2017 NBA Hack-A-Thon
Team Application

Proposed Solutions to Question #2

Ashwin Ghadiyaram
Graham Pash
Vinit Ranjan
Jason Thompson

Trust the Process

Introduction

This problem tasked us with determining when a team may be eliminated from playoff contention as the season progresses. We chose to produce our solution using the R Programming Language and utilized the readxl, magrittr, lubridate, dplyr packages. At a high level our code starts with the first game day of the season, tallies the scores from that day (i.e. point differential, wins, losses for teams involved), generates a best case scenario for the remaining teams in the playoff hunt, determines which teams are still in playoff contention, and then continues the loop by advancing to the next day.

Note that we are aware of some bugs in our code, but we present the solution here as a solid foundation step towards the true solution. See the final section for known bugs and possible fixes that we would wish to implement.

Main Loop

The flowchart in Figure 1 diagrams how our program works.

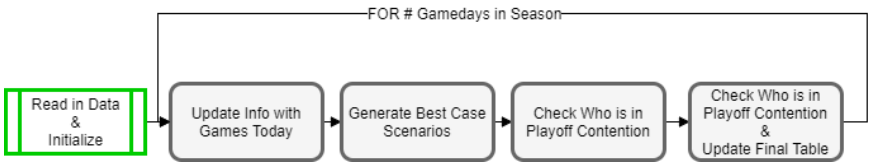

Figure 1: Pseudo-code of our solution

`tallyScores()`

This function loops through the games that occured during `currentDay` and updates the wins, losses, point differential, conference wins/losses, and division wins/losses in the master data frame `teams`, which stores the actual results from the season.

`generateBestCase()`

This function generates a "best case" scenario for each team, i.e. simulates the circumstances that would lead to the best end season result possible for a given team. While it is not computational feasible to simulate every possible season from the start of the season, there are a few rules that can be followed to greatly reduce the number of games which results need to be checked. These rules are:

- Since playoffs are determined by conference, when looking at a given team only games that teams in their conference are involved in need to be simulated.
- Clearly it is always in the best interest of a team to win out when trying to make the playoffs.
- Since the each team in the conference plays two games against each team in the other it is in the best interest of a team to have all of the other teams in their conference lose these games since it drags competitors to the playoffs down, while giving them an advantage (assuming they win all of these games).
- Since division leaders win in one of the tie-breaker scenarios, it is in a team's best interest if the other teams in their division lose games outside of the division.

After these rules, the only two types of games that need to be simulated are:

- Games between games in the same division as the team being investigated.
- Games between other divisions in the same conference as the team in question that are not involving teams in the division of the team in question.

`checkPlayoffTeams()`

This function takes in the set of results, which has been taken up to a certain date and generated thereafter. It takes as input a specific team in question because only one conference needs to be considered at a time. Using the given team as input, it generates the playoff contention teams for that team's specific conference and then returns a list of those teams. Overall, this function can be considered a driver function to take the information from games and extract results.

`twoTeamLogic()`

This function takes in the game results and the two teams in contention that are tied. It runs the logic according to the the priority order given in the problem statement and returns the team that wins the tiebreaker. If it comes down to point differential, then we simply say both teams are still playoff contenders. We only aim to simulate game results without going to detail of points. This is valid because if a tiebreaker comes down to point differential then neither team is mathematically eliminated.

`threePlusTeamLogic()`

This function is similar to the prior function except it is built to handle a variable amount of teams larger than three. It also uses the tiebreakers outlined in the problem statement. The code handles multiple teams due to the recursive strategy of going through tiebreaker criteria on smaller and smaller subsets. This continues until a sufficient number of teams are found. Similar to the previous function, point differential is not used as an elimination factor and all teams at that stage are still mathematically in contention.

Known Bugs

Due to lack of computational power, our randomness scheme is not given a sufficient amount of trials to run. So, results may eliminate some teams early when they are still in contention because the limited number of trials were unable to find cases to stay in contention. Also, once a team is eliminated, they are gone and we do not consider them anymore. This is likely the cause of any strange results we get for eliminations.