Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 1895–1906
computers & operations research
www.elsevier.com/locate/cor

A constraint programming approach to the multiple-venue, sport-scheduling problem

Robert A. Russell*, Timothy L. Urban

Operations Management, The University of Tulsa, 600 South College Avenue, Tulsa, Ok 74104, USA

Available online 5 November 2004

Abstract

In this paper, we consider the problem of scheduling sports competitions over several venues which are not associated with any of the competitors. A two-phase, constraint programming approach is developed, first identifying a solution that designates the participants and schedules each of the competitions, then assigning each competitor as the "home" or the "away" team. Computational experiments are conducted and the results are compared with an integer goal programming approach. The constraint programming approach achieves optimal solutions for problems with up to sixteen teams, and near-optimal solutions for problems with up to thirty teams.

Keywords: Scheduling; Sports management; Constraint programming; Timetabling

1. Introduction

The subject of scheduling sport competitions has received a great deal of interest in the operations research literature during the past quarter of a century. It is an interesting problem to the operations research community due to its wide application (not only for professional or collegiate sports, but also the multitude of recreation league offerings) as well as its straightforward, but computationally burdensome, presentation. These types of problems may take on a variety of structures, such as being temporally relaxed (more time periods available than games needed to be played) or temporally constrained. There may be an explicit objective, such as minimizing travel requirements, or the goal may simply be identifying a feasible schedule, particularly when there are a large number of side constraints that must be satisfied.

* Corresponding author. Tel.: +918 631 3135; fax: +918 631 2037.
E-mail addresses: rrussell@utulsa.edu (R.A. Russell), urbantl@utulsa.edu (T.L. Urban).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.09.029

Moreover, the proliferation of sport-scheduling software further emphasizes the demand as well as the difficulty of solving these types of problems.

The sports-scheduling research literature has focused on league scheduling and round-robin tournament play. Graph theory was used by de Werra [1,2] to develop schedules with desirable properties, such as the minimum number of breaks in the sequences of home and away games. Schreuder [3] constructed a timetable for the Dutch professional soccer leagues that minimizes the number of schedule breaks. Russell and Leung [4] developed a schedule for the Texas Baseball League that satisfied stadium availability and various timing restrictions with the objective of minimizing travel costs. Nemhauser and Trick [5] created a schedule for the Atlantic Coast Conference men's basketball season, taking into consideration numerous conflicting requirements and preferences. Other applications that have been studied include professional hockey [6,7], basketball [8,9], and baseball [10] leagues, as well as cricket [11–13] and tennis [14] tournaments.

Urban and Russell [15] presented a distinctive sport-scheduling problem in which the competitions take place on several venues that are not associated with one of the competitors. This extended the traditional sport-scheduling problem in that the selection of the venue for each competition now becomes part of the decision-making process and imposes additional constraints on the schedule. An integer goal programming model was used to solve the problem; however, this approach could not identify optimal solutions to problems involving more than ten teams.

Recently, constraint programming has been proposed as an appropriate means of solving highly-constrained, combinatorial problems such as timetabling and scheduling (see Brailsford et al. [16] for a review of constraint satisfaction problems). Concerning sport scheduling in particular, Schaerf [17] presented a two-step approach, first using known graph-theoretic results to generate a tournament pattern, then using constraint programming to assign the teams to the pattern. Trick [18] used constraint programming to find a schedule of the teams (ignoring home and away patterns) in the tournament, then used an integer programming model to appropriately assign the home and away teams. Henz [19] presented a constraint programming approach to the college basketball scheduling problem introduced by Nemhauser and Trick [5]; he found that problems previously requiring a day to solve, could now be solved in less than one minute using this approach. Henz et al. [20] have recently analyzed the effect of specific constraint programming propagation algorithms on the efficient solution of the sport-scheduling problem.

The purpose of this paper is to evaluate constraint programming as a solution methodology for the multiple-venue, sport-scheduling problem. In Section 2, we present a formal definition of the problem as well as a mathematical programming formulation that has been used to solve the problem. We then present a constraint programming approach in Section 3, finding a solution that satisfies all of the goals for moderately-sized problems, and all but the least important goal for larger problems. The results are compared to those obtained from the mathematical programming approach. Finally, in Section 4, we conclude the paper.

2. The multiple-venue, sport-scheduling problem

The multiple-venue, sport-scheduling problem (MVSSP) pertains to situations in which the scheduling of sports competitions occurs over several venues that are not associated with any of the competitors. This situation frequently arises in recreation-league play as well as in tournament play in which

round-robin schedules (i.e., every competitor plays every other competitor) are necessary. The distinguishing characteristic of this type of sport-scheduling problem is that the scheduling of the venue becomes part of the overall scheduling process.

The problem was introduced by Urban and Russell [15] in response to a major-college football coach, who was scheduling intra-squad team competitions on various drill stations for spring training. The squad had been subdivided into eight teams, each of which was to compete simultaneously on four different drill exercises. The coach had hoped to schedule the competitions while satisfying the following conditions, listed in order of increasing importance:

1. Each team participates at each station twice.
2. Each team competes against each of the other teams at least once.
3. Any two teams should not compete against each other twice at the same station.

Urban and Russell [15] then formally developed the MVSSP model and provided an integer goal program to solve it. The model was presented as follows: there are a number of competitors (teams, players, etc.: $i = 1, 2, \ldots, n$) to be paired for a series of competitions. Each competition occurs on one of the available venues (site, court, stadium, drill station, etc.: $s = 1, 2, \ldots, n/2$) during a period of time ($t = 1, 2, \ldots, n$). Let p represent a competition such that $p = (i, j)$ is a competition pairing competitors i and j, and let P_i represent the set of competitions in which competitor i participates. The decision variables are defined as:

$$x_{pst} = \begin{cases} 1 & \text{if competition } p \text{ is assigned to venue } s \text{ during period } t, \\ 0 & \text{otherwise.} \end{cases}$$

Similar to timetabling problems, the objective of the MVSSP is to identify a feasible schedule that satisfies a number of constraints. Goal programming is an appropriate solution methodology, since some of the constraints must be considered as hard constraints—e.g., only one competition can take place on a given venue at a particular time period—while other conditions can be treated as soft constraints—e.g., each competitor is expected to play an equal number of competitions on each of the venues. The integer goal program can then be formulated in the following manner [15]:

$$\text{Minimize } Z = \sum_i \sum_s \lambda_1 d^-_{1is} + \sum_p \lambda_2 d^-_{2p} + \sum_p \sum_s \lambda_3 d^+_{3ps}. \tag{1}$$

$$\text{Subject to:} \quad \sum_p x_{pst} = 1, \quad \forall s, t, \tag{2}$$

$$\sum_{p \in P_i} \sum_s x_{pst} = 1, \quad \forall i, t, \tag{3}$$

$$\sum_{p \in P_i} \sum_t x_{pst} + d^-_{1is} \geqslant 2, \quad \forall i, s, \tag{4}$$

$$\sum_s \sum_t x_{pst} + d^-_{2p} \geqslant 1, \quad \forall p, \tag{5}$$

$$\sum_t x_{pst} - d^+_{3ps} \leqslant 1, \quad \forall p, s, \tag{6}$$

$$x_{pst} = \{0, 1\}, \quad \forall p, s, t, \tag{7}$$

where λ_k are the weights (or priorities) assigned to each of the sets of deviation variables, d_k^-, d_k^+. Constraint sets (2) and (3) are hard constraints, ensuring that only one competition will take place on a particular venue each time period and that each competitor will compete at only one venue each time period. The remainder of the constraints are treated as soft constraints, such that each competitor will compete at each venue twice (constraint set 4), that each competitor will compete against every other competitor at least once (constraint set 5), and that any competition (pair of competitors) should not compete at the same venue more than once (constraint set 6).

The model, as presented, was able to find solutions satisfying all constraints for eight- and ten-competitor problems. Solutions for up to sixteen-team problems were also presented; however, the integer goal program was unable to find a solution that satisfied all constraints within 14 CPU hours. The reported solutions only violated the least important constraint (i.e., each team competes at each venue twice). To improve on the solutions provided by the goal program, a "quick-and-dirty" post-processing procedure was used.

For many sport-scheduling applications, such as recreation-league play or round-robin tournaments, a "home" team is necessary. As an example, the home team gets the last at bat in baseball games. To formulate this situation, define $p=(i, j)$ as an *ordered* pair representing a competition pairing competitors i and j, where i is the home team, and let $P_i^+(P_i^-)$ represent those competitions in which competitor i is the home (away) team. To ensure each competitor is the home team about as often as it is the away team, we formulate a set of constraints to be included in the integer goal program as follows:

$$\sum_{p\in P_i^+}\sum_t x_{pst} - \sum_{p\in P_i^-}\sum_t x_{pst} - d_{4i}^+ \leqslant 0, \quad \forall i, s. \tag{8}$$

However, the addition of this constraint set to the integer goal programming model increases the computation time substantially. Therefore, the balancing of home-away team assignments would be best accomplished with a heuristic or with the second constraint programming model described in the following section.

Due to the inability of the mathematical programming formulation to optimally solve problems with more than ten competitors—or to provide reasonable solutions to problems with more than sixteen competitors—we now turn our attention to the use of constraint programming for the MVSSP.

3. Solving the MVSSP using constraint programming

The MVSSP can be formulated as a constraint satisfaction problem (CSP). A CSP consists of a set of variables with a finite domain of possible values and a set of constraints. Each constraint affects a subset of the variables and limits the values that the variables can simultaneously assume. A solution to a CSP determines values for the variables, which satisfy all the constraints. A pure CSP has no objective function; however, some constraint programming languages such as OPL can accommodate optimization, which permits the maximization or minimization of an objective function.

Our approach to solving the MVSSP will consist of two phases. In phase one, we assign teams to pairings (games) and pairings to venues during the planning horizon of time periods. In the second phase, we assign the teams in each pairing to home or away positions in order to balance the number of home or away competitions. This type of approach was suggested by Trick [18] in those sport-timetabling applications where the requirements on the schedule do not necessarily involve the home-away patterns.

To formulate the CSP for the multiple-venue, sports-scheduling problem, we first describe the nature of the variables and constraints and then present an OPL model to solve the CSP. The model is an extension of Van Hentenryck's "simple" OPL model for the round-robin, sport-scheduling problem [21]. An n-period MVSSP is comprised of a set of n teams $\{1, 2, \ldots, n\}$, $n/2$ venues $\{1, 2, \ldots, n/2\}$, n time periods $\{1, 2, \ldots, n\}$, and 2 slots $\{one, two\}$. The slots are used to represent the fact that each competition requires two teams or opponents. Let the variable $team(v, p, s)$ equal the value (i.e., $1, 2, \ldots, n$) of the team assigned to play at venue v, in time period p, and in slot position s.

To reduce symmetries in the MVSSP solution space, define each team pairing (i, j) to be restricted to pairings such that $i < j$. This reduces solution time and enables the solution of larger problems. The second phase of our model will be used to restore the balance between home and away competitions.

For each competition (i, j), a unique game value can be defined as $n(i-1)+j$. The game value is used to ensure that each team will play every other team. The CSP for the MVSSP consists of the following constraints:

1. The first constraint ensures that all n teams play in each period.

   For each period $p = 1, 2, \ldots, n$,
   $team\ (v, p, s)$ are all different $\quad \forall v = 1, 2, \ldots, n/2$ and $s = \{\text{one, two}\}$.

2. The following constraint stipulates that each team competes at each venue twice.

   For each venue $v = 1, 2, \ldots, n/2$ and team $t = 1, 2, \ldots, n$,
   cardinality $\{team(v, p, s) \mid team(v, p, s) = t\} = 2 \qquad \forall p = 1, 2, \ldots, n$ and $s = \{\text{one, two}\}$.

3. The following constraint forces each team to play every other team within the first $n-1$ time periods. This requirement is more restrictive then necessary, but is utilized to prune the search space much more efficiently.

   $game\ (v, p)$ are all different $\quad \forall v = 1, 2, \ldots, n/2$ and $p = 1, 2, \ldots, n-1$.

4. This constraint ensures that no pair of teams competes more than once at the same venue over the n time periods.

   For each venue $v = 1, 2, \ldots, n/2$,
   $game\ (v, p)$ are all different $\quad \forall p = 1, 2, \ldots, n$.

The OPL model implementation for the MVSSP is shown in Fig. 1. We introduce the notion of venues and add three constraints required to solve the MVSSP as opposed to the simpler round-robin, sports-scheduling problem.

The OPL code in Fig. 1 initializes the range of values for the parameters and variables in the problem. The arrays "occur" and "values" are assigned values and are used to enforce cardinality constraints used in the model. Two variables, team and game, are defined over the relevant venues, periods, and slots. The index EPeriods is defined over n periods and is a one-period extension of Periods, which is defined over $n-1$ periods. A typical round robin set of competitions with n teams will involve $n-1$ periods of play; the MVSSP problem involves n periods.

The team and game variables must be linked so that the unique game value is associated with the correct pair of teams. The three-field record, Plays, specifies the set of legal games. For a 10 team problem, the

```
int nbTeams = 16;
range Teams 1..nbTeams;
range Venues 1..nbTeams/2;
range Periods 1..nbTeams-1;
range EPeriods 1..nbTeams;
range Games 1..nbTeams*nbTeams;
enum Slots = {one,two};

int occur[t in Teams] = 2;
int values[t in Teams] = t;

var Teams team[Venues,EPeriods,Slots];
var Games game[Venues,Periods];

struct Play {int a;int b;int g;};
{Play} Plays = {<i,j,(i-1)*nbTeams+j>|ordered i,j in Teams};
predicate link(int a,int b,int g) in Plays;

solve {
   forall(p in EPeriods)
      alldifferent(all(v in Venues & s in Slots) team[v,p,s]);
   forall(v in Venues)
      cardinality(occur,values,all(p in EPeriods & s in Slots) team[v,p,s]);
   alldifferent(game);
   forall(v in Venues & p in Periods)
      game[v,p] <> (team[v,nbTeams,one]-1)*nbTeams+team[v,nbTeams,two];
   forall(v in Venues) {
      team[v,nbTeams,two] > team[v,nbTeams,one];
      team[v,1,one] = 2*v-1; team[v,1,two] = 2*v;
      };
   forall(v in Venues & p in Periods)
      link(team[v,p,one],team[v,p,two],game[v,p]);
};
search {
   generate(game);
};
```

Fig. 1. OPL model for the multiple-venue, sport-scheduling problem.

pairing of teams 5 and 7, for example, would result in a game value of $10 \times (5 - 1) + 7 = 47$. Thus, one of the tuples in Play would be $\langle 5, 7, 47\rangle$. The instruction, predicate, is a symbolic constraint used to enforce the relation between the game and team variables.

The first alldifferent statement is a global constraint that requires all teams playing in period p to be different. For a given period, p, the aggregate operator "all" is applied over the $n/2$ venues and 2 slots. The cardinality constraint specifies that for each venue a team plays exactly twice during the n time periods. The declaration of alldifferent(game) stipulates that games at all venues during the first $n - 1$ time periods must be different. This global constraint is stronger than the MVSSP problem requires, since it is only necessary for all teams to have played each other once during all n periods. However, this global constraint was necessary to achieve "reasonable" computation times and in our experiments was found to be the most efficient way to require that each team play every other team.

The constraint:

```
forall(v in Venues & p in Periods)
   game[v,p] <> (team[v,nbTeams,one]-1)*nbTeams+team[v,nbTeams,two];
```

specifies that for each venue and periods 1 to $n-1$, the game value must be different than the computed game value of the teams playing in period n. Thus, all games will involve a different pair of teams at each venue.

The two constraints:

```
forall(v in Venues){
    team[v,nbTeams,two] > team[v,nbTeams,one];
    team[v,1,one] =2*v-1; team[v,1,two] =2*v;
    };
```

preserve the ordering of teams in period n, (which is outside the domain of game) such that the lower numbered team always appears in the first slot. The second constraint listed further reduces symmetries in the solution space by specifying the teams to play in period 1. The last constraint, link, specifies that $game[v, p]$ consists of $team[v, p, one]$ and $team[v, p, two]$.

Constraint programming algorithms typically employ a search procedure to enumerate assignments of values to variables. When the value of a variable is fixed, constraint propagation is applied to restrict the domains of other variables whose values are not currently fixed. OPL has the capability of using different propagation algorithms for various constraints. Arc consistency is a fundamental constraint satisfaction technique from artificial intelligence. Arc consistency algorithms can be applied as a preprocessing stage thereby reducing the size of some variables' domains and making the problem easier to solve. OPL enforces arc consistency on the cardinality global constraint, the predicate constraint, and the symbolic link constraint [21]. OPL will enforce arc consistency for the alldifferent global constraint if the keyword "onDomain" is specified; for example:

```
alldifferent(game) onDomain;
```

however, this did not improve the solution time for this particular application.

The default search strategy in OPL is the heuristic known as the "first fail principle". This principle chooses the variable with the fewest possible values first. The first fail principle is used in the search specifications generate(game) (as shown in Fig. 1), which generates values for all the games and subsequently to the team variable by constraint propagation.

The OPL model shown in Fig. 1 generates games to be played at each venue over the n time periods. However, the team requirement that $i < j$ means that team 1 will be assigned to slot one n times and team n will be assigned to slot two n times. To generate schedules that balance home and away games, we create a second OPL model. This second model reads the game pairings created by the first model and creates a balanced home-and-away schedule. The home-away model is shown in Fig. 2. The first constraint retains the assignment of team pairings to venues in each period and reassigns the home-away position. The first cardinality constraint requires that each team is a home team once at each venue. The second cardinality constraint specifies that each team will be a home team for exactly half of the n games played. The last constraint specifies that no game with the same pair of teams and home-away assignment in period n has occurred at any venue during periods 1 to $n-1$. This model was able to identify balanced home-away assignments for each problem in less than 0.1 s.

Table 1 compares the results of using integer goal programming and constraint programming to solve the MVSSP. The computation times reflect the time required to obtain a feasible assignment of team pairings to venues and time periods, and does not include the time required to obtain balanced play with respect to home and away. Urban and Russell [15] have shown that for problems with the number of

```
int nbTeams = ...;
range Teams 1..nbTeams;
range Venues 1..nbTeams/2;
range EPeriods 1..nbTeams;
enum Slots = {home,away};

int occur1[t in Teams] = 1;
int occur2[t in Teams] = nbTeams/2;
int values[t in Teams] = t;

int preSol[Venues,EPeriods,Slots] = ...;
{int} choose[v in Venues, p in EPeriods] = { preSol[v,p,s] | s in Slots };

var Teams team[Venues,EPeriods,Slots];

solve {
   forall(v in Venues & p in EPeriods) {
      team[v,p,home] in choose[v,p];
      team[v,p,away] in choose[v,p];
      team[v,p,home] <> team[v,p,away];
   };
   forall(v in Venues)
      cardinality(occur1,values,all(p in EPeriods) team[v,p,home]);
   cardinality(occur2,values,all(v in Venues & p in EPeriods)
      team[v,p,home]);
   forall(v,w in Venues & p in EPeriods : p < nbTeams)
      ((team[v,p,home]=team[w,nbTeams,home]) &
         (team[v,p,away]=team[w,nbTeams,away])) = 0;
};
```

Fig. 2. OPL model for balanced home-away assignments.

Table 1
CPU requirements (in seconds)

Problem size (N)	Mathematical[a] programming	Constraint[b] programming
8	2.1	0.2
10	86.6	0.2
12	*	3.0
14	*	14.9
16	*	13,416.9

*Solution not identified within 14 h (50,400 s) on a 2.0-GHz Pentium 4 computer.
[a]Solved using AMPL/CPLEX 6.5.
[b]Solved using OPL Studio 3.6.1.

teams of size $n = 4$ and $n = 6$, it is not possible to satisfy all constraints. Integer goal programming was able to find optimal solutions for $n = 8$ and $n = 10$, but not for problems of size $n = 12$ or larger. The CSP model was able to solve problems up to size $n = 16$ which are much larger combinatorial problems. Interestingly, the CSP model reported finding no feasible solution for problems of size $n = 18$ and $n = 20$. We do not know whether solutions exist or whether the specification of the constraint alldifferent(game)

is too restrictive for these larger problems. However, if we relax the alldifferent(game) constraint, the computational effort increases dramatically. We hypothesize that feasible solutions do exist for $n \geqslant 18$ based simply on the fact that they do exist for $n = 8,10,12,14$, and 16.

In the appendix, the optimal solutions for the multiple-venue, sport-scheduling problems with $n = 12$, 14, and 16 are shown. These schedules also include the balanced home and away assignments with the home team being listed first.

We should note that Van Hentenryck [21] also developed a more efficient "round-robin model" for the sports-scheduling problem. This model is capable of solving larger problems more efficiently than the "simple model". Symmetries in the search space are reduced by specifying the pairings for each period. However, when constraints were added to solve the MVSSP as defined in this paper, the more efficient model failed to achieve a feasible solution for problems of any size; the game specifications prove to be too restrictive.

Shown in the appendix is a near-optimal solution for $n = 20$. Executing Van Hentenryck's round-robin model without adding constraints to satisfy the MVSSP, yields a schedule that violates constraint set (6) in the integer goal program. In period n, at least one game was repeated at the same venue, thereby violating the most critical constraint. A manual switching of one team in any of the repeated games with another team eliminates the violation of constraint set (6) and creates a violation of the less important constraint set (4) yielding an objective function underachievement value of only 2. This was accomplished for all even-numbered team totals from $n = 18$ to $n = 30$. The computation time was quite small ranging from 0.3 to 143 s on a 2.0-GHz Pentium 4 computer. Thus, it is possible to obtain near-optimal solutions to large problems in small amounts of time. It is interesting to note that the mathematical programming approach of integer goal programming could only achieve objective function values of 8 for smaller problems of size $n = 14$, and $n = 16$. Furthermore, the CPU time required was orders of magnitude larger.

4. Conclusion

We have presented a constraint programming approach to solving the multiple-venue, sports-scheduling problem. The two-phase approach involves first solving a constraint satisfaction problem (CSP) to determine a feasible solution that pairs teams and schedules the competitions at the appropriate venues over the course of the planning horizon. The second phase uses another CSP to designate home and away teams for each competition so that balanced home-away assignments are achieved.

Compared to our previous integer goal programming approach, the constraint programming approach solves significantly larger problems in less time. Sport scheduling involving up to 16 teams can be solved optimally using the two CSP models. Additionally, near-optimal solutions to larger problems can be generated with a modification of Van Hentenryck's round-robin model.

Although constraint programming was more effective than integer programming in this application, its success depends on formulations that facilitate a significant amount of constraint propagation. That is, variable instantiation should facilitate a domain reduction in other variables in the model. If significant domain reduction is not achieved in the constraint programming formulation, a mathematical programming approach might prove to be more effective.

Finally, for constraint programming to be effectively applied to real-world sports scheduling situations, it must become a user-friendly component of an overall decision support system. Although constraint

programming software has improved considerably in the decade since Schreuder [22] noted there "is still a long way to go" in developing such a system, a thorough understanding of the particular software package is still necessary to solve a specific problem. Commercial packages such as OPL Studio can help with modeling the application, as well as integrating mathematical programming and constraint programming components, but still require a level of understanding beyond that of a novice.

Appendix. Solutions to the multiple-venue, sport-scheduling problem (see Table 2).

Table 2
Solutions to the multiple-venue sport-scheduling problem

Period	$n = 12$ Venue					
	1	2	3	4	5	6
1	(1,2)	(4,3)	(6,5)	(7,8)	(9,10)	(12,11)
2	(3,1)	(2,4)	(5,7)	(8,6)	(11,9)	(10,12)
3	(2,3)	(1,5)	(7,11)	(6,10)	(8,12)	(9,4)
4	(4,5)	(6,11)	(9,3)	(12,1)	(2,8)	(7,10)
5	(6,4)	(3,10)	(1,8)	(5,11)	(12,7)	(2,9)
6	(5,8)	(9,6)	(2,10)	(4,12)	(7,3)	(11,1)
7	(7,6)	(12,2)	(10,1)	(9,5)	(4,11)	(3,8)
8	(9,7)	(11,8)	(3,12)	(10,4)	(5,2)	(1,6)
9	(8,10)	(5,12)	(11,2)	(1,9)	(3,6)	(4,7)
10	(12,9)	(7,1)	(8,4)	(11,3)	(10,5)	(6,2)
11	(10,11)	(8,9)	(12,6)	(2,7)	(1,4)	(5,3)
12	(11,12)	(10,7)	(4,9)	(3,2)	(6,1)	(8,5)

Period	$n = 14$						
	1	2	3	4	5	6	7
1	(1,2)	(3,4)	(5,6)	(8,7)	(9,10)	(12,11)	(13,14)
2	(3,1)	(4,2)	(7,5)	(6,8)	(11,9)	(13,10)	(14,12)
3	(2,3)	(1,5)	(9,14)	(7,4)	(10,6)	(11,13)	(12,8)
4	(4,5)	(11,3)	(6,13)	(10,14)	(2,8)	(7,12)	(1,9)
5	(6,4)	(8,10)	(12,2)	(1,11)	(14,5)	(3,9)	(7,13)
6	(5,8)	(9,12)	(10,3)	(4,13)	(1,7)	(2,14)	(6,11)
7	(7,6)	(2,11)	(14,1)	(3,12)	(8,13)	(10,5)	(9,4)
8	(9,7)	(13,1)	(2,10)	(14,6)	(12,4)	(8,3)	(11,5)
9	(8,9)	(7,14)	(4,11)	(13,3)	(5,12)	(6,2)	(10,1)
10	(10,11)	(5,13)	(8,4)	(12,1)	(3,14)	(9,6)	(2,7)
11	(12,10)	(14,8)	(13,9)	(2,5)	(7,11)	(4,1)	(3,6)
12	(11,14)	(12,6)	(3,7)	(5,9)	(13,2)	(1,8)	(4,10)
13	(13,12)	(10,7)	(11,8)	(9,2)	(6,1)	(14,4)	(5,3)
14	(14,13)	(6,9)	(1,12)	(11,10)	(4,3)	(5,7)	(8,2)

Table 2 (*cotinued*)

Period	$n = 16$							
	Venue							
	1	2	3	4	5	6	7	8
1	(1,2)	(3,4)	(5,6)	(8,7)	(9,10)	(12,11)	(13,14)	(16,15)
2	(3,1)	(4,2)	(7,5)	(6,8)	(11,9)	(10,12)	(15,13)	(14,16)
3	(2,3)	(5,1)	(16,13)	(7,4)	(10,6)	(11,8)	(14,9)	(15,12)
4	(4,5)	(1,11)	(15,7)	(9,12)	(14,8)	(13,6)	(16,2)	(10,3)
5	(6,4)	(15,10)	(12,8)	(5,13)	(1,14)	(9,2)	(3,16)	(11,7)
6	(5,8)	(7,14)	(1,10)	(13,2)	(15,4)	(16,9)	(11,3)	(12,6)
7	(7,6)	(8,13)	(14,2)	(10,16)	(12,1)	(3,15)	(9,4)	(5,11)
8	(9,7)	(14,3)	(8,4)	(16,1)	(13,12)	(15,5)	(6,11)	(2,10)
9	(8,9)	(16,5)	(10,14)	(15,11)	(7,3)	(1,13)	(4,12)	(6,2)
10	(10,11)	(12,7)	(13,3)	(14,5)	(6,15)	(4,16)	(2,8)	(1,9)
11	(13,10)	(2,15)	(4,11)	(12,3)	(8,16)	(6,14)	(7,1)	(9,5)
12	(11,13)	(9,6)	(2,12)	(1,15)	(16,7)	(14,4)	(5,10)	(3,8)
13	(12,14)	(13,9)	(11,16)	(3,6)	(5,2)	(7,10)	(8,15)	(4,1)
14	(16,12)	(10,8)	(9,15)	(11,14)	(3,5)	(2,7)	(1,6)	(13,4)
15	(14,15)	(6,16)	(3,9)	(4,10)	(2,11)	(8,1)	(12,5)	(7,13)
16	(15,16)	(11,12)	(6,1)	(2,9)	(4,13)	(5,3)	(10,7)	(8,14)

Period	$n = 20$									
	1	2	3	4	5	6	7	8	9	10
1	(2,1)	(3,20)	(4,19)	(18,5)	(17,6)	(16,7)	(8,15)	(14,9)	(10,13)	(11,12)
2	(1,3)	(2,4)	(20,5)	(6,19)	(18,7)	(8,17)	(9,16)	(15,10)	(14,11)	(12,13)
3	(6,2)	(4,1)	(5,3)	(20,7)	(19,8)	(9,18)	(17,10)	(11,16)	(12,15)	(13,14)
4	(3,7)	(1,5)	(8,2)	(4,6)	(9,20)	(10,19)	(11,18)	(17,12)	(13,16)	(14,15)
5	(4,8)	(10,2)	(1,6)	(9,3)	(7,5)	(11,20)	(19,12)	(13,18)	(17,14)	(15,16)
6	(10,4)	(11,3)	(7,1)	(12,2)	(5,9)	(19,14)	(16,17)	(20,13)	(15,18)	(8,6)
7	(5,11)	(6,10)	(2,14)	(8,1)	(13,3)	(12,4)	(15,20)	(16,19)	(9,7)	(17,18)
8	(13,5)	(12,6)	(3,15)	(1,9)	(16,2)	(7,11)	(10,8)	(4,14)	(18,19)	(20,17)
9	(14,6)	(5,15)	(11,9)	(2,18)	(20,19)	(4,16)	(7,13)	(12,8)	(3,17)	(10,1)
10	(7,15)	(14,8)	(6,16)	(5,17)	(12,10)	(13,9)	(3,19)	(1,11)	(20,2)	(18,4)
11	(8,16)	(17,7)	(13,11)	(14,10)	(6,18)	(2,3)	(20,4)	(9,15)	(1,12)	(5,19)
12	(17,9)	(15,11)	(18,8)	(10,16)	(14,12)	(1,13)	(5,2)	(3,4)	(6,20)	(19,7)
13	(18,10)	(19,9)	(16,12)	(15,13)	(11,17)	(3,6)	(14,1)	(8,20)	(5,4)	(7,2)
14	(16,14)	(13,17)	(12,18)	(19,11)	(15,1)	(20,10)	(4,7)	(5,6)	(8,3)	(2,9)
15	(12,20)	(18,14)	(19,13)	(17,15)	(1,16)	(5,8)	(2,11)	(6,7)	(4,9)	(3,10)
16	(15,19)	(16,18)	(14,20)	(3,12)	(2,13)	(17,1)	(6,9)	(10,5)	(7,8)	(4,11)
17	(19,17)	(20,16)	(10,7)	(13,4)	(3,14)	(15,2)	(12,5)	(18,1)	(11,6)	(9,8)
18	(20,18)	(7,12)	(9,10)	(11,8)	(4,15)	(14,5)	(13,6)	(2,17)	(19,1)	(16,3)
19	(9,12)	(8,13)	(17,4)	(7,14)	(10,11)	(6,15)	(18,3)	(19,2)	(16,5)	(1,20)
20	(11,13)	(9,14)	(15,17)	(16,20)	(8,4)	(18,12)	(1,19)	(7,3)	(2,10)	(6,5)

References

[1] de Werra D. Geography, games and graphs. Discrete Applied Mathematics 1980;2(4):327–37.
[2] de Werra D. Minimizing irregularities in sports schedules using graph theory. Discrete Applied Mathematics 1982;4(3): 217–26.
[3] Schreuder JAM. Combinatorial aspects of construction of competition Dutch professional football leagues. Discrete Applied Mathematics 1992;35(3):301–12.
[4] Russell RA, Leung JMY. Devising a cost effective schedule for a baseball league. Operations Research 1994;42(4): 614–25.
[5] Nemhauser GL, Trick MA. Scheduling a major college basketball conference. Operations Research 1998;46(1):1–8.
[6] Ferland JA, Fleurent C. Computer aided scheduling for a sport league. INFOR 1991;29(1):14–25.
[7] Costa D. An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR 1995;33(3):161–78.
[8] Bean JC, Birge JR. Reducing travelling costs and player fatigue in the National Basketball Association. Interfaces 1980;10(3):98–102.
[9] Wright MB. Scheduling fixtures for Basketball New Zealand. Working Paper, Lancaster University Management School, 2003.
[10] Cain WO. The computer-aided heuristic approach used to schedule the major league baseball clubs. In: Ladany SP, Machol RE, editors. Optimal strategies in sports. Amsterdam: North-Holland; 1977. p. 32–41.
[11] Armstrong J, Willis RJ. Scheduling the cricket world cup—a case study. Journal of the Operational Research Society 1993;44(11):1067–72.
[12] Willis RJ, Terrill BJ. Scheduling the Australian state cricket season using simulated annealing. Journal of the Operational Research Society 1994;45(3):276–80.
[13] Wright M. Timetabling county cricket fixtures using a form of tabu search. Journal of the Operational Research Society 1994;45(7):758–70.
[14] Della Croce F, Tadei R, Asioli PS. Scheduling a round robin tennis tournament under courts and players availability constraints. Annals of Operations Research 1999;92:349–61.
[15] Urban TL, Russell RA. Scheduling sports competitions on multiple venues. European Journal of Operational Research 2003;148(2):302–11.
[16] Brailsford SC, Potts CN, Smith BM. Constraint satisfaction problems: algorithms and applications. European Journal of Operational Research 1999;119(3):557–81.
[17] Schaerf A. Scheduling sport tournaments using constraint logic programming. Constraints 1999;4(1):43–65.
[18] Trick MA. A schedule-then-break approach to sports timetabling. In: Burke EK, Wilhelm E, editors. Practice and theory of automated timetabling III, Lecture notes in computer science, vol. 2079. Heidelberg: Springer; 2001. p. 242–53.
[19] Henz M. Scheduling a major college basketball conference—revisited. Operations Research 2001;49(1):163–8.
[20] Henz M, Müller T, Thiel S. Global constraints for round-robin tournament scheduling. European Journal of Operational Research 2004;153(1):92–101.
[21] Van Hentenryck P. Constraint and integer programming in OPL. INFORMS Journal on Computing 2002;14(4):345–72.
[22] Schreuder JAM. Constraint satisfaction for requirement handling of football fixture lists. Belgian Journal of Operations Research 1995;35(2):51–60.