Isaac Schaal
CS111 - Final Project
December 15th, 2017

Mathematical Modeling for Sports Scheduling

INTRODUCTION

Sporting events have been a form of entertainment enjoyed by those around the world for centuries. Many sports, like football, basketball and cricket, have national and international leagues and tournaments, which attract millions of fans. One fundamental problem encountered when organizing tournaments is how to create the schedule. With a variety of factors and constraints to incorporate, and several different aspects to optimize, the area of sports scheduling is rich for applications of mathematical modeling. In this paper, I present several different subproblems of mathematical modeling. I start with important definitions relevant to sports scheduling. I then identify basic ways that schedules can be represented visually using graph theory. I then move on to a specific optimization problem that necessitates a more complicated model, incorporating aspects of advanced linear programming. Finally, I discuss the benefits and drawbacks of this model and ways that it can be modified to fit to real life.

DEFINITIONS

Throughout the paper, I will use specific terms relevant to sports scheduling. A specific set of definitions is standard in the field. (Ribeiro, 2012). We will start with a basic tournament, which is played by an even number n teams. Each event is a game played between two teams. The basic tournament design that we will be considering is a round robin tournament. In a single round robin tournament, each team plays each other team exactly once. This can be extended to include multiple phases. For example, in a double phase round robin tournament, each team must play every other team exactly twice and must play them exactly once in each phase. Another possible aspect of scheduling is the location of the game. In some tournaments, each team has a home venue. Of the two teams playing, one is the home team and the other is the away team, and the game is played at the home team's venue. In double round robin tournaments, it is often stipulated that the second time two teams play each other, the designation of home and away must be switched.

MODELING WITH GRAPHS

For our first model, we will represent a tournament using graphs. The AIM explains how graphs can be used to model sports scheduling problems. (Malkevitch, No Date) The complete graph K_n, where each team is a node and each game is an edge, shows a round robin tournament between n teams. This can be seen in Figure 1.

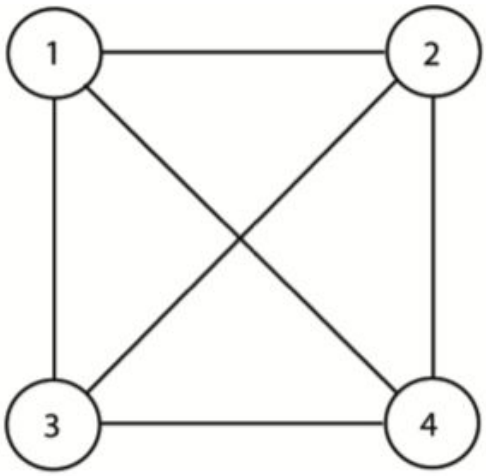
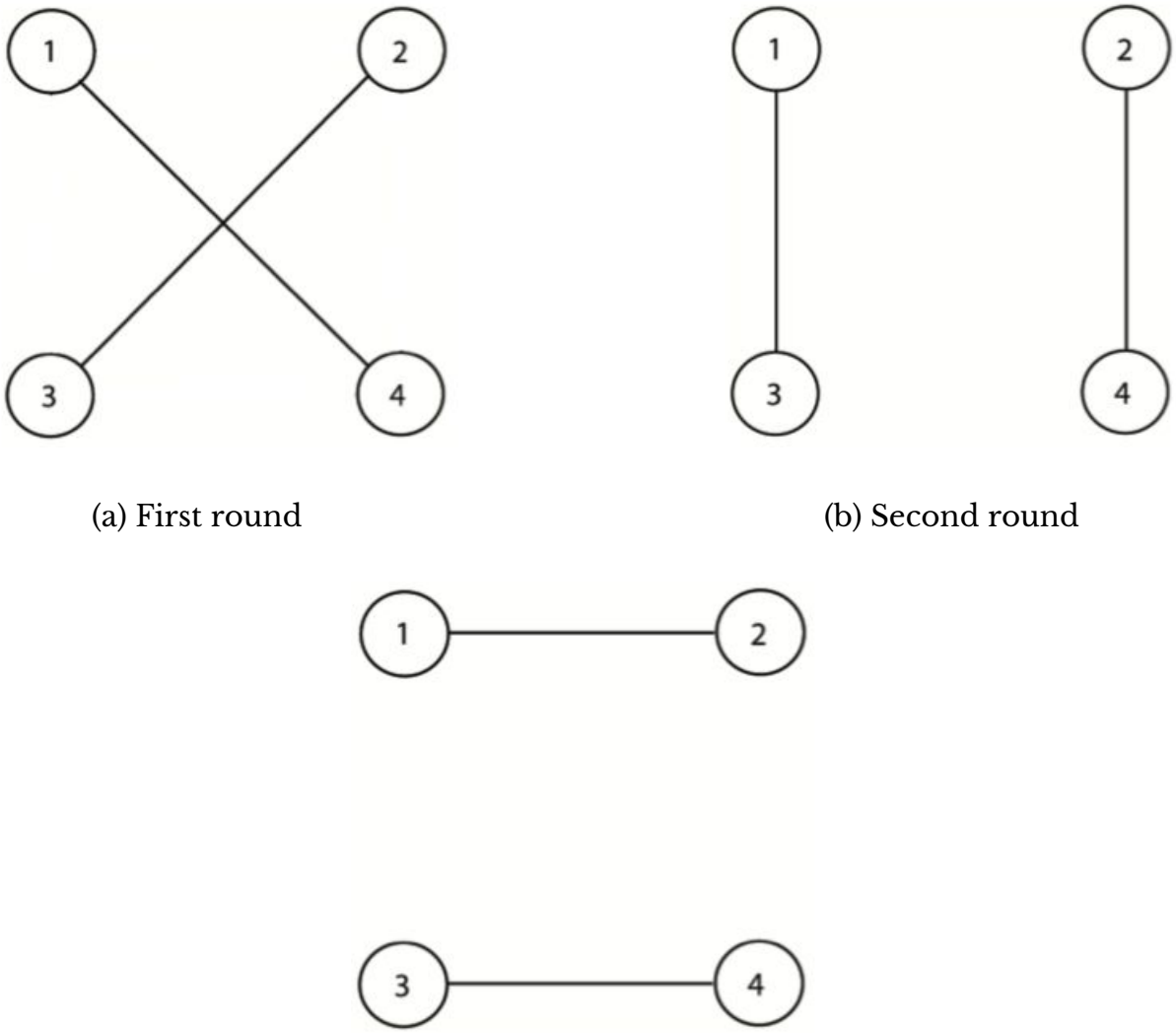


FIGURE 1. The graph K_4, showing a single round robin tournament.

Graphs such as Figure 1 can then be used in order to solve scheduling problems. We must now determine which games can be played at what times, in order to create our schedule. To do this, we must create a 1-factorization of Figure 1. We do this by creating an edge coloring with n -1 colors, which partitions the edges into n-1 groups, called 1-factors, each with $n / 2$ non-adjacent edges. Each of the 1-factors consist of the games that can be played in each round. An ordered 1-factorization of graph K_n is a timetable that will work for our single round tournament. Figures 2 (a) to 2 (c) show an example of this with K_4.



(a) First round

(b) Second round

(c) Third Round

FIGURE 2. A timeline of a single round robin tournament made using 1-factors of K_4.

Using this method, we have easily created a schedule for our tournament. Each team is playing each round, and after three rounds, all teams have played all other teams. However, there are several ways that we can extend it even further. For example, what would we do if we had an odd number of teams? It is impossible to have each team playing each round, as one team will be left without an opponent. We can adopt our model to account for this possibility.

In sports scheduling, there is a concept known as an open slot[1]. If a team has an open slot, they don't play any team that round and get a rest. In order to model this, we would create the graph K_{n+1} and have one node be designated as the open node. If a team ends up on the schedule playing that node, they get a open slot. For example, we could use Figure 2 to create a schedule with n = 3 teams, and designate team 4 as the open slot team. During the first round, teams 1 and 3 would play each other, while team 1 gets an open slot. We can also use this graph model to schedule a tournament that has home and away teams. We simply change Figure 1 into a directed graph, as seen in Figure 3.

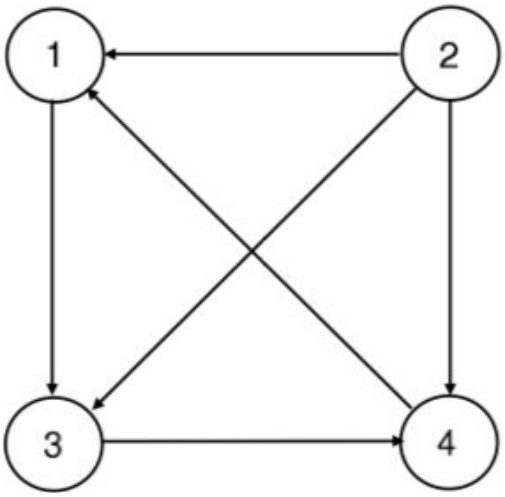


FIGURE 3. A directed graph of K_4, showing a single round robin tournament with home and away teams.

In this model, a directed edge from node i to node j indicates that i is the home team and j is the away team. Thus, when team 1 plays team 3, team 1 will be the home team.

Graphs such as these are very useful for creating simple round robin tournaments. A double round robin tournament could be created by having the first round made using the 1-factorization of Figure 3, while the second round would be the same except with of all directed edges reversed. Thus, each team would play each other team twice, once as the home team and once as the away team. The main benefit of this model is that it is very easy to understand and implement. People with minimal mathematical knowledge could use this model to create their own round robin tournament, so this model is especially useful for small scale sports

[1] Some people mistakenly interpret this "open slot" to be a "bye". However, a bye is a concept specific to an elimination tournament, and implies that the team with a bye does not play and advances to the next round. These are often given to the best teams as a reward, but are sometimes assigned randomly. It is improper to refer to an open slot as a bye.

scheduling. For those with some coding experience, the NetworkX module in python can be used to create graphs such as the figures above, which could be useful for creating larger tournaments. However, with larger tournaments, there are more factors to consider. With huge amounts of money being invested into sporting schedules like the NBA or NFL, having graphs that are optimized on key outcomes can be beneficial. In order to most efficiently do this, we will need to upgrade our model.

THE TRAVELING TOURNAMENT PROBLEM

We will focus on a problem that forces us to upgrade our model in order to solve it. The problem is known as the traveling tournament problem. (Ribeiro, 2012) In this problem, we must create a double phase round robin tournament. Each team also has a home venue, and if they play at another venue (are the away team) they must travel to this venue. These teams may bring hundreds of players, coaches, trainers and staff, and thus minimizing the distance that they must travel throughout the season can dramatically lower costs as compared to a schedule that simply has each team playing the other twice. Thus, we must optimize by minimizing total distance traveled by all teams. To do this, we can set up an objective function and a set of constraints and use linear programming methods to optimize.

There are a few more specifics of the traveling tournament problem that we must consider. We are given n teams numbered 1 to n and distances d_{ij} between two venues (for example, if New York is team 1 and Philadelphia is team 2, then $d_{1,2}$ is the distance between New York and Philadelphia) for every $i, j = 1, \ldots, n$ (with $d_{ij} = 0$ if $i = j$). We are also given two integer numbers, L and U, such that all sequences of consecutive home games or consecutive away games played by any team is at least L games and at most U games. Traditionally, L=1 and U= 3. Every team must also begin the tournament at their home venue and must return to their home venue after their last away game. Finally, no repetitions (two teams playing against each other in two consecutive rounds) are allowed. We must first define two decision variables

x_{ijk} =1, if team i plays away against team j in round k,
0, otherwise;

and

y_{tijk} =1, if team t travels from the venue of team i to that of team j between rounds k and $k + 1$,
0, otherwise.

For example, let New York be team 1, Philadelphia be team 2, and Washington be team 3. In round 1, New York plays at Philadelphia. $x_{1,2,1} = 1$ because team 1 plays against team 2 in round 1 and $x_{1,3,1} = 0$ because team 1 did not play team 3 in round 1. In round 2, New York plays at Washington, and $y_{1,2,3,1} = 1$ because New York traveled between Philadelphia and Washington.

We can then define our objective function to be minimized.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} \cdot x_{ij1} + \sum_{t=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{2n-3} d_{ij} \cdot y_{tijk} + \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ji} \cdot x_{ij,2n-2}$$

The objective function accounts for the total distance traveled by separating it into three parts: the distance traveled by teams playing away in the first round, the distance traveled after the first and before the last game by all teams, and the distance traveled by teams returning home by teams who played away in the last round. By minimizing this objective function, we obtain the optimal values of the x_{ijk}'s and the y_{tijk}'s that form a complete tournament schedule.

However, in order to optimize this function, we must set up a system of constraints. A full list of these constraints and their explanations can be found in index (a).

The constraints effectively and creatively create a system that enforces the rules of a double phase round robin tournament upon our minimization function. For example, the rule that a team can't play itself, or that a team must travel between the venues that it is playing at in two consecutive games. We can then use optimization methods to find the optimal schedule of games to minimize overall travel distance across the whole tournament. In order to do this, traditionally we would use an optimization method like the simplex method, which finds all possible optimal points and calculates what the outcome (total travel distance) is at those points, and finds the minimum. However, this would be forgetting a fundamental aspect of the system we are modeling.

The variables in this model are discrete and not continuous. In fact, x_{ijk} and y_{tijk} are either 0 or 1 at all *t,i,j,* and *k.* We must have an additional set of constraints, which can be found in the appendix. Thus our problem is no longer a linear programming problem, but an integer programming problem.

Integer programming adds an entirely new element to the linear programming that we previously learned. While the concept is clear (the variables must be integers, or in our case binary values), the implications are immense. We can no longer find the possible optimal points which are the intersection of the edges of our feasible region and then find the optimum of those, as those intersection can occur at non-binary points. One could attempt to solve this problem by finding all possible points and evaluating the objective function at each one, but this becomes incredibly complicated with only a few variable and is not solvable in any reasonable time by modern computational techniques (this type of problem is called NP-hard). (Trick, 2004).

There are a variety of approaches to solve this problem. The first is called linear programming relaxation. Linear programming relaxation relies on the idea that integer programming is very similar to linear programming. The approach works by changing the constraints that the variables (in our case x_{ijk} ,y_{tijk}) must be 0 or 1 to the constraint that they must be greater than or equal to 0 and less than or equal to 1. This problem is then solved as a normal linear relaxation problem. However, this only outputs a bound on what the integer programming solution would be. It does not give us the point we want, but instead tells us that whatever point is optimal with the integer constraint will produce an output less optimal than what our relaxation problem outputs. Unfortunately, linear program relaxation gives us very weak results, and are not very informative. (Ribeiro, 2012)There are several methods that can, however, make this problem solvable (at least for relatively small n). Trick proposes a method of solving this using the Mosel language with the XPRESS - MP package. (Trick, 2005) This language and package helps to solve integer programming problems and has been proven to find the optimal solution to the traveling tournament problem. In order to do so, a user must input the objective function and constraints, and the package will output a schedule. However, an optimal solution for large values of n (for example, $n = 16$) has never been found. Running on 20 processors, the solution for $n = 8$ took over 4 days. Further research is being conducted in an attempt to solve the $n = 10$ case. Thus, this model is limited both by the scale at which it can be applied and also by its complications, as many people would not be able to understand or apply this model.

We can now turn our attention to the key assumptions of this model, as well as ways that it can be adapted for real life situations. Firstly, we are assuming that teams on an away game streak will travel immediately from the venue of the team they previously played to the venue of the team they will play. In real life, this assumption may not be true. For example, a team may want to return home for the night if doing so was not too far out of their way. This would add a complication to the model as the distance matrix would need to be updated and with the updates only called upon in specific situations. We also must examine the

assumption that we are trying to minimize travel distance. We could just as easily want to minimize time or cost involved with travel. Teams could fly or drive depending on the distance which would have varied effects for minimizing time or cost. Finally, the model makes the assumption that away streaks of more than 3 games (constrained by the U value) are to be avoided. This does prevent situations where a team may not be at home for a long stretch of time, but allowing longer away streaks could dramatically decrease travel distance in certain scenarios. The ideal U value is up to debate.

While not being able to produce provably optimal results for large values of n, this model can be used to help solve the incredibly difficult problem of scheduling, and it is at the forefront of the field. Researchers who developed and apply this model have gotten contracts for scheduling some of the world's largest sports scheduling problems, like the Major League Baseball schedule in the US. (Hoshino, 2013) When adapting this model for real life conditions, many other constraints and factors can come into play. Certain teams that have rivalries or histories may be emphasized to meet in certain locations, longer streaks of away games may be allowed, and considerations of holidays or stadium availability must be considered. Even more complicated tournament designs, like mirrored schedules or certain routes that must be followed can also be incorporated. Modelers must also meet demands of distributors, like not having popular games at the same time so that TV networks can have the maximum number of viewers. These factors and more go into the design and implementation of major sports scheduling models, and innovations can have large scale impacts on a huge industry. Understanding the diverse field of mathematical modeling and its implications is essential to solving problems, as multiple methods and models are often combined to create and understand a final solution.

BIBLIOGRAPHY

Hoshino, R., and Kawarabayashi, K. (2013). Graph theory and sports scheduling. Notices of the American Mathematical Society. http://www.ams.org/notices/201306/rnoti-p726.pdf

Malkevitch, J. (No Date). Mathematics and Sports. American Mathematical Society. http://www.ams.org/samplings/feature-column/fcarc-sports

Ribeiro, J. (2012) Sports Scheduling: Problems and Applications. International Transactions in Operational Research, 19(1-2): 201-226 https://pdfs.semanticscholar.org/4af6/5185dbf7f201aa1171ba46ada4cb02f1dd36.pdf

Trick, M. (2004) Using Sports Scheduling to Teach Integer Programming. INFORMS Transactions on Education 5(1):10-17. https://pubsonline.informs.org/doi/pdf/10.1287/ited.5.1.10

Trick, M. (2005). Formulations and Reformulations in Integer Programming. CPAIOR. http://mat.gsia.cmu.edu/trick/formul04.pdf

INDEX

(a) INITIAL CONSTRAINTS

$$x_{iik} = 0, \qquad i = 1, \ldots, n, \quad k = 1, \ldots, 2n-2$$

The first set of constraints enforce that no team may play itself.

$$\sum_{j=1}^{n} (x_{ijk} + x_{jik}) = 1, \qquad i = 1, \ldots, n, \quad k = 1, \ldots, 2n-2$$

The second set of constraint enforce that each team must play exactly once each round.

$$\sum_{k=1}^{2n-2} x_{ijk} = 1, \qquad i, j = 1, \ldots, n : i \neq j$$

The third set constraints enforce that each team will play an away game against each opponent exactly once.

$$L \leq \sum_{\ell=0}^{U} \sum_{j=1}^{n} x_{ij,k+\ell} \leq U, \qquad i = 1, \ldots, n, \quad k = 1, \ldots, 2n-2-U$$

The fourth set of constraints enforce that in all sequences of *U*+1 consecutive games, a team will play at least *L* and at most *U* away games.

$$x_{ijk} + x_{jik} + x_{ij,k+1} + x_{ji,k+1} \leq 1, \qquad i, j = 1, \ldots, n, \quad k = 1, \ldots, 2n-3$$

The fifth set of constraints enforce that there will be no repetitions.

$$z_{iik} = \sum_{j=1}^{n} x_{jik}, \qquad i = 1, \ldots, n, \quad k = 1, \ldots, 2n-2$$

The sixth set of constraints create a new variable, z_{iik}, and enforces that z_{iik} is equal to 1 if team *i* plays in round *k*.

$$z_{ijk} = x_{ijk}, \qquad i, j = 1, \ldots, n : i \neq j, \quad k = 1, \ldots, 2n-2$$

The seventh set of constraints enforce that that team i should be at the home venue of team j if the former plays away against the latter. This makes z_{ijk} =1 when team i is playing at the home venue of team j during game k.

$$y_{tijk} \geq z_{tik} + z_{tj,k+1} - 1, \qquad t, i, j = 1, \ldots, n, \quad k = 1, \ldots, 2n-3$$

The eight set of constraints enforce that team t travels from the home venue of team i to that of team j if it plays in the respective cities in two consecutive rounds.

(b) BINARY CONSTRAINTS

$$x_{ijk} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \quad k = 1, \ldots, 2n-2,$$

$$z_{ijk} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \quad k = 1, \ldots, 2n-2,$$

$$y_{tijk} \in \{0, 1\}, \qquad t, i, j = 1, \ldots, n, \quad k = 1, \ldots, 2n-2$$

These three constraints enforce that x, z, and y must be binary variables (either 0 or 1).