Constraint-based Round Robin Tournament Planning

Martin Henz
School of Computing
National University of Singapore
Singapore 117543
henz@comp.nus.edu.sg

Abstract

Sport tournament planning becomes a complex task in the presence of heterogeneous requirements from teams, media, fans and other parties. Existing approaches to sport tournament planning often rely on precomputed tournament schemes which may be too rigid to cater for these requirements. Existing work on sport tournaments suggests a separation of the planning process into three phases. In this work, it is shown that all three phases can be solved using finite-domain constraint programming. The design of Friar Tuck, a generic constraint-based round robin planning tool, is outlined. New numerical results on round robin tournaments obtained with Friar Tuck underline the potential of constraints over finite domains in this area.

1 Introduction

In a sport competition, n given teams play against each other over a period of time according to a certain scheme. The round robin scheme is popular in many team sports like football and basketball. It determines that every team t plays against every other team a fixed number of times r during the competition. If r is 1, the scheme is called single round robin, and if r is 2, it is called double round robin. The matches take place at one of the opponents' facilities. If a team uses its own facilities in a match, it is said to have a home match, otherwise it has an away match.

This paper focuses on temporally dense round robin tournaments (abbreviated as RR throughout the paper), in which the $rn(n-1)/2$ matches are distributed over a minimal number d of dates such that every team plays at most one match per date. If n is even, d is $r(n-1)$. A RR with an odd number of teams consists of rn dates in each of which $n-1$ teams play and one team does not. This team is said to have a bye.

Table 1 shows a RR for $n = 5$ and $r = 1$. The integral value in row t and column j tells the team against which team t plays in date j; the $+$ or $-$ symbol indicates that the match is a home match or an away match, respectively, for team t; and b indicates a bye. A similar notation is used by Schreuder [16] and Russell and Leung [14].

This basic setup can be refined by additional requirements in various ways. The following list contains common requirements in tournament planning practice.

- A match between two given teams may be fixed to a certain date. We shall refer to such constraints as FIX-AGAINST.

		dates				
		1	2	3	4	5
teams	1	+2	−4	b	+3	−5
	2	−1	+3	−5	b	+4
	3	+5	−2	+4	−1	b
	4	b	+1	−3	+5	−2
	5	−3	b	+2	−4	+1

Table 1: A Single Round Robin with 5 Teams

- A team may be required to play home, or to play away, or to have a bye at a certain date (FIX-HAB).
- The number of home and away games that a team plays at certain dates may be fixed (EXACT-HAB). For example, in Table 1, every team plays twice at home and twice bye, making for a balanced tournament.
- The number of home matches (or away matches or byes) that a team plays at certain dates may be limited (ATMOST-HAB). For example, in Table 1, every team plays at least once home in the first two dates.
- The number of home matches (or away matches or byes, or certain combinations of these) that a team plays in sequence may be limited (ATMOST-HAB-SEQUENCE). For example, in Table 1, no team plays more than one away match in a row. Furthermore, there are no occurrences of sequences of more than two away games or byes in a row.
- There may be a limit on the number of times in which a team plays in sequence against one of a specified set of (supposedly strong) teams (ATMOST-AGAINST-SEQUENCE). For example, in Table 1, no team plays in a row against team 1 and team 2.

Double round robin tournaments—particularly popular in many sports—often have to fulfill additional constraints like the following.

- If the first match between two teams t_1 and t_2 is is a home match for t_1, then the second match is a home match for t_2 and vice versa (EVEN-HAB).
- The tournament may be split into two halves, requiring that each team play against the other teams in the same order in both halves (MIRROR). Such a perfectly mirrored RR can be obtained from Table 1 by simply repeating it once.

Commercial products that support round robin timetabling typically have a fixed set of timetables up to some n preloaded. The timetabling process then consists of assigning the right teams to the rows of this timetable. For professional sports leagues, where fans, media and teams pose highly irregular constraints (for an interesting collection of such constraints, consider [12]) these timetables typically do not lead to acceptable solutions.

Existing systematic approaches to round robin planning [3, 1, 2, 16, 14, 15, 12] proceed in two main stages. In the first stage, feasible sets of n patterns are generated. A pattern is a sequence of d home/away/bye tokens that indicates home matches, away matches or byes for a team throughout the tournament. In the second stage, timetables are generated based on pattern sets from the first stage.

The combination of the following properties makes pattern sets attractive as an intermediate step towards timetables. On one hand, a pattern set characterizes a significant aspect of a timetable, namely in which way teams may alternate between home and away matches and byes. On the other hand, pattern sets are generic in a sense that they fix neither the teams that play according to its element patterns, nor the opponent teams. Often many timetables can be constructed from a given pattern set.

The author showed in [6] that using constraint programming for all phases of a complex problem provides a vast improvement over techniques reported in [12]. The aim of this paper is to explain the constraint programming approach to all phases of the timetabling process. To this goal, a number of common constraints are given and corresponding constraints for constraint programming models of the solution phases are outlined.

Whereas in this work, all phases are handled with constraint programming, Schaerf [15] provides a constraint-based solution only to the second phase. McAloon, Tretkoff and Wetzel [10] deal with a related problem in which the concept of home and away games is replaced by resources called periods.

Existing models of the subproblems are described in more detail in the next section. In Section 3, a brief introduction to the basic ideas of finite domain constraint programming is given before constraint programming models of pattern generation, pattern set generation and timetable generation are outlined in Sections 4, 5 and 6, respectively. Section 7 presents Friar Tuck, a constraint-based tool for RR planning in which these models are accessible through a graphical user interface which also allows the user to enter a variety of specialized constraints. Section 8 gives the results of computational experiments conducted with Friar Tuck.

2 Solution Phases

2.1 Patterns

A pattern is a sequence of values from the set of symbols $\{+, -, b\}$ that fulfills the given constraints. Each symbol indicates either a home match $(+)$, an away match $(-)$ or a bye (b) in the date that corresponds to the position of the symbol in the sequence. For example, the following pattern for $n = 5$ indicates an away match in date 4.

	dates				
	1	2	3	4	5
pattern	$-$	$+$	b	$-$	$+$

All possible patterns are generated that meet the required constraints among EXACT-HAB, ATMOST-HAB, ATMOST-HAB-SEQUENCE, EVEN-HAB and MIRROR.

2.2 Pattern Sets

From these patterns, sets of n patterns are generated that have the right number of occurrences of $+$, $-$ and b per date. For even n, there must be

$n/2$ occurrences of $+$ and $-$ each, and for odd n, there must be $(n-1)/2$ occurrences of $+$ and $-$ each and one b. Such sets are called pattern sets.

2.3 Timetables

This step generates complete timetables from a given pattern set. Two distinct ways to decompose the problem have been taken for this step.

The first decomposition—sketched by Cain [2]—assigns teams to pattern sets first. This assignment is subject to the constraints FIX-HAB. Finally, opponent teams are assigned to the entries of the pattern set such that the result is a RR that fulfills all other constraints.

For the second problem decomposition—described by Schreuder [16] and used by Nemhauser and Trick [12]—it is convenient to introduce the notion of a team placeholder. Each of n team placeholders stands for a team, but the assignment of teams to team placeholders is not fixed. The idea is that a timetable of team placeholders can stand for as many different actual timetables as there are such assignments. Now matches between team placeholders are assigned to the entries of the pattern sets such that the result is a RR of placeholders. Finally, every team is assigned to a different placeholder such that all relevant constraints are fulfilled.

3 Finite Domain Constraint Programming

For solving the various combinatorial search problems that arise in these phases, we use finite domain constraint programming. A central notion is the constraint store that contains basic constraints on the possible values of integer (finite domain) variables. During computation, these constraints become stronger as integer values are excluded from the domain of variables. More complex constraints cannot be expressed in general by the constraint store and thus are operationalized by propagators instead. Propagators observe the constraint store and amplify it whenever possible by adding basic constraints to it, implementing a certain notion of consistency. After exhaustive propagation, typically not all finite domain variables are fixed to unique values. The operational notion of search explores a tree of possible constraint stores by adding new constraints or propagators according to a search strategy. Before branching, exhaustive propagation is performed. The constraint programming approach is described in detail in [9].

4 Patterns

With the usual problem sizes and constraints in tournament planning, pattern generation is a straightforward task. Nemhauser and Trick [12] show that for the problem they study (double round robin for 9 teams), exhaustive enumeration of all patterns is computationally feasible. In that case, a declarative description of the constraints can be used for filtering out admissible patterns. For bigger n, a more algorithmic approach would be needed.

The advantage of constraint programming here is that a declarative formulation of the constraints on patterns can be used for bigger n. A straightforward model consists of 0/1 variables h_j, a_j (and b_j, if n odd), $1 \leq j \leq d$. A team that plays according to the pattern represented by these variables plays home (away, bye) at date j if and only if $h_j = 1$ ($a_j = 1$, $b_j = 1$). An obvious constraint is that for every j, the sum of h_j, a_j and b_j must be 1. The constraints EXACT-HAB, ATMOST-HAB, ATMOST-HAB-SEQUENCE, EVEN-HAB and MIRROR can be represented with the summation propagator and combinations of 0/1 propagators like conjunction and disjunction. For example, an ATMOST-HAB constraint that expresses that there must be at least one home game in the first two dates is represented by a propagator for the disequation $h_1 + h_2 \geq 1$. An ATMOST-HAB-SEQUENCE constraint that expresses that all sequences of home matches should be shorter than s is expressed by propagators corresponding to constraints of the form

$$h_1 + \cdots + h_s < s \wedge \cdots h_{n-s+1} + \cdots h_n < s$$

This approach to such sequence constraints in the context of finite domain constraint programming is present in [17]. It seems to be adequate with the problem sizes at hand; more powerful propagation algorithms are presented in [13].

In the presence of sequence constraints, the most effective search strategy seems to be to enumerate the h, a and b variables date-wise, i.e. in the order $h_1, a_1, b_1, h_2, a_2, b_2, \ldots, b_d$.

5 Pattern Sets

Graph-theoretical results cover the existence and generation of pattern sets with useful properties. A well-studied property of pattern sets concerns the presence of breaks, which are sequences of two home matches or two away matches in a row or separated only with a bye. Minimizing the number of breaks in the schedule is a specialization of the constraint ATMOST-HAB-SEQUENCE. De Werra [4] and Schreuder [16] show that for odd n, there are pattern sets with no breaks, and for even n, the minimal number of breaks is $n - 2$.

In the presence of irregular constraints on patterns, pattern set generation degenerates to a combinatorial search problem and can be formulated as a variant of graph coloring. Nemhauser and Trick [12] employ integer programming to solve this problem, using the following model. Let P be the set of all feasible patterns. The 0/1 variables for home, away and bye of a pattern in P with index i for date j are denoted by $h_{i,j}$, $a_{i,j}$ and $b_{i,j}$, respectively. For each pattern index i, a 0/1 variable x_i indicates whether this pattern occurs in the desired pattern set. For odd n, the constraints are as follows.

$$\sum_i h_{i,j} x_i = (n-1)/2, \sum_i a_{i,j} x_i = (n-1)/2, \sum_i b_{i,j} x_i = 1, \text{for all dates } j.$$

For even n the constraints are

$$\sum_i h_{i,j} x_i = n/2, \sum_i a_{i,j} x_i = n/2, \text{for all dates } j.$$

Nemhauser and Trick reformulate this model as an optimization problem and solve it with integer programming.

In [6], the author shows that constraint programming exhibits better performance for pattern set generation on the problem studied by Nemhauser and Trick. Constraint programming systems provide summation propagators for such linear equations.

Trick [18] suggests excluding pairs of patterns in which there is no possible meeting date for corresponding teams. More formally, for every pair i, i' s.t. $i \neq i'$, $\forall_j (h_{i,j} = 0 \vee a_{i',j} = 0)$ and $\forall_j (a_{i,j} = 0 \vee h_{i',j} = 0)$ the equation $x_i + x_{i'} \leq 1$ must hold.

Adding corresponding propagators is often crucial for a good performance. In the example reported in [6], an overall speedup of the scheduling process by a factor of three is achieved.

When it is not necessary to generate all feasible pattern sets, performance is greatly improved if not all feasible patterns are used, but a small number $s \geq n$ of patterns is randomly selected as input for pattern set generation. If no pattern set is found, the process is repeated with a new selection. As enumeration strategy, the most naive one, enumerating x in the order $x_1, x_2, \ldots, x_s$ was found to be most effective.

6 Timetables

The two models for timetable generation mentioned in Section 2.3 are described in detail here. Given is a pattern set in form of $n \times d$ matrices H, A and B of 0/1 variables whose entries $H_{i,j}$ ($A_{i,j}$, $B_{i,j}$) indicate home matches (away matches, byes) for pattern i in date j.

Furthermore, common to both approaches is a target timetable, represented by

- an $n \times d$ matrix α, whose entries variables $\alpha_{t,j}$ range over $0, \ldots, n$ and tell the opponent team against which team t plays in date j (0 stands for a bye), and
- matrices $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ of 0/1 variables whose entries $\mathcal{H}_{t,j}$ ($\mathcal{A}_{t,j}$, $\mathcal{B}_{t,j}$) tell if team t plays home (plays away, has a bye) in date j.

The constraints FIX-HAB and FIX-AGAINST are represented by basic constraints on variables of α, $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ in an obvious way. Constraints ATMOST-AGAINST-SEQUENCE can be expressed by sets of propagators over α similar to the constraints ATMOST-HAB-SEQUENCE in Section 4.

6.1 Cain's Method

Cain's method first assigns teams to patterns of the given pattern set. Accordingly, we introduce n finite domain variables p_i, $1 \leq i \leq n$. Each team t plays according to the pattern in row p_t of H, A and B. The constraints on the vector p and the matrix α are listed in Table 2.

These constraints can be easily implemented by propagators. Of particular importance is the last one, which links the given pattern set with the desired timetable, using the variables p_i as indices. It is expressible with the so-called element propagator [5]. The element propagator takes as arguments a finite domain variable k, a vector of integers v and a finite domain variable w.

$$element(k, v, w)$$

The semantics is $v_k = w$. Propagation can restrict the possible values for k, if a value in v is eliminated from w, and it can eliminate a number x from w,

1. For all dates j and teams t, at most one of the values $\alpha_{1,j}, \ldots, \alpha_{n,j}$ is t.
2. For all dates j and teams t_1 and t_2, $\alpha_{t_1,j} = t_2$ if and only if $\alpha_{t_2,j} = t_1$.
3. For all teams t_1 and t_2, the number of dates r for which $\alpha_{t_1,j} = t_2$ is r.
4. For all dates j and teams t_1, t_2, if $\alpha_{t_1,j} = t_2$, then
$$((\mathcal{H}_{t_1,j} = 1) \wedge (\mathcal{A}_{t_2,j} = 1)) \ \vee \ ((\mathcal{A}_{t_1,j} = 1) \wedge (\mathcal{H}_{t_2,j} = 1))$$
5. If n is odd, for all dates j and teams t, $\alpha_{t,j} = 0$ if and only if $\mathcal{B}_{t,j} = 1$.
6. For all dates j and all teams t : $H_{pt,j} = \mathcal{H}_{t,j}$, $A_{pt,j} = \mathcal{A}_{t,j}$; and $B_{pt,j} = \mathcal{B}_{t,j}$ for odd n.

Table 2: Constraints on α

if the last index that pointed to an x in v is eliminated from k. The following propagator implements one of the desired constraints, namely $H_{p_1,2} = \mathcal{H}_{1,2}$.

$$element(p_1, H_2, \mathcal{H}_{1,2})$$

where H_2 stands for the second column of matrix H.

The search strategy of Cain's method first assigns patterns to teams by enumerating the variables in p. Note that in the presence of FIX-HAB constraints, the just-mentioned element propagators can prune the search tree in this process. Next, the variables in α are enumerated. Here, the propagators corresponding to the first three constraints in 2 may be able to prune the remaining search tree. Apparently, the best strategy is to enumerate date-wise, i.e. in the order $\alpha_{1,1}, \alpha_{2,1}, \ldots, \alpha_{n,1}, \alpha_{1,2}, \ldots, \alpha_{n,d}$.

6.2 Schreuder's Method

Schreuder proposes to construct a timetable of team placeholders first. For this purpose, a matrix β similar to α is introduced. Its entries $\beta_{i,j}$ fix the team placeholder against which team placeholder i "plays" in date j. The first three constraints on α given in Table 2 must hold correspondingly for β. Date-wise enumeration of β constructs a timetable of team placeholders. The patterns described by H, A, and B are associated with the team placeholders in sequential order, i.e. team placeholder 1 "plays" at date j according to $H_{1,j}$, $A_{1,j}$ and $B_{1,j}$.

The last step assigns teams to team placeholders. For this assignment, we introduce n finite domain variables q_i, $1 \leq i \leq n$. The constraints that link the given pattern set represented by H, A and B, q and $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ are similar to the last constraint in Table 2. The relationship between α and β is a little more complicated. For all dates j and all teams t:

$$\beta_{qt,j} = q_{\alpha_{t,j}}$$

Consider the following instance of this constraint.

$$\beta_{q_1,2} = q_{\alpha_{1,2}}$$

This instance can be expressed by two element propagators, introducing an auxiliary finite domain variable *against*, ranging from 1 to n.

$$element(q_1, \beta_2, against), element(\alpha_{1,2}, q, against).$$

where β_2 stands for the second column of matrix β. The element propagator as described in [5] takes as second argument a list of integers. Initially the second argument in both element propagators are vectors of finite domain variables. A standard implementation of the element propagator as provided for example by the Mozart system [11] will wait until all components of the second argument vector are determined. Thus these propagators become able to prune the search tree at a relatively late stage. A more powerful version of the element propagator that is able to propagate even if the vector is not determined, would doubtless improve the performance of this implementation of Schreuder's approach to timetable generation.

7 Friar Tuck

Crucial for planning an irregular RR tournament is a careful design of the constraints. It seems that a basic understanding of the solution process is necessary. The user needs to fine-tune the constraints as well as the solution process in order come to a satisfactory timetable. Thus, a useful tool should provide interactive access to all phases and allow fine-tuning of the solution process and constraints.

For this purpose, the tool Friar Tuck was designed. Friar Tuck is available at `http://www.comp.nus.edu.sg/~henz/projects/FriarTuck/` in binary and source form. Friar Tuck is implemented using the finite domain constraint programming system Mozart 1.0 [11].

7.1 Structure

The user interface displays a graph representing the solution process as shown in Figure 1. Clicking on the nodes in this graph results in the display of a control panel for the corresponding phase on the right. In Figure 1, the user chose the control panel for timetable generation. Cain's method of assigning teams to patterns was selected.

7.2 Entry of Constraints

The interface for entering and editing tasks is displayed on the right after clicking on phase "Task" on the left. All constraints described in Section 1 can be entered by mouse-click, stored to and retrieved from files. The interface for the constraints FIX-HAB and FIX-AGAINST performs constraint propagation and consistency checking and thus supports semi-automatic timetable construction. Figure 2 shows the corresponding dialog window.

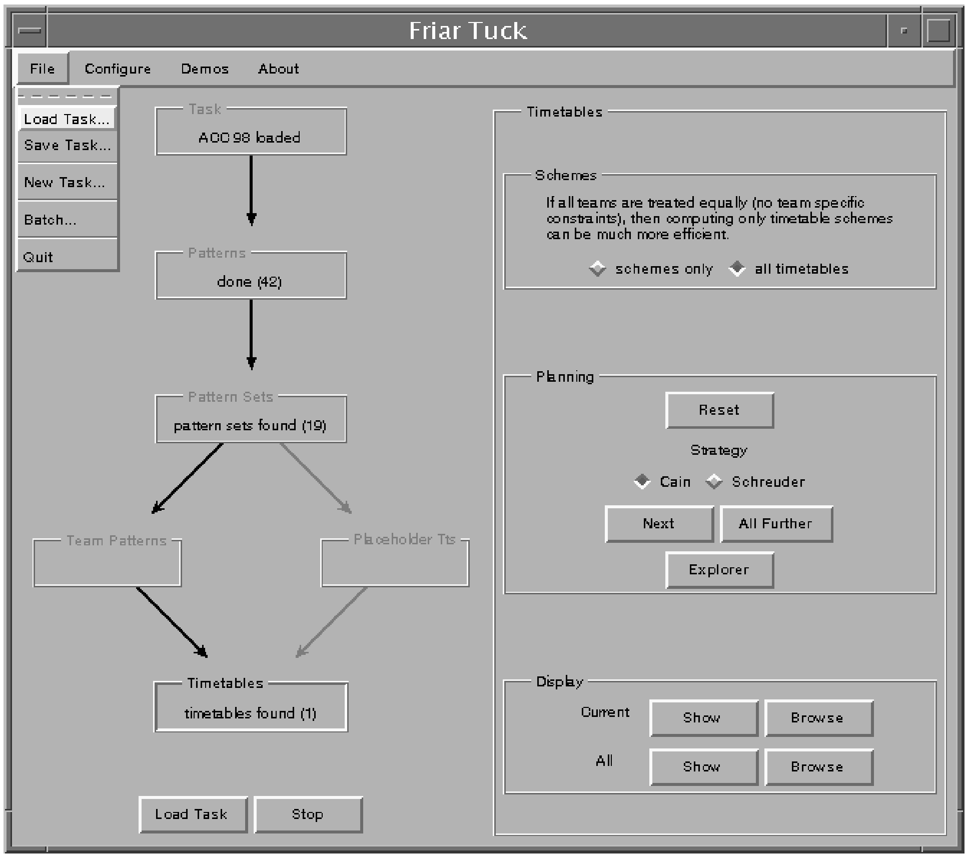

Figure 1: Friar Tuck (timetable interface)

7.3 Solution Process

After loading a task, asking for the “Next” timetable in the “Timetable” panel results in asking the pattern set phase for a pattern set. The pattern set phase in turn triggers the computation of all feasible patterns by the pattern phase. Based on a selection of these patterns, the pattern set phase computes the first pattern set which is sent to the timetable phase. The timetable phase now tries to construct a timetable based on this pattern set. If it succeeds, this timetable is provided to the user, and if it fails, it asks the pattern set phase for the next pattern set, etc.

In addition to this automatic mode, the user can interact with the first two phases separately and thus conduct experiments on pattern and pattern set generation. A batch mode supports automation of experiments such as the ones reported in the next section.

This interactive access is possible, since each of the phases' “Patterns”, “Pattern Sets”, and “Timetables” are implemented by instances of the class `Search.object`, which is provided by the Oz Standard Modules [7]. Search objects encapsulate constraint-based search and provide an interface to search from a more conventional programming environment. Newer versions of the C++ constraint programming library Ilog Solver [8] are based on the same idea of encapsulation.

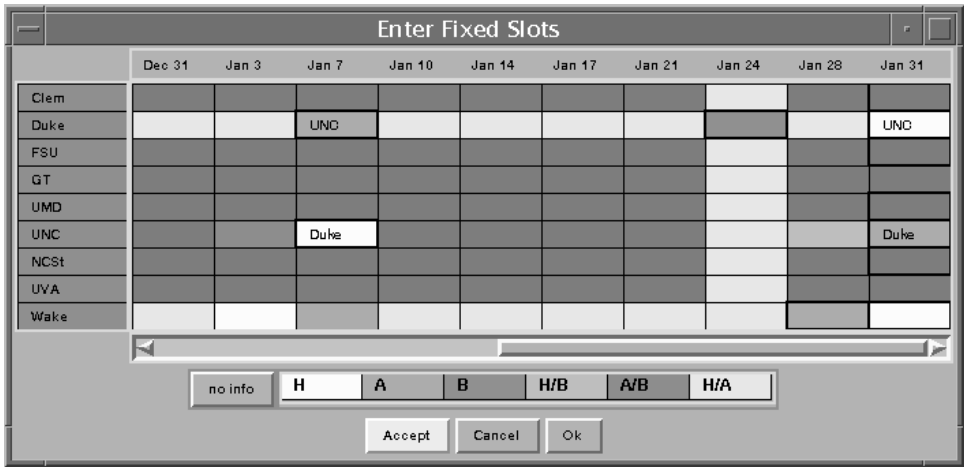

Figure 2: Friar Tuck (task interface)

		P		PS		T		Total
		S	R	S	R	S	R	R
# teams	2	2	0.05	1	0.05	8	0.50	0.60
	4	8	0.05	8	0.12	16	0.07	0.24
	6	32	0.06	3824	77.6	32768	2134.0	2211.0
	8	128	0.14	?	.	?	.	.
	3	12	0.05	8	0.50	8	0.10	0.65
	5	80	0.11	7776	246.2	6144	1654.5	1900.8
	7	448	2.31	?	.	?	.	.

Table 3: Single round robin, no restrictions; P: pattern generation; PS: pattern set generation; T: timetable generation; S: number of solutions; R: runtime in seconds.

8 Computational Results

In most examples that we encountered, Cain's method performs better than Schreuder's, including the ACC example reported in detail in [6]. All experiments reported in this section are run using Cain's method on a PC with a 223 MHz Pentium processor and 64 MBytes main memory running Mozart 1.0 under Windows 95.

8.1 No Restriction

In the unrestricted case, there are no closed formulae known to compute the number of pattern sets and timetables for a given n. The numbers grow very quickly with n as shown in Table 3. With no team-specific constraints, one given timetable can be used to generate $n!$ timetables which are identical up to team permutation. To generate only timetables that are different modulo team permutation, Friar Tuck allows the fixing of the vectors p (Cain's method) and q (Schreuder's method). In this section, only timetables that are different modulo team permutation are counted.

# teams	P		PS		T		Total
	S	R	S	R	S	R	R
2	2	0.05	1	0.01	1	0.01	0.07
4	6	0.05	1	0.11	2	0.02	0.18
6	10	0.06	1	0.21	8	1.76	2.03
8	14	0.07	1	3.8	320	17.1	20.9
10	18	0.07	1	5.0	61440	344.1	349.1

Table 4: Single round robin, minimal number of breaks for even n; P: pattern generation; PS: pattern set generation; T: timetable generation; S: number of solutions; R: runtime in seconds.

8.2 Minimal Number of Breaks

For odd n, the timetables with no breaks are unique (up to switching home and away throughout the timetable) and have a very regular structure. They can be computed by Friar Tuck up to about $n = 35$.

For even n, the situation is more interesting. De Werra [4] and Schreuder [16] show that the minimal number of breaks in this case is $n - 2$. Table 4 shows the number of possible pattern sets and timetables with minimal number of breaks for even $n \leq 10$. The number of timetables forms an interesting sequence, although a conjecture for a closed formula is not obvious.

9 Conclusion

Constraint programming proves to be an effective solution technique for various known models of subproblems of round robin tournament planning.

The tool Friar Tuck provides a user interface to the planning process, from problem definition to fully automatic generation of timetables. Furthermore, Friar Tuck supports experimentation on subproblems such as pattern and pattern set generation. Friar Tuck is an example for a hierarchical problem solver whose implementation benefits from an object-oriented encapsulation of constraint-based search.

New experimental results regarding the number of patterns, pattern sets and timetables with regular constraints were presented. Such numbers may provide new insights into the algebraic and graph-theoretical structure of the underlying problems.

A separate study provides evidence that constraint programming is competitive with integer programming for pattern set and timetable generation [6]. More such studies and experiments are needed to assess the relative merits of competing approaches in this field.

Concerning the constraint programming approach, round robin tournaments provide a rich application field for specialized propagation techniques.

- More powerful sequence constraints for pattern generation may be considered [13].
- Pattern set generation may benefit from a specialized global propagator.
- The implementation of Schreuder's method may benefit from a generalized element propagator that allows the argument vector to contain

finite domain variables instead of insisting on a vector of fixed integers.

Experience with challenging timetabling problems is needed to assess the relative efficiency of the described constraint-based and other approaches. Optimization criteria such as minimization of travel distance [3] should be supported.

References

[1] B. C. Ball and D. B. Webster. Optimal scheduling for even-numbered team athletic conferences. *AIIE Transactions*, 9:161–169, 1977.

[2] William O. Cain, Jr. The computer-assisted heuristic approach used to schedule the major league baseball clubs. In Shaul P. Ladany and Robert 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.

[3] Robert T. Campbell and Der-San Chen. A minimum distance basketball scheduling problem. In Shaul P. Ladany; under the general supervision of Donald G.Morrison Robert E. Machol, editor, *Management Science in Sports*, number 4 in Studies in Management Sciences, pages 15–25. North-Holland Publishing Co., Amsterdam, New York, Oxford, 1976. Special issue of the journal Management Science.

[4] D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

[5] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In Yves Kodratoff, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 290–295, Munich, Germany, August 1988. Pitman Publishers, London.

[6] Martin Henz. Scheduling a major college basketball conference—revisited. *Operations Research*, 2000. to appear.

[7] Martin Henz, Martin Müller, Christian Schulte, and Jörg Würtz. The Oz standard modules. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1997.

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

[9] Kim Marriott and Peter J. Stuckey. *Programming with Constraints*. The MIT Press, Cambridge, MA, 1998.

[10] Ken McAloon, Carol Tretkoff, and Gerhard Wetzel. Sports league scheduling. In *Proceedings of the 1997 ILOG Optimization Suite International Users' Conference*, Paris, July 1997.

[11] 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.

[12] George L. Nemhauser and Michael A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46(1), 1998.

[13] Jean-Charles Régin and Jean-François Puget. A filtering algorithm for global sequencing constraints. In Gert Smolka, editor, *Principles and Practice of Constraint Programming—CP97, Proceedings of the Third International Conference*, Lecture Notes in Computer Science 1330, pages 32–46, Schloss Hagenberg, Linz, Austria, October/November 1997. Springer-Verlag, Berlin.

[14] Robert A. Russell and Janny M. Y. Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42(4):614–625, 1994.

[15] Andrea Schaerf. Scheduling sport tournaments using constraint logic programming. In *Proceedings of the European Conference on Artificial Intelligence*, pages 634–639, Budapest, Hungary, 1996. John Wiley & Sons.

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

[17] Barbara Smith. Succeed-first or fail-first: A case study in variable and value ordering. In *Proceedings of the 1996 ILOG Solver and ILOG Scheduler International Users' Conference*, Paris, July 1996.

[18] Michael Trick. Modifications to the problem description of "scheduling a major college basketball conference". WWW at `http://mat.gsia.cmu.edu/acc_mod.html`, 1998.