Sports League Scheduling

Ken McAloon, Carol Tretkoff, Gerhard Wetzel

Logic Based Systems Lab

Brooklyn College and CUNY Graduate Center, Brooklyn, NY 11210, USA

Email: {tretkoff,mcaloon,gw}@sci.brooklyn.cuny.edu

1. Introduction

Sports scheduling is an area of increasing interest in constraint programming. As amateur and professional sports leagues proliferate and grow in size and complexity, organizers are increasingly turning to computer assisted scheduling [Nemhauser and Trick]. The scientific literature in this area is also growing and one can begin to get a sense of the range and mathematical difficulty of the problems encountered. These can include classical challenges such as set covering problems and quadratic assignment problems. In this note we concentrate on a version of a core problem that invariably comes up: determining whether a set of constraints on the schedule is feasible. This is often called the "timetabling" problem of the scheduling process.

Here we consider the timetabling problem for a "round robin" schedule: the case in which every team must play every other team exactly once. (Later we will discuss related problems that come up in other sports situations such as ones involving home and away games between each pair of teams.) Typically a league game will be scheduled on a certain field or court, at a certain time, etc. This kind of combination will be called a period. These periods can vary in desirability due to such factors as lateness in the day, the location and the condition of the field, etc. Therefore, the problem is to schedule the games such that the different periods are assigned to the teams in an equitable manner over the course of the season. To fix ideas, we make the following stipulations:

1. There are N teams (N even) and every two teams play each other exactly once.

2. The season lasts N-1 weeks.

3. Every team plays one game in each week of the season.

4. There are N/2 periods and, each week, every period is scheduled for one game.

5. No team plays more than twice in the same period over the course of the season.

A particular period in a particular week will be called a slot. The meeting between two teams will be called a matchup. For example, a valid schedule for 8 teams named 0,1,2,3,4,5,6,7 would be given by filling in the slots with matchups as in the Table below.

2. Algorithms

The 8 team problem instance is very simple and can be done by brute force if necessary. However, the combinatorics of this scheduling problem are very explosive. For an N team league, there are N/2*(N-1) matchups (i,j) with 0i<j<N to be played. A schedule can be thought of as a permutation of these matchups. So for N teams the search space size is (N/2*(N-1))! . In other words the search space size grows as the factorial of the square of N/2. This means that algorithms cannot be expected to scale nicely; and, as we shall see, they do not.

An 8 Team Round Robin Timetable

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

There are two ways of formulating a solution strategy. The primal method is to start with the slots and to seek to find matchups to place in them. The dual method is to start with the matchups and to look for the slots to place them in. The primal approach makes formulating the constraints more natural and it is the one we discuss now. We discuss the dual method briefly further on.

As in the Table, the solution can be represented as a two-dimensional array with columns for weeks and rows for periods. There is a "cardinality" constraint on each row, namely that no team appear more than twice. On each column we have the constraint that each team occur exactly once; this is equivalent to an "all different" constraint on the column, namely that the teams appearing in the column be distinct. Finally there is a global constraint on the entire array, namely that each matchup occur exactly once or, equivalently, that all matchups be different. So the primal algorithm comes down to filling the slots with matchups subject to the above constraints. Note that one has the choice for the column constraints and the global constraint of the problem of using "cardinality = 1" constraints or of using "all different" constraints.

An important tool for this timetabling phase of sports scheduling is integer programming (IP). In this approach one uses the cardinality constraints rather than the all different constraints, the former being elegant in this approach and the latter being both inelegant and inefficient. The 0-1 integer programming model for the primal approach to a solution can be summarized as follows. For each pair of teams i<j<N, for each row k and each column m, there is a binary variable $x_{i,j,k,m}$ which will be 1 if i plays j in the slot in row k and column m and 0 otherwise. The constraints arre

for all i and j	$\sum_{k,m} x_{i,j,k,m} = 1$	each team plays each other team once
for all i and m	$\sum_{k,j} x_{i,j,k,m} = 1$	team i plays once in column m
for all i and k	$\sum_{k,m} x_{i,j,k,m} <= 2$	each team plays at most twice in a period
for all k and m	$\sum_{i,j} x_{i,j,k,m} = 1$	each slot has 1 game

This 0-1 integer program is elegant in its simplicity. However, it suffers from the fact that as N increases the number of 0-1 variables and the number of constraints grow exceedingly large. Even with symmetry breaking variable bindings added, this formulation was not able to find a solution for N=14, although it performed creditably for N<= 12. For reference, these analyses were made using Cplex, a leading commercial integer programming package.

The ILOG Solver offers the possibility of using all different constraints in lieu of the cardinality constraints that were used in the IP model to express the facts that each team plays once in each column and that each matchup occurs exactly once in the schedule.

To do this in ILOG Solver, we introduce a row object with two kinds of structural constraints: constraints to insure that each matchup involves different teams and the cardinality constraints on the row. Note that the row has length 2*(N-1) because each matchup requires a pair of teams.

```
class Row {
public:
```

```
      IlcIntVarArray row;
      Row();
};
Row::Row(): row(2*(N-1),0,N-1){
      IlcInt j;
      for(j=0;j<2*(N-1);j=j+2)
            IlcPost(row[j] < row[j+1]);
      IlcIndex I;
      for(j=0;j<N;j++)
            IlcPost(IlcCard(I,row[I] == j) <= 2);
}
```

Then the two-dimensional schedule is represented by a vector of rows

```
      Sched *rows = new (IlcHeap()) Row[N/2];
```

Another object is introduced to obtain a column view of the schedule:

```
class Column {
public:
      IlcIntVarArray column;
      Column();
};
Column::Column(): column(N,0,N-1){
      // Constraint delayed until assignments are made
      // IlcPost(IlcAllDifferent(column,whenValue));
}
```

Then the schedule is organized from a column view as a vector of column objects

```
      Column *columns = new (IlcHeap()) Column[N-1];
```

The column entries are assigned the appropriate row entries. At this point, the all different constraints on the columns can be posted.

Finally, we have to assert that all the matchups are distinct. To that end we introduce a "pairing function":

```
IlcIntVar Array pairs(N*(N-1));
```

We then assign to the elements of this array an IlcIntExp for each i < N and each even j<2*(N-1),

```
      pairs[i*(N-1)+j/2] =     N*Sched[i].row[j] + Sched[i].row[j+1];
```

and tighten the upper and lower bounds and prune the domains. The global requirement that each matchup only occur once becomes

```
IlcPost(IlcAllDifferent(pairs,whenValue));
```

At this point we can call

```
IlcSolve(IlcGenerate(pairs,IlcChooseMinSizeInt));
```

However, generating pairs means deciding both teams i and j (i<j) for a slot at each branching point in the search. More flexibility is afforded by separating these into a decision on i and a decision on j. To this end, an IlcIntVarArray is introduced to embody the entire schedule and its entries are generated in a call to IlcSolve. This gives the best performance. In fact it solves the N=14 problem in 45 minutes on an UltraSparc.

3. Remarks

With some very simple changes this code can be adapted to deal with the timetabling problems that come up in other kinds of league scheduling. For example, let us look at double round robin tournaments, a most common form for amateur and professional leagues. Here each team plays each other team twice, once at home and once away. The season can be broken into two halves, the first being a simple round robin schedule and the second being a mirror of the first. Looking at the first half schedule, each team will play some games at home and some away; fairness dictates that these numbers be close, even equal if possible, and that consecutive sequences of home or away games be at most 2 (say) in length. A set of N home/away patterns with the property that the number of home teams equals the number of away teams for each game day is determined. Finding such sets is itself an interesting set covering problem. Given such a set, determining a feasible schedule is a timetabling problem much akin to the one we discussed above though much simpler. The cardinality constraints on the rows can be dropped. One adds column constraints to the effect that two opposing teams cannot both be home or both be away. (Though this is a much simpler problem, in practice it is often infeasible and verifying this can take a bit more time than one would like.) As before, there is an integer programming formulation, but ILOG Solver offers the more scalable solution with all different constraints replacing the cardinality constraints of the integer programming model. One more point: typically, the sets of home and away patters are determined for abstract teams 0,...,N-1. Only after timetables have been developed does one try to assign actual teams to 0,...,N-1. For the computationally masochistic, this turns out to be a quadratic assignment problem [Schreuder], another classically difficult combinatorial problem. But this one is well-studied and enumeration techniques prove sufficient to the task, at least for small enough N. It is also simplified by team preferences and other criteria that restrict the freedom in these assignments, Examples include Munich's wanting to play at home during the Oktoberfest and the Minnesota Twins' wanting to be away on the first day of hunting season.

Let us make some remarks on the dual approach to our original single round robin problem. For one thing there is no clear natural way of formulating the dual as an integer program. On the other hand, the ILOG Solver provides the tools for a very elegant formulation. However, our code does not perform nearly so well as the primal code. We are continuing experiments with the dual code since it is compact and might have better scaling behavior in the end. In point of fact, we also formulated primal and dual code in clp(FD). Here the primal performance was less satisfactory than with ILOG and the instance N=14 was not solved. However, the performance of the dual algorithm in clp(FD) for N<=12 was comparable to the primal in clp(FD).

The hope for the dual algorithm is also bolstered by the performance of a randomized local search version of it: one randomly assigns a permutation of the matchups to slots, then one makes random swaps getting column after column to satisfy the all different constraints. If that succeeds, one makes swaps within a column from row to row to try to satisfy the cardinality constraints on the rows. This code's performance is delicate; it depends heavily on the random number generator used and on several critical parameters that have to be set. With lucky

settings, this algorithm can solve the N=14 problem in 10 minutes; with unlucky settings, the algorithm does not succeed.

In point of fact, we do not know if this problem has a solution for N=16. It would be nice to have a proper mathematical analysis of the problem in its pure form. Even were that available. however, computer-based solutions would still be needed to deal with the infinite variations that occur in practical situations.

The round robin scheduling problem we have been considering is something of a classic. We would like to thank Bob Daniels for suggesting it and for giving us some of the folklore as to its origin: it reportedly came up first in scheduling "five on a side" football and the desirable periods afforded more uninterrupted pub time.

4. References

[Nemhauser and Trick] G. Nemhauser and M. Trick, Scheduling a Major College Basketball Conference, Georgia Tech Technical Report, 1997.

[Schreuder] J.A.M. Schreuder, Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues, Discrete Applied Mathematics 35 (1992) 301-312.