all-different constraint for one-factor was not crucial and the pattern sets were obtained faster with trivial propagation for the one-factor constraint. For $n = 18$, the arc-consistent propagation for the one-factor constraint allows to prove the infeasibility of 17 pattern sets; without it, the number of pattern sets generated is 101.

Intermural Tournaments

The intermural benchmarks in this section show that the pruning obtained from arc-consistent one-factor not always outweighs its computational effort. The best strategy for intermural tournament problems is to first find so-called pattern sets [3]. Table 3 shows the run times for finding all solutions of intermural tournament problems, all of which except the last one are randomly constrained as in the previous section. The last problem is the ACC 1997/98 problem [15,8], which is tightly constrained by a variety of conditions. Since we are not concerned with the computation of the pattern sets, we fix a particular pattern set for the benchmarks, except for the ACC 1997/98 problem. The numbers for this last problem include the effort for generating pattern sets, since the one-factor constraint already achieves some additional pruning during pattern set generation.

We observe that the effort for neither arc-consistent one-factor nor arc-consistent all-different is is justified for these intermural tournaments. The reason is that pattern sets already enforce one-factor to such an extent that arc-consistent one-factor achieves almost no additional pruning of the search trees. Note that the arc-consistent one-factor constraint could be sped up in this case by exploiting that the variable graph is bipartite, which would lead a complexity of $O(n^2\sqrt{n})$, similar to the arc-consistent all-different.

Table 3 Benchmarks on intermural tournaments; n: number of teams; S: number of solutions; F: number of failures in the search tree; UT: user time

			neq / eq		*all-diff*		*one-factor 2*	
file	n	S	F	UT	F	UT	F	UT
`i_8_yes`	8	7	0	0.138	0	0.182	0	0.142
`i_12_yes`	12	3	0	0.440	0	0.510	0	0.436
`i_16_yes`	16	4	6	1.30	4	1.51	4	1.22
`i_20_yes`	20	10	35	4.23	22	4.63	22	3.61
`acc97/98`	9	179	273	19.5	273	28.4	268	22.1

7 Conclusion

We analyzed the use of the global constraints all-different and one-factor for constraint-based search for round robin tournament schedules. We conclude from an extensive experimental evaluation that arc-consistent propagation for the all-different constraint is crucial for efficient solution of all tournament scheduling problems considered.

Arc-consistent propagation for the one-factor constraint is essential for intramural tournaments. For large unconstrained and tightly constrained single and multiple round robin tournaments, we observe a typical reduction of the search tree and runtime by one order of magnitude. Intermural tournaments do not benefit much from the arc-consistent one-factor constraint.

Using these algorithms, we could establish new lower bounds for the minimization of carry-over effects for intramural single round robin tournaments and enumerate the all feasible pattern sets for intramural tournaments with a minimal number of breaks for up to 16 teams. For 18 teams, there are 4 open cases.

Acknowledgments

The authors would like to thank Ernst Althaus, Kurt Mehlhorn and Tony Tan for many helpful discussions, Jan A. M. Schreuder for providing the motivation for the work on pattern sets, Michael Trick for pointing out omissions and mistakes, Marleen van Brandenburg for carrying out initial experiments on round robin tournament benchmarks, and the reviewers for pointing out the potential of the optimization for intermural tournaments.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows : theory, algorithms and applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] I. Anderson. *Combinatorial Designs and Tournaments*. Oxford University Press, 1997.

[3] W. O. Cain, Jr. The computer-assisted heuristic approach used to schedule the major league baseball clubs. In S. P. Ladany and R. E. Machol, editors, *Optimal Strategies in Sports*, number 5 in Studies in Management Science and Systems, pages 32–41. North-Holland Publishing Co., Amsterdam, New York, Oxford, 1977.

[4] D. de Werra. Scheduling in sports. In P. Hansen, editor, *Studies on Graphs and Discrete Programming*, pages 381–395. North-Holland, Amsterdam, 1980.

[5] J. Edmonds. Maximum matching and a ployhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[6] J. Edmonds. Paths, trees, and flowers. *Canadian Journal on Mathematics*, pages 449–467, 1965.

[7] M. Henz. Constraint-based round robin tournament planning. In D. De Schreye, editor, *Proceedings of the International Conference on Logic Programming*, pages 545–557, Las Cruces, New Mexico, 1999. The MIT Press, Cambridge, MA.

[8] M. Henz. Scheduling a major college basketball conference—revisited. *Operations Research*, 49(1), Jan. 2001.

[9] M. Henz, T. Müller, S. Thiel, and M. van Brandenburg. Benchmarks and results for round robin tournaments. `http://www.comp.nus.edu.sg/~henz/roundrobin_benchmarks`, 2000.

[10] ILOG Inc., Mountain View, CA 94043, USA, `http://www.ilog.com`. *ILOG Solver 4.0, Reference Manual*, 1997.

[11] K. McAloon, C. Tretkoff, and G. Wetzel. Sports league scheduling. In *Proceedings of the 1997 ILOG Optimization Suite International Users' Conference*, Paris, July 1997.

[12] K. Mehlhorn and S. Näher. *LEDA: a Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, November 1999.

[13] Mozart Consortium. The Mozart Programming System. Documentation and system available from `http://www.mozart-oz.org`, Programming Systems Lab, Saarbrücken, Swedish Institute of Computer Science, Stockholm, and Université catholique de Louvain, 1999.

[14] T. Müller. Constraint extensions tutorial. available from `http://www.mozart-oz.org`, Programming Systems Lab, Saarbrücken, Swedish Institute of Computer Science, Stockholm, and Université catholique de Louvain, 1999.

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

[16] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the AAAI* 12^{th} *National Conference on Artificial Intelligence*, pages 362–367. AAAI Press, 1994.

[17] J.-C. Régin. The symmetric alldiff constraint. In T. Dean, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 1, pages 420–425, Stockholm, Sweden, Aug. 1999. Morgan Kaufmann Publishers, San Mateo, CA.

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

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

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

[21] G. Smolka. The Oz programming model. In J. van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science 1000, pages 324–343. Springer-Verlag, Berlin, 1995.

[22] M. A. Trick. A schedule-then-break approach to sports timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III, Selected Papers of PATAT 2000*, Lecture Notes in Computer Science 2079, Konstanz, Germany, 2000. Springer-Verlag, Berlin.

[23] W. D. Wallis. *One-Factorizations*. Kluwer Academic Publishers, Dordrecht / Boston / London, 1997.

[24] J. P. Walser. *Domain-Independent Local Search for Linear Integer Optimization*. PhD thesis, Universität des Saarlandes, D 66041 Saarbrücken, Germany, Aug. 1998.