Scheduling Tennis Tournament - Algorithm Design

1 Problem Statement

Scheduling tennis tournaments is contingent upon several crucial constraints (as outlined below). Our group aims to probe different optimization techniques to determine the most optimal schedule for each day of the tennis tournament, yielding maximum revenue. The rest of this report will provide a brief overview of the problem we are trying to solve.

2 Description

1. **Basic Rules Considered:**

   (a) **Number of players:**
   Since a match is played between 2 players and the tournament is being played in a knockout format, the number of players N required for the tournament should be a power of 2, i.e. $log_2 N$ should be an integer (in the context of this assignment, we have considered only singles matches).

   (b) **Player Popularity:**
   We have assumed direct relation between the popularity of a player and the audience they attract for their match. Further, the popularity would decide the percentage of tickets for a court which would be sold due to a single player. For example, if a player has popularity of 0.5, they will attract a 50% audience capacity of any given court. This forms the basis of our algorithm and how we schedule the matches on most profitable courts. Hence, we have restricted the popularity in the range $0 \leq p \leq 0.5$ such that the joint popularity of players for any match does not exceed 1.

   (c) **Formation of the Draw:**
   For tennis tournaments, draws are the most essential decision making tool for tennis fans. The draw places players into 2 halves and 4 quarters such that the top ranked players play against each other only during the latter stages (Quarterfinals, Semifinals and Final) of the tournament. Simply put, we can safely say the 1^{st} and 2^{nd} rank players can never meet before the final, considering they win all their matches up-to the final. That is the beauty of tennis draws. Hence, oftentimes some popular match-ups are projected during the initial rounds of the tournament and fans wait before buying tickets to see on which particular court that match is scheduled. Hence, scheduling popular match-ups on the most profitable court in terms of $c \times t$, where c is the seating capacity and t is the ticket price, is essential for optimizing the overall revenue of the tournament.
   Reference for how tennis draws work: https://mytennishq.com/how-do-tennis-draws-work-the-ultimate-guide

   (d) **Number of days to complete the Tournament:** To schedule a tournament with N players over d days, we can conclude the tournament in d_{min} days. Here $d_{min} = 2log_2 N$. This can be derived as:
   For a player to reach the final in a knockout tournament, he has to play at least $log_2 N$ matches. After every knockout round, the number of players are halved. Thus, $N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow ...1 = log_2 N$. Since, a player cannot play on consecutive days, the minimum number of days required for a tournament is $2log_2 N$.
   We choose d_{min} to maintain consistency and fairness in the algorithm. If additional days are given, we can certainly maximize those days to increase the overall revenue earned. However, here we want to ensure that the algorithm maintains fairness, i.e. for every match the opponents and all players in the draw have had same number of rest days except for one match as explained in the proof of fairness below. The minimum number of rest days required as per our constraint is 1 and to ensure that, 1 half of the draw needs to complete their matches on a single day. Further, there can be cases where we increase the number of rest days for each player and conclude the tournament in d > d_{min}, however even in such scenarios, based on the condition that a half needs to complete their matches on the same day, the revenue generated would remain the same and we can safely conclude the tournament in d_{min} days. Moreover, concluding the tournament in fewer days while respecting all constraints would mean additional savings of overhead costs from an organizer's perspective and would be an additional benefit of the algorithm.

   Proof of Fairness
   Players should have equal number of rest days throughout the tournament except for the Quarterfinals and Semifinals where 2 players of a particular half get an additional rest day to maintain fairness for the final match. Let us consider there are 128 players in our draw. In that scenario, we conclude matches as shown below:

Day	Half	Round	Players Eliminated
1	1st Half	Round 1	32
2	2nd Half	Round 1	32
3	1st Half	Round 2	16
4	2nd Half	Round 2	16
5	1st Half	Round 3	8
6	2nd Half	Round 3	8
7	1st Half	Round 4	4
8	2nd Half	Round 4	4
9	1st Half	Quarterfinals	2
10	2nd Half	Quarterfinals	2
11	-	-	-
12	Both	Semifinals	2
13	-	-	-
14	Both	Final	-

Table 1: Realization of draw for 128 players

From the table 1 we can see that the tournament concludes in $2 \times log_2 128 = 14$ days which is the required d_{min} where Day 11 and 13 are complete rest days having no matches. Here, fairness is maintained in the sense that all players get equal number of rest days except for the players in the first half who get an additional day off between their Quarterfinal and Semifinal match. We do this to ensure that for the finals, both players have got equal amount of rest days. Hence, in this way, the algorithm will maintain overall fairness if the tournament is to be concluded in d_{min} days and that is the constraint we follow going forward. Lastly, a point to be noted is that either half can start first. The example illustrates starting with the first half but in practice either half can go first as it would bear no effect on the overall revenue of the tournament.

(e) **Number of courts required for successfully scheduling the Tournament:**
We now have a constraint on the number of days required for the tournament. To satisfy the constraint of concluding the tournament in d_{min} days, we now need to check if a valid schedule is possible, i.e. whether a given half can complete their matches on the same day. Hence, we now need a lower bound on the number of courts available at our disposal to check for the validity of the Tournament. Since a given half plays on a single day, for 1^{st} round, we have $\frac{N}{2}$ players i.e. $\frac{N}{4}$ matches on Day 1. From data collected till date, an average tennis match lasts 3 hours (inclusive of warm-up, on-court interviews and court cleaning post matches). This would enable us to schedule up to 4 matches on a single court in a day. Therefore, to complete $\frac{N}{4}$ matches, we have the constraint on c, where c is the number of courts, as:

$$4c \geq \frac{N}{4} \implies c \geq \frac{N}{16} \tag{1}$$

2. **Formal Specification:**

(a) **Inputs**:

i. Integer N - number of players in the draw

ii. Integer d - number of days over which the tournament can be held. For $d \geq d_{min}$ days, the tournament will conclude in d_{min} days.

iii. $Courts[1...c]$ - list of courts, where c is the number of courts and each entry in the list is a tuple of form (String, Integer, Float) representing the name of the court/stadium, seating capacity, and the ticket price respectively.

iv. $Players[1...N]$ - list of players of size N where each entry in the list is a tuple of the form (String, Integer, Float) representing the name of the player, their ranking amongst all players in the tournament and their popularity respectively. Player popularity p satisfies $0 \leq p \leq 0.5$

Dynamic input for the optimization function

v. $Fixture[1...\frac{N}{4}]$ - a list of fixtures of size $\frac{N}{4}$ for a given day where each entry in the list is a tuple of the form (Integer, Integer) representing the ranks of the first and second players in a match. This is a dynamic list getting generated based on the available players N remaining in the draw. **Note:** For our implementation of the algorithm, this input will be dynamic for each passing day of the tournament. The size $\frac{N}{4}$ represents the number of matches in a day, where N is remaining players in the draw. By maximizing the revenue for the instance of a single day, we can generalize that the overall revenue earned over the course of the tournament would also be maximum as the same algorithm would be run on the input of remaining days. This input keeps changing for each knockout stage and cannot be determined in advance as it depends on the results of preceding rounds. Hence, we can say

that this dynamic input is taken by the organizers and our algorithm is run on a single instance of input and by running it for the entire tournament, we get the overall revenue which would be maximum based on proofs below.

(b) **Output**: $Matches[1...m]$ - a list of matches to be played on a given day, where each entry in the list would be a tuple of the form (String, Fixture[i]) representing the name of the court and the match(fixture) to be played. The fixture is of the form (Integer, Integer) representing the ranking of first and second players of the match respectively. No court would be present in the list more than 4 times as the maximum number of matches which can be played on a court is 4. Further, each fixture would be unique and the players would have had the same number of rest days before the fixture so as to maintain fairness in scheduling.

(c) **Optimization Task:** Our algorithm would aim to schedule fixtures on courts on the basis of popularity so as to maximize the tournament revenue. This idea is based on the assumption that popular players would attract more audience for their matches. The equation we optimize is:

$$\sum_{i=1}^{m}[(Matches_i(Popularity_{Player1}) + Matches_i(Popularity_{Player2})) \times (Matches_i(CP))] \tag{2}$$

where

$$\begin{aligned} CP &= \text{seating\_capacity} \times \text{ticket\_price} \\ m &= \text{Number of matches} \end{aligned}$$

3. **High Level Algorithm and Decision Making:**
We considered multiple approaches and scenarios to solve this optimization algorithm.

(a) **Backtracking:** For knockout tournaments, the number of matches per day reduces as the tournament progresses. Hence, if the input $d > d_{min}$, we could possibly assign additional days to a given knockout round so as to maximize profit. Since the number of matches are the maximum during the 1^{st} round, logically assigning the additional days to the first round would be a plausible optimization solution. Also as listed in constraints, the draw is such that the 1^{st} half and 2^{nd} half do not play on the same day. Hence, we would have to explore for all possibilities amongst the the two halves wherein assigning additional days to a given half would maximize profit. However, this would lead to the scenario where there are uneven rest days for players, thus, breaching the fairness policy for a given fixture in future or result in a situation where the tournament cannot be completed in d_{min} days. Hence, we eliminated this approach to maintain fairness and consistency in our algorithm.

(b) **Divide and Conquer:** The draws follow a tree structure starting from leaf nodes to the root, and hence divide and conquer may seem to be an intuitive approach. At the knockout stage level, one half of the current pool of players play on one day while the other plays on the next. A half H needs to complete their matches on the same day to maintain fairness (equal rest days for all players) for future fixtures. However, since we cannot predict outcomes of previous stage fixtures, we can only schedule matches on a per knockout stage basis. In such a case, there are no sub-problems that the main problem can be divided into that could suit the divide and conquer approach given the constraints.

(c) **Greedy Solution:** After considering the above approaches, Greedy algorithm is the one we finalized. For the greedy solution, fixtures in $Matches$ would be sorted in descending order on basis of the joint popularity of the players in the fixture i.e. $p_1 + p_2$. Post that, we sort $Courts$ in descending order on the basis of the value (seating_capacity $\times$ ticket_price). Finally, we assign the fixtures with highest popularity to the courts yielding most profit until 4 matches have been assigned to a court. Then we move on to the next court and assign the next 4 most popular fixtures and so on. Here, we will not run out of available courts for the day as we check the feasibility of the tournament before starting the optimization algorithm. Hence, we keep assigning the fixtures to courts until there are no more fixtures left for the given day. The proof of correctness can be found in section 6 of this report.

3 Algorithm Working

Each day of the tournament can be treated independently as all the courts are again available to schedule the matches on the next day and the fixtures are also different on each day which results in different revenue generation. Thus, optimizing the revenue on each day will maximize the total revenue for the complete tournament.
Let us now see the steps that needs to performed in order to use the functions mentioned below under Pseudocode section for organizing the complete tournament.

1. We take N, c, d, i.e. Number of Players, Number of Courts, and Number of Days over which tournament can be held, as inputs and validate whether the tournament is possible using the function *IsTournamentPossible*.

2. Sort the courts in descending order using the value (seating_capacity × ticket_price).
3. For each day of the tournament, do the following:
   (a) Get the fixture for the current day of the tournament using *GetFixturesForADay* function and passing the current day as input.
   (b) Invoke *GetSchedule* and pass the list of fixtures obtained in the previous step. This will return a list of matches which will have each fixture associated with a court.

4 Pseudocode

Algorithm 1

function IsTournamentPossible(N, d, $Courts$)
  if N is not a power of 2 **or** $d < 2 \times log_2(N)$ **or** $len(Courts) < \frac{N}{16}$ **then**
    return false
  end if
  return true
end function

function GetFixturesForADay(d)
  return list of fixtures for the day d of the tournament.
end function

function GetSchedule($Courts$, $Fixtures$, $Players$)
  Sort $Fixtures$ in descending order based on the joint popularity of players in each of the fixture
  matchIndex ← 0
  courtIndex ← 0
  Matches ← An array of tuples of size equal to len($Fixtures$)
  while $matchIndex < len(Fixtures)$ **do**
    $Matches[matchIndex] \leftarrow (Courts[courtIndex][0], Fixtures[matchIndex])$
      ▷ Courts[courtIndex][0] fetches the name of court at index courtIndex
    Increment matchIndex by 1
    if $matchIndex$ MOD $4 = 0$ **then**
      Increment courtIndex by 1
        ▷ Once 4 matches are scheduled on a court, we move to the next most profitable court
    end if
  end while
  return Matches
end function

5 Time Analysis

Complexity of the algorithm for one day in a tournament:

- Initially, *IsTournamentPossible* is called to check if the input provided is valid or not, which is O(1).
- The next step is to call *GetFixturesForADay* to get fixtures for a given day of the tournament. This adds another O(1) to the time complexity.
- The third step is to call *GetSchedule* method which returns the fixture on a given day. This function first sorts fixtures based on joint popularity of players which results in $O(\frac{N}{4}log(\frac{N}{4}))$, where N is the number of available players in the draw $\frac{N}{4}$ is the number of matches played on a day. Lastly, the loop iterates $\frac{N}{4}$ times until all the fixtures are assigned a court. Therefore, the total time complexity of *GetSchedule* function for a day is $O(\frac{N}{4}log(\frac{N}{4}) + \frac{N}{4})$

  Since a knockout stage comprises of 2 days - Time complexity = $O(\frac{N}{2}log(\frac{N}{4}) + \frac{N}{2})$ where N is the number of available players in the draw.

Complexity of the complete tournament:
Extending the above logic for the entire tournament, we can compute the total time complexity for scheduling the tournament, which would be the sum over all the knockout stages. Additionally, we would need to sort the courts once on the basis of (seating_capacity × ticket_price) for the entire tournament. Note that the size of the player pool reduces by half at each stage.

Sorting of courts : $clog(c)$
If N is the total number of players at the beginning of the tournament -
Knockout stage 1: $\frac{N}{2}log(\frac{N}{4}) + \frac{N}{2}$
Knockout stage 2: $\frac{N}{4}log(\frac{N}{8}) + \frac{N}{4}$
Knockout stage 3: $\frac{N}{8}log(\frac{N}{16}) + \frac{N}{8}$ and so on.

Summing over all knockout stages = $\frac{N}{2}log(\frac{N}{4}) + \frac{N}{4}log(\frac{N}{8}) + \frac{N}{8}log(\frac{N}{16})... + \frac{N}{2} + \frac{N}{4} + \frac{N}{8}... = O(2Nlog(n) - 3Nlog(2) + N)$
Total time complexity = $O(clog(c) + Nlog(N) - 3Nlog(2) + N) = O(clog(c) + Nlog(N))$

6 Proof of correctness

Claim: The greedy algorithm returns the maximum revenue for each day and thus maximizing the overall revenue for the complete tournament.

Proof using Greedy Stays Ahead
Proof:
Base case : When there is only one fixture, the algorithm will choose a court which has the highest value of (seating_capacity × ticket_price), which is correct as that will maximize the revenue. So the base case holds true.

Induction Step:

1. Let us consider an optimal schedule so far, OS, which selects a court c for k^{th} fixture which results in the maximum revenue.
2. Let us now consider another optimal solution so far, GS, which selects court c' for k^{th} fixture based on the reasoning that c' is the court which has a maximum value of (seating_capacity × ticket_price) and $c \neq c'$.
3. Based on the reasoning for the selection of c', we can say that (seating_capacity of c × ticket_price of c) ≤ (seating_capacity of c' × ticket_price of c'). If selecting c is maximizing the revenue, selecting c' will always result in either a better or an equally optimal schedule that maximizes the revenue.
4. Thus, we can say revenue of $OS \leq$ revenue of GS if we select c' using greedy approach. Thus the induction step also holds true.

Since, greedy algorithm always selects a court which results in a revenue that is higher or equal to an optimal solution and both base case and induction step holds true, we can say that greedy algorithm always result in the maximum revenue for each day and thus maximize the overall revenue for the complete tournament.

Proof using Regular Induction
Proof:
Base case : When there is only one fixture, the algorithm will choose a court which has the highest value of (seating_capacity × ticket_price), which is correct as that will maximize the revenue. So the base case holds true.

Induction step :

1. Let us assume, that we have scheduled $k-1$ fixtures using the available courts and that resulted in the maximum revenue so far. Let us denote the maximum revenue so far as $R = \sum_{i=1}^{k-1} r_i$, where r_i is the revenue generated from each court i.
2. For k^{th} fixture, we choose a court c_k that has the highest value of (seating_capacity × ticket_price). Choosing this court will result in a revenue r_k.
3. Since the courts and fixtures are sorted in descending order on the value of (seating_capacity × ticket_price) and $(p_1 + p_2)$ respectively where p_1 and p_2 is the popularity of the two players, we can say that choosing c_k would result in the maximum revenue because there would be no other fixture that would result in a value greater than r_k using court c_k as well as there would be no other court that would result in a value greater than r_k.

4. Therefore, after selecting court c_k for the k^{th} fixture and since R is maximum so far, the total revenue would be $(R + r_k)$ would also result an optimal schedule that generates the maximum revenue. Thus, the induction step holds true.

Since, the base case and induction step both holds true, the above algorithm will maximize the revenue for each day, and thus maximizing the overall revenue for the complete tournament.