Research on Math Expectation of the Number of Suits in N Cards

Junchen Zhao, Hao Zhou

1. Abstract

In this paper, we aim to design an algorithm to calculate the expectation associated with card at a stochastic process. Techniques used include combinatorics, probability, and statistics. Code in C++ using a recursive function is also provided to substantiate our theory.

2. Introduction

We are inspired by a board game called “Killers of Three Kingdoms”. In this board game, a hero’s spell is defined by taking cards with different suits among the first five cards on the card pile. (For example, when two spades, two diamonds, and one heart are taken, one spade, one diamond, and one heart are obtained by the player.) We want to calculate the math expectation of the cards obtained according to this spell, further getting a formula to express the number of suits in n cards taken from an infinite card pile. Our research is aimed at getting a deeper understanding of combinatorics and solving real-life problems using knowledge learned in class. Three independent tools are employed to analyze this process to calculate the math expectation: combinatorics, exhaustive algorithm, and recursion relation analysis; they all give out the same result. In our essay, the new concepts of “similarity” and “dissimilarity” are defined.

3. Method employed

A. In this essay, the method such as (1+4) and (1+1+3) is used to analyze suit combinations. For example, (1+4) means a total of 5 cards, where there are two groups, one of which has one card in a suit, the other of which has four cards in another suit; (1+1+3) means a total of 5 cards, where there are three groups, which have 1, 1, 3 cards in different suits respectively.

B. When allocating suits to cards, the number of combinations (C) is used to calculate “similar” cards, and the number of permutations (P) is used to calculate “dissimilar” cards. For example, there are three conditions when 6 cards with 2 suits are drawn from the card pile, two of which are (2+4) and (3+3). For (2+4), “2” and “4” are “dissimilar”, so the formula to allocate suits is P_4^2 , which means to assign suits (picking 2 from 4 suits regardless of their permutation) to each group of cards. For (3+3), “3” and “3” are “similar”, so the formula to allocate suits is C_4^2 , which means to assign suits (picking 2 from 4 suits only considering combination) to each group of cards. That’s because when considering the arrangements of the cards, the reverse of “3” and “3” has already been included. If the reverse is considered again, one condition will be recorded once again, so the number of combinations (C) is used to calculate the suit-distribution conditions.

C. According to the Multiplication Rule, the number of cases satisfying the respective condition is obtained by multiplying the arrangement of cards and combination of suits. ***[1]***

4. Combinatorics analysis

4-1. Analysis of taking 5 cards

When there are 5 cards, the number of total conditions is $4^5 = 1024$.

Get 1 card

Obviously, if only 1 card is obtained, each card must be of the same suit. So there are 4 conditions. The possibility is $P_1 = \frac{4}{1024}$.

Get 2 cards

If 2 cards are obtained, the suit combinations of 5 cards are (1+4) or (2+3).

i. The total number of (1+4) is $C_5^1 \times C_4^4 \times P_4^2 = 60$. C_5^1 means the number of arrangements of "1" in 5 cards; C_4^4 means the number of arrangements of the remaining 4 cards; P_4^2 means the number of suit combinations regardless of their permutation.
ii. For the same reason, the total number of (2+3) is $C_5^2 \times C_3^3 \times P_4^2 = 120$.

In summary, the possibility of getting 2 cards in 5 cards is $P_2 = \frac{180}{1024}$.

Get 3 cards

If 3 cards are obtained, the suit combinations of 5 cards are (1+2+2) or (1+1+3).

i. The total number of (1+2+2) is $C_5^1 \times C_4^2 \times C_2^2 \times C_4^2 \times 2 = 360$. C_5^1 means the number of arrangements of "1" in 5 cards. C_4^2 means the number of arrangements of a "2" in the remaining 4 cards. C_2^2 means the number of arrangements of the last "2". The final C_4^2 means the number of suit combinations of two "similar" "2". The last factor 2 means the two possibilities of the suit of "1".
ii. For the same reason, the total number of (1+1+3) is $C_5^1 \times C_4^1 \times C_3^3 \times C_4^2 \times 2 = 240$.

In summary, the possibility of getting 3 cards in 5 cards is $P_3 = \frac{600}{1024}$.

Get 4 cards

If 4 cards are obtained, the suit combination of 5 cards is (1+1+1+2).

Same as the condition of getting 3 cards, the total number of (1+1+1+2) is $C_5^1 \times C_4^1 \times C_3^1 \times C_2^2 \times C_4^3 \times 1 = 240$. So the possibility is $P_4 = \frac{240}{1024}$.

Combining all the above analysis of taking 5 cards, we could get the math expectation:
$E(5) = 1 \times P_1 + 2 \times P_2 + 3 \times P_3 + 4 \times P_4 = 3.05078$

4-2. Analysis of taking 6 cards

When there are 6 cards, the number of total conditions is $4^6 = 4096$.

Get 1 card

Same as the case of 5 cards, the possibility here is $P_1 = \frac{4}{4096}$.

Get 2 cards

If 2 cards are obtained, the suit combinations of 6 cards are (1+5), (2+4), or (3+3).

i. When it is (1+5), the formula is $C_6^1 \times C_5^5 \times P_4^2 = 72$.
ii. When it is (2+4), the formula is $C_6^2 \times C_4^4 \times P_4^2 = 180$.
iii. When it is (3+3), the formula is $C_6^3 \times C_3^3 \times C_4^2 = 120$.

In summary, the possibility of getting 2 cards in 6 cards is $P_2 = \frac{372}{4096}$.

Get 3 cards

If 3 cards are obtained, the suit combinations of 6 cards are (1+1+4), (1+2+3), and (2+2+2).

i. When it is (1+1+4), the formula is $C_6^1 \times C_5^1 \times C_4^4 \times C_4^2 \times 2 = 360$.
ii. When it is (1+2+3), the formula is $C_6^1 \times C_5^2 \times C_3^3 \times P_4^3 = 1440$.
iii. When it is (2+2+2), the formula is $C_6^2 \times C_4^2 \times C_2^2 \times C_4^3 = 360$.

In summary, the possibility of getting 3 cards in 6 cards is $P_3 = \frac{2160}{4096}$.

Get 4 cards

If 4 cards are obtained, the suit combinations of 6 cards are (1+1+1+3) and (1+1+2+2).

i. When it is (1+1+1+3), the formula is $C_6^1 \times C_5^1 \times C_4^1 \times C_3^3 \times C_4^3 \times 1 = 480$.
ii. When it is (1+1+2+2), the formula is $C_6^1 \times C_5^1 \times C_4^2 \times C_2^2 \times C_4^2 \times 1 = 1080$.

In summary, the possibility of getting 4 cards in 6 cards is $P_4 = \frac{1560}{4096}$.

Combining all the analysis above, we could get the math expectation:
$E(6) = 1 \times P_1 + 2 \times P_2 + 3 \times P_3 + 4 \times P_4 = 3.28809$

4-3. Analysis of taking 7 cards:

When 7 cards are taken, the number of all possible situations is $4^7 = 16384$.

Same as the method above, we can easily conclude that P_1, the probability of obtaining 1 card, equals $\frac{4}{16384}$; P_2 equals $\frac{756}{16384}$; P_3 equals $\frac{7224}{16384}$; and P_4 equals $\frac{8400}{16384}$.

According to these results, E(7), the math expectation of the number of suits in 7 cards, equals 3.46606.

5. Verification by exhaustive algorithm in computer science *[2]*

In order to verify the results, a piece of C++ code using exhaustive algorithm is designed to enumerate all possible situations.

($P_i(n)$ below represents the probability of obtaining i cards when n cards are taken from the card pile.)

n	$P_1(\mathrm{n})$	$P_2(\mathrm{n})$	$P_3(\mathrm{n})$	$P_4(\mathrm{n})$	Math expectation
5	0.003906250	0.175781250	0.585937500	0.234375000	3.050781250
6	0.000976562	0.090820312	0.527343750	0.380859375	3.288085938
7	0.000244141	0.046142578	0.440917969	0.512695312	3.466064453
8	0.000061035	0.023254395	0.353759766	0.622924805	3.599548340
9	0.000015259	0.011672974	0.276947021	0.711364746	3.699661255
10	0.000003815	0.005847931	0.213546753	0.780601501	3.774745941

6. Formula obtained by recursion relation analysis

E(n), the math expectation of cards obtained when n cards are taken from the card pile, is calculated by recursion relation analysis.

We can conclude: $E(n) = 1 \times P_1(n) + 2 \times P_2(n) + 3 \times P_3(n) + 4 \times P_4(n)$

$P_i(n) \times 4^n$ is firstly calculated to get the number of all the situations when i cards are obtained by taking n cards from the card pile. Subsequently, we determine whether the suit of the $(n+1)^{th}$ card occurred or not, therefore confirming the overall number of suits in these (n+1) cards. Finally, these situations are re-classified according to current number of suits, and then divided by current sum of situations, 4^{n+1}, to get $P_i(n+1)$.

The deduction of the recursion relation of $P_i(n)$:

When i=1, the $(n+1)^{th}$ card taken must be the same as the suit of previous n cards, so its suit is certain:

$$P_1(n+1) = \frac{P_1(n) \times 4^n \times 1}{4^{n+1}}$$

When i=2, there are two possibilities: there are 2 suits in previous n cards, the suit of the $(n+1)^{th}$ card taken is the same as one of them (2 cases); there is 1 suit in previous n cards, the suit of the $(n+1)^{th}$ card taken is different from either one of them (3 cases):

$$P_2(n+1) = \frac{P_2(n) \times 4^n \times 2 + P_1(n) \times 4^n \times 3}{4^{n+1}}$$

When i=3, there are two possibilities: there are 3 suits in previous n cards, the suit of the $(n+1)^{th}$ card taken is the same as one of them (3 cases); there are 2 suits in previous n cards, the suit of the $(n+1)^{th}$ card taken is different from either one of them (2 cases):

$$P_3(n+1) = \frac{P_3(n) \times 4^n \times 3 + P_2(n) \times 4^n \times 2}{4^{n+1}}$$

When i=4, there are two possibilities: there are 4 suits in previous n cards, the suit of the $(n+1)^{th}$ card taken is the same as one of them (4 case); there are 3 suits in previous n cards, the suit of the $(n+1)^{th}$ card taken is different from either one of them (1 case):

$$P_4(n+1) = \frac{P_4(n) \times 4^n \times 4 + P_3(n) \times 4^n \times 1}{4^{n+1}}$$

In order to obtain a general formula, n=1 is used as the initial condition. Obviously, only one suit will be obtained if there is only 1 card taken from the card pile, so:

$$P_1(1) = 1,\ P_2(1) = 0,\ P_3(1) = 0,\ P_4(1) = 0,\ E(1) = 1.$$

Finally, the 5 equations above, with initial condition, are solved as a set of simultaneous equations:

$$E(n) = 4 \times [1 - (\frac{3}{4})^n],\ n > 0$$

7. Conclusion

Three independent methods give out the same result respectively, therefore proving their validity. The problem is also furthered to calculate the math expectation of the number of suits in n cards. By analyzing the impact of taking the $(n+1)^{th}$ card, the recursive formula of $P_i(n)$ is deduced, and the general formula of E(n) is solved subsequently.

8. Acknowledgements

Firstly we would like to appreciate Ms. Y. Chen; her ideas about researching mathematics inspired us to pick up this problem associated with cards, further extending it to “n dimensions” to do deeper research. We are also very grateful to Mr. J. Lu; he not only inspired us to use recursion relation analysis, but also helped us revise this paper very carefully.

[1]:

We assume each suit is represented by one specific shape:

(2+4) is represented by OOXXXX. The permutation of O and X is determined first, which is given by $C_6^2 \times C_4^4$ (firstly select 2 positions for O in all 6 positions; then select 4 positions for X in the remaining 4 positions); then different suits are assigned to O and X: …… O<—spade, X<—diamond……O<—diamond, X<—spade……; later, the total number of suit-assignment is calculated, which is given by P_4^2.

(3+3) is represented by OOOXXX. If we continue our analysis according to the method above, the permutation of O and X counted will be $C_6^3 \times C_3^3$, and the total number of suit-assignment will be P_4^2. All the possibilities considered is $C_6^3 \times C_3^3 \times P_4^2$, as is shown in the table below:

	o=spade x=diamond	o=diamond x=spade
ooxoxx	I	II
xxoxoo	III	IV

However, it is obvious that I&IV (II&III) are the same situation but counted twice, which is in contrast with reality.
Therefore, when two groups are “similar”, P should not be used to count the suit-assignment after their permutation is determined by C. Instead, C, which does not take permutation into account, should be employed. So the total possibilities are given by $C_6^3 \times C_3^3 \times C_4^2$ to avoid over-counting, as is shown in the table below:

	o=spade x=diamond	o=heart x=diamond
ooxoxx	I	II
xxoxoo	III	IV

[2]:

All the results given by our recursive function in C++ are shown in the table below:

n	$P_1(n)$	$P_2(n)$	$P_3(n)$	$P_4(n)$	Math expectation
1	1.000000000	0.000000000	0.000000000	0.000000000	1.000000000
2	0.250000000	0.750000000	0.000000000	0.000000000	1.750000000
3	0.062500000	0.562500000	0.375000000	0.000000000	2.312500000
4	0.015625000	0.328125000	0.562500000	0.093750000	2.734375000

5	0.003906250	0.175781250	0.585937500	0.234375000	3.050781250
6	0.000976562	0.090820312	0.527343750	0.380859375	3.288085938
7	0.000244141	0.046142578	0.440917969	0.512695312	3.466064453
8	0.000061035	0.023254395	0.353759766	0.622924805	3.599548340
9	0.000015259	0.011672974	0.276947021	0.711364746	3.699661255
10	0.000003815	0.005847931	0.213546753	0.780601501	3.774745941
11	0.000000954	0.002926826	0.163084030	0.833988190	3.831059456
12	0.000000238	0.001464128	0.123776436	0.874759197	3.873294592
13	0.000000060	0.000732243	0.093564391	0.905703306	3.904970944
14	0.000000015	0.000366166	0.070539415	0.929094404	3.928728208
15	0.000000004	0.000183094	0.053087644	0.946729258	3.946546156
16	0.000000001	0.000091550	0.039907280	0.960001169	3.959909617

And here is our code:

```
#include <cstdio>
#include <iostream>
using namespace std;

const int maxn=100;
long long a[maxn],u[maxn],res[maxn];
//a[0--n-1] represent n cards, each of which has value in {1,2,3,4},
representing 4 suits
//u[i in range[1,4]] represent whether or not suit i has occurred, each of
which has value 0 or 1
//res[i in range[1,4]] represents the time i suits has occurred

int N; //the number of cards taken

void check()
{
    //count the number of suits occurred and add to res[]
    int ans=0;
    for(int i=1;i<=4;i++)
        if(u[i])
            ans++;
    res[ans]++;
    return;
}
void work(int n)
{
```

```
    //generate all situations by recursion
    if(n==N)
    {
        //check suits when n cards generated
        check();
        return;
    }

   //generate cards and record suits at the same time to increase efficiency
    for(int i=1;i<=4;i++)
    {
        a[n]=i;
        u[i]++;
        work(n+1);
        u[i]--;
    }
    return;
}

int main()
{
    //run with n from 1 to 10
    for(N=1;N<=16;N++)
    {
        cout<<endl<<endl<<"When "<<N<<" cards are taken from the card
           pile:"<<endl;

        //initialize arrays
        for(int i=0;i<maxn;i++)
            a[i]=u[i]=res[i]=0;

        //generate all possibilities and count
        work(0);

        double sum=0;   //record all possibilities
        for(int i=1;i<=4;i++)
        {
            sum+=res[i];
        }

        for(int i=1;i<=4;i++)
        {
            cout<<"The chance occurring "<<i<<" suits: ";
            printf("%.9lf\n",(double)res[i]/sum);
        }

        double n=0;
        for(int i=1;i<=4;i++)
            n+=1.0*i*res[i]/sum;
        printf("The math expectation is: %.9lf",n);
    }
    cout<<endl<<endl;
    return 0;
}
```