National Basketball Association Scheduling Simulation

21-393 Final Project, Fall 2016

Shengqi Chai, Yutong Li, Liyunshu Qian, Ming Yang

Department of Mathematics
Carnegie Mellon University
Pittsburgh, PA 15213

Table of Contents

I. Abstract

Sport scheduling is a complex task in the presence of a myriad of conflicting requirements and preferences. In this work, our primary goal is to find a feasible and approximately optimal schedule in terms of travel distance for the 30 teams in National Basketball Association. We focus on the schedule for the regular season, which usually spans over a 5-month duration. Existing approaches to build a schedule from scratch tends to suffer from substantial runtime overhead. In particular, it is computationally infeasible to solve the problem directly using linear programming and constraint programming due to the complicate formats and rules for NBA scheduling. Thus for the sake of simplification, we adopted assumptions so that integer programming is applicable. Additionally, we approached the problem using divide and conquer to reduce computational complexity. Apart from Operations Research techniques, methods from Machine Learning and Data Collection are also exploited in finding the solution. Our approach yields reliable schedules in a reasonable runtime, and the algorithm should be applicable, with slight modifications, to any scheduling problems in single-round robin or double-round robin fashion.

II. Problem Background

National Basketball Association is the preeminent men's professional basketball league in North America, and is widely considered as one of the best basketball leagues in the world. It now consists of 30 teams, with 29 teams from the United States and 1 team from Canada. The NBA and its teams earned in excess of $5 billion in revenue in 2015, with a large portion coming from television networks broadcasting the games and from gates receipts [1].

In an official NBA schedule, each team plays 82 games each season, including four games each against its four in-division opponents (16 games), four games each against six out-of-division opponents (24 games), three games each against the remaining four out-of-division opponents (12 games) and two games each against the 15 teams in the opposite conference (30 games). The four out-of-division opponents that are played three times each are determined by a five-year rotation.

In addition to this set of hard constraints, NBA schedule needs to be fair to every team. For example, instances of back-to-back games should be reduced to all-time lows, and either too many or too few consecutive home or away games in any period is not preferred. The schedule also needs to account for a range of soft constraints, such as arena availability, broadcasting preference, and holiday conflicts. In essence, scheduling can have heavy impact on revenues. Television networks generally require that 'high-quality' games to be spread out in the season and take place on special dates. Having certain rivals with high commercial values, such as

Celtics vs. Lakers or Cavaliers vs. Warriors, play on Thanksgiving Day or Christmas usually brings a substantial amount of extra profit. Furthermore, in consideration of individual team's income from ticket sales and television deals, there should be a reasonable number of home games on weekends for each team throughout the season.

Professional approaches to NBA scheduling is confidential due to its extremely high value to the league as well as all the parties affiliated. In fact, the schedule was produced manually until recent years. In the next section, we will explain our approach to the problem, which produces a schedule that is fair to every team and meanwhile satisfies as many constraints as possible.

III. Solution

1. Assumptions and Simplification

Our main objective function is to minimize the total travel distance of all teams, while maintaining fairness by ensuring that each team will travel similar distances during the season (e.g. we don't want some teams to travel 100,000 miles a season while having other teams traveling only 50,000 miles). Meanwhile, we require the schedule to minimize back-to-back games and break up consecutive home/away games. Other constraints such as broadcast requirement or game preference on specific nights are treated as softer constraints that will be exploited once we have an initial schedule that optimizes our primary objective function.

The official NBA schedule is not symmetric due to the fact that certain teams play against each other 3 times per season. This can be a real technical difficulty in constructing the schedule especially in terms of computational complexity. Thus we make assumptions for simplification that each team plays 4 games each against its in-division opponents (16 games) and 2 games each against all other opponents (50 games), which total up to 66 games. Under this assumption, the NBA schedule can be considered as a double-round robin tournament. In other words, the second half of the schedule is an approximate mirror of the first half (if the Cavaliers plays the Warriors at home in the first half of the season, then it should play an away game against the Warriors in the Second half).

Additionally, constructing an initial feasible schedule with 30 teams is computationally infeasible, as it gives billions of candidates to choose from. Thus, we use divide and conquer to construct our schedule. In particular, we treat each division as a single team and produce feasible minimal distance schedule of divisions. We then complete the schedule with specific teams based on our division schedules. This approach will greatly reduce our computational complexity so that the schedule can be produced with our algorithm in a relatively short amount of time.

2. Data Collection

Python web-crawlers are created to retrieve information about the location of NBA teams and distances between all pair of teams from Google Map. We also scrape location coordinates of all teams in a division, and use the average coordinate to generate distance across divisions [2].

3. Approach

3.0 Preliminary

The general idea of our approach is divide and conquer. In particular, we first treat each of the six NBA divisions as a single team and construct a feasible schedule for division games. We then expand the schedule by filling in specific teams for cross-division and in-division games, and perform multiple optimality checks before finalizing the schedule. Before explaining the detailed approaches to reach our final schedule, we will first define several terms that we will use throughout the solution.

- **Pattern Bits** are used to denote the position of a certain team in a certain time slot. Particularly, 'H' means that the team plays at home in that time slot and 'A' means that the team plays away games in the time slot. Furthermore, we have an 'I' bit, which is only defined when we treat division as team. It means that in-division games are played (teams within that division play against each other) in that time slot.
- **Patterns** are sequences of pattern bits. Patterns of divisions should contain pattern bits 'H', 'A' and 'I' while patterns of single team should only have 'H' and 'A' as pattern bits. The length of patterns is the number of available time slot in the schedule.
- **Pattern sets** are sets of feasible patterns. The cardinality of pattern sets equals to the number of teams.

With these definitions and constructions, we can now proceed on explaining our detailed solution.

3.1 Step One -- Finding the optimal map for divisions

3.1.1 Find feasible patterns and pattern sets

We use letter ‘A’ through ‘F’ to represent the six NBA divisions. Our goal is to find the optimal map between letters and the actual divisions. (i.e A maps to Pacific Division, B maps to Central Division, and so on). To accomplish this, we first generate all feasible patterns for division A through F. For each feasible pattern, the following conditions are required to be met.

- It contains five ‘H’ bits (because it plays with five other divisions exactly once each at home), and five ‘A’ bits (because it plays with five other divisions exactly once each away). In addition, the pattern should contain four ‘I’ bits because teams within a division plays with other teams four times per season. However, the ‘I’ bits will only be inserted after we find feasible time tables for the corresponding pattern sets.
- Without the ‘I’ bits, first half of the pattern should be exact mirror of second half of the pattern.
- The pattern does not contain at most two consecutive ‘H’ or ‘A’ bits. These consecutive bits must be broken up by the four ‘I’ bits inserted later.

Note that if division E is playing division B at home, this means all five teams in division E and all five teams in division B play five games in a row using simple rotation. (Figure 1). Thus we cannot have two ‘H’ or ‘A’ bits next to each other, as it represents 10 consecutive home/away games, which exceeds the hard limit. With these constraints, there are a total of 12 feasible patterns without ‘I’ bits.

Round 1 --- B1@E1, B2@E2, B3@E3, B4@E4, B5@E5
Round 2 --- B1@E2, B2@E3, B3@E4, B4@E5, B5@E1
Round 3 --- B1@E3, B2@E4, B3@E5, B4@E1, B5@E2
Round 4 --- B1@E4, B2@E5, B3@E1, B4@E2, B5@E3
Round 5 --- B1@E5, B2@E1, B3@E2, B4@E3, B5@E4

Figure 1: We see that every team in division E plays with every team in division B exactly once, with teams in E playing home, teams in B playing away

Next, we match the feasible divisions patterns together to create pattern sets. The pattern set should have cardinality of 6 because there are 6 divisions in total and each pattern represents one division. There are constraints in constructing pattern sets as well. In particular, at any time slot i (the i^{th} bit in all patterns), the pattern set should have equal number of ‘H’s and ‘A’s. That

is, whenever a team is playing at home, there should be a team that's playing away as its opponent. Besides, we in general prefer pattern sets in which patterns are lexicographic different in many places. This usually provides more flexibility for assigning games in future steps. However, the strong assumption of mirroring already significantly reduced the size of the pattern space. It turns out that all patterns differing in at least one places is sufficient. (If there are patterns in the same pattern set that differs in no place, the divisions represented by these patterns cannot play against each other in any time slot, which is undesirable) . With these constraints, we generated 22 different feasible pattern sets. Figure 2 illustrates a sample feasible pattern set.

```
Team A: HAAHHAHHAA
Team B: HAAHAAHHAH
Team C: HAHAHAHAHA
Team D: AHAHAHAHAH
Team E: AHHAAHAAHH
Team F: AHHAHHAAHA
```

Figure 2: Sample feasible pattern set

3.1.2 Find feasible time tables

We then match division letters (A-F) with patterns in the pattern sets to produce feasible timetables. We only need to construct the timetable for the first half of the pattern sets. The second half, with mirroring, should be the same except that roles of 'H's and 'A's are swapped. The algorithm generating the time tables is constructed based on Integer Programming. Specifically, we let x_{ijk} be 1 if division i plays against division j at time k, and 0 otherwise. Note that x_{ijk} is only defined when the kth pattern bits of division i and j are complementary - i.e one of them is 'H' and the other is 'A'. In addition, in each time slot, each division only plays one game, and all 15 cross-division game (single-round) need to played by the end of the first half of the schedule. The constraints are summarized below in mathematical equations. Since we have not yet assigned specific divisions to letters, we do not care about the objective function as long as it generates feasible solution sets of x_{ijk}.

Minimize $\sum_{(i,j,k)} x_{ijk}$

Subject to $\sum_{,k} x_{ijk} = 1$ for all $i, j \in$ {'A', 'B', 'C', 'D', 'E', 'F'}, $i \neq j$, $k = 1,...,5$

$\sum_{,j} x_{ijk} = 1 + \sum_{,j} x_{jik} \leq 1$ for all $i \in$ {'A', 'B', 'C', 'D', 'E', 'F'}, $k = 1,...,5$

The 22 feasible pattern sets generate 30 feasible time tables. The pattern bits representing home/away status are now replaced with divisions letters ‘A’- ‘F’ to indicate where the games are played in the time slot. For instance, if division ‘F’ plays at home against division ‘A’ in time slot 3, the 3rd bit of the patterns in the set representing division ‘A’ and ‘F’ will both be replaced by ‘F’. The ‘I’ bits representing in-division games are now to be inserted. Specifically, for each pattern in the pattern set, if there are two consecutive home/away games, a new time slot is added in between, and ‘I’ is assigned to the time slot. In any patterns, there exist at most 4 sets of two consecutive home/away games due to our constraints of feasible patterns. Therefore, assigning four ‘I’ bits guarantees to break up the all consecutive home/away games in any patterns. Usually more than four time slots need to be added for a pattern set as a whole because the breakpoints for each pattern in the pattern set do not necessarily overlap. After inserting ‘I’s necessary for breaking up consecutive games, we can assign the remaining ‘I’s arbitrarily to the newly added time slots. The unassigned new time slots are filled with ‘X’ bit, denoting a bye slot for certain teams. With ‘I’ bits inserted, there are 162 total feasible timetables including both cross-division and in-division games. Again, the time slot in the timetable is a period during which all games in these matchups will be completed, rather than a specific day. Figure 3 shows a sample feasible time table with in-division games inserted.

```
Team A: AFIEAIAXDAIACIB
Team B: BDICBXAIEBIBFIB
Team C: CEICAICXFCIBCID
Team D: ADIFDICXDBIDEID
Team E: BEIEDIFXECIAEIE
Team F: CFIFBXFIFAIDFIE
```

Figure 3: Sample feasible pattern set

3.1.3 Produce optimal time schedule for actual divisions

The final step is to select the optimal timetable with divisions ‘A’ - ‘F’ matched to actual divisions (Southwest, Northeast, etc.) from all available timetables. For each of the 162 feasible timetable, we iterate through all possible 720 (6!) maps from ‘A’ - ‘F’ to actual divisions to find the division schedule with minimum total traveling distance as well as similar traveling distances for each team. Among the minimums, we then return the one timetable that minimizes travelling distance. This is the optimal division schedule. At this point, we have completed our first step, and what’s left is to break up the divisions into single teams find an optimal schedule of team matchups for each division.

3.2 Step Two -- Finding the optimal map for teams

Each division contains 5 teams, each labeled with subscription '1' to '5' respectively. For example, teams in division A would have label 'A_1' to 'A_5'. We construct pattern for each team in the division. Initially, the pattern for each team is the same pattern for the entire division. Then, we expand the 'H' and 'A' pattern bits to single game matchups. Each of these pattern bits can be expanded into 5 single games because each team in the division plays every team on the opposite division exactly once.

Next, we expand 'I' pattern bits, which represent in-division games. A dummy team labeled as '6' is added for each division. Any team matched with team '6' has a bye status in in-division team matchups. Each 'I' bit represents four games in total, which takes the similar format of single round robin tournament. We use team distance data collected previously to determine the matchup and order of these games, and eventually replace 'I' bit in each team pattern with places that the team plays at. For example, in a certain 'I' bit if team 1 plays team 2 at home and then plays team 5 away, then the I bit in the pattern for team 1 becomes "15". After we perform these procedures on every division, we will have a total of 30 patterns in total with each representing a specific team. We then match teams '1'- '5' in each division to specific teams by minimizing the total travel distance in similar fashion as step one. At this point, we have an initial schedule of 66 games for each team.

Finally, we have timetables for each specific team with divisions 'A'- 'F' already fixed. Using the algorithm implemented before for division assignment solution (see 3.1.3), we enumerate all possible assignments of specific teams and find the minimum travel distance according to the timetables we have. Now we get a full time schedule with local minimum travel distance and we can start to put dates for each game in the schedule.

3.3 Step Three --- Match games to calendar

At this point, we can match games with calendar and produce a final schedule. We enforce general constraints like no Tuesday and Thursday games, more valuable match-ups at weekends and holidays etc. Eventually we check that no team is playing excessive amount of back-to-back games in final schedule. Note that the simplification of reducing from 82 games per team throughout the season to 66 games has crucial effects on number of back-to-back games, as clearly there is more flexibility. There are about 160 game days in the regular season. As a result, it is possible fit 66 games per team to completely eliminate back-to-back games from the schedule.

IV. Results

Team Name	Total Distance (km)	Team Name	Total Distance (km)
ATL	36069	IND	35016
WSH	38564	SAS	37960
ORL	42206	MEM	34153
CHA	42354	HOU	40191
MIA	45399	DAL	40868
OKC	43001	NOP	38958
POR	51246	BOS	41698
DEN	44168	TOR	34861
UTA	49892	BKN	39270
MIN	50149	NYK	41855
MIL	34806	PHI	39254
CLE	32526	GSW	45286
CHI	35886	PHX	38286
DET	39531	LAC	43964
SAC	46919	LAL	43686

Table 1: the total travel distance for each team

The travel distances for each team during the regular seasons is shown in Table 1. On examination, teams in middle U.S (e.g CLE, CHI, MEM) tends to have relatively small travel distances. Teams like Portland Trail (POR) and Minnesota Timberwolves (MIN) have large travel distances because their location is relatively far to other teams in the conference. Overall, the travel distance for all teams appear to be fairly balanced.

The Complete schedule can be found in the Excel file as an attachment to the paper. In the final schedule, the length of the season is 19 weeks, during which each team will play 66

games as specified before. Game days are Mondays, Wednesdays, Fridays, Saturdays and Sundays. Note that Saturdays and Sundays usually have less game than weekdays. This is done intentionally so that teams playing on Saturday will have a rest day on Sunday and vice versa. In this way, we can guarantee that no teams in the schedule will be playing three consecutive games in any three days. Furthermore, this allocation is more desirable to broadcast networks as well because in weekends there will be fewer games at each time slot, which leads to less broadcasting conflicts. Thus, more need of all audiences can be satisfied and more values can be generated to the networks. Additionally, in the final days of the season number of games per day is limited as well because teams will be fighting for their playoff seats and thus these games usually have more values than mid-season games.

V. Conclusion

We do not account for potential violation of TV broadcasting or arena availability constraints in our project, as these detailed information are unobtainable. Once those constraints become available, we can use integer and constraint programming to implement these constraints and produce a schedule that satisfies these constraints. Possible future study direction includes how to efficiently inserting the remaining 16 games such that the structure of official NBA schedule can be maintained without breaking the least travelling distance property of the schedule.

Reference

1. *National Basketball Association, Wikipedia, retrived from "https://en.wikipedia.org/wiki/National_Basketball_Association"*
2. *, 'NBA Teams.' Map of NBA Teams, retrieved from "http://www.sportmapworld.com/map/basketball/north-america/nba/"*
3. *Scheduling a Major College Basketball Association, July 1997, George L. Nemhauser, Michael A. Trick. Retrieved from "http://mat.gsia.cmu.edu/trick/acc.pdf"*