Dealing it multiple times does not change the EV

(Mircea Digulescu)
28.05.2015

ABSTRACT

In this article I study the following problem: “Given a bin of N balls, X of which are red what is the expected number of ‘wins’ if you are taking out S balls at a time for a total of K times? A win is a ‘drawing’ (a taking out of S balls) which contains at least 1 red ball.”. The interesting case of this problem is when the drawing of balls is made without replacement. The problem is a generalisation of the classic holdem poker problem of determining the Expected Value (EV) of dealing the turn and river more than once (from the same deck, without replacement) once two players are all in. The main and only result of this work is a proof that the expected number of wins in the case without replacement is the same as in the case with replacement, namely K times the probability of winning for K=1 (a single draw). This shows that the Expected Value of multiple draws, when each draw is worth the same fraction of the total pot is precisely the same as that of a single draw.

INTRODUCTION

The problem has been stated in the Abstract. The texas-holdem poker special case is equivallent to the parameters N = 45, X = number of ‘outs’ of first player, S = 2, K = number of times the turn and river are delt, assuming a heads-up all-in situation on the flop. The surprising fiding is that no matter X or K, the EV of dealing it more than once is precisely the same as that of dealing it just once.
In the general case this is even more suprising (to the author) since the probability of hitting a ‘win’ the j-th draw is heavily dependent on both the prior number of wins, but also their structure (how many red balls were in each win). Intuitively, the author was tempted to believe the EV decreases if X is sufficiently large and S is sufficiently large too, since situations could arise when some outs are wasted by showing up in groups of more than 1 in a winning draw. However this is not the case. The Expected Number of wins is precisely K times the probability of winning it once (as in the case with replacement), which is interesting.

While the knowledge that dealing it more than once does not change the EV has been around in card rooms for a while, the author has been unable to find a formal proof of this result, so he present his own proof here.

THE CLAIM

Let

g(n,k,x,s) = "expected number of times to win if you have x red balls with n balls in total, making k draws, each time drawing s balls at random".

Then

g(n,k,x,s) = k * g(n,1,x,s), for any valid choice of n,k,x,s.

Futhermore an efficient formula for g(n,1,x,s) is given.

THE APPROACH

There are several computer science approaches to computing g(n,k,x,s). In this section I present two non-optimal approaches to underlying the pitfalls of formalizing a probability problem in different ways:

- Dynamic programming in O(s*x^2) time: The basic approach is to compute g(n,k,x,s) recursively noticing that
  g(n,k,x,s) = Sum (alpha(n,s,h) * g(n-h, k-1, x-h, s)) for all h in [0..min(x, s]) representing the number of red balls "hit" on the first drawing and alpha() representing the probability of hitting precisely h red balls.
  This leads to the program presented in file BasicPokerDealCalculator.cs.

- Dynamic programming in O(x^2) time: A qualitative jump is made by changing the way in which we look at the problem. Instead of trying to figure out the EV by looking at the probability on having a win on a certain draw (which leads to difficulties), we can do the same by looking at the probability of *a certain red ball* contributing or not to a win. Care is required: an out can only contribute to winning a "fresh" drawing (one that was not already won by another red ball). The formula of interest is
  `EV[whichOut, hitsSoFar] = alpha * f_calc(whichOut + 1, hitsSoFar + 1) + (1 - alpha) * f_calc(whichOut + 1, hitsSoFar)` where alpha is the probability of the red ball number whichOut causing a win if there were already hitsSoFar wins from the prior drawings.
  This leads to the program presented in ImprovedPokerDealCalculator.cs.

- Precise formula in O(1) time: Taking the approach from the prior solution further, we can compute the EV directly, in O(1) (excluding one-time times to compute a log-table). This leads to the program presented in BestPokerDealCalculator.cs.

All there approaches are correct and the programs produce the same result, save some very minute differences due to numerical instability. In this paper I present the

final result, but it could be interesting to note how I came to arrive to it (both in noticing it is correct and in discovering the proof), based on prior approaches.

THE BASIC CASE

I start by giving a formula for the basic case, that is dealing it precisely once. What is the expected number of wins here? It is exactly the probability of a red ball (i.e. at least one) showing up in the s balls chosen at random from the n available in the single draw. There are several ways to look at this, but the one which is useful is actually considering the probability of loosing. What is it?
Performing random draws is equivallent to having some initial suffle of the n-ball bin which produces some order in which the balls are drawn (which is precisely what happens in poker game special case). However we are not interested in precisely what this order is precisely, but more importantly we care about where are the x red balls located. If they are all beyond the first s positions, the first (and only in this case) draw is not a win. There are precisely Comb(n-s, x) ways in which the n-ball bin can be shuffled so that all x red balls appear on some of the n-s non winning positions. There are, naturally, Comb(n, x) total ways in which the n-ball bin can be shuffled without restrictions.
Here **Comb(n,x) = "number of possible combinations of choosing x elements from n"**.

Therefore, the expected value of a single draw (dealing it once) is:

$$g(n,1,x,s)=1-\frac{Comb(n-s,x)}{Comb(n,x)}=1-\frac{\frac{(n-s)!}{x!\cdot(n-s-x)!}}{\frac{n!}{(n-x)!\cdot x!}}=1-\frac{(n-s)!\cdot(n-x)!}{(n-s-x)!\cdot n!}$$ for any s<=n, x<n

This is precisely 1 time g(n,1,x,s), thus the claimed result holds for k = 1.

THE GENERAL CASE

Since the result holds for k = 1, we can use induction:

By induction:

$$g(n,k,x,s)=k\cdot g(n-1,1,x,s)=k\cdot\left(1-\frac{(n-s)!\cdot(n-x)!}{(n-s-x)!\cdot n!}\right)$$ for any n<N x<n, k*s<=n

Also

$$g(N,1,x,s)=1-\frac{(N-s)!\cdot(N-x)!}{(N-s-x)!\cdot N!}$$ for any s<=n, x<n

Now, the trick is again to consider the contribution of the first red-ball (weather it causes a win or not). The first such "out" has k*s positions out of the n total on which it could land to cause a win.

Let $a=\frac{k \cdot s}{N}$ be the probability the first out of x contributes to the EV pozitively

$p=g(N-1,1,x-1,s)=1-\frac{(N-s-1)!\cdot(N-x)!}{(N-s-x)!\cdot(N-1)!}$ the expected number of wins of dealing it once with x-1 outs, n-1 remaining deck

Notice that:

Since

$$g(N,k,x,s)=a\cdot(1+g(N-1,k-1,x-1,s))+(1-a)\cdot g(N-1,k,x-1,s)$$

(which results directly by considering the recursion given considering weather the first out hits [probability a] – increasing the number of wins by 1 or misses [probability 1-a]; notice that if the first out hits, then the entire s-ball draw which it hit is already a win as such the remaning positions in it are no longer winning positions for the remaining outs).

Then

$$g(N,k,x,s)=a\cdot(1+(k-1)\cdot p)+(1-a)\cdot k\cdot p=a+a\cdot k\cdot p-a\cdot p+k\cdot p-a\cdot k\cdot p=a\cdot(1-p)+k\cdot p$$

$$g(N,k,x,s)=a\cdot(1-p)+k+k\cdot(p-1)=k+(1-p)\cdot(a-k)$$

$$g(N,k,x,s)=k+\frac{(N-s-1)!\cdot(N-x)!}{(N-s-x)!\cdot(N-1)!}\cdot\left(\frac{k\cdot s}{N}-k\right)=k+\frac{(N-s-1)!\cdot(N-x)!\cdot k\cdot(s-N)}{(N-s-x)!\cdot(N-1)!\cdot N}$$

$$g(N,k,x,s)=k-k\cdot\frac{(N-s-1)!\cdot(N-x)!\cdot(N-s)}{(N-s-x)!\cdot(N-1)!\cdot N}=k-k\cdot\frac{(N-s)!\cdot(N-x)!}{(N-s-x)!\cdot N!}=k\cdot g(N,1,x,s)$$

qed

The first two lines result by simple arithmetic. The forth line results from the third by explicitating p and a. The remainder is again simple arithmetic. An innovative step occurred on line two in expressing k*p as k + k * (p – 1). This idea came since, already suspecting the end result, a free 1*k was desired.

CONCLUSION

I presented a proof of the claimed result. Considering that each of the k draws is "worth" 1/k of the total "pot", then we have:

$$EV = \frac{g(N,k,x,s)}{k} = \frac{k \cdot g(N,1,x,s)}{k} = g(N,1,x,s)$$

qed

In arriving at the proof, two (lightly) innovative steps were performed:

- Considering the problem in terms of the j-th out hitting instead of the j-th draw being winning or not;
- Rewritting the expression on line two in the prior section.

In conclusion this article is an example of how an incremental approach, starting with brute-force experimenting and being enhanced by some innovative steps can lead to an efficient and elegant result.

The source code for the three approaches presented is available for download here: [1 – Source Files].

References

[1] — *Mircea Digulescu (2015 April) – Poker Deal Calculator Research Project Source Files* - http://www