Genshin 5 Star Estimation(G5SE) with Dynamic Programming

ChACiooh(Github)

Abstraction

2020년 9월 런칭한 RPG 게임 '원신'의 gacha(뽑기) system은 한 stage마다 획득할 수 있는 재화가 3성, 4성, 그리고 5성이 있으며, 이 중 5성을 한 번의 try에 획득할 확률은 0.6%이다. 그리고 그 중에서 우리가 원하는 5성 캐릭터를 획득할 확률은 조금 특별하다. 직전 stage에서 유저가 픽업 캐릭터를 획득하지 못했을 경우 이번 stage에 등장하는 5성 캐릭터는 반드시 픽업 캐릭터이며, 그렇지 않을 경우 픽업 캐릭터의 등장 확률은 5성 획득 확률의 50%이다. 또한 원신의 천장 시스템(반드시 5성 뽑기가 보장되는 지점을 제공하는 시스템)은 단순히 90번째에 반드시 5성을 획득하는 것뿐만 아니라, 특정 번째부터 5성 캐릭터를 획득할 확률이 비약적으로 상승하는 통계도 나타났다. 이러한 조건을 기반으로 우리가 가진 한정된 재화를 통해 원하는 캐릭터를 얼마나 뽑을 수 있을지 예측하고자 하였으며 이러한 estimation을 위해 Dynamic Programming(이하 DP) 기법을 사용하여 문제를 해결하였다. 기존에 공개된 몬테카를로 기법의 시뮬레이터는 시간복잡도가 O(N^M; N~10e7)에 이르렀으나 본 연구를 통해 DP 알고리즘을 사용하여 O(S*N*M*K; S~89, N~7)까지 줄이는데 성공하였다.

Introduction

gacha system에서는 각 stage마다 확정적으로 5성 캐릭터를 획득할 수 있는 상한이 존재하며 원신의 경우 90번째에서는 반드시 5성 캐릭터를 획득할 수 있다. 이러한 시스템을 '천장'이라고 하며, 이전 stage에서 픽업 캐릭터를 뽑았거나 한 번도 gacha system을 이용하지 않았을 경우에는 원하지 않는 5성 캐릭터도 등장하므로 이를 '반천장 상태'라고 한다. 반천장 상태에서 5성 캐릭터를 뽑았을 경우 픽업 캐릭터가 등장할 확률은 5성 캐릭터가 등장할 확률의 50%이므로 0.3%임을 알 수 있다.

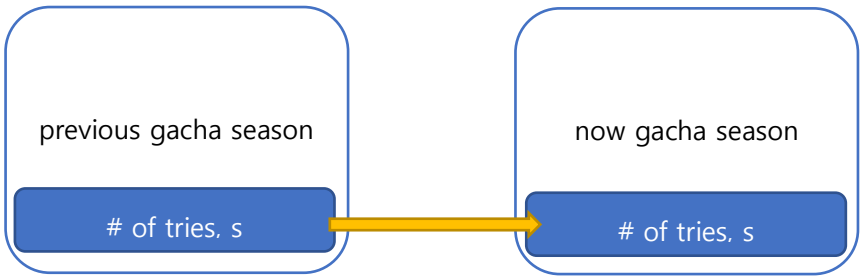
그러나 단순히 90번째까지 계속 시도하지 않고도 충분히 5성 캐릭터를 뽑을 수 있다는 사실이 통계적으로 증명되었는데, 무수히 많은 유저들의 뽑기 데이터를 통해 추출한 결과 한 stage 당 74번째부터 확률이 약 32.37%로 상승함을 알 수 있었다. 즉, 유저가 각 stage에서 try한 횟수에 따라 구간을 나누어서 확률을 계산할 필요가 있었다.

$$P_i(k, p_i, q_i) = \begin{cases} p_i(1-p_i)^{k-1}, & 0 < k < 74 \\ (1-p_i)^{73}(1-q_i)^{k-73}q_i, & 74 \le k < 90 \\ 1, & k = 90 \end{cases}$$

위 식에서 P는 k번째에 원하는 한정 5성 캐릭터를 뽑을 확률을 의미한다. p는 유저가 원하는 한정 5성 캐릭터를 뽑을 공개된 확률, q는 유저들이 통계적으로 찾은 74번째부터의 보정확률이다. 반천장 상태와 천장 상태를 구분하기 위해 i가 홀수라면 p_i의 값은 0.003, q_i의 값은 약 0.161781이며, 짝수라면 각각 0.006, 0.323561의 값을 가지도록 정의했다. 그리고 이 두 값은 근사치이므로 보다 정확한 계산을 위해 $\sum_{k=1}^{90} P_i(k, p_i, q_i) = 1$을 만족하도록 보정해주었다. 이러한 기본 확률 계산 모델을 근거로 DP를 설계하였다.

Stack system

앞서 Introduction에는 소개하지 않은 내용으로 stack 속성이 존재한다. 이는 직전 gacha state에서 진행한 try 수이며 만약 그 수가 남아 있는 채로 gacha season이 바뀌게 되면 초기화되지 않고 누적된 stack 값은 그대로 이월된다. 따라서 현재 어느 정도 시도를 한 후 다음 시즌으로 넘어가더라도 유저는 s 값에 이어서 gacha를 진행할 수 있다.



즉 이제부터 계산하게 될 확률 모델에서는 현재 state가 직전 gacha season에서의 결과로부터 반천장 및 천장 상태를 정의하고 남아있는 stack의 수 s만큼 누적된 값에서 출발했을 때의 한정 캐릭터를 뽑을 확률을 구하게 되는 것이다. 현재 j개의 try를 했다 하더라도 실제로는 이미 누적된 s만큼의 실패 이후의 j번의 try를 한 것과 같으므로, 이를 식으로 나타내면 다음과 같다.

$$P_{i,s} = P_i(k, p_i, q_i|s) = \frac{P_i(k+s, p_i, q_i)}{C_{i,s}(p_i, q_i)}$$

이때,

$$C_{i,s}(p_i, q_i) = \begin{cases} 1, & where\ s \equiv 0\ mod\ 90 \\ \dfrac{P_i(k, p_i, q_i)(1-p_i)}{p_i}, & where\ 0 < s < 74 \\ \dfrac{P_i(k, p_i, q_i)(1-q_i)}{q_i}, & where\ 74 \le s < 90 \end{cases}$$

으로 계산된다.

DP design

이렇게 stack s의 값과 현재 gacha state에 대한 정보 i가 주어졌을 때 유저는 한정 캐릭터를 n번뽑고 싶을 때 자신이 얼마나 try하면 좋을지 궁금하다. 우선 자신의 지갑 사정을 고려하여 구매 가능한 창세의 결정을 토대로 뒤얽힌 인연(gacha를 try할 수 있는 아이템)의 수를 계산할 것이다. 한화 119,000 원에 8080개의 창세의 결정을 구입할 수 있으므로, 50.5 tries / 119,000₩임을 알 수 있다. 약 100만 원의 예산이 있다면 그 안에 try할 수 있는 최대 수는 404회이다(119,000₩ 미만의 상품도 존재하나 계산하기 쉽게 해당 가격의 상품만 있다고 가정한다).

동일한 캐릭터를 여러 번 뽑아서 잠재능력을 개방할 수 있으며, 이는 최대 6회까지 가능하다. 따라서 아무리 많이 뽑았다 하더라도 7회까지 유효하다. 만약 100만 원의 예산이 있을 때 한정 픽업 캐릭터를 7회 뽑고자 한다면 얼마의 확률로 이것이 이루어질까? 다시 말해 404회의 try 안에 한정 픽업 캐릭터가 7번 이상 나올 확률은 얼마나 될까?

이러한 문제를 해결하기 위해 가장 먼저 유저로부터 현재 gacha stage가 반천장 상태인지 천장 상태인지를 알아야 한다. 반천장 상태의 경우 i는 홀수부터 시작하며, 그렇지 않을 경우 i는 짝수부터 시작한다. 그리고 i=0일 때의 모든 확률은 0으로 본다. 그리고 DP의 점화식을 다음과 같이 정의한다.

$$DP[n][j] = \begin{cases} \max\limits_{k}\left(DP[n][j], DP[n-1][j-k] * \ prob\ on\ pic(k, NOWANT)\right), & where\ n\ is\ even \\ \max\limits_{k}\begin{pmatrix} DP[n][j], \\ DP[n-2][j-k] * prob\ on\ pic(k, WANT) \\ +DP[n-1][j-k] * prob\ on\ nopic(k) \end{pmatrix}, & where\ \ n\ is\ odd \end{cases}$$

n은 일단 stage의 순서라고 생각하자. j는 내가 try하는 gacha의 수이다. 이때 prob on pic은 직전의 gacha state에서 한정 픽업 캐릭터를 뽑은 상태에서의 확률 모델이고, prob on nopic은 직전의 gacha state에서 한정 픽업 캐릭터가 아닌 5성 캐릭터가 나온 상태(이하 픽뚫 상태)에서의 확률 모델이다. prob on pic에서는 원하는 한정 캐릭터와 원치 않는 5성 캐릭터가 모두 나올 수 있으므로, DP[n]에서 n이 짝수일 때에는 픽뚫 확률을, n이 홀수일 때에는 원하는 한정 캐릭터가 나올 확률을 구하고 maximum value를 저장한다. 여기서 j-k의 의미는 현재 gacha state에서 k개를 이용하여 try할 것이기 때문에, 그 전까지는 j-k개를 이용해 직전 단계의 DP값이 정의되어 있어야 하고 또 이를 이용하는 것이다. 따라서 n = 2m – 1의 식을 고려해본다면 m개의 한정 캐릭터를 j번의 gacha 안에 뽑는 확률이 도출된다.

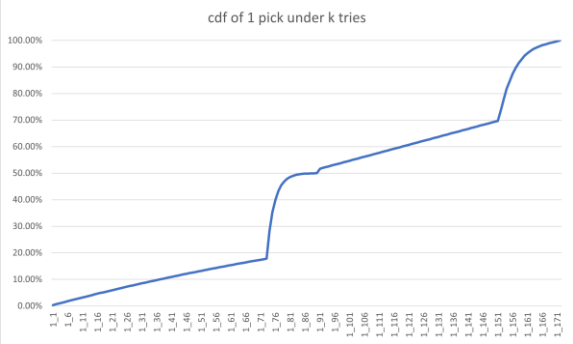
Apply on code

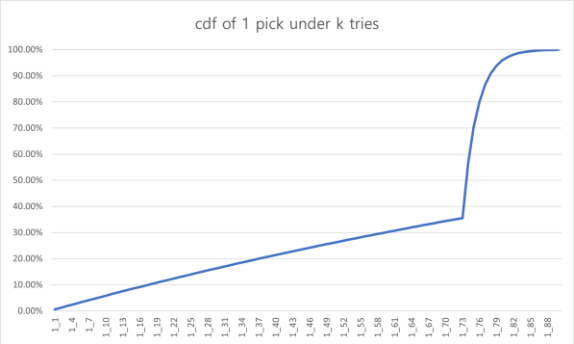
DP 모델에 적용시키기에 앞서, stack s의 값은 항상 n = 1일 때에만 유효한데 그 이후는 stack이 아닌 try 수에 포함되기 때문이다. 그리하여 n = 1때부터 유저가 원하는 중복 획득 수까지의 DP를 구하면 유저가 원하는 결과를 보여줌과 동시에 한정 캐릭터를 얻고자 하는 수만큼 획득하기 위한 확률값과 try 수를 확률이 100%가 될 때까지 구할 수 있다.

지금까지 세운 모델을 코드로 작성하면 다음과 같다:

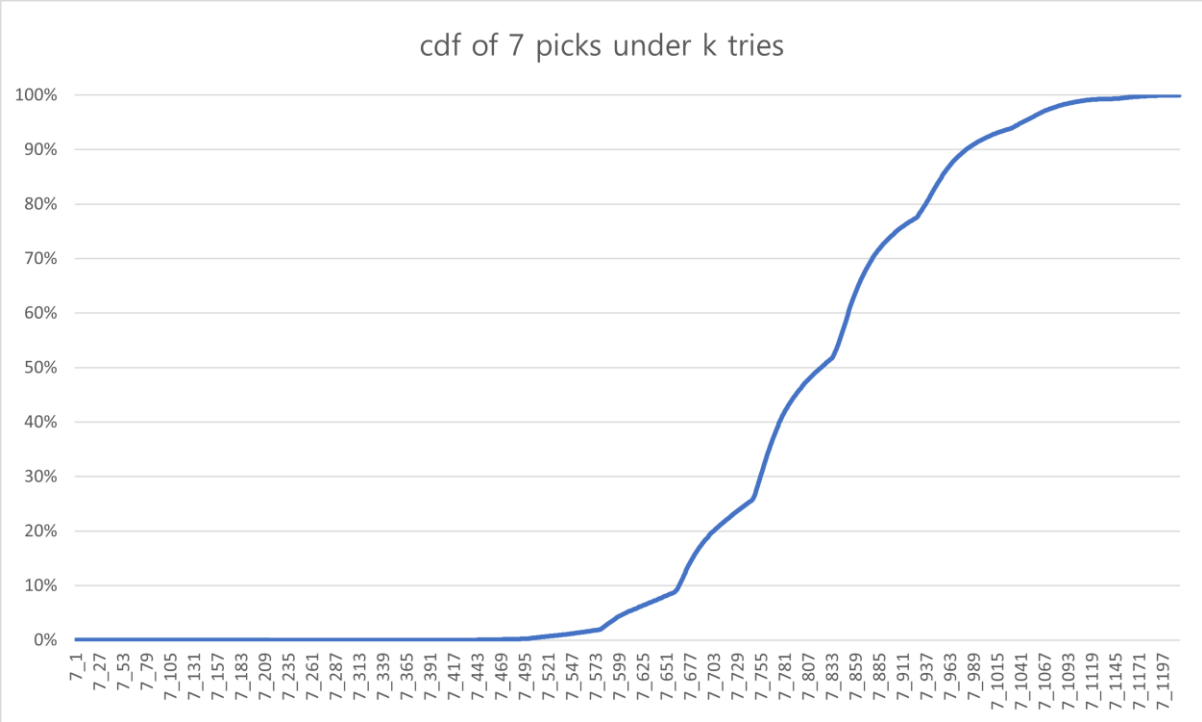
```
/* 첫 스테이지 */
for(int j = 1; j <= MAX_GACHA;++j) // 전체 가챠 스테이지에서 사용하는 가챠권 j개
{
    dp[1][j].SetProb(first_pic(j, WANT));    // 기본 첫 스테이지
    dp[1][j].SetStack(j);
    if(j > GACHA_SIZE && getPic)
    {
        for(int k = 1; k < j && k <= GACHA_SIZE; ++k)   // 이번 가챠 스테이지에서 사용하는 가챠권 k개
        {
            double dp1j = dp[1][j].GetProb();
            double np = picWhatIdonWantInN(stack_, j-k, getPic) * picWhatIwantInN(0, k, false);
            double dn = dp[1][j-k].GetProb() + np;

            if(dp1j < dn) {
                dp[1][j].SetProb(dn);
                dp[1][j].SetStack(k);
            }
        }
        if(dp[1][j].GetProb() >= 1.0) dp[1][j].SetProb(1.0);
    }
}

/* 원하는 돌파 수까지 구하는 과정 */
for(int n = 2; n <= DOL_SIZE; ++n)
{
    for(int j = 1; j <= MAX_GACHA; ++j) // 전체 가챠 스테이지에서 사용하는 가챠권 j개
    {
        for(int k = 1; k < j && k <= GACHA_SIZE; ++k)   // 이번 가챠 스테이지에서 사용하는 가챠권 k개
        {
            double dpnj, pp;
            if(n % 2 == 0)  // 픽뚫 확률 (짝수는 못 뽑는 거)
            {
                dpnj = dp[n][j].GetProb();
                pp = dp[n-1][j - k].GetProb() * prob_on_pic(k, NOWANT);
                if(dpnj < pp)
                {
                    dp[n][j].SetProb(pp);
                    dp[n][j].SetStack(k);
                }
            }
            else    // 픽업-픽업 또는 픽뚫-픽업 (홀수는 뽑는 거)
            {
                pp = dp[n-2][j-k].GetProb() * prob_on_pic(k, WANT);
                double np = dp[n-1][j-k].GetProb() * prob_on_nopic(k);
                dpnj = dp[n][j].GetProb();
                if(dpnj < pp + np)
                {
                    dp[n][j].SetProb(pp + np);
                    dp[n][j].SetStack(k);
                }
            }
            if(dp[n][j].GetProb() >= 1.0) dp[n][j].SetProb(1.0);
            dp[n][j].SetStack(k);
        }   // for k end
    }   // for j end
}   // for n end
```

Result





두 그래프 모두 s=0, n=1일 때 k회 시도할 때에 대한 cdf 확률 모델의 결과이다. 이때 좌측은 반천장 상태이며 우측은 천장 상태이다.



위 그래프는 반천장 상태에서 s=36, n=7일 때 k회 시도할 때에 대한 cdf 확률 모델의 결과이다.

본 모델의 설계 당시 cdf의 값이 50%일 때 가장 가능도가 높을 것으로 예상하였으나, 실제 사례들을 인터뷰한 결과 n, s, 천장 및 반천장 상태에 대하여 cdf의 값이 20% 즈음에서 목표가 달성된 사례들이 다수였다. 이를 토대로 본인의 상태인 반천장 상태, s=36, n=7의 값을 갖고 cdf의 값이 20%인 곳을 찾아보았더니 k=707이 나타났으며 직접 실험해 본 결과 놀랍게도 정확히 707번째에 설정된 목표가 이루어졌다.

About Modifying Probability Model(MPM)

그렇다면 왜 cdf의 값이 20% 즈음에서 이 문제가 해결된 것일까? 그것은 본 연구는 한 번의 시도에 한 번 뽑기를 한다고 가정했기 때문이다. 10회 연속 뽑기를 할 경우 반드시 한 번은 4성 이상의 아이템 및 캐릭터가 등장하므로, 10회마다 확률 보너스가 적용된다고 할 수 있다. 뿐만 아니라 4성 이상의 아이템 획득 시 다른 재화 포인트가 누적되며 이를 통해 다시 뽑기권을 구입할 수 있다. 10회 연속 뽑기 시스템에 의한 확률 보정 보너스와 4성 이상의 아이템 획득 시에 대한 페이 백(Pay-back) 시스템이 보다 낮은 기댓값에서도 목표를 이룰 수 있었던 것이었다.

따라서 이 시스템을 적용하기 위해서는 모든 시도를 10회 연속 뽑기로 했을 경우와 4성 이상 아이템의 획득 확률에 대한 페이 백의 기댓값도 고려해야 하지만 지금까지 낸 결론을 토대로 낮은 확률로 나타났음에도 목표가 이루어졌을 때 상대적으로 더 큰 기쁨을 만끽할 수 있으므로 이러한 즐거움을 위해 여기서 종료하기로 하였다.

또한 이러한 결과를 다른 유저들과의 교차 검증을 통하여 본 모델의 cdf값이 20%인 구간에서 가장 높은 빈도로 목표가 이루어졌음을 알 수 있었다(도움을 주신 분: J.Young 외 유저 일동 다수).

Conclusion

본 연구를 통해 보다 빠른 속도로 유저의 남은 stack 수와 천장 상태 및 뽑고자 하는 한정 캐릭터의 수를 입력 받아 전체 가능한 경우에 대한 확률값을 모두 구할 수 있었고, 실제 실험 통계와 유저 데이터로부터 얻은 가장 가능성이 높은 cdf의 값이 20% 구간에 대한 신뢰성을 확보할 수 있었다. 이를 토대로 원하는 캐릭터를 원하는 만큼 뽑고자 할 때 예산을 토대로 보다 현명한 소비와 지출 계획을 세울 수 있을 것이다.