Efficient MLB Schedule Creation Algorithm

Karl Kiesel
Andrei Cerci

Section 1: Introduction

Every team in Major League Baseball (MLB) plays a total of 162 games during the regular season. This includes 81 games being played at a team's home stadium, and then another 81 games being played away at another team's stadium. All of this traveling to and from different cities to play baseball games can take up a lot of time and money for the teams, as well as increase pollution to the environment. Our goal is to create an algorithm to design the most efficient schedule possible for each team to help reduce travel costs as well as pollution. We are using the dedicated number of homesteads and road trips to optimize the team's efficiency. We plan to have this algorithm take in a list of what teams need to play each other during the course of the season, as well as distances between the different cities where the teams are located, and then create a schedule that reduces unnecessary traveling as much as possible.

Section 2: Literature Review

In this paper titled, "Adaptive mechanism for schedule arrangement and optimization in socially-empowered professional sports games", the authors look to find a way to decrease the cost of traveling for professional sports leagues. They describe an algorithm that can be used to create a schedule for all the teams in the league. Their algorithm first takes in a set of constraints, such as the number of teams and divisions, as well the length of the season. The algorithm then randomly generates a schedule that meets these constraints. That random schedule is then analyzed to find the total cost of travel among all the teams in the schedule. The algorithm then randomly generates another schedule and finds the total travel cost of the new schedule. If the total cost of the new schedule is less than the first schedule, then the new schedule is swapped with the old one. The algorithm then repeats this cycle of randomly generating new schedules and comparing them to the previous best schedule for a set number of iterations. [1]

In this paper, titled "A Framework for a Highly Constrained Sports Scheduling Problem" the authors seek to create a framework for a sports tournament schedule. The first part of their framework involves defining 36 different constraints. These constraints involve when and where the games are played, as well as what round those games are played and even dividing teams into different strength groups and ensuring that a team will only play a certain number of opponents from the same strength group. The paper then looks at some real-world examples and describes how those examples were solved, with the common solutions involving a metaheuristic algorithm or a multiphase algorithm. The authors state that it is important to consider all the constraints as early as possible when developing a solution to help simplify the algorithm that will be used. [2]

In this paper titled, "Scheduling the finnish major ice hockey league", the authors describe how to create a way to schedule a season for the 14 teams in the Finnish hockey league. The schedule involves dividing the season up into several different rounds, or weeks, depending on the number of teams in the league. Every team will play once during each round. For each team to play every other team once, the number of rounds must be equal to the number of teams minus 1 (t - 1). For teams to play each other twice, home and away, the number of rounds must be 2(t - 1). Therefore, the total number of rounds for the Finish hockey league must be 2(14 - 1) = 26 rounds. When deciding what teams play each other, the algorithm first generates a random schedule which meets all the constraints. The algorithm then uses a greedy hill climbing mutation algorithm to generate a new solution candidate from the previous solution. As the schedules are refined and improved, the old non-optimal schedules are replaced by the better ones, and the algorithm will continue this process until a set amount of time is reached. Another important consideration is to see what individual requirements each team has, such as what days they would prefer to play or not to play, and make sure those are taken into account in the algorithm. This however, does make the algorithm much more complicated depending on the complexity and number of requirements. [3]

Sports scheduling is incredibly complicated. As stated in the problem definition there are so many factors that go into what teams play who when. This paper titled “A Polynomial Algorithm for the Degree Constrained Minimum K Tree Problem” explains how Marshall L. Fisher used a greedy type algorithm and an MST that turned a team's travel schedule into a graph that had weighted costs on each leg. The goal is to satisfy the requirements while keeping cost at a minimum. He concluded that by building a K-tree and running his algorithm he could average an O(n^3) time.[4]

The traveling tournament problem is defined in this paper by Dr. Nitin Choubey titled “A Novel Encoding Scheme for Traveling Tournament Problem using Genetic Algorithm”. The traveling tournament problem refers to when there is an event that needs to happen, in this case a sports game, and travel is a constraint. The paper explains how he adapted the genetic algorithm to the problem. In short summary he used a penalty system to build a set of suitable schedules and every time there was a conflict there was a penalty. Then he put them into categories of optimal or sub-optimal. I think this is important because it recognises the fact that there may be no perfect answer where every team has the optimal schedule. However, it is important to find a way to have as many as possible. [5]

In this Paper the author Jean-Philippe Hamiez applies a linear-time algorithm to the problem that we are trying to solve. They applied a simple undirected graph to represent all the requirements for a team schedule. They listed out all the properties and then decided whether they were satisfied or not. They applied the E3S algorithm which according to them is a simplified L3S algorithm. They ended up successfully achieving a linear time for (T-1) where number of team mod 3 does not equal 0.[6]

Section 3: Problem Definition

Variable	**Meaning**
HN	The name of the team that a particular schedule is being created for. We use this variable to find the home team name within the TBP set.
OND	name of an opponent team in the same division
ONID	name of inter divisional opponent team
ONIL	name of inter league opponent team
OC	city where an opponent baseball team is located
HC	city where the home team plays games
DF	Distance needed to fly from the home city to an opponent city (miles)
FCF	Fuel consumed while flying (gallons)
DD	Distance needed to drive from the home city to an opponent city (miles)
FCD	Fuel consumed while driving (gallons)

TBP	set of teams that will be played
BS	the resulting schedule
MLB	Set of all teams in the MLB
T, t	The team for which the schedule is being created. Little t is a T in the set TBP.
RT, c	The set of all trips, including the flying distance DF and the driving distance DD. Little c is a RT in the set TBP.

The purpose of our algorithm is to make a schedule for each of the 30 teams, *HNs*, in the MLB. Over the course of a season, each *HN* will play 162 games. These games are divided into sets of three games called a series, for a total of 54 series over the course of a season. Every *HN* in the division must play 24 divisional series, 20 inter division series, and 10 inter league series. A given *HN* will play each of its 4 divisional opponents, *ONDs*, in 6 series with 3 of those series being at the home city, *HC*, and 3 being at an away city, *OC*. That same *HN* will play each of its 10 inter division opponents, *ONIDs*, in 2 series with 1 series being at the *HC* and 1 series being at the *OC*. A given *HN* will then play 5 inter league opponents (all within the same division), *ONILs*, in 2 series with 1 series being at the *HC* and 1 series being at the *OC*. All of these opponents will be put into the *TBP* set to get the resulting *BS* set. To find the most energy efficient way of creating a schedule. To be energy efficient it will balance the net cost of air travel versus ground and calculate how to schedule their opponents. Our algorithm needs to use the distance between each of the *OCs* that the *HN* will travel to, as well as the distance between the *OCs* and the *HC*. These distances are stored within the *DD* variable if the distance is less than 350 miles, or else those distances are stored in the *DF* variable meaning the *HN* must fly to the *OC*. The amount of fuel consumed in either *FCF* or *FCD* will depend on the value of the respective distance travelled variable, *DF* or *DD*.

Section 4: Final Algorithm

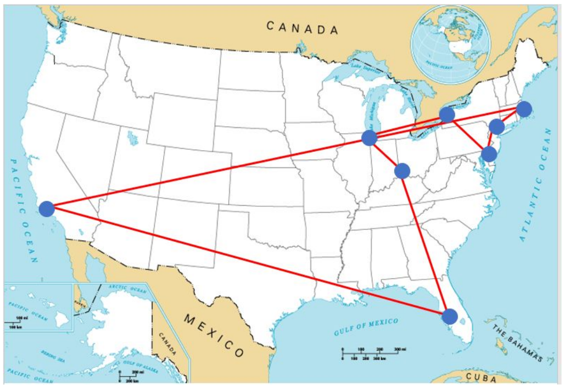
This algorithm is designed to create the most energy efficient schedule for each of the 30 teams in Major League Baseball. This is accomplished by grouping road games to cities with the least amount of distance between each other in order to reduce the amount of traveling in between cities. This is accomplished by using a modification of Kruskal’s shortest path algorithm, with each city that must be visited forming the nodes, and the distance between cities forming the weights of the various paths.

Algorithm Pseudo Code:

```
1   SCHEDULE(MLB):
2   For T ∈ MLB
3           MAKE set TBP
4           KRUSKALS(HN, TBP):
5                   BS = ∅
6                   For each T ∈ TBP
7                           MAKE SET (t)
8                   For each trip (c ,t) ∈ RT ordered increasing by (c,t)
9                           if FIND SET (c) ≠ FIND SET(t)
10                          BS = BS U {(c,t)}
11                          UNION(c,t)
12                  return BS
```

The algorithm runs through two steps as stated in the pseudocode. It starts in line 1 with the constructor initializing the schedule algorithm. Next in line 2 and 3 is a method which first builds the set TBP where all the teams in the schedule are made into a weighted undirected graph. Line 4 initializes Kruskal's method. Line 5 through 11 runs Kruskal's algorithm building the set. In line 12 the Best scheduling algorithm returns the best possible set.

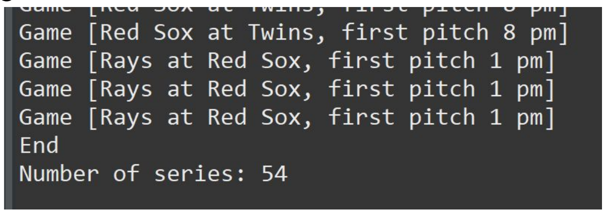
Example Case: This algorithm will run for the set of all teams in major league baseball. For this example we will run for the Boston Red Sox. The Red Sox are a team T in MLB. First a set is built called teams to be played with a for loop adding the set of division, league, and inter league games totalling 162 games and 54 series in line 2 and 3. Next that set is entered as a weighted undirected graph into kruskals and runs in line 4 through 11. It is undirected because the team must travel both ways. Their best schedule is the empty set to start. Then a set of all their teams in TBP is made. For each trip with cost c and team t in set RT (road trips if cost c is not in set trips t then add the union of the set (c,t) to the set BS. The for loop orders the trips from lowest to highest cost. Then once it terminates it returns the BS in line 12. For the Red Sox, it will first add the teams closest like the New York Yankees and the Baltimore Orioles. The road trips to the west coast will likely be at the end due to the cost it takes to cross the country. Then the algorithm will move on to the next team T in MLB and repeat the process. It will build a set TBP and then build an MST called BS. It does this for all teams in the MLB.



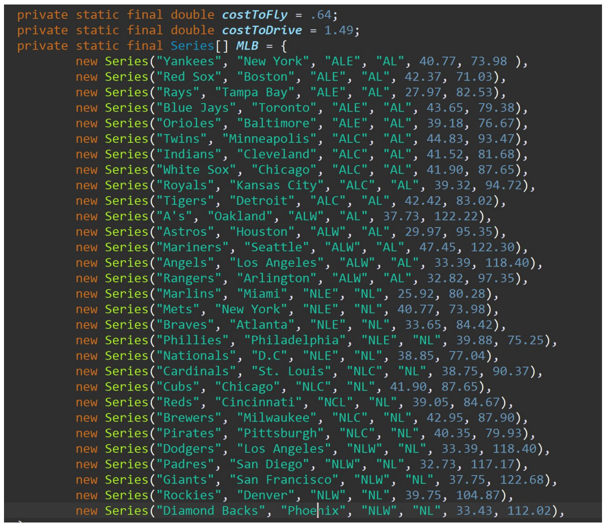
Cost Analysis: For Kruskal's Algorithm, the time complexity is O(ElogV), where V is the number of vertices and E is the number of edges. Since each team plays in its home stadium and the stadiums of 19 different teams, the value of V will be 20 for each team. We also know that the worst case value of E will be V squared (each city has a path to every other city). This means E's worst case value will be 20 squared, or 400, so the Kruskal Algorithm section of our algorithm will have a time complexity of O(400 * log20). Within the Kruskal algorithm section, there is a for each loop that runs through the TBP set, so the time complexity is O(TBP). Also within the Kruskal algorithm section, there is another loop that goes through each of the trips within the set of city distances, to the worst case for that loop is the size of the set of DD and DF values O(size of RT. So, the total time complexity is O((400 * log 20) * (TBP + RT)).

Section 5: Performance Analysis

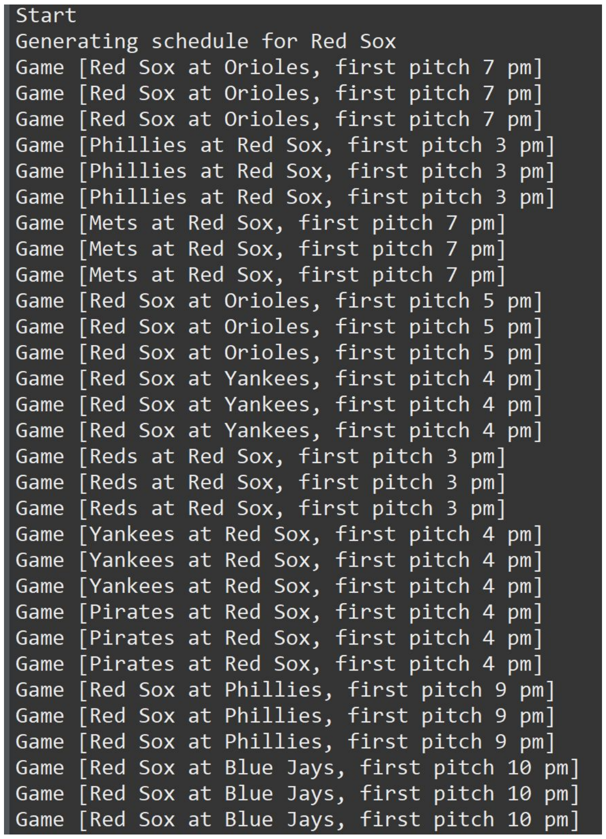
To measure the performance of our algorithm, we analyzed the schedules that were created to ensure they followed all of the constraints that we mentioned previously. The number of series is 54 every time, so 100% accuracy there. There are also always 3 games per series, so also 100% accuracy there. The home and away games are determined randomly. Since there is a 50% chance of any series being home or away, it is likely that the spread will be close to 50/50, but it is not guaranteed.


```
Game [Red Sox at Twins, first pitch 8 pm]
Game [Rays at Red Sox, first pitch 1 pm]
Game [Rays at Red Sox, first pitch 1 pm]
Game [Rays at Red Sox, first pitch 1 pm]
End
Number of series: 54
```

No matter what specific team the user puts in to create a schedule for, the cost of driving, flying, and all the data for each team remains constant. This is shown in the following screenshot of our code.

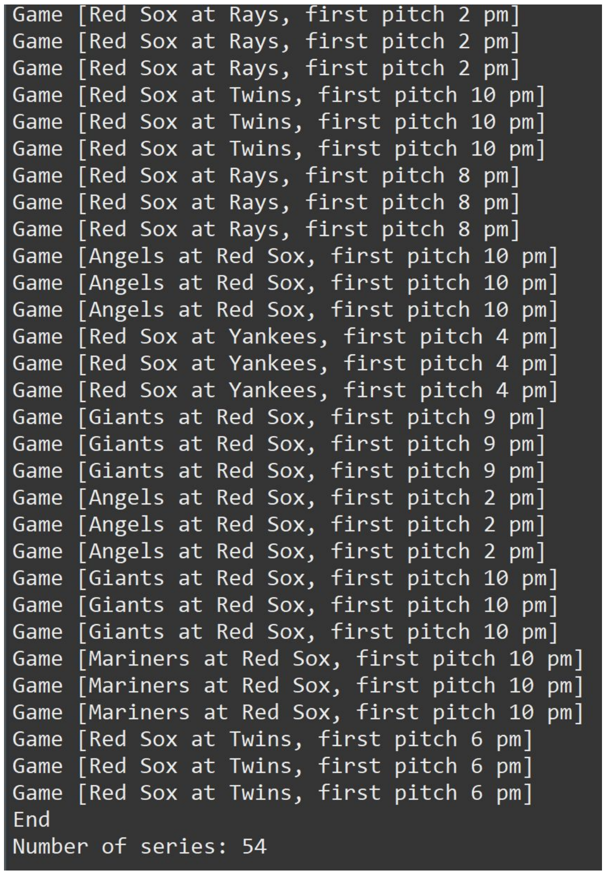

```
private static final double costToFly = .64;
private static final double costToDrive = 1.49;
private static final Series[] MLB = {
        new Series("Yankees", "New York", "ALE", "AL", 40.77, 73.98 ),
        new Series("Red Sox", "Boston", "ALE", "AL", 42.37, 71.03),
        new Series("Rays", "Tampa Bay", "ALE", "AL", 27.97, 82.53),
        new Series("Blue Jays", "Toronto", "ALE", "AL", 43.65, 79.38),
        new Series("Orioles", "Baltimore", "ALE", "AL", 39.18, 76.67),
        new Series("Twins", "Minneapolis", "ALC", "AL", 44.83, 93.47),
        new Series("Indians", "Cleveland", "ALC", "AL", 41.52, 81.68),
        new Series("White Sox", "Chicago", "ALC", "AL", 41.90, 87.65),
        new Series("Royals", "Kansas City", "ALC", "AL", 39.32, 94.72),
        new Series("Tigers", "Detroit", "ALC", "AL", 42.42, 83.02),
        new Series("A's", "Oakland", "ALW", "AL", 37.73, 122.22),
        new Series("Astros", "Houston", "ALW", "AL", 29.97, 95.35),
        new Series("Mariners", "Seattle", "ALW", "AL", 47.45, 122.30),
        new Series("Angels", "Los Angeles", "ALW", "AL", 33.39, 118.40),
        new Series("Rangers", "Arlington", "ALW", "AL", 32.82, 97.35),
        new Series("Marlins", "Miami", "NLE", "NL", 25.92, 80.28),
        new Series("Mets", "New York", "NLE", "NL", 40.77, 73.98),
        new Series("Braves", "Atlanta", "NLE", "NL", 33.65, 84.42),
        new Series("Phillies", "Philadelphia", "NLE", "NL", 39.88, 75.25),
        new Series("Nationals", "D.C", "NLE", "NL", 38.85, 77.04),
        new Series("Cardinals", "St. Louis", "NLC", "NL", 38.75, 90.37),
        new Series("Cubs", "Chicago", "NLC", "NL", 41.90, 87.65),
        new Series("Reds", "Cincinnati", "NCL", "NL", 39.05, 84.67),
        new Series("Brewers", "Milwaukee", "NLC", "NL", 42.95, 87.90),
        new Series("Pirates", "Pittsburgh", "NLC", "NL", 40.35, 79.93),
        new Series("Dodgers", "Los Angeles", "NLW", "NL", 33.39, 118.40),
        new Series("Padres", "San Diego", "NLW", "NL", 32.73, 117.17),
        new Series("Giants", "San Francisco", "NLW", "NL", 37.75, 122.68),
        new Series("Rockies", "Denver", "NLW", "NL", 39.75, 104.87),
        new Series("Diamond Backs", "Phoenix", "NLW", "NL", 33.43, 112.02),
```

For a test run, we used the Boston Red Sox as a team to create a schedule for. The result was a list of 54 series to be played. A portion of these series are shown in the following screenshot.



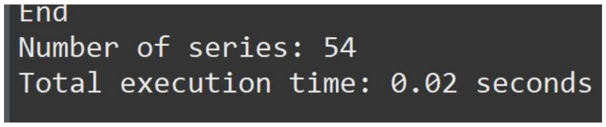
```
Start
Generating schedule for Red Sox
Game [Red Sox at Orioles, first pitch 7 pm]
Game [Red Sox at Orioles, first pitch 7 pm]
Game [Red Sox at Orioles, first pitch 7 pm]
Game [Phillies at Red Sox, first pitch 3 pm]
Game [Phillies at Red Sox, first pitch 3 pm]
Game [Phillies at Red Sox, first pitch 3 pm]
Game [Mets at Red Sox, first pitch 7 pm]
Game [Mets at Red Sox, first pitch 7 pm]
Game [Mets at Red Sox, first pitch 7 pm]
Game [Red Sox at Orioles, first pitch 5 pm]
Game [Red Sox at Orioles, first pitch 5 pm]
Game [Red Sox at Orioles, first pitch 5 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Reds at Red Sox, first pitch 3 pm]
Game [Reds at Red Sox, first pitch 3 pm]
Game [Reds at Red Sox, first pitch 3 pm]
Game [Yankees at Red Sox, first pitch 4 pm]
Game [Yankees at Red Sox, first pitch 4 pm]
Game [Yankees at Red Sox, first pitch 4 pm]
Game [Pirates at Red Sox, first pitch 4 pm]
Game [Pirates at Red Sox, first pitch 4 pm]
Game [Pirates at Red Sox, first pitch 4 pm]
Game [Red Sox at Phillies, first pitch 9 pm]
Game [Red Sox at Phillies, first pitch 9 pm]
Game [Red Sox at Phillies, first pitch 9 pm]
Game [Red Sox at Blue Jays, first pitch 10 pm]
Game [Red Sox at Blue Jays, first pitch 10 pm]
Game [Red Sox at Blue Jays, first pitch 10 pm]
```

The screenshot above shows the first 10 series to be played by the Red Sox. All of the following teams on this list are located in the Northeastern United States and Canada. This is because our algorithm tries to group games with similar geographic locations together in order to reduce travel time. The last few games of the schedule are mostly against teams that are farther away from Boston, since those farther teams are grouped together to reduce travel time between them. This is shown in the screenshot below.



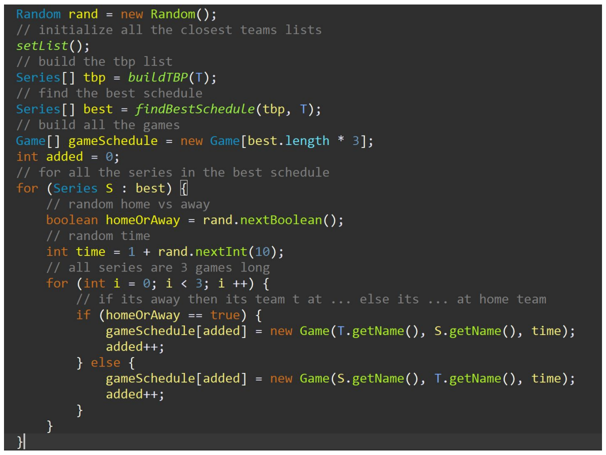
```
Game [Red Sox at Rays, first pitch 2 pm]
Game [Red Sox at Rays, first pitch 2 pm]
Game [Red Sox at Rays, first pitch 2 pm]
Game [Red Sox at Twins, first pitch 10 pm]
Game [Red Sox at Twins, first pitch 10 pm]
Game [Red Sox at Twins, first pitch 10 pm]
Game [Red Sox at Rays, first pitch 8 pm]
Game [Red Sox at Rays, first pitch 8 pm]
Game [Red Sox at Rays, first pitch 8 pm]
Game [Angels at Red Sox, first pitch 10 pm]
Game [Angels at Red Sox, first pitch 10 pm]
Game [Angels at Red Sox, first pitch 10 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Red Sox at Yankees, first pitch 4 pm]
Game [Giants at Red Sox, first pitch 9 pm]
Game [Giants at Red Sox, first pitch 9 pm]
Game [Giants at Red Sox, first pitch 9 pm]
Game [Angels at Red Sox, first pitch 2 pm]
Game [Angels at Red Sox, first pitch 2 pm]
Game [Angels at Red Sox, first pitch 2 pm]
Game [Giants at Red Sox, first pitch 10 pm]
Game [Giants at Red Sox, first pitch 10 pm]
Game [Giants at Red Sox, first pitch 10 pm]
Game [Mariners at Red Sox, first pitch 10 pm]
Game [Mariners at Red Sox, first pitch 10 pm]
Game [Mariners at Red Sox, first pitch 10 pm]
Game [Red Sox at Twins, first pitch 6 pm]
Game [Red Sox at Twins, first pitch 6 pm]
Game [Red Sox at Twins, first pitch 6 pm]
End
Number of series: 54
```

Every run of the algorithm takes approximately 0.02 seconds to complete one team's schedule.



```
End
Number of series: 54
Total execution time: 0.02 seconds
```

To break it down further, the algorithm first builds the list of teams to be played, which has a complexity of O(TBP), which can be simplified to O(1). Next, the algorithm builds the best schedule, which runs through all the teams in TBP. From here the algorithm organizes the series list generated by building the best schedule. This means going through the list of games in the best schedule and the trips involved, so the time complexity is O(RT). In our theoretical analysis of the algorithm, we realized that many elements of our algorithm would remain constant, and the variables that did change value would still always have the same size, such as the size of TBP. However, the actual values of RT and TBP are subject to some change, since we do not know how many road trips must be taken until the teams to be played (TBP) are established. Therefore, the time complexity of our algorithm is largely determined by the values within the TBP set, which then affects the values of the RT value, as well as some other constant constraint values. This is consistent with what we predicted in our theoretical analysis.



```
Random rand = new Random();
// initialize all the closest teams lists
setList();
// build the tbp list
Series[] tbp = buildTBP(T);
// find the best schedule
Series[] best = findBestSchedule(tbp, T);
// build all the games
Game[] gameSchedule = new Game[best.length * 3];
int added = 0;
// for all the series in the best schedule
for (Series S : best) {
    // random home vs away
    boolean homeOrAway = rand.nextBoolean();
    // random time
    int time = 1 + rand.nextInt(10);
    // all series are 3 games long
    for (int i = 0; i < 3; i ++) {
        // if its away then its team t at ... else its ... at home team
        if (homeOrAway == true) {
            gameSchedule[added] = new Game(T.getName(), S.getName(), time);
            added++;
        } else {
            gameSchedule[added] = new Game(S.getName(), T.getName(), time);
            added++;
        }
    }
}
```

Contribution Clarifications

Karl Kiesel - 50% of Section 1, First half of Section 2, 70% of table in Section 3, 40% of explanation in Section 3, 100% Intro and cost analysis for Section 4. 80% of Section 5, 30% of code. 50% of powerpoint

Andrei Cerci - 50% of Section 1, First half of Section 2, 30% of table in Section 3, 60% of explanation in Section 3, 100% Pseudocode and the example for Section 4. 20% of Section 5, 70% of code. 50% of powerpoint

References

[1] Hung, J.C., Yen, N.Y., Jeong, H. et al. Adaptive mechanism for schedule arrangement and optimization in socially-empowered professional sports games. Multimed Tools Appl 74, 5085–5108 (2015). https://doi.org/10.1007/s11042-014-1852-2

[2] Nurmi, K., et al. "A Framework for a Highly Constrained Sports Scheduling Problem." ResearchGate, 19 Mar. 2010. https://www.researchgate.net/profile/Frits_Spieksma/publication/44260913_A_Framework_for_a_Highly_Constrained_Sports_Scheduling_Problem/links/0fcfd510f65d93d19c000000/A-Framework-for-a-Highly-Constrained-Sports-Scheduling-Problem.pdf

[3] J. Kyngas and K. Nurmi, "Scheduling the finnish major ice hockey league," 2009 IEEE Symposium on Computational Intelligence in Scheduling, Nashville, TN, 2009, pp. 84-89, doi: 10.1109/SCIS.2009.4927019. https://ieeexplore.ieee.org/abstract/document/4927019

[4] Fisher, Marshall L. "A polynomial algorithm for the degree-constrained minimum k-tree problem." *Operations Research* 42, no. 4 (1994): 775-779.

[5] Choubey, Nitin. (2010). A Novel Encoding Scheme for Traveling Tournament Problem using Genetic Algorithm. International Journal of Computer Applications. ecot. 10.5120/1536-139.

[6] Hamiez, Jean-Philippe & Hao, Jin-Kao. (2004). A linear-time algorithm to solve the Sports League Scheduling Problem (prob026 of CSPLib). Discrete Applied Mathematics. 143. 252-265. 10.1016/j.dam.2003.10.009.