Scheduling a Major College Basketball Conference—Revisited

Technical Note, Revised Version (first submission: March 4 1998)

Martin Henz

School of Computing
National University of Singapore
Singapore 117543
email: henz@comp.nus.edu.sg

January 15, 2002

Abstract

Nemhauser and Trick presented the problem of finding a timetable for the 1997/98 Atlantic Coast Conference (ACC) in basketball. Their solution, found with a combination of integer programming and exhaustive enumeration, was accepted by the ACC.

Finite-domain constraint programming is another programming technique that can be used for solving combinatorial search problems such as sports tournament scheduling. This paper presents a solution of round robin tournament planning based on finite-domain constraint programming. The approach yields a dramatic performance improvement, which makes an integrated interactive software solution feasible.

1 Introduction

Round robin tournaments are popular in many sports disciplines. They consist of successions of dates in which each team plays each other team a fixed number of times (twice in a double round robin). Such tournaments become computationally interesting, if constraints prevent the reuse of well-known timetables. Such a case is presented by Nemhauser and Trick (1998) where a host of criteria coming from teams, fans and media must be satisfied. They show how the criteria are exploited in three phases of a solution process, implemented using integer programming and explicit enumeration, and report a "turn-around-time" of 24 hours, which means it takes one day of computing time from specifying/modifying the criteria using feedback from the ACC organizers to proposing new solutions.

An alternative approach to solving such a combinatorial search problem is finite-domain constraint programming. This paper shows how constraint programming can be used for solving all three phases of the process and reducing the turn-around-time to below one minute on a computer with similar performance as the one used by Nemhauser and Trick. This speedup makes it practical to encorporate the approach in an interactive tool for round robin scheduling.

Finite-domain constraint programming is introduced in the next section. The ACC 1997/98 tournament scheduling problem is given in Section 3. Section 4 explains the decomposition into three phases. Sections 5, 6 and 7 describe the modeling and implementation of these phases using finite-domain constraint programming.

2 Constraint Programming

Finite-domain constraint programming is a technique designed for solving combinatorial search problems. It evolved from research in constraint logic programming languages—described by Jaffar and Maher (1994)—and led to the development of constraint programming languages such as CHIP described by Dincbas et al. (1988), and Oz described by Smolka (1995), and the constraint programming library ILOG Solver described by Puget (1994). Stuckey and Marriott (1998) explain the approach in detail and Wallace (1996) presents an overview of applications of finite-domain constraint programming.

Every variable of the model is represented by a finite-domain variable. A constraint store stores information on such a variable in the form of the set of possible values that the variable can take; this set is called the current domain of the variable. More formally, the constraint store is a conjunction of constraints of the form $x \in S$, where S is a set of integers. Computation starts with an initial domain for each variable as given in the model. Some constraints can be directly entered in the constraint store. For example, the constraint $x \neq 5$ can be expressed in the constraint store by removing 5 from the domain of x.

Other more complex constraints are translated by the programmer into computational agents called propagators. Each propagator observes the variables given by the corresponding constraint in the problem. Whenever possible, it strengthens the constraint store with respect to these variables by excluding values from their domain according to the corresponding constraint. For example, a propagator for the constraint $x \leq y$ observes the upper and lower bounds of the domains of x and y. A possible strengthening consists of removing all values from the domain of x that are greater than the upper bound of the domain of y. Apart from arithmetic constraints, propagators can express complex symbolic relationships between variables, such as the constraint that the values of given variables should be distinct integers.

The process of propagation continues until no propagator can further strengthen the constraint store. The constraint store is said to be stable. At this point, many problem variables typically have still non-singleton domains. Thus the constraint store does not represent a solution, and search becomes necessary.

Search for solutions is implemented by choice points. A choice point generates a constraint c. From the current stable constraint store cs, two new constraint stores are created by adding c and $\neg c$, respectively, to cs. It is quite possible that the new constraint stores are not stable, in other words c or $\neg c$ trigger some propagators in the respective new store. After stability is reached, this branching process is continued recursively on both sides until every leaf of the resulting search tree is either inconsistent or every variable has a unique value. The choice of the constraint c at each branching determines the shape and size of the search tree and thus is a crucial factor for performance of the search. A mechanism to systematically generate these constraints is called a search strategy. A common search strategy is based on variable enumeration, i.e. the constraints c have the form $x = v$ for some variable x and some value v from its domain.

A constraint programming system takes care of activating propagators, reaching stability and performing the search. The programmer can concentrate on translating the constraints into appropriate propagators and specifying the search strategy. Finite-domain constraint programming systems support this task through libraries of propagators and search strategies.

3 The ACC 1997/98 Tournament Scheduling Problem

The double round robin scheme determines that every team i plays against every other team exactly twice during the competition, once at the place of team i (a home match for i) and once at the other team's place (an away match for i). The first of the two matches is called the first leg, the second is the return match.

A temporally dense double round robin (DDRR) for n teams distributes the $n(n-1)$ matches over a minimal number of dates such that every team plays at most one match per date. If n is even, the number of dates is $2(n-1)$. A DDRR with an odd number of teams consists of $2n$ dates in each of which $n-1$ teams play and one team does not. This team is said to have a bye.

The ACC 1997/98 schedule in male basketball described by Nemhauser and Trick (1998) was a DDRR consisting of nine teams: Clemson (abbreviation Clem; team 1), Duke (Duke; 2), Florida State (FSU; 3), Georgia Tech (GT; 4), Maryland (UMD; 5), North Carolina (UNC; 6), North Carolina State (NCSt; 7), Virginia (UVA; 8), and Wake Forest (Wake; 9). The problem was to find a DDRR timetable whose 18 dates are distributed over the period of nine weeks starting December 31 1997 (date 1, a Wednesday) and ending March 1 1998 (date 18, a Sunday) such that there is one weekday date and one weekend date per week, subject to a number of criteria. For the purpose of comparison, only the criteria that are employed by Nemhauser and Trick (1998) and Trick (1998) are considered here.

1. **Mirroring.** The dates are grouped into pairs (r_1, r_2), such that each team will get to play against the same team in dates r_1 and r_2. Such a grouping is called a mirroring scheme. Nemhauser and Trick fix the mirroring scheme to

   $$m = \{(1,8),(2,9),(3,12),(4,13),(5,14),(6,15),(7,16),(10,17),(11,18)\}$$

   to cater for one of the idiosyncratic criteria (see Criteria 9 below). To ease the comparison with their work, this mirroring is used throughout this paper.

2. **Initial and Final Homes and Aways.** Every team must play home on at least one of the first three dates. Every team must play home on at least one of the last three dates. No team can play away on both last dates.

3. **Home/Away/Bye Pattern Criteria.** No team may have more than two away matches in a row. No team may have more than two home matches in a row. No team may have more than three away matches or byes in a row. No team may have more than four home matches or byes in a row.

4. **Weekend Pattern.** Of the weekends, each team plays four at home, four away, and one bye.

5. **First Weekends.** Each team must have home matches or byes at least on two of the first five weekends.

6. **Rival Matches.** Every team except FSU has a traditional rival. The rival pairs are Clem-GT, Duke-UNC, UMD-UVA, and NCSt-Wake. In the last date, every team except FSU plays against its rival, unless it plays against FSU or has a bye.

7. **Popular Matches in February.** The following pairings must occur at least once in dates 11 to 18: Duke-GT, Duke-Wake, GT-UNC, UNC-Wake.

8. **Opponent Sequence Criteria.** No team plays in two consecutive dates away against Duke and UNC. No team plays in three consecutive dates against Duke, UNC and Wake (independent of home/away).

9. **Idiosyncratic Criteria.** UNC plays its rival Duke in the last date and in date 11. UNC plays Clem in the second date. Duke has a bye in date 16. Wake does not play home in date 17. Wake has a bye in the first date. Clem, Duke, UMD and Wake do not play away in the last date. Clem, FSU, GT and Wake do not play away in the first date. Neither FSU nor NCSt have a bye in last date. UNC does not have a bye in the first date.

Among all solutions that fulfill these criteria, Nemhauser and Trick chose those solutions that suitably satisfy a number of further preferences, for final selection by the ACC. These preferences and the resulting postprocessing is beyond the scope of this work.

4 Computing Round Robin Tournament Schedules

Previous works on round robin scheduling by Cain (1977), de Werra (1988), Schreuder (1992), Schaerf (1996), and Nemhauser and Trick (1998) generally agree on a decomposition of the scheduling process into three phases, namely pattern generation, pattern set generation and timetable generation. McAloon, Tretkoff and Wetzel (1997) use constraint programming to solve a related problem, in which the concept of home and away games is replaced by resources called periods, and therefore this decomposition is not applicable.

A pattern indicates the way in which a team can play home, away and bye throughout the tournament. In the ACC 1997/98 case, patterns are subject to Criteria 1 through 5 given in the previous section. The following pattern meets these criteria; here, a home game is represented by the symbol $+$, an away game by $-$ and a bye by b.

dates	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
pattern	$-$	$+$	b	$-$	$+$	$+$	$-$	$+$	$-$	$-$	$+$	b	$+$	$-$	$-$	$+$	$+$	$-$

After generating all patterns that meet the given criteria, sets of n patterns are computed. The patterns in such a pattern set must have properties that enable the construction of a timetable based on its patterns.

From a given pattern set, timetables are constructed by deciding which team plays according to which pattern and against which opponent in each round.

5 Patterns

A suitable constraint programming model for pattern generation consists of 0/1 variables h_j, a_j and b_j, $1 \leq j \leq 18$. A team that plays according to the pattern represented by these variables plays home (away, bye) at date j if and only if $h_j = 1$ ($a_j = 1$, $b_j = 1$). The constraints on these variables are given in Table 1.

1. For all dates j: $h_j + a_j + b_j = 1$
2. For all pairs of dates $(j, j') \in m$ (m is given in Constraint 1 in Section 3): $h_j = a_{j'}$, $a_j = h_{j'}$, $b_j = b_{j'}$
3. $h_1 + h_2 + h_3 \geq 1$, $h_{16} + h_{17} + h_{18} \geq 1$, $a_{17} + a_{18} < 2$
4. For all $j \leq 16$: $a_j + a_{j+1} + a_{j+2} < 3$, $h_j + h_{j+1} + h_{j+2} < 3$, for all $j \leq 15$: $a_j + b_j + a_{j+1} + b_{j+1} + a_{j+2} + b_{j+2} + a_{j+3} + b_{j+3} < 4$, for all $j \leq 14$: $h_j + b_j + h_{j+1} + b_{j+1} + h_{j+2} + b_{j+2} + h_{j+3} + b_{j+3} + h_{j+4} + b_{j+4} < 5$
5. $(\sum_{j \in \{2,4,\ldots,18\}} h_j) = 4$, $(\sum_{j \in \{2,4,\ldots,18\}} a_j) = 4$, $(\sum_{j \in \{2,4,\ldots,18\}} b_j) = 1$
6. $(\sum_{j \in \{2,4,6,8,10\}} h_j + b_j) \geq 2$
7. $b_1 + a_{18} < 2$, $b_1 + h_{17} < 2$, $b_{16} + a_{18} < 2$

Table 1: The Constraints for Patterns

The first constraint expresses that a team plays either home, away or has a bye in each round. Constraints 2 through 6 correspond to the first five ACC 1997/98 criteria in Section 3. Constraints 7 stem from Criteria 9 in Section 3, since only one team can have a bye in a given date.

Constraint programming systems provide propagators for all these arithmetic constraints. The most effective search strategy in our case seems to be to enumerate the h, a and b variables date-wise, i.e. in the order $h_1, a_1, b_1, h_2, a_2, b_2, \ldots, b_{18}$.

For ACC 1997/98, Nemhauser and Trick argue that explicit enumeration ("generate and test") finds all 38 patterns in reasonable time, but do not give runtimes. Using the above model, a constraint program performs all solution search in 0.44 seconds using 117 choice points. This and all other runtimes given in this work were obtained by the constraint-based round robin tournament planning software Friar Tuck developed by the author and given in Henz (1999), on a PC with a 233 MHz Pentium II processor and 64 MBytes of RAM. Friar Tuck is implemented using the constraint programming system Mozart, Mozart Consortium (1999).

6 Pattern Sets

The next step is to generate suitable sets of 9 patterns. Graph-theoretical results cover the existence and generation of pattern sets with useful properties; see de Werra (1988) and Schreuder (1992). In the presence of irregular constraints on patterns such as Criteria 3 in Section 3 the pattern set problem becomes a combinatorial search problem. The following model is suitable for a solution using constraint programming.

The 38 feasible patterns are represented by three 38×9 matrices h, a, and b of 0/1 values, where $h_{i,j}$, $(a_{i,j}, b_{i,j})$ indicate whether pattern i fixes date j to a home game (away game, bye). For each pattern i, a 0/1 variable x_i indicates whether this pattern occurs in the resulting pattern set. The constraints are

1. $\sum_i x_i = 9$
2. For all dates j: $(\sum_i h_{i,j} x_i = 4), (\sum_i a_{i,j} x_i = 4), (\sum_i b_{i,j} x_i = 1)$

Following Trick (1998) we exclude pairs of patterns in which there is no possible meeting date for corresponding teams. More formally,

3. For every pair i, i' s.t. $i \neq i'$, $\forall_j (h_{i,j} = 0 \vee a_{i',j} = 0)$ and $\forall_j (a_{i,j} = 0 \vee h_{i',j} = 0)$: $x_i + x_{i'} \leq 1$

Nemhauser and Trick model pattern set generation as an optimization problem and use integer programming to solve it. They report that all 17 pattern sets are found by solving a sequence of 17 integer programs with an overall runtime under 1 minute on a Sun Sparcstation 20 with CPLEX version 4.0. In each run, a constraint was added to the previous integer program that precludes the previous solution.

For constraint programming, there is no need for introducing an objective function and therefore the modeling and implementation is simpler than for the integer programming approach. A constraint program for the above model performs all solution search in 3.1 seconds with 176 choice points, using simple enumeration of the variables x_i.

7 Timetables

This step generates feasible timetables from a given pattern set. Two approaches for timetable generation have been described. The first approach—proposed by Schreuder (1992) and used by Nemhauser and Trick—generates timetables of "placeholder teams" first and then assigns teams to placeholders. This approach is also taken in previous work on using constraint programming for DDRR scheduling by Schaerf (1996). The disadvantage of this approach for ACC 1997/98 is that the team-specific Criteria 6 through 9 cannot be exploited while generating timetables of placeholders. Nemhauser and Trick report a runtime of 24 hours on a Sun Sparcstation 20 to compute all timetables based on the 17 pattern sets for this approach using integer programming and explicit enumeration.

The second approach—sketched by Cain (1977)—assigns teams to pattern sets, and then opponent teams for each team and date. This work recasts Cain's approach in the framework of finite-domain constraint programming. The following model assumes that the pattern set is given in the form of 9×18 matrices H, A and B of 0/1 values whose entries $H_{i,j}$ ($A_{i,j}$, $B_{i,j}$) indicate home matches (away matches, byes) for pattern i in date j.

The target timetable is represented by a 9×18 matrix α, whose variables $\alpha_{i,j}$ range over $0, \ldots, 9$ and tell the opponent team against which team i plays in date j (0 stands for bye), and three 9×18 matrices $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ of 0/1 variables whose entries $\mathcal{H}_{i,j}$ ($\mathcal{A}_{i,j}$, $\mathcal{B}_{i,j}$) tell if team i plays home (plays away, has a bye) in date j.

The constraints on α, $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ are given in Table 2. Here, `distinct` is a constraint that forces all its arguments to be distinct integers. $(x \doteq y)$ stands for a 0/1 variable which is 1 if and only if x is equal to y; $(x \in s)$ stands for a 0/1 variable which is 1 if and only if x is an element of s. Such constraints that reflect the validity of another constraint in a 0/1 variable were introduced in the context of constraint programming by Older and Benhamou (1993). For such constraints, the term reified constraint was coined by Gert Smolka and appeared first in Henz and Würtz (1995). Expressions formed by the logical operations $\Rightarrow$, $\wedge$ and $\vee$ represent constraints that operate on 0/1 variables. Constraints 1 through 7 describe general properties of DDRRs with an odd number of teams. Note

1. For all dates j: $\texttt{distinct}(\alpha_{1,j}, \ldots, \alpha_{9,j})$
2. For all teams i, i', $i \neq i'$, and dates j: $(\alpha_{i,j} \doteq i') = (\alpha_{i',j} \doteq i)$
3. For all teams i, i', $i \neq i'$: $(\sum_{j \in \{1,\ldots,18\}} (\alpha_{i,j} \doteq i')) = 2$
4. For all teams i and dates j: $\alpha_{i,j} \neq i$
5. For all teams i and dates j: $(\alpha_{i,j} \doteq 0) = \mathcal{B}_{i,j}$
6. For all teams i, i', $i \neq i'$, and dates j: $(\alpha_{i,j} \doteq i') \Rightarrow ((\mathcal{H}_{i,j} \wedge \mathcal{A}_{i',j}) \vee (\mathcal{A}_{i,j} \wedge \mathcal{H}_{i',j}))$
7. For all teams i and dates j: $\mathcal{H}_{i,j} + \mathcal{A}_{i,j} + \mathcal{B}_{i,j} = 1$
8. For all pairs of teams $(i, i') \in \{(1,4), (2,6), (5,8), (7,9)\}$:
   $(\alpha_{i,18} \doteq i') \vee (\alpha_{i,18} \doteq 3) \vee (\alpha_{i,18} \doteq 0)$
9. For all pairs of teams $(i, i') \in \{(2,4), (2,9), (4,6), (6,9)\}$:
   $(\sum_{j \in \{11,\ldots,18\}} (\alpha_{i,j} \doteq i')) \geq 1$
10. For all teams i and dates $j \in \{1, \ldots, 17\}$:
    $(\alpha_{i,j} \in \{2,6\}) + \mathcal{A}_{i,j} + (\alpha_{i,j+1} \in \{2,6\}) + \mathcal{A}_{i,j+1} < 4$,
    for all teams i and dates $j \in \{1, \ldots, 16\}$:
    $(\alpha_{i,j} \in \{2,6,9\}) + (\alpha_{i,j+1} \in \{2,6,9\}) + (\alpha_{i,j+2} \in \{2,6,9\}) < 3$
11. $\alpha_{6,11} = 2$, $\alpha_{6,18} = 2$, $\alpha_{6,2} = 1$, $\mathcal{B}_{2,16} = 1$, $\mathcal{H}_{9,17} = 0$, $\mathcal{B}_{9,1} = 1$, $\mathcal{A}_{1,18} = 0$, $\mathcal{A}_{2,18} = 0$, $\mathcal{A}_{5,18} = 0$, $\mathcal{A}_{9,18} = 0$, $\mathcal{A}_{1,1} = 0$, $\mathcal{A}_{3,1} = 0$, $\mathcal{A}_{4,1} = 0$, $\mathcal{A}_{9,1} = 0$, $\mathcal{B}_{3,18} = 0$, $\mathcal{B}_{7,18} = 0$, $\mathcal{B}_{6,1} = 0$.

Table 2: The Constraints for Timetables

that Constraints 4 are redundant due to Constraints 3. Adding redundant constraints to achieve earlier pruning of the search tree is an important constraint programming technique. Constraints 8 through 11 correspond to the ACC 1997/98 Criteria 6 through 9 in Section 3.

To connect the given pattern set to the target timetable, nine finite-domain variables p_i, $1 \leq i \leq 9$, ranging over $1, \ldots, 9$ are introduced. Each team i plays according to the pattern in row p_i of H, A and B. The following relationship links the given pattern set with the target timetable, using the variables p_i as indices.

$$\text{For all teams } i \text{ and dates } j\text{: } H_{p_i,j} = \mathcal{H}_{i,j}, A_{p_i,j} = \mathcal{A}_{i,j}, B_{p_i,j} = \mathcal{B}_{i,j}$$

This relationship is expressible with the so-called element constraint described by

Dincbas, Simonis and Van Hentenryck (1988). The element constraint takes as arguments a finite-domain variable k, a vector of integers v and a finite-domain variable w.

$$\mathtt{el}(k, v, w)$$

The semantics is $v_k = w$. A corresponding propagator 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, if the last index that pointed to an x in v is eliminated from k. Both propagation directions are essential here. The following Constraints 12 describe the desired relationship; here H_j (A_j, B_j) stands for the j^{th} column of the matrix H (A, B).

12. For all teams i and dates j: $\mathtt{el}(p_i, H_j, \mathcal{H}_{i,j})$, $\mathtt{el}(p_i, A_j, \mathcal{A}_{i,j})$, $\mathtt{el}(p_i, B_j, \mathcal{B}_{i,j})$.

All major constraint programming systems provide propagators for all constraints required, including `el`, `distinct`, logical and reified constraints.

The search strategy of Cain's method first assigns patterns to teams by enumerating the variables in p. In this process, the matrices $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{B}$ are gradually being determined. The propagators for the element constraints, together with the propagators for the constraints 8 through 11 in Table 2, achieve a dramatic pruning of the search tree in this process.

Next, the variables in α are enumerated. Here, the propagators corresponding to the constraints in Table 2 achieve further pruning. Apparently, the best strategy is to enumerate date-wise, i.e. in the order $\alpha_{1,1}, \alpha_{2,1}, \ldots, \alpha_{n,1}$, $\alpha_{1,2}, \ldots$, $\alpha_{n,d}$. A constraint program for this approach performs all solution search successively on all 17 pattern sets in 53.7 seconds using 476 choice points, leading to 179 solutions.

8 Conclusion

Modeling and solving sports tournament scheduling problems with finite-domain constraint programming can lead to efficient computation of timetables. All 179 solutions to the ACC 1997/98 tournament scheduling problem presented by Nemhauser and Trick (1998) are found in less than one minute using constraint programming, whereas Nemhauser and Trick report an overall runtime of about 24 hours (both on computers with similar performance) using exhaustive enumeration and integer programming.

Nemhauser and Trick report that problems of this kind require multiple cycles of problem refinement in collaboration with the tournament organizers to obtain a timetable that satisfies all parties involved. In such a scenario, the advantage of an interactive and efficient software becomes even more obvious. The results reported in this paper show that finite-domain constraint programming can provide the base for such a software, as realized in the round robin scheduler Friar Tuck, Henz (1999).

Acknowledgements

George Nemhauser mentioned this problem in his invited talk at the conference "Theory and Practice of Constraint Programming CP96" in Boston. He and Michael Trick shared the problem specification with the author. Joachim Paul Walser and Lua Seet Chong pointed out errors in previous versions of this paper and suggested improvements. Jörg Würtz pointed out the first appearance of the element propagator. The author thanks the editor and the referees for many helpful comments on previous versions.

References

CAIN, W. O., JR. 1977. 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.

DINCBAS, M., H. SIMONIS, AND P. VAN HENTENRYCK. 1988. 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.

DINCBAS, M., P. VAN HENTENRYCK, H. SIMONIS, A. AGGOUN, AND T. GRAF. 1988. The constraint logic programming language CHIP. In Proceedings International

Conference on Fifth Generation Computer Systems, pages 693–702, Tokyo, Japan, December 1988. Springer-Verlag.

HENZ, M. 1999. Friar Tuck 1.1: A Constraint-based Round Robin Planner. Software available via WWW at `http://www.comp.nus.edu.sg/~henz/projects/FriarTuck`.

HENZ, M., AND J. Würtz. 1995. Using Oz for College Time Tabling, In E.K.Burke and P.Ross, editors, *The Practice and Theory of Automated Time Tabling*, Lecture Notes in Computer Science, vol. 1153, Springer-Verlag, Berlin, 162–177.

JAFFAR, J., AND M. MAHER. 1994. Constraint logic programming—a survey. Journal of Logic Programming 19/20, 503–582.

MARRIOTT, K., AND P. STUCKEY. 1998. Programming with Constraints. MIT Press, Cambridge, MA.

MOZART CONSORTIUM. 1999. The Mozart Programming System. Available via WWW at `http://www.mozart-oz.org`, Programming Systems Lab, German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3, D–66123 Saarbrücken, Germany.

NEMHAUSER, G., AND M. TRICK. 1998. Scheduling a major college basketball conference. Opns. Res. 46(1).

OLDER, W., AND F. BENHAMOU. 1993. Programming in CLP(BNR). Position Papers for the First Workshop on Principles and Practice of Constraint Programming, Newport, RI, USA, 239–249.

PUGET, J.-F. 1994. A C++ implementation of CLP. In Proceedings Second Singapore International Conference on Intelligent Systems, Singapore.

SCHAERF, A. 1999. Scheduling Sport Tournaments using Constraint Logic Programming. CONSTRAINTS 4(1) 43–65.

SCHREUDER, J. A. M. 1992. Combinatorial aspects of construction of competition dutch professional football leagues. Discrete Applied Mathematics, 35, 301–312.

SMOLKA, G. 1995. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, Volume 1000, Springer-Verlag, Berlin, 324–343.

TRICK, M. 1998. Modifications to the Problem Description of "Scheduling a Major College Basketball Conference". WWW at `http://mat.gsia.cmu.edu/acc_mod.html`.

WALLACE, M. 1996. Practical applications of constraint programming. Constraints, 1(1&2), 139–168.

de WERRA, D. 1988. Some models of Graphs for Scheduling sports competitions. Discrete Applied Mathematics, 21, 47–65.

McALOON, K., C. TRETKOFF AND G. WETZEL. 1997. Sports League Scheduling, Proceedings of the 1997 ILOG Optimization Suite International Users' Conference, Paris.