DMC April 2017 Geek Challenge Solution

Mike Deck, michaelanthonydeck@gmail.com

April 9, 2017

1 Introduction

The problem is stated on the DMC, Inc. website - click here to access.

2 Solution

2.1 Main solution

First, we set forth the probability of having a 100% correct bracket in a given year:

$$p^n \tag{1}$$

where p is the constant probability of the favorite winning a particular game, and n is the number of games held in that year's tournament to determine the NCAA champion. It follows that the probability of having **any errors whatsoever** in the bracket is:

$$1 - p^n \tag{2}$$

We can quickly see that the probability of having imperfect brackets in all 50 independent years is:

$$(1 - p^n)^{50} \tag{3}$$

Thus, the probability of having **at least one** perfect bracket in this set of 50 years, is:

$$1 - (1 - p^n)^{50} \tag{4}$$

Now to resolve that pesky n. The vast majority of the time, your bracket pool will ask you to make picks for 63 games. If this is the case for you, set (4) equal to 50%, plug in $n = 63$, and solve for p:

$$
\begin{aligned}
0.5 &= 1-(1-p^{63})^{50} \\
-0.5 &= -(1-p^{63})^{50} \\
0.5 &= (1-p^{63})^{50} \\
\sqrt[50]{0.5} &= 1-p^{63} \\
p^{63} &= 1-\sqrt[50]{0.5} \\
p &= \sqrt[63]{1-\sqrt[50]{0.5}} \\
p &\approx 93.424\%
\end{aligned}
$$

Huzzah! The answer is C.

2.2 General formula for p

From the above, it is easy to generalize the formula for p in terms of the number of brackets filled out:

$$p = \sqrt[63]{1-\sqrt[b]{c}} \tag{5}$$

where b is the number of brackets filled out, c is the probability of **at least one** perfect bracket.

3 But wait, there's more!

3.1 What if my pool picks the play-in games too?

I noted that there are 63 games in your pool most of the time. But since 2001, there have been extra "play-in" games that happen prior to the official start of Round 1. From 2001-2010, there was a single extra game for a total of 64, and since 2011 there have been four extra games for a total of 67. You can plug those values into (4) above instead to find alternative solutions that take the play-in games into account.

For 2001-2010:

$$
\begin{aligned}
p &= \sqrt[64]{1-\sqrt[50]{0.5}} \\
p &\approx 93.523\%
\end{aligned}
$$

For 2011-present:

$$
\begin{aligned}
p &= \sqrt[67]{1-\sqrt[50]{0.5}} \\
p &\approx 93.804\%
\end{aligned}
$$

3.2 An even more general formula for p

We can modify (5) to account for a different number of play-in games:

$$p = \sqrt[n]{1 - \sqrt[b]{c}} \tag{6}$$

by simply replacing 63 with n to represent the total number of games in the bracket.

4 Hold on, check the problem

The wording of the problem is slightly ambiguous. For the preceding solutions, it has been assumed that the questioner was seeking the probability that **at least one** of the brackets is perfect. That means the cases where any number of your brackets (including getting 2 perfect, 3 perfect, all the way up to all 50) are perfect are included. But what if the questioner intended to ask how large p should be if you want to get **exactly one** bracket correct in 50 years?

This solution does not initially appear difficult to set up. Refer back to (3). This formula expresses the case where we are imperfect 50 times. If we only have one perfect bracket, we will have 49 imperfect brackets, so we can use (3) in this solution by decreasing 50 to 49. The probability of a single perfect bracket is expressed in (1). Combining the adjusted (3) with (1), we get:

$$p^n(1 - p^n)^{49} \tag{7}$$

Which can easily be identified as (almost) an example of the binomial distribution's probability mass function. However (7) does not take into account the number of places in which the single perfect bracket may occur - to take this into account, multiply by the number of ways the single perfect bracket can appear (the binomial coefficient):

$$\binom{50}{1} p^n(1 - p^n)^{49} \tag{8}$$

Plugging in $n = 63$ and setting this equal to 0.5:

$$\binom{50}{1} p^{63}(1 - p^{63})^{49} = 0.5 \tag{9}$$

Attempting to solve for p algebraically here is impossible per the Abel-Ruffini theorem - in its simplified form, this is effectively a 50th order polynomial. With a spreadsheet it is easy to plug in a range of values between 0 and 1 for p^{63} and see that the probability of exactly one bracket out of 50 being perfect (probably) peaks out around 37.16% when $p^{63} \approx 0.0200$. So, if the text of the problem is taken literally to mean that we seek the value of p for which the probability of having **exactly one** perfect bracket is 50%, it does not appear that there is a valid solution.

5 A final reality check

The NCAA has changed the number of teams/games at least 10 times since the inception of the tournament in 1939. Based on the trend, it is likely that there will be more than 67 games at some point in the future. How do we account for this in our answer?

Given the excessive levels of complexity we are introducing here, it's time to break out some code. I've chosen Python to define a function **solve_p()** that will accept a list of future annual values for n, a target value for the probability of getting 1 bracket correct, and a value for how granularly to search for a solution between 0 and 1, and returns its approximation of p:

```
def prob_imperfect(p, num_games):
    return (1.0 - p ** num_games)

def find_closest(dict_solutions, target):
    adj_dict = {}
    for key in dict_solutions:
        adj_dict[key] = abs(dict_solutions[key] - target)
    return min(adj_dict, key=adj_dict.get)

def solve_p(n_list=([63] * 50), target=0.5, vals=100000):
    num_brackets = len(n_list)
    candidates = {}
    for i in [float(j) / vals for j in range(0,vals,1)]:
        temp_val = 1.0
        for b in n_list:
            temp_val = temp_val * prob_imperfect(i, b)
        candidates[i] = 1.0 - temp_val
    return find_closest(candidates,target)
```

The helper function **prob_imperfect()** calculates the probability of an imperfect bracket from p and the number of games. The other helper function **find_closest()** finds the candidate solution for p whose probability is closest to the target. The default arguments for **solve_p()** are the list [63,63,...,63] and the target 0.5, which represent the values for our main solution as provided in section 2.1, so we can test that the function works for the static number-of-games scenario by simply running the function with no arguments:

```
>>> import dmc_geek as dg
>>> dg.solve_p()
0.93424
```

Great! Now we can see how high p needs to be if, say, the NCAA decides to add 1 game per year for the next 134 years (assuming I am in a pool where every game is picked):

```
>>> nl = list(range(67,201))
>>> dg.solve_p(n_list=nl)
0.95074
```

Good to know! Now, what if I only need a 10% chance of at least one bracket being correct in those 134 years?

```
>>> dg.solve_p(n_list=nl,target=0.1)
0.92943
```

Fantastic. Let's say I want to waste more time on this and would like a more precise answer. How can I increase the granularity of my search space beyond the default of 100,000 test values for p?

```
>>> v = 1000000
>>> dg.solve_p(n_list=nl,target=0.1,vals=v)
0.929429
```

6 Conclusion

As you can see, that p needs to be pretty high for you to have any greater than - how shall I put this - a snowball's chance in hell of ever picking a perfect bracket in your lifetime. But hey, you don't need to be perfect to win your coworkers' cash - just be a little bit better than everyone else and eventually you might win your pool. Thanks for reading and good luck with your future brackets!

Solution .tex file and accompanying Python code will be published at github.com/m-deck after the close of the challenge period on May 5, 2017.