Scheduling a Major College Basketball Conference

George L. Nemhauser
Industrial And Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205
e-mail: george.nemhauser@isye.gatech.edu

Michael A. Trick
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213-3890
e-mail: trick+@cmu.edu

February, 1997, Revised May, 1997

Accepted for Publication: *Operations Research*

Abstract

The nine universities in the Atlantic Coast Conference (ACC) have a basketball competition in which each school plays home and away games against each other over a nine-week period. The creation of a suitable schedule is a very difficult problem with a myriad of conflicting requirements and preferences. We develop an approach to scheduling problems that uses a combination of integer programming and enumerative techniques. Our approach yields reasonable schedules very quickly and gave a schedule that was accepted by the ACC for play in 1997–1998.

1 Introduction

The Atlantic Coast Conference (ACC) is a group of nine universities in the southeastern United States which compete against each other in a number of sports. The most important sport for the ACC in terms of overall revenue is basketball. The ACC and its universities earned in excess of $30 million in basketball revenue in 1995. Almost all of this revenue came from television and radio networks showing the games, and from gate receipts.

These revenue streams are affected by the scheduling of the teams. Television networks need a steady stream of “high quality” games. Spectators want neither too few nor too many home games in any period.

In addition to these revenue aspects, there are many other effects that a schedule can have. Some teams in the ACC schedule games with non-ACC opponents during the season and need the ACC schedule not to conflict. Some teams are traditionally strong teams, and no team wishes to play a series of such teams consecutively. Teams also have preferences that last from year to year. For instance, every team likes to have the final game of the season at home. Since it is impossible to meet this request for every team every year, it is important that each team does not have consecutive years finishing with away games. These are just a few of the examples of the schedule requirements and effects.

Until the 1996–97 schedule, scheduling was done by hand in the conference office. This arduous process created adequate schedules, but was getting more difficult due to the addition of a new team in 1991–92 and increased attention to the television schedule requests. In 1996–97, a constraint programming approach was attempted, with limited success for reasons described in section 4. For the 1997-98 season, we were approached to create the schedule. One of the schedules we recommended was accepted. While this paper concentrates on the men's basketball schedule, we also provided the ACC with a women's basketball schedule that was accepted.

In the past few years there have been a number of papers on sports scheduling. We begin by giving a summary and classification of sports scheduling systems that allows us to contrast our scheduling system with others. We then detail the requirements of an ACC schedule and show how our scheduling system specializes in this case. Finally, we present the schedule accepted by the ACC and discuss the difficulties in getting it.

2 Sports Scheduling

Sports schedules come in two broad types: temporally–constrained schedules and temporally–relaxed schedules. In a temporally–constrained schedule, the number of *slots*, or time periods in which a game may appear, is equal to the number of games that each team must play plus any necessary byes for leagues with an odd number of teams. For instance, a double round robin tournament has every team play every other team twice, once at home and once away. Such a tournament among 9 teams requires at least eighteen slots. A temporally–constrained schedule has eighteen slots available. A temporally–relaxed schedule has more than eighteen slots, and perhaps significantly more.

In a temporally–relaxed schedule, it is possible to assign games sequentially and end up with a feasible schedule. Furthermore, local improvement heuristics seem prevalent in this environment. Examples of this include Bean and Birge (1980) in scheduling the National Basketball Association, Ferland and Fleurent (1991) in scheduling the National Hockey League and other hockey leagues, and Armstrong and Willis (1993), Willis and Terrill (1994), and Wright (1994), all of whom studied scheduling cricket matches.

In this paper, we are concerned with temporally–constrained schedules. In such schedules, greedily assigning games typically leads to infeasibilities. Furthermore, while some local improvement heuristics have been found, they tend to be rather limited in scope and heavily dependent on finding a good initial solution. Examples of such schedules include the work of Campbell and Chen (1976) and Ball and Webster (1977) for scheduling college basketball, Schreuder (1992) for Dutch Football, Cain (1977) for major league baseball, and Russell and Leung (1994) for a minor league baseball league. All of these use the same framework for finding good schedules. Since we also use this framework, we examine it in some detail.

We can divide the search for a good schedule into three steps:

1) A pattern is a string consisting of H (home), A (away), and B (bye), of length equal to the number of slots in the schedule. In step 1, we find a set of *patterns* with cardinality

equal to the number of teams. This set is called a *pattern set*.

2) In step 2, we assign games to the pattern set consistent with the HAB letters. The result from this stage is a *timetable*.

3) In step 3, we assign the teams to the patterns. Together with the timetable, this gives a *schedule*.

For instance, for a four team round robin, with teams a,b,c,d, one method of finding a schedule is to chose the pattern set:

1:HHA
2:AHA
3:HAH
4:AAH

This means, for example, that team 1 plays at home in slots 1 and 2 and away in slot 3. However, we have not yet specified which team among {a,b,c,d} is team 1.

We then assign games consistent with this pattern set to get the timetable:

1:(-2)(-3)(4)
2:1(-4)3
3:(-4)1(-2)
4:32(-1)

(where x in row i means i is at x and $(-x)$ in row i means x is at i).

Then we would assign teams to patterns based on, say, their preferences for being home in the given slots, to get the schedule:

d:(-b)(-a)c
b:d(-c)a
a:(-c)d(-b)
c:ab(-d)

Any round robin schedule can be extended to a double round robin by *mirroring*. In a mirrored schedule, a round robin schedule is repeated, reversing the home and away teams. In the above example, a mirrored double round robin tournament would be:

d:(-b)(-a)cba(-c)
b:d(-c)a(-d)c(-a)
a:(-c)d(-b)c(-d)b
c:ab(-d)(-a)(-b)d

Scheduling systems differ in how each step is done, and in which order. For example, Russell and Leung (1994) began with steps 1 and 2 and used combinatorial design theory to give timetables to which they then assign teams in step 3 using enumeration. Cambell and Chen (1976) began by combining their 10 teams into 5 pairs and then used a latin square to implement steps 1 and 2. In their model, teams had no preferences on patterns, so the final step 3 was arbitrary. Schreuder (1992) also used combinatorial design for steps 1 and 2, and then formulated step 3 as a quadratic assignment problem. This was heuristically solved by reduced enumeration. Finally, Cain (1977) combined steps 1 and 3, assigning teams to patterns, and then completed step 2 by placing in all "forced games" and using enumerative methods for the remainder.

The main problem with this approach is the difficulty in finding a solution to steps 1 and 2. DeWerra (1980,1988) and others in combinatorial design (Mendelsohn and Rosa, 1985) have made great advances in determining feasible timetables, but there is no known characterization of all feasible timetables. The known timetables all revolve around some particular property, like maximizing or minimizing the number of alternations of home and away in the patterns, which may not be critical or appropriate in any particular application. Furthermore, finding feasible timetables is so laborious that very few are known for any given problem size.

In this work, we avoid the use of combinatorial design and generate many timetables by use of integer programming and enumeration. This gives us many feasible timetables (826 in this case) in a very short period of time. We begin with step 1, which we solve by enumeration combined with integer programming. Step 2 is then solved by integer programming. Finally, step 3 is solved by enumeration.

The factors that go into the decision of the order in which to do the steps include size of problem (what can be enumerated? what leads to problems that are too large to solve at all?), specificity of team requests, and desired objective. For instance, a schedule that depends on team travel may need to identify teams with patterns earlier in the process.

In our case, the size of the problem and the requirements on patterns made enumerating the feasible patterns relatively easy. Furthermore, since the team requests were relatively few, we could delay assigning teams to patterns until late in the process.

After going into detail on the particular requirements of the conference, we describe each step of the scheduling process.

3 ACC Schedule Requirements

Almost every paper in the literature of sports scheduling comments on the wide variety of constraints and objectives that occur in any real sports scheduling problem. For instance, Ferland and Fleurent (1991), when discussing scheduling the National Hockey League, begin with issues like availability of arenas, game pattern restrictions (no more than two games in three days, and so on), times between revisits, and distance restrictions, and end with "etc.", denoting a large number of other constraints that are critical to the feasibility of a schedule. This paper is no different since there are many constraints that go into a feasible ACC schedule.

The ACC consists of nine universities: Clemson (Clem), Duke (Duke), Florida State (FSU), Georgia Tech (GT), Maryland (UMD), North Carolina (UNC), North Carolina State (NCSt), Virginia (UVA), and Wake Forest (Wake). Every year, their basketball teams play a double round robin tournament in January and February (possibly including a game in December and/or March). Every team plays every other team twice, once at home and once away.

In general, every team will play twice in a week, often on Wednesday and Saturday. The exact day may differ, however, so we will refer to the two slots as the "weekday" and "weekend" slots. Since there are an odd number of teams, there will be one team with a bye in each slot. The length of the schedule is set up in such a way that there will be four conference games in each slot. This implies that the schedule is nine weeks long, and consists of 18 slots. The schedule always ends on a weekend, so the first slot is a weekday slot. The starting slot in 1997-98 will occur in December, and is denoted slot 0. The final slot is therefore slot 17.

Every team plays 8 slots at home, 8 away, and 2 bye slots. Teams generally value weekend slots higher than weekday slots. To be fair to the teams, we require each team to have 4 weekend home slots, 4 weekend away slots, and 1 weekend bye slot.

That gives the basic structure of the schedule. There are a number of additional constraints and objectives. We classify these in a way that corresponds to the algorithmic step at which they are enforced.

3.1 Pattern constraints

The pattern of home games and away games is important due to wear-and-tear on the teams, issues of missing class time, and spectator preferences. No team should play more than two away games consecutively, nor more than two home games consecutively. A bye is generally thought of as an away game, but a series Away-Bye-Away is not as bad as three consecutive aways. Similarly Home-Bye-Home-Home is not illegal, but definitely not preferred.

Just as a long series of away slots is not preferred, a long series of away weekend slots is also not liked. Rules similar to the above apply to weekend slots (no more than 2 at home consecutively and so on). In addition, the first five weekends are used for recruiting future student–atheletes, so each team must have at least two home or one home and one bye weekend among the first five. A bye may be acceptable here because the open slot could be used to schedule a non-conference home game.

The final week of the season has great importance for all teams, so no team can be away for both of the slots in the final week. Ideally, no team will be away for the first two slots.

These constraints are enforced in step 1 of our algorithm.

3.2 Game Count constraints

Another set of constraints that is enforced in step 1 is the basic requirement that as many teams play in a slot as is possible. This implies that there are four home teams in a slot, four away teams, and one team with a bye.

3.3 Team Pairing constraints

Since every team plays two games against every other team, the conference desires a large separation between meetings. A separation of nine slots can be achieved by mirroring a round robin schedule. We will see that a complete mirror is impossible in our case, but a similar concept will be used to create large separation. These separation constraints are enforced in step 1 and step 2. All others in this section are enforced in step 3.

The final weekend of the season is the most important slot, and is reserved for "rival" games: the pairings Duke-UNC, Clem-GT, NCSt-Wake, and UMD-UVA are traditionally played on that day. With the addition of FSU, which has no traditional rival yet, not all the rival games may be played, but if neither of the two teams in a rival pair has the bye in the last slot, then they must play each other. For example, if UMD has the bye in the last slot, then the pairings for the final slot are Duke-UNC, Clem-GT, NCSt-Wake, and FSU-UVA.

Duke-UNC is the most critical pairing in the schedule. It must occur in slot 17 and also needs to occur in slot 10. It is this latter requirement, caused by the preferences of a television network, that prohibits full separation of the pairwise meetings.

The preferences of the television networks are perhaps the most convoluted. The ACC has television contracts with both ESPN and Raycom and both broadcast games thoughout the season. ESPN made a request that UNC play Clem in slot 1, and otherwise is looking for appealing games in slots 1, 3, and 5. Raycom had a much more extensive set of preferences for February. Individual games are considered either A games, B games or neither. Table 1 summarizes the game evaluation, where an "A" entry denotes an A game, with A1 meaning the game is valued that way only if it appears during the week, and A2 denoting games that are valued only on the weekend. B games are similarly noted. Each slot in February is either an A-slot (best), B-slot, or Bad-slot (worst). If an A game appears in a slot, or if two B games appear, then the slot is an A-slot. If one B game appears, it is a B-slot. If no A or B games appear, it is a Bad-slot.

Table 1: Game Quality

	Away								
Home	Clem	Duke	FSU	GT	UMD	UNC	NCSt	UVA	Wake
Clem	—					B1/A2			
Duke		—		B	A	A2		B	B
FSU			—						
GT		B1		—	B	A1			B
UMD		A1		B1	—	A1		B1	
UNC	B1	A1		B	B	—			
NCSt		B				B	—		B
UVA		B						—	B1
Wake		B		B		B	B		—

Finally, since we require both Duke-UNC games to be in Februrary, some pairings must occur twice in January. To avoid popular pairings from occuring twice in January, the following pairings are specified to occur at least once in February: Wake-UNC, Wake-Duke, GT-UNC, GT-Duke.

3.4 Team Requirement constraints

Other slots are reserved for other team pairings and requirements. For 1997-98, these requirements are that Duke have a bye in slot 15, Wake not be at home in slot 16 and Wake have a bye in slot 0.

Clem, Duke, UMD, and Wake ended the 1996-97 season with an away game, so none should end with an away game in 1997-98. Similarly, Clem, FSU, GT, and Wake began with an away game in 1996-97 so should not begin with one in 1997-98. Neither FSU nor NCSt should end with a bye since they have recently done so, nor should UNC begin with a bye.

The constraints are enforced in step 3 of our algorithm, and are used to shorten the enumeration time.

3.5 Opponent Ordering Constraints

In almost every case, a team returns home after an away game. Therefore, there are no travel constraints per se. However, there is a restriction on a sequence of visits not related to travel distance. For 1997-98, teams have a requirement on the order in which they face UNC, Duke, and Wake. No team should play the three consecutively and no team should play UNC and Duke consecutively. This is due to the traditional and recent strength of the programs. This is enforced in Step 3.

3.6 Other constraints

While that is the entire list of explicit requirements, there are other aspects that are harder to quantify. In addition, there are several things listed above that are "not preferred" or otherwise permitted but not liked. One of us (GLN) has worked with the ACC and can evaluate schedules for some of these aspects. Schedules that pass his evaluation get sent on to the ACC office where the Associate Director for ACC Basketball is able to evaluate a small number of schedules, and choose the best.

4 Creating the Schedule

The previous section, with its laundry list of requirements, likes, and dislikes, makes the job of creating a schedule formidable. For the 1996-97 schedule, the ACC used a consultant who used constraint programming to create the schedule shown in Figure 1, where shaded entries are home games; rectangles represent weekend slots.

This schedule had different requirements than that outlined above and some of its defects, including the unevenness in weekends and the consecutive away sequences, were caused by some last minute flipping of games. Fundamentally, however, the schedule is a poor one due to lack of separation of meetings, an over-reliance on Home-Bye-Home(-Home) sequences, and an inability to meet television needs. We are not claiming, of course, that this performance inevitably results from constraint programming. A more concerted effort would doubtless create much more satisfactory schedules.

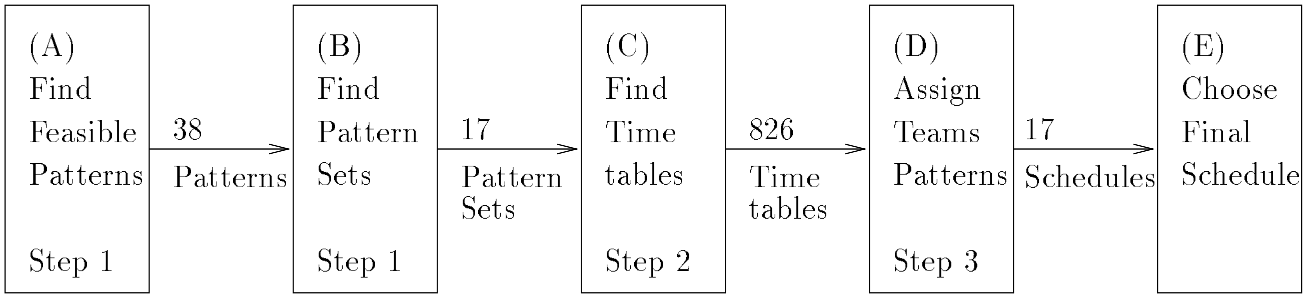
ACC BASKETBALL SCHEDULE (1997)

Slot	Clem	Duke	FSU	GT	UMD	UNC	NCSt	UVA	Wake
Dec	@UVA	FSU	@Duke	@UMD	GT	Bye	Wake	Clem	@NCSt
1/4	Bye	@GT	NCSt	Duke	UVA	@Wake	@FSU	@UMD	UNC
1/8	Duke	@Clem	UVA	Wake	@UNC	UMD	Bye	@FSU	@GT
1/11	@FSU	Wake	Clem	Bye	@NCSt	@UVA	UMD	UNC	@Duke
1/15	@UMD	Bye	@GT	FSU	Clem	NCSt	@UNC	@Wake	UVA
1/18	NCSt	UVA	Bye	@UNC	@Wake	GT	@Clem	@Duke	UMD
1/22	Wake	@NCSt	UNC	@UVA	Bye	@FSU	Duke	GT	@Clem
1/25	@UNC	@UMD	@Wake	NCSt	Duke	Clem	@GT	Bye	FSU
1/29	GT	UNC	UMD	@Clem	@FSU	@Duke	@UVA	NCSt	Bye
2/1	@NCSt	GT	@UVA	@Duke	Wake	Bye	Clem	FSU	@UMD
2/5	Bye	@Wake	@UNC	UVA	NCSt	FSU	@UMD	@GT	Duke
2/8	UMD	NCSt	GT	@FSU	@Clem	UVA	@Duke	@UNC	Bye
2/12	@Wake	@UVA	@UMD	Bye	FSU	@NCSt	UNC	Duke	Clem
2/15	UVA	@FSU	Duke	UNC	Bye	@GT	@Wake	@Clem	NCSt
2/19	@Duke	Clem	Bye	UMD	@GT	Wake	UVA	@NCSt	@UNC
2/22	FSU	Bye	@Clem	@NCSt	UNC	@UMD	GT	Wake	@UVA
2/26	UNC	UMD	@NCSt	@Wake	@Duke	@Clem	FSU	Bye	GT
3/1	@GT	@UNC	Wake	Clem	@UVA	Duke	Bye	UMD	@FSU

Minimum Difference between repeating games: 4
Length 3 or more Home (Bye=Away): 2
Length 3 or more Home (Bye=Home): 10
Length 3 or more Away (Bye=Away): 1
Length 3 or more Away (Bye=Home): 0
3 consecutive weekends at home: 1
3 consecutive weekends away: 0
Teams starting Away/Away: 0
Teams ending Away/Away: 1
Teams with away in first slot two years in a row: 1
Teams with away in last slot two years on a row: 2
Consecutive Duke/UNC (never A/A): 3

Figure 1: 1996-97 Official Schedule

We now describe each step of our algorithm with an emphasis on how it was used for the ACC. This process is illustrated by the flow chart in Figure 2 that also summarizes which constraints are enforced in each step.



Constraints

(A)	(B)	(C)	(D)	(E)
Pattern	Game Count	Team Pairing	Final Slot	Preferences
		Mirroring	TV Preferences	Intangibles
			Team Requirements	
			Opponent Ordering	

Figure 2: Algorithm Flow Chart

4.1 Step 1. Find Patterns and Pattern Sets

This step, occuring early in the process, is critical to the success of the system. If we generate many pattern sets that do not lead to feasible schedules, much time could be wasted. On the other hand, this step must be powerful enough to generate enough pattern sets to "survive" through the following steps. We divide this step into two phases.

The first issue is how to generate patterns that have a reasonable chance of being included in a feasible schedule, with a particular emphasis on the separation requirement. Clearly, a pattern needs to have an appropriate length (18), number of H, A, and B (8,8, and 2, respectively), number of H, A, and B in the weekend slots (4,4,1), limits on consecutive A and consecutive H (2), and other limits on basic structure given in the previous section.

A collection of nine such patterns would have little chance, however, of leading to a feasible schedule. It seems that almost invariably there is at least one, and usually many, pairs of teams whose meetings are insufficiently separated. We therefore impose a restriction that the schedule be "almost-mirrored".

The pattern for a mirrored schedule for this problem has a specific structure: Slot i is an H (A, B respectively) if and only if slot $i + 9 \bmod 18$ is an A (H, B respectively) . So slots 0 and 9 mirror each other, as do 1 and 10 and so on. In particular, slots 8 and 17 mirror each other. Since we are required to have Duke play UNC in both slots 10 and 17, we will require all teams to mirror in slots 10 and 17. We do that by "flipping" slots 8 and 10 in a mirror schedule, so slot 1 now mirrors 8 and 10 mirrors 17. We can also flip other slots in order to generate additional patterns. We were most successful by also flipping slots 7 and 9, so slot 0 mirrors slot 7 and 9 mirrors 16. Note that in order to have each team with a weekend bye, it is necessary to have even slots mirror odd slots.

With this restriction, we can generate all feasible patterns by generating patterns of length 9 consisting of Hs, As and one B, mirroring each, and flipping slots 8 and 10, and 7 and 9. There are only $9(2^8) = 2304$ such patterns, so we can enumerate them, discarding infeasible patterns. This is Box A in Figure 2. With the restrictions we have, we generated 38 feasible patterns. There are some simplifications made. For example, since we know Duke has both a bye in period 15 and is home in period 17, we can discard any pattern with a bye in period 15 and an away in period 17.

We need now to find sets of nine patterns that have a reasonable chance of resulting in feasible timetables, see Box B in Figure 2. We find these sets by integer programming. There are some obvious constraints on the patterns: for instance, in every slot, there must be four chosen patterns with an H, four with an A, and one with a B. We would, in general, also like patterns that differ in many places so that there are many places to assign games in Step 2. With the mirroring we require, however, all patterns differ in at least 4 places, which most of the time turns out to be sufficient for later stages to be successful.

There are less preferred patterns in our set. For instance a pattern that corresponds to a team beginning with two away games is definitely not preferred. It turns out with the restrictions we have so far, there is no feasible pattern set that does not include at least one such pattern, but we restricted ourselves to only one such bad pattern.

This gives the following integer program. Let P be the set of patterns, and T be the set of slots. Create a variable x_i for each pattern i that will be 1 if the pattern is in the set and 0 otherwise. For pattern i and slot k let h_{ik} be 1 if pattern i has an H in slot k; 0 otherwise. Similarly, let a_{ik} be 1 if pattern i has an A in slot k; 0 otherwise. Finally, let b_i be 1 if pattern i is a less preferred pattern, and 0 otherwise. The formulation is

$$
\begin{array}{llcll}
\text{Minimize} & \sum_{i\in P} b_i x_i & & & \\
\text{Subject to} & & & & \\
\textbf{(Home)} & \sum_{i\in P} h_{ik} x_i & = & 4 & \text{for all } k \in T \\
\textbf{(Away)} & \sum_{i\in P} a_{ik} x_i & = & 4 & \text{for all } k \in T \\
 & x_i & \in & \{0,1\} & \text{for all } i \in P
\end{array}
$$

We generated all feasible solutions by optimizing with respect to this objective, then including a constraint that precludes the resulting solution ($\sum_{i\in S} x_i \leq 8$ for solution set S worked fine) and reoptimizing until infeasibility. In this case, infeasibility means that the objective goes to 2, denoted two less prefered patterns. This small (38 variable, 18 constraint) integer program solves in under 1 minute on a Sun Sparcstation 20, with CPLEX©version 4.0 (CPLEX, 1995) as the integer programming solver. This led to 17 pattern sets.

4.2 Step 2. Finding Timetables

Given a pattern set, we next assign games, see Box C in Figure 2. This, again, is done by integer programming. In this case, we have a variable x_{ijk} denoting pattern i plays at pattern j in slot k. This variable is only defined if the ith pattern has an A in its kth position, and the jth pattern has an H in its kth position. Let S be the set of patterns and F be the set of feasible (i, j, k) triplets.

This integer program has an arbitrary objective, since at this point we still have not identified teams with patterns, so there are no preferences to include, and a small number of constraints. Of course, every pattern must play at every other pattern (the Visit constraints in the following formulation), and the "almost-mirroring" condition must hold (Mirror constraints). The final set of constraints force a team to play at most one game in a slot (OneGame constraints). The resulting formulation is

$$
\begin{array}{llcll}
\text{Minimize} & \sum_{(i,j,k)\in F} x_{ijk} & & & \\
\text{Subject to} & & & & \\
\textbf{(Visit)} & \sum_{\{k:(i,j,k)\in F\}} x_{ijk} & = & 1 & \text{for all } i \in S, j \in S, i \neq j \\
\textbf{(OneGame)} & \sum_{\{j:(i,j,k)\in F\}} x_{ijk} + & & & \\
 & \sum_{\{j:(j,i,k)\in F\}} x_{jik} & \leq & 1 & \text{for all } i \in S, k \in T \\
\textbf{(Mirror)} & x_{ijk} & = & x_{jik'} & \text{for all } (i,j,k) \in F, (j,i,k') \in F, \\
 & & & & \quad k \text{ and } k' \text{ must mirror} \\
 & x_{ijk} & \in & \{0,1\} & \text{for all } (i,j,k) \in F.
\end{array}
$$

This leads to an integer program with 234 constraints and approximately 300 variables (the exact number depends on the pattern set). Again this can be solved quickly, in under 10 minutes on a Sparcstation 20 using CPLEX©4.0 as the integer programming solver and constraints can be added to generate all feasible solutions. The 17 feasible pattern sets generated 826 feasible timetables.

4.3 Step 3. Assigning Teams to Patterns

Our next step is to take a timetable and assign teams to patterns in the pattern set, see Box D in Figure 2. This process, the most time consuming part of the procedure, is done through enumeration. There are $9! = 362,880$ assignments of teams to patterns for each timetable. Each of these is checked for feasibility (are the finishing games ok? are there consecutive games against Duke-UNC?) and for preferred aspects (how many slots are A-slots for TV? how many are bad slots?).

We needed to look at $826 * 362,880 = 299,738,880$ possible assignments, which took about 24 hours on a Sun SPARC-20 workstation. This lead to 17 feasible schedules. From those schedules, we used dominance aspects to reduce the list to three schedules for submission to the ACC. The ACC, consulting with their television partners, then chose one as the official schedule. This process is Box E in Figure 2. The accepted schedule is shown in Figure 3.

5 Possible Extensions

There are a number of interesting extensions and questions. One important question is to characterize the feasible pattern sets. We have used the bare minimum constraints in step 1, and many of the generated sets are infeasible for step 2. It would be interesting to have a characterization of feasible pattern sets, or to show that no compact characterization exists by showing, for instance, that the question of determining if a set is feasible is NP-complete.

ACC BASKETBALL SCHEDULE (1998) *Official Schedule, Accepted: November 1996*

Slot	Clem	Duke	FSU	GT	UMD	UNC	NCSt	UVA	Wake
Dec	UMD	UVA	UNC	NCSt	@Clem	@FSU	@ GT	@Duke	Bye
1/3	UNC	@UMD	@NCSt	Bye	Duke	@Clem	FSU	Wake	@UVA
1/7	@Wake	NCSt	@UMD	@UNC	FSU	GT	@Duke	Bye	Clem
1/10	Bye	@FSU	Duke	Wake	@NCSt	UVA	UMD	@UNC	@ GT
1/14	FSU	@Wake	@Clem	@UVA	UNC	@UMD	Bye	GT	Duke
1/17	@Duke	Clem	@ GT	FSU	@Wake	Bye	@UVA	NCSt	UMD
1/21	UVA	Bye	Wake	UMD	@ GT	@NCSt	UNC	@Clem	@FSU
1/24	@UMD	@UVA	@UNC	@NCSt	Clem	FSU	GT	Duke	Bye
1/28	@UNC	UMD	NCSt	Bye	@Duke	Clem	@FSU	@Wake	UVA
1/31	NCSt	GT	Bye	@Duke	UVA	@Wake	@Clem	@UMD	UNC
2/4	@ GT	@UNC	@UVA	Clem	Bye	Duke	Wake	FSU	@NCSt
2/7	Wake	@NCSt	UMD	UNC	@FSU	@ GT	Duke	Bye	@Clem
2/11	Bye	FSU	@Duke	@Wake	NCSt	@UVA	@UMD	UNC	GT
2/14	@FSU	Wake	Clem	UVA	@UNC	UMD	Bye	@ GT	@Duke
2/18	Duke	@Clem	GT	@FSU	Wake	Bye	UVA	@NCSt	@UMD
2/21	@UVA	Bye	@Wake	@UMD	GT	NCSt	@UNC	Clem	FSU
2/25	@NCSt	@ GT	Bye	Duke	@UVA	Wake	Clem	UMD	@UNC
2/28	GT	UNC	UVA	@Clem	Bye	@Duke	@Wake	@FSU	NCSt

Minimum Difference between repeating games: 7
Length 3 or more Home (Bye=Away): 0
Length 3 or more Home (Bye=Home): 3
Length 3 or more Away (Bye=Away): 3
Length 3 or more Away (Bye=Home): 0
3 consecutive weekends at home: 0
3 consecutive weekends away: 0
Teams starting Away/Away: 1
Teams ending Away/Away: 0
Teams with away in first slot two years in a row: 0
Teams with away in last slot two years on a row: 0
Consecutive Duke/UNC (never A/A): 0
A slots: 4; B slots: 2; Bad slots: 2.

Figure 3: 1997-98 Official Schedule

Furthermore, not every sports schedule is a round-robin, thus how can feasible patterns and pattern sets be characterized for arbitrary game requirements?

Another related question involves the computational effort at each of the steps. Can any of our integer programs or enumerative steps be replaced by more efficient algorithms? The limiting factor in our approach is the final need to enumerate the possible assignments of teams to patterns of the timetable. It would be possible to extend this approach directly to 10 or 11 teams by generating fewer timetables by, for instance, having more restrictions in the pattern or game assignment steps. It would also be possible to extend these results to slightly larger leagues by a more aggresive pruning of infeasible assignments. For instance, if we first assigned Duke and UNC to patterns, then we could immediately identify cases where some team plays each consecutively and ignore any such permutation. Schreuder (1992) points out that the final assignment of teams to patterns is a quadratic assignment problem, so any methods for that problem apply here. For larger problems, a heuristic approach, like a greedy assignment followed by 2-exchanges, may lead to adequate schedules.

Also, note that while the majority of the constraints used to create this schedule were treated as constraints, there are many opportunities to use cost coefficients instead. For instance, while we required no more more than one "less preferred" pattern in our set when finding the pattern set, we could also have had a richer evaluation that gives a cost to each pattern and then minimize the total cost of the pattern set. Such flexibility is particularly valuable in problems that have many feasible schedules.

6 Conclusions

Of course, our exposition represents an impossibly clean view of the process. In reality, we did not generate just 300 million possible schedules to get down to 17 then 3 then 1. We actually generated, conservatively, 10 billion possible schedules and iterated many times between the authors, and between the authors and the ACC. Many of the restrictions that are listed in section 3 actually arose late in the process, as the ACC and the television networks clarified and modified their requests. Clearly, the ACC, and we would guess most conferences, finds it difficult to articulate what it really requires in a schedule. The best way to determine those needs is to have a system that can quickly generate sample schedules and have the decision makers update their requirements based on those schedules. The system we created is capable of turning around schedules in a day, and that speed was instrumental in its success.

The main algorithmic advance of this paper is to point out that an appropriate combination of enumeration and integer programming is sufficient to generate many suitable patterns, replacing the tedious and difficult combinatorial design techniques. The success of this approach is shown by creating real schedules, meeting and exceeding the requirements of a real college basketball conference.

Acknowledgement: *The authors would like to thank Fred Barakat of the Atlantic Coast Conference for his patience and efforts in helping us understand the real issues in scheduling the conference and in pointing out the flaws of our early schedules. We would also like to thank the referees for helpful and prompt suggestions for improvement.*

References

ARMSTRONG, J. AND R.J. WILLIS. 1993. Scheduling the cricket world cup — a case study, *Journal of the Operational Research Society* **44**, 1067–1072.

BALL, B.C. AND D.B. WEBSTER. 1997. Optimal scheduling for even–numbered team athletic conferences, *AIIE Transactions* **9**, 161–169.

BEAN, J.C. AND J.R. BIRGE. 1980. Reducing traveling costs and player fatigue in the National Basketball Association, *Interfaces* **10**, 98–102.

CAIN, W.O., JR. 1977. A computer assisted heuristic approach used to schedule the major league baseball clubs, in *Optimal Strategies in Sports*, S.P. Ladany and R.E. Machol (eds.), North Holland, Amsterdam, 32–41.

CAMPBELL, R.T. AND D.–S. CHEN. 1976. A minimum distance basketball scheduling problem, in *Management Science in Sports*, R.E. Machol, S.P. Ladany, and D.G. Morrison (eds.), North–Holland, Amsterdam, 15–25.

CPLEX OPTIMIZATION, INC. 1995. "Using the CPLEX©Callable Library", Tahoe, NV.

DE WERRA, D. 1980. Geography, games, and graphs, *Discrete Applied Mathematics* **2**, 327–337 (1980).

DE WERRA, D. 1988. Some models of graphs for scheduling sports competitions, *Discrete Applied Mathematics* **21**, 47–65 (1988).

FERLAND, J.A. AND C. FLEURENT. 1991. Computer aided scheduling for a sports league, *INFOR* **29**, 14–24.

MENDELSOHN, E. AND A. ROSA. 1985. One–factorization of the complete graph – a survey, *Journal of Graph Theory* **9**, 43–65.

RUSSELL, R.A. AND J.M LEUNG. 1994. Devising a cost effective schedule for a baseball league, *Operations Research* **42**, 614–625.

SCHREUDER, J.A.M. 1992. Combinatorial aspects of construction of competition Dutch Professional Football Leagues, *Discrete Applied Mathematics* **35**, 301–312.

WILLIS, R.J. AND B.J. TERRILL. 1994. Scheduling the Australian state cricket season using simulated annealing, *Journal of the Operational Research Society* **45**, 276–280.

WRIGHT, M. 1994. Timetabling county cricket fixtures using a form of tabu search, *Journal of the Operational Research Society* **45**, 758–770.