



**Efficient Scheduling of matches in Double Round RobinTournament using Discrete Mathematics**

**Project Review**

Submitted by

PRIYANSHI SHARMA (19BCE7487)

SAHIL KUMAR MISHRA (19BCN7044)

BIREN NAHAK (18BCE7021)

ASHUTOSH KUMAR SINGH (18BCE7082)

DHRUVE SINGH (18BCE7093)

PHANI SAI PRANAV (18BCE7168)

Submitted To

Prof. Tanuj Kumar

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
| **1** | **Aim of Project** | 3 |
| **2** | **Abstract** | 3 |
| **3** | **Introduction** | 3 |
| **4** | **Problem Definition** | 4 |
| **5** | **Implementation** | 4 - 14 |
| **6** | **Advantages** | 15 |
| **7** | **Summary** | 15 |
| **8** | **References** |  |

**AIM**

Solution of the project gives an official schedule with respect to minimizing and balancing travel distance and breaks, while satisfying more requirements and making profit for the organization by higher attendance and TV viewership, lower costs and increased fairness .

# **ABSTRACT**

Discrete math forms the basis of many real-world scientific fields -- especially computer science. The primary techniques learned in a discrete math course can be applied to many different fields. It has a lot of advanced prediction algorithms in order to solve day-to-day problems and Efficient Scheduling of matches in the Double Round Robin Tournament is also one of them.

In a tournament, every team plays with every other team twice. This is a double round robin. That means, in a tournament of eight teams, each team gets to play 14 matches. The winner is decided based on all the matches.

But the major drawback of Double Round-robin is that it suffers from being too long compared to other tournament types, and hence higher expenses for travelling, staying etc.

This project gives an idea about how the above-mentioned problem can be solved with the help of Discrete mathematics. For this, the use of Python and JAVA has been made along with scheduling and travelling salesman problems.

**INTRODUCTION**

Professional Sports League are big businesses around the world.

One key to such an income level is the schedule team plays. No right holder wants to pay large sums only to get unattractive teams playing on a prime date. Teams do not want to see their large investment in players and infrastructure undermined by poor scheduling. Fans, who ultimately provide the income for the leagues, are also greatly affected by the schedules.

In addition, travels become an important issue for the team. Teams are also concerned with the more traditional issues with regards to their home and away patterns. No teams like to be away more than two weeks or so, nor do teams want to be home for longer than that period.

From above, we can clearly see that the major problem is to schedule the matches in order to reduce travels for teams. This problem can be solved by the combination of Double round robin tournament and Travelling salesman problem.

**PROBLEM DEFINITION:**

Here as an example, we have taken IPL. There are many constraints such that every team plays with others twice, one on its own home town and another on the opponent home town. As we all know that of double round robin in IPL suffer from being too long to complete and hence required a lot of days to get over resulting in higher expenses of travelling, staying etc and hence we are trying to balance travel distance and breaks, while satisfying more requirements and making profit for the organization by higher attendance and TV viewership, lower costs and increased fairness.

**IMPLEMENTATION:**

1. SCHEDULING MATCHES

We used Double Round Robin tournament scheduling to schedule matches for our tournament. A round robin tournament is a tournament where every team plays every other team exactly once. Each game is a head-to-head match between two teams where the winner earns a point. At the end of the tournament, the overall winner is the team with the most amount of points.

Double round robin is a variant where every team plays every other team twice instead of once. This is a convenient option if the organizer wants to provide the team with a choice of arena for the matches since each team can make an arena choice.

The general formula for the number of matches played by a team in double round robin tournament are (n teams)

Each team – 2 \* (n-1)

Total number of matches – n \* (n - 1)

CODE:

|  |
| --- |
| import itertools  import random  competitors = ['1', '2', '3', '4']  print (' '\*40,"The competitors are: " + " ".join(competitors))  print(' '\*38,'-'\*31)  def do\_round():    matches = itertools.permutations(competitors,2)  l=list(matches)    random.shuffle(l)  for match in l:  print (' '\*38,'|',' ',match[0],' ','vs',' ', match[1],' ','|')  print(' '\*38,'-'\*31)  return matches  do\_round() |



1. **MINIMIZING TRAVEL DISTANCE**

The minimization of travel distance becomes relevant when teams travel from one away game to the next without returning home. In this setup huge savings can be obtained when long trips are applied and teams located close together are visited on the same trip. To solve the problem, a 2-phase approach is applied.

In Phase 1, the optimal trips for each team are derived and the authors show that, for a tournament with an even number of teams, it is equivalent to pair the teams two and two such that the distances between the paired teams are minimized.

In Phase 2, the optimal pairing is translated into a number of feasible sequences using a constructive approach. This approach takes all the constraints into account and the result is an optimal schedule which minimizes the total travel distance.

We used the travelling salesman problem to solve the distance shortening constraint.   
Travelling salesman problem is the most notorious computational problem. We can use a brute-force approach to evaluate every possible tour and select the best one. For **n** number of vertices in a graph, there are **(*n* - 1)!** Number of possibilities. Here the solution can be obtained in less time, though there is no polynomial time algorithm.

|  |
| --- |
| import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Scanner;  public class TSP {  public static void main(String[] args) throws IOException {    BufferedReader userinput = new BufferedReader(new  InputStreamReader(System.in));  //Get file name  String fileName = "";  System.out.println("Place dataset in root folder and type filename (ex: data8.txt):");  fileName = userinput.readLine();  long startTime = System.currentTimeMillis();    Scanner readFile;  readFile = new Scanner(new File(fileName));  //find size of problem based on input file  int n = 0;  do{  readFile.nextInt();  n++;  }while(readFile.hasNext());  n = (int) Math.sqrt(n);  readFile.close();  //Create distance matrix  int[][] cityDistances = new int[n][n];  int i = 0;  int j = 0;  readFile = new Scanner(new File(fileName));  //Populate distance matrix  for(j=0;j<n;j++){  for(i=0;i<n;i++){  cityDistances [i][j] = readFile.nextInt();  }  }  readFile.close();//close file  int ha = 0;    while(ha > n-1 || ha <= 0){  System.out.println("Input consecutive maximum home/away games: ");  ha = Integer.parseInt(userinput.readLine());  if(ha > n-1 || ha <= 0){  System.out.println("Invalid value entered. Value must be greater than 0 and less than (# of Teams - 1): ");  }  }  int [][]starWeight = new int [n][1];  int indexWeight = 0;  for(j=0;j<n;j++){  for(i=0;i<n;i++){  starWeight[j][0] += cityDistances[i][j];  }  }    int tempWeight = 10000000;  for(i=0;i<n;i++){  if(starWeight[i][0]<tempWeight){  tempWeight = starWeight[i][0];  indexWeight = i;  }  }    int[]chosenPath = new int [n+1];  int distance = 0;    for(i=1;i<n;i++){  if(i!=indexWeight){  chosenPath[i] = i;  }  }    chosenPath [0] = indexWeight;  chosenPath [n] = indexWeight;  for(i=0;i<n;i++){  distance += cityDistances[chosenPath[i]][chosenPath[i+1]];  }    int tempDistance=0;  int [] tempPath = new int[n+1];  int tempHold = 0;    for(i=0;i<n+1;i++){  tempPath[i] = chosenPath[i];  }    int count = 0;  while(count!=100){    for(j=1;j<n-1;j++){  for(i=1;i<n-1;i++){  tempHold=tempPath[j];  tempPath[j]=tempPath[i];  tempPath[i]=tempHold;  tempDistance = 0;    for(int z=0;z<n;z++){  tempDistance += cityDistances[tempPath[z]][tempPath[z+1]];  }  if(tempDistance<distance){  for(int b=0;b<n+1;b++){  chosenPath[b]=tempPath[b];  }  distance = tempDistance;      }    else{  tempHold=tempPath[i];  tempPath[i]=tempPath[j];  tempPath[j]=tempHold;  }  }  }    count++;  }  int pathFormatted[] = new int [n];      for(i=0;i<pathFormatted.length;i++){  pathFormatted[i]=chosenPath[i+1];  }      int [][]schedule = new int [n][2\*n-2];      schedule = part2.part2(ha,pathFormatted,n);    int []teamDistance= new int [n];  int totalDistance = 0;  for(j=0;j<n;j++){  for(i=0;i<(2\*n-3);i++){  if(i==0 && schedule[j][i]<0){    teamDistance[j] += cityDistances[j][Math.abs(schedule[j][i])-1];  }    //2x home games or (1home 1away)  if(schedule[j][i]>0){    //not going anywhere 2x home  if(schedule[j][i+1]>0){  teamDistance[j] += cityDistances[j][j];  }  //1 home, 1 away  else{  teamDistance[j] += cityDistances[j][Math.abs(schedule[j][i+1])-1];  }  }  //1away 1home  else if(schedule[j][i+1]>0){  teamDistance[j] += cityDistances[Math.abs(schedule[j][i])-1][j];  }  //2 away  else{teamDistance[j] += cityDistances[Math.abs(schedule[j][i])-1][Math.abs(schedule[j][i+1])-1];  }    //If last game away return home  if(i==(2\*n-4) && schedule [j][i+1]<0){  teamDistance[j] += cityDistances[j][Math.abs(schedule[j][i+1])-1];  }  }  //Print out each team's travel distance  System.out.println("Team "+(j+1)+ " has a total travel distance of: " + teamDistance[j]);  totalDistance += teamDistance[j];  }  //Total distance traveled by all teams  System.out.println();  System.out.println("Total Team Travel Distance: " + totalDistance);  System.out.println();  //Total computation time  long endTime = System.currentTimeMillis();  System.out.println("Total time: " + (endTime-startTime) + "ms");  //Write schedule to file  BufferedWriter out= new BufferedWriter(new FileWriter("Solution "+fileName));  for (int a=0; a<n;a++){  for(int b=0;b<2\*n-2;b++){  out.write(String.valueOf(schedule[a][b]));  out.write("\t");  }  out.newLine();  }  out.close();  }  } |

**SAMPLE INPUT:**

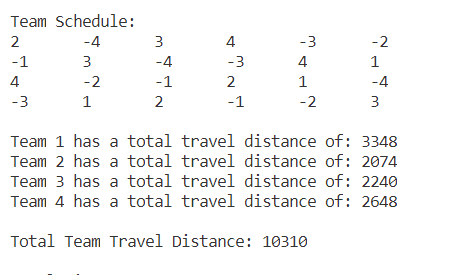
0 665 745 929

665 0 80 380

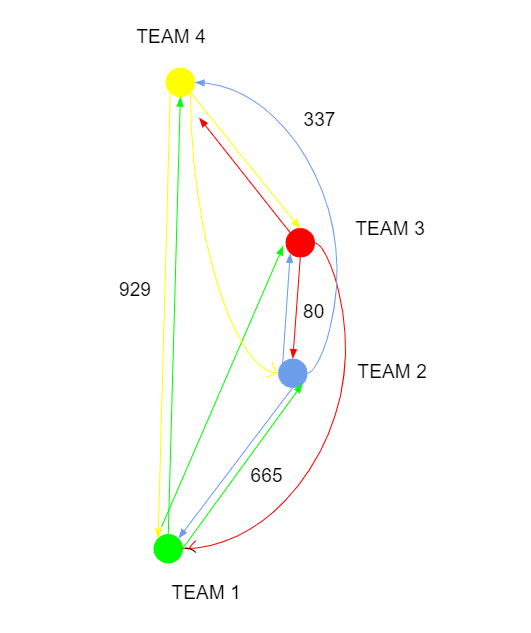
745 80 0 337

929 337 380 0

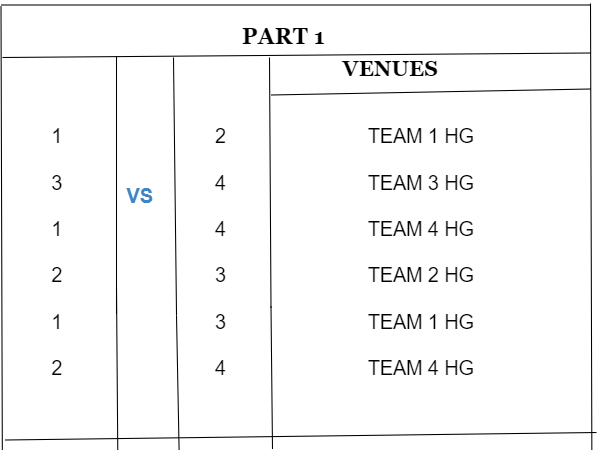
**OUTPUT:**

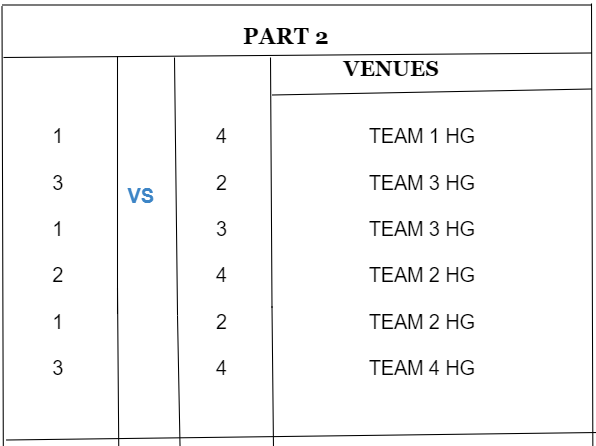


**BY USING ABOVE OUTPUT:**



**FINAL SCHEDULE**

****

****

**Advantages:**  
Having a good schedule for a tournament will always be an advantage, some of them are:

1. Reduces travelling distance and cost.
2. Provides enough time for the players to rest which automatically reduces the chances of injuries.
3. A good schedule can also lead to the increase in income of tournaments.

**Summary:**

We can summarise that , after effectively scheduling matches it provides the number of tournament schedules for the same tournament with the same constraints and help to save the partiality means it does allow the partiality for all the team. With the help of travelling salesman problem, we calculated the shortest possible distance of each team during the whole tournament. Minimizing the travel distance will automatically lead to less expenditure in travelling and hence promote increase in fan interest .

**Reference:**

1. <https://www.baeldung.com/java-simulated-annealing-for-traveling-salesman#:~:text=The%20Travelling%20Salesman%20Problem%20>(TSP,please%20take%20a%20look%20here.

2.<http://www.oxfordcroquet.com/manage/doubleroundrobin/index.asp#:~:text=A%20double%20round%20robin%20is,different%20court%20the%20second%20time>