Winner Probability Estimation in a Knockout Tournament

Prasansa Mapara *201901097*
Aniruddh Muley *201901222*
Hasti Doshi *201901302*
Srushti Menpara *201901307*
Adit Shah *201901454*
Darshil Shah *201901467*

***Abstract*—In this paper, we have worked on finding the probability of a team/player winning in a tournament, conducted in a single-elimination Knockout format. The paper adopts a sequential approach of solving the problem, where it tries to find out the solution initially without any time constraints (brute force approach), and later optimizes the time taken.**

I. Problem Statement

Given a scenario of a knockout tournament conducted on a large scale, this paper tries to solve the problem of predicting the winner of the tournament based on the previous record of the participants/teams. We will have access to the previous record of all matches between each pair of teams involved in the knockout tournament. We consider the tournament played without injuries, giving results(no draw or tie).

II. Introduction

A. Aim

The aim of this paper is to come up with an approach to estimate the probability of a team winning in a knockout tournament.

B. Objective

The main objective of this paper is to provide a working mechanism that will optimally determine the winning percentage.

C. Background

A large number of tournaments are conducted for a variety of sports. The format of tournaments is decided by the nature of the sport, the time taken for each match, and available time. This gives rise to a number of formats in which tournaments can be conducted like Round Robin, Knockout, Swiss-system etc. The nature of sports tournaments organized around the world has preferably been of single-elimination knockout nature. They are generally preferred when we have a large number of participants and the time taken to conduct this is less than the format carried out in formats of league stages. In the Olympics, a variety of events are conducted in Single Elimination format, like Archery, Tennis etc. This helps in conducting the events with ease, and comparatively in less time.

The format of a Knockout tournament is such that for every duel, the winner gets promoted to the next round, and the losing player gets eliminated from the tournament. It takes less time compared to other alternatives like round-robin formats. To prevent the top players from getting eliminated in the initial rounds, we can also consider seeding the players, which is arranging the players in decreasing order of performance/skill.

The below diagram illustrates the way the fixtures of knockouts are organized.

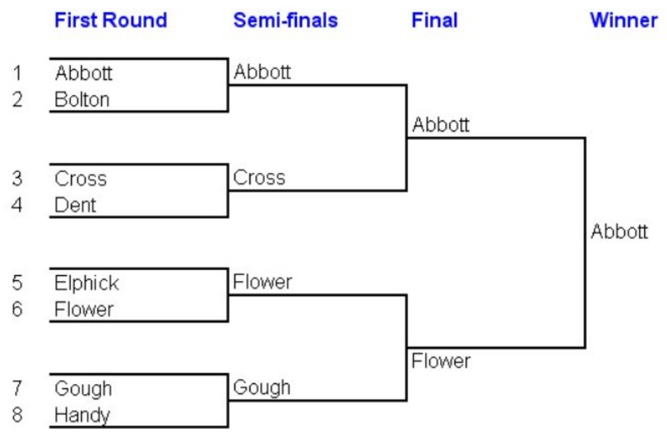


Fig. 1. Single Elimination Knockout tournament Format [9]

The sports industry has become a business in itself, be it for the players competing, or the businesses investing. As a sport gets popular, the money invested in it increases, leading to the sport entering domains like branding, advertisements and increased use of technology. So the winner of the competition matters for the sponsors/brands so that they can invest in these players early on, get them to endorse their products and earn profits. Getting deeper into the business aspect, the talent handling companies have a watch on a particular set of players, who they feel can achieve great success, and they start investing in them from their ripe ages.

This then spreads over to the concept of franchises which is now a popular phenomenon, not just for a sport that has gained importance, but also as a way of increasing the popularity of a sport.

The competition studied in this paper is hierarchically shaped, which has been attracting audiences by enabling the status quo where topmost players, or in technical terms, highly seeded players compete with each other in the higher stages of the competition. The article considers one such scenario, where the players are participating in a knockout tournament, and its end goal lies in finding out the probable winner of the tournament, using Probability Estimation Techniques, in the most optimized manner. Using the concept of Dynamic Programming, we develop efficient, stochastic and deterministic sub-optimal algorithms to find the solution of our problem statement, after considering the required assumptions. The paper will present experimental results derived from the CPP and Python implementation of the proposed algorithms, considering the factors like optimality of solutions and running time efficiency.

III. Brief Literature Review

The team hovered over a large number of resources available online, helping in understanding many things like the overall concept of dynamic programming coupled with bit-masking, the effectiveness of the algorithms and how widely knockout tournaments are played throughout the world.

We looked over the details of the Knockout tournament format on Wikipedia [1], understanding why knockout tournaments are popular in the world of sports. The Knockout Tournament format pushes even the best of the sportsmen to perform to their best abilities, in a bid to avoid elimination, resulting in a good display of the game. The second advantage it poses is that the tournament ends quickly and hence the cost of conducting the tournament is less. However, along with these advantages, the team members also discovered various flaws, like the use of Byes in cases when a participant does not take part in a duel for any reason. [2]

The paper [3] on dynamic programming by Biswajit Bhowmik helped in understanding the basics of dynamic programming and how to implement it effectively. Using the HackerEarth platform [4] also helped in understanding how bit masking helps to solve questions in dynamic programming. For the mathematics and understanding of bit masking, the article referred to was [5]

IV. Plan of Work

A. *Week 1*

We explored various domains where optimization can be used before choosing our final topic, a subject that is relatable to a larger group of people, which helped us in zeroing down to the sports industry. After a lot of discussions, our team decided to work on the problem of estimating the probability of a player/team winning in a knockout tournament.

B. *Week 2*

We referred to various papers published in academia related to knockout tournaments and the probabilities involved in knockout tournaments, the time complexities behind this estimation problem, and how those complexities can be optimized further using data structures, and optimization algorithms.

C. *Week 3*

We had a thorough discussion regarding the algorithm that we can use for our project. We decided to go ahead with dynamic programming and bitmask as it is best suited to the problem which we are solving. The fact that the team can either be in the tournament or get eliminated, led us towards bitmasking, and moving forward we observed that there were recurring sub problems, which paved our way towards using Dynamic Programming.

D. *Week 4*

This week, we mainly focused on developing our algorithm. We worked on formulating the algorithm we are planning to implement, in layman terms, and tried to evaluate the possible DP states and corresponding cases possible. We also formulated a brute force solution for the problem.

E. *Week 5*

This week we finalized the maximum number of teams we were going to have for our code and also implemented our algorithm in the bruteforce method.

F. *Week 6*

This week we focused on how to implement our optimized code using DP and bitmasking, and what functions we can use. The motivation behind using DP was to memoize the previously calculated results for later use. We then came up with a C++ code for our planned bitmask i.e. valid mask and submask for the permutations of winning and losing teams. We then converted our logic for the DP approach into a C++ code.

G. *Week 7*

We looked into various tech options available and feasible to convert this code into a working prototype. We decided to build a web application in Django. Since Django is a python based framework, we first decided to convert our C++ code in python. Thereafter, we went on making a basic frontend design for taking input from the user.

H. Week 8

We focused on backend part of the application and connected it with sqlite3 database to store data of previous matches between various teams. We worked on making our interface more user friendly.The reason behind making this user-friendly interface was to make our algorithm more practical since a real user would not like working with this code on a terminal. The user would prefer to interact with an interface where they just need to make some clicks on the interface to exploit the algorithm we're proposing.

V. Technology and Literature review

The algorithm to the solution of the problem statement was implemented in C++ language, both the Brute Force approach and the optimized one. In the brute force approach, we have used some functionalities from the Standard Template Library available in C++, namely vector, and a map. Vector is a form of a dynamic array having the ability to resize itself as required. Map is a form of a user defined dictionary where we can have key, value pairs. In map, we can only have unique keys. We have used map of vectors in our brute force code.

We have also made a working web application implementing this algorithm. The web application is based on Django [11], which is a python based framework. Frontend part of the application is made using HTML and CSS while the backend is based on Python, its numpy [12]library and sqlite3 database.

The present system does not require any hardware to be used. Instead, we then tried our hands over creating a working prototype of the model by developing a web application as mentioned above.

VI. Approach to the solution

We explored different topics for choosing our projects.
The key points involved while exploring various topics were:

- It should solve an existing, and impactful real-life problem.
- Its solution should not be already available and hence it could be helpful if we want to build something onto it.
- It should solve one of the most important optimization problems. This helps to make the system highly responsive when dealing with large data.
- It should have some real-life application.

A. Topics explored for selecting project

1) Binary Search: This works in optimising time complexity when data is sorted in a particular order, like ascending or descending. This helps in optimising search time complexity drastically. However, a majority of real-life problems don't have their data in sorted order. Instead, they prefer using some other data structures like trie or segment tree for reducing search time as they work better with data of the real world.

2) Using machine learning model: The idea of machine learning as an optimisation problem was mainly due to the wide range of practical applications which it serves. It is used everywhere, in search engines for recommendations, in automobiles for making self-driving cars, in medicine for cancer detection etc. We have many algorithms available and they have proven to be highly useful. However if we develop a new algorithm using machine learning concepts, its effectiveness isn't guaranteed unless deployed in practical scenarios. Hence we did not go further with machine learning.

3) Dynamic Programming (DP): The basic idea of dynamic programming is to pre-compute the values that will be used again and again, and use them whenever required directly without spending the time computing them again. This concept is highly useful which reduces time complexity drastically. However, we struggled to find something that actually solves an impactful problem. Dynamic Programming by itself doesn't have many applications. However, when we couple it with bit masking, we find a better range of problems to be solved. This is because, in the real world, we can visualise that an entity can have a limited number of states. For example, students can either be enrolled in a course or not. This paves the way for dynamic programming with bit masking which we have actually selected as our topic for this project.

B. Towards the solution of the problem

1) Approaching the solution: As stated earlier, we are considering a knockout tournament. Here each team either sustains in the tournament or gets eliminated. Each round ends with half the number of teams getting eliminated. For example, if there are 16 teams in qualifiers, 8 teams will be there in the quarterfinals, 4 teams will be there in semifinals, 2 teams in finals and 1 player shall finally win. The idea is to implement this in the most optimised way possible.

2) Assumptions for writing the code: We are assuming several things while writing the code. The first assumption is that the total number of teams will be in powers of 2. However if it is not a power of 2, we make the number of teams to the next power of 2 by giving bye. Since the knockout tournament system can be applied to players also , we will assume that the player is not injured or unavailable. We also assume that any match doesn't end up in a tie and one team will be declared as the winner that moves to the next round. We also have previous record of matches available for each pair of teams/players involved in the tournament.

3) Brute for approach: The brute force approach uses the most basic idea that in each round all teams play one match and either qualifies to next round or get eliminated. The process is repeated till only one team remains and all other teams get eliminated. The pseudo code for the same is as follows :

4) How will DP help: The basic idea as discussed above is to store the already computed values so that we don't need to recompute that. Let us understand this with an example. Let's assume that teams 1,2,3 and 4 are in one pool. Similarly teams 5,6,7 and 8 are in one pool and so on. Let's assume teams 4 and 8 enter the semi-finals. Now from team 4's perspective, it doesn't matter how team 8 entered the semifinal, defeating team 5,6 or 7. So we don't need to calculate the same thing differently. The only thing that matters is storing the probability of team 8 into the semifinals and hence we will store it for future use. This is exactly what we are doing by applying the knowledge of dynamic programming.

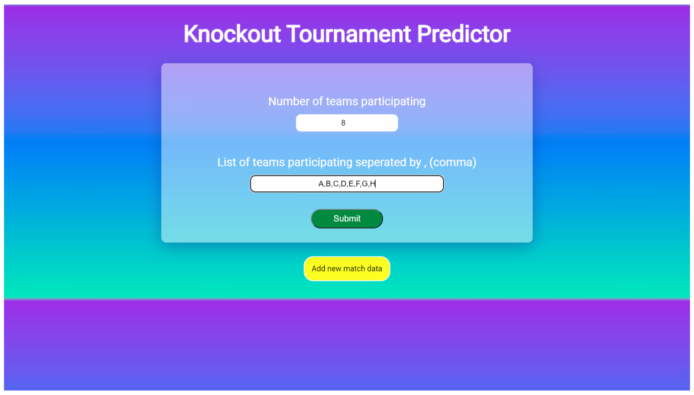
5) The DP states: We will have to make two DP tables to handle the problem. The first DP table will have two states. The first state indicates that the team is still in the tournament and the second state will indicate that the team is eliminated from the tournament. The second DP table will show whether the team is already eliminated, expected to win or expected to lose. This DP table will help us to make transitions for the first DP table.

6) The final solution: We are working on the code for implementing the same. We are implementing the code currently in the CPP language that will serve as logic and working code for the idea which we intend to solve. Later on, we shall attempt to make a working interface using the same logic and code which we develop in CPP.

VII. System Description

A. Home Screen of Application

Home screen of the application contains 2 input fields, asking the user to enter number of teams and comma-separated list of teams respectively. As soon as the user presses submit button, the data entered by user is sent to the server using POST method for further processing.


Knockout Tournament Predictor

Number of teams participating

8

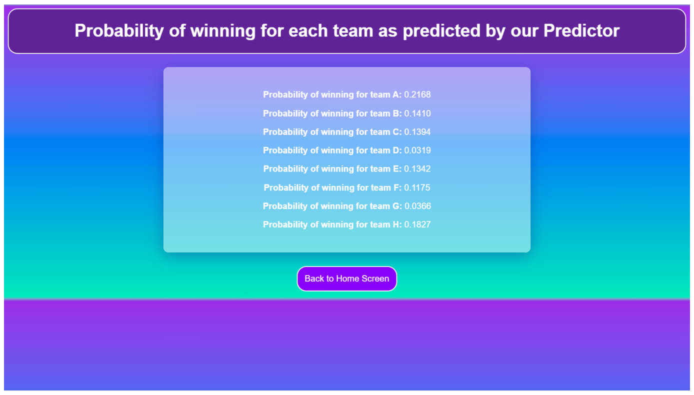
List of teams participating seperated by , (comma)

A,B,C,D,E,F,G,H

Fig. 2. Interface that collects list of participatin g teams

B. Generic Backend Processing

First of all, on receiving the data from user, a validation is run and a warning is generated if the number of teams is not a power of 2, or list of teams is incomplete. This warning is displayed to the user and no further processing happens for this inappropriate data. If the data passes validation check, the code moves towards generating a probability matrix prob where prob(i,j) denotes the probability of team i winning over team j. This value is calculated by dividing total wins of i over j by total matches played between i and j.This previous record of matches for all pair of teams is already stored in a sqlite3 database, it is fetched from there and these computations are made on it. From here, solve function is called which is described later in this section and after performing all the calculations, probability of winning the tournament for each team is returned back here from solve function. These values are sent back to the client end for user to see the prediction results.


Probability of winning for each team as predicted by our Predictor

Probability of winning for team A: 0.2168
Probability of winning for team B: 0.1410
Probability of winning for team C: 0.1394
Probability of winning for team D: 0.0319
Probability of winning for team E: 0.1342
Probability of winning for team F: 0.1175
Probability of winning for team G: 0.0366
Probability of winning for team H: 0.1827

Fig. 3. Interface that displays results

C. Feature to dynamically add data of new matches

The database of previous matches can not be kept purely static. It will get outdated after a certain point of time if data of new matches happening is not added into the database. To avoid this issue and to keep the database updated, there's a feature to add data of new match where user needs to enter name of winning team and losing team in a match and corresponding data will be updated in the database for both the teams.

Various functions used in the code of our application are described below.

D. Mask for a dp1 array

It will help us to depict which teams which are eliminated from the tournament and which are still in the tournament. This is depicted by a bitmask with base 2. The representing bits can be 0 or 1.

- 0 represents that the team is eliminated
- 1 represents that the team is still in the tournament and not eliminated.

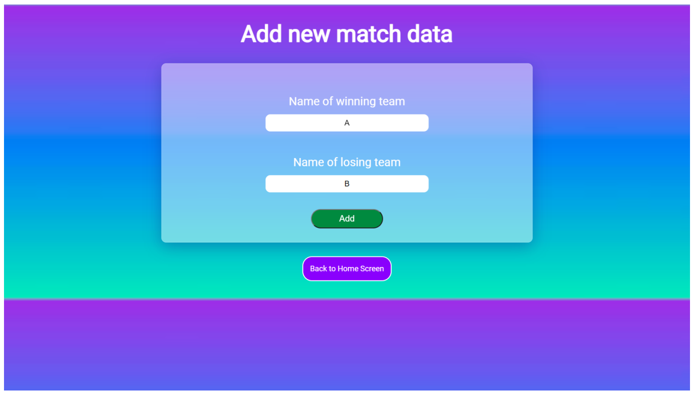


Fig. 4. Interface that collects data of new matches

E. Mask for a dp2 array

It will help us to depict which teams are eliminated from the tournament and which are still in the tournament and expected to win or lose. This is depicted by a bitmask with base 3 . The representing bits can be 0, 1 or 2.

- 0 represents that the team is already eliminated.
- 1 represents that the team is expected to win and go to next round.
- 2 represents that the team is expected to lose and get eliminated.

F. The solve function

This function helps us to calculate the final answer and store it in dp1 which we are filling by bottom-up approach. This solve function makes calls to functions like valid_mask, valid_submask and calculate_probability which are explained below. The final answer for any team will be when a mask of dp1 will only have 1 bit set. We store this in the answer array from dp1.

G. The Prob_Mask_To_Submask function

This function has 2 tasks :

- Convert mask for dp2 by taking arguments as mask and submask of dp1. This is achieved by comparing the set and unset bits of mask and submask of dp1. If the bit is set in dp1 mask , then it may be set or unset in submask depending upon whether it is expected to win(bit 1) or lose(bit 2). If the bit is unset in the mask it is bound to be unset in the submask as the team is already eliminated(bit 0).
- Calculate probability of this new mask of dp2 generated above. This is achieved by calling a function called calculate_Probability. The working and description of this function is given below.

H. The valid_mask function

It takes the mask as an argument and checks whether this mask is practically possible. To do so it checks whether the number of set bits is a power of 2 or not because at any round, the number of teams remaining in the tournament is power of 2.

I. The calculate_probability function

This function is responsible for filling the dp2. We are filling the dp2 array by top-down approach and hence the calculate_probability function is a recursive function. We give the mask as an argument to this function. The function works as shown below:

- There will be some bits 0, some bits 1 and remaining bits 2. The number of 1 bits and 2 bits shall be equal as the number of winning and losing teams are equal. Now we choose the last losing bit and pair it with all possible winning bits.
- For each pair, we multiply the probability of winning for the chosen winning team against the chosen losing team. Now we continue the same till all the teams have either won or lost or are already eliminated. To do so , we make a recursive call on calculate_probability again.
- This time when we make the recursive call , we pass the same mask again but with the chosen pair of bits set to zero as one of them has won and moved to the next round and the other is already eliminated. So both these teams are inactive for this recursive call and hence set to 0.
- Finally we memoise the answer so that next time we are given the same mask , we don't need to recalculate the same again. This is indeed very necessary for a top-down dp.

J. Base cases for dp1 and dp2

The following conditions serves as base cases for dp1 and dp2 :

- For dp1 - Initially all teams will be in the tournament. So the probability for the mask=$(2^N - 1)$ will be 1. For all other masks, the initial probability will be 0 as will be calculated by the solve function.
- For dp2 - We know that eventually all teams will either be eliminated or go to the next round or will be eliminated in this round. So, after all matches are played , the final mask will be 0 when all the bits will be set to 0. So, the probability of mask=0 will be 1. For all other masks, initial probability will be set to -1 , which will indicate that probability for this mask is not calculated yet. It will be calculated later on by calculate_probability function.

K. The auxiliary functions

The following auxiliary functions are also used :

- The count_set_bits : This function counts the number of set bits for a 2 base mask which is there for dp1. It is achieved by iterating all the bits of the mask.
- The get_bit function - This function is used to find the ith bit of the 3 base mask which is there for dp2. This is achieved by first taking the modulo of the mask with 3^{i+1} to make all bits on the left of ith bit to 0 and then dividing with 3^i to remove all bits on right of ith bit. So we are left with ith bit which we return.

L. *Time and Space complexities*

The time and space complexities is as follows :-

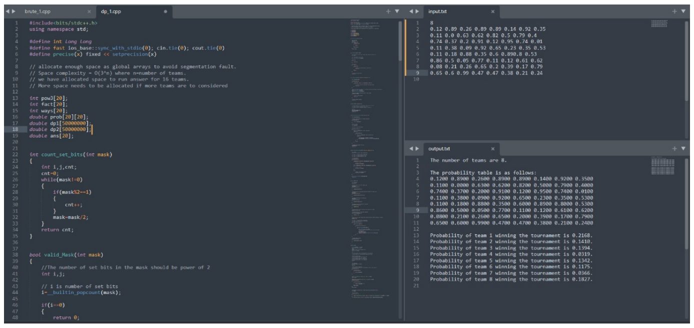
- Time complexity : The time complexity of the code is O($N * 3^N$) , where N is number of teams. This is because the for filling dp1, we have used the optimisation [10] for generating all sub-masks for given masks which is a great optimisation over doubly nested for loop each of O(2^N) which would result in overall O(4^N). For dp2, we have used the calculate_probability function which calculates answer for each mask in O(N) which will result in overall complexity of O($N * 3^N$). All other calculations take negligible time compared to this. Hence for $N <= 16$, we code will take a few seconds to run.

- Space Complexity : The space required for storing dp2 array is O(3^N) and that of dp1 is O(2^N). Sizes of all other arrays are negligible in comparison to these arrays and hence overall space complexity is O(3^N). Hence for $N <= 16$, the space consumed will be around 100 MB.

VIII. Result, Discussion, and Conclusion

A. *Result and Discussion*

Below attached is the simulation over the optimized code for number of teams = 8. Given the probabilities of teams winning over every another team we get the probability of a team winning the tournament individually.


brute_1.cpp
dp_1.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;

#define int long long
#define fast ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0)
#define precise(x) fixed << setprecision(x)

// allocate enough space as global arrays to avoid segmentation fault.
// Space complexity = O(3^n) where n=number of teams.
// we have allocated space to run answer for 16 teams.
// More space needs to be allocated if more teams are to considered

int pow3[20];
int fact[20];
int ways[20];
double prob[20][20];
double dp1[50000000];
double dp2[50000000];
double ans[20];


int count_set_bits(int mask)
{
    int i,j,cnt;
    cnt=0;
    while(mask!=0)
    {
        if(mask%2==1)
        {
            cnt++;
        }
        mask=mask/2;
    }
    return cnt;
}


bool valid_Mask(int mask)
{
    //The number of set bits in the mask should be power of 2
    int i,j;

    // i is number of set bits
    i=__builtin_popcount(mask);

    if(i==0)
    {
        return 0;
```

input.txt

```
8
0.12 0.89 0.26 0.89 0.89 0.14 0.92 0.35
0.11 0.0 0.63 0.62 0.82 0.5 0.79 0.4
0.74 0.37 0.2 0.91 0.12 0.95 0.74 0.01
0.11 0.38 0.09 0.92 0.65 0.23 0.35 0.53
0.11 0.18 0.88 0.35 0.6 0.890.8 0.53
0.86 0.5 0.05 0.77 0.11 0.12 0.61 0.62
0.08 0.21 0.26 0.65 0.2 0.39 0.17 0.79
0.65 0.6 0.99 0.47 0.47 0.38 0.21 0.24
```

output.txt

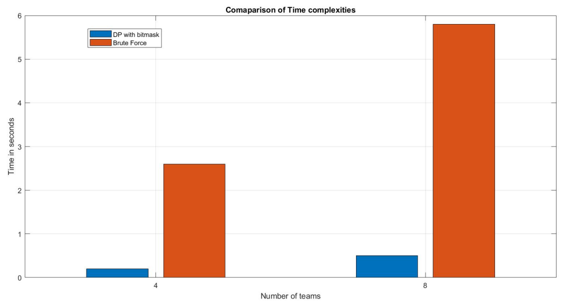
```
The number of teams are 8.

The probability table is as follows:
0.1200 0.8900 0.2600 0.8900 0.8900 0.1400 0.9200 0.3500
0.1100 0.0000 0.6300 0.6200 0.8200 0.5000 0.7900 0.4000
0.7400 0.3700 0.2000 0.9100 0.1200 0.9500 0.7400 0.0100
0.1100 0.3800 0.0900 0.9200 0.6500 0.2300 0.3500 0.5300
0.1100 0.1800 0.8800 0.3500 0.6000 0.8900 0.8000 0.5300
0.8600 0.5000 0.0500 0.7700 0.1100 0.1200 0.6100 0.6200
0.0800 0.2100 0.2600 0.6500 0.2000 0.3900 0.1700 0.7900
0.6500 0.6000 0.9900 0.4700 0.4700 0.3800 0.2100 0.2400

Probability of team 1 winning the tournament is 0.2168.
Probability of team 2 winning the tournament is 0.1410.
Probability of team 3 winning the tournament is 0.1394.
Probability of team 4 winning the tournament is 0.0319.
Probability of team 5 winning the tournament is 0.1342.
Probability of team 6 winning the tournament is 0.1175.
Probability of team 7 winning the tournament is 0.0366.
Probability of team 8 winning the tournament is 0.1827.
```

Here is the github link for CPP code for simulation of stated scenario.

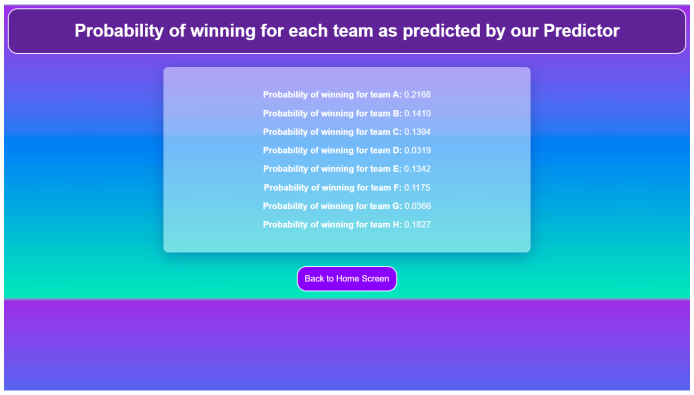
Below attached is the graph for comparison between the brute force results and optimized dynamic programming with bitmasking results. It is quite evident from time complexities stated in above section that our optimized code is much efficient than brute force algorithm.

Here is the drive link showing images in which time taken for different simulations.

So we can clearly witness the difference between brute force's factorial complexity and exponential dynamic programming's approach complexity.



Below is the output window of web application using we made using the same probabilities of inputs shown in CPP code above.


Probability of winning for each team as predicted by our Predictor

Probability of winning for team A: 0.2168
Probability of winning for team B: 0.1410
Probability of winning for team C: 0.1394
Probability of winning for team D: 0.0319
Probability of winning for team E: 0.1342
Probability of winning for team F: 0.1175
Probability of winning for team G: 0.0366
Probability of winning for team H: 0.1827

Here is the github link for web application.

B. *Conclusion*

From results and our observations we conclude that dynamic programming optimization technique is indeed quite useful in areas which are calculation intensive and use data based on past events. In this paper, we focused on knockout tournaments which are globally popular and which require a lot of calculation for its predictions. Hence this paper provides an optimal way with a working mechanism for determining the same, using past data of team's relative performance against other teams.

References

[1] https://en.wikipedia.org/wiki/Tournament.

[2] WHAT IS THE CORRECT WAY TO SEED KNOCKOUT TOURNAMENT. https://tinyurl.com/3byxu736 by Allen J. Schwenk.

[3] DYNAMIC PROGRAMMING – ITS PRINCIPLES, APPLICATIONS, STRENGTHS, AND LIMITATIONS by Biswajit R Bhowmik (https://tinyurl.com/s7fjsrkh)

[4] https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/.

[5] https://medium.com/analytics-vidhya/bits-bitmasking-62277789f6f5) by Aman Agarwal.

[6] https://inginious.org/course/competitive-programming/dp-knockout?lang=en .

[7] https://www.jstor.org/stable/2238623?read-now=1refreqid=excelsior

[8] https://www.mykhel.com/cricket/ipl-head-to-head-records-s4/

[9] https://www.allabouttabletennis.com/running-a-tournament.html

[10] https://cp-algorithms.com/algebra/all-submasks.html

[11] https://docs.djangoproject.com/en/3.2/

[12] https://numpy.org/doc/stable/