Simulation/Analysis of Mini Cactpot

Alexander Hu

Introduction

“Mini Cactpot” is a minigame from the MMORPG (Massively Multiplayer Online Role Playing Game) Final Fantasy XIV. This minigame is similar to a scratch-off lottery ticket. Non-repeating integers from 1 to 9, inclusive, are distributed randomly across a 3x3 grid. One of the tiles is revealed to the player at the start; all other tiles are hidden.

The player chooses to reveal three additional tiles on the grid. The player reveals the tile one at a time and sees the hidden information from a newly flipped tile before getting to flip a new tile. After three additional tiles are flipped, the player finally chooses a row, column, or diagonal on the grid.

All remaining tiles are flipped and the three numbers residing on the line chosen are added together. This resulting sum is then correlated with a rewards table and the corresponding reward is given to the player.

The game playing process can be seen in Figure 1.

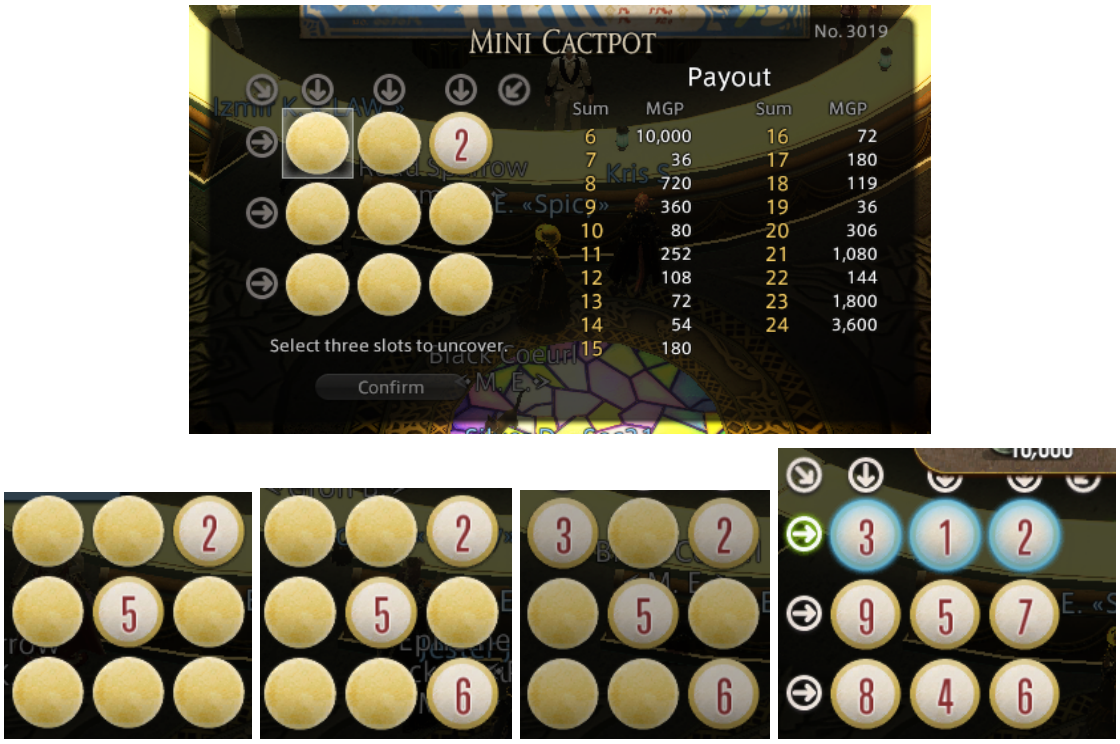

Figure 1: Mini Cactpot example. The game first introduces a board with one revealed tile. The player chooses the middle tile, then the bottom right, and then the top left. Finally, the player chooses the top row, conveniently winning the coveted prize of 10,000 points.

The goal of this project is to simulate this mini game using two different algorithms - one baseline algorithm and one algorithm powering an online solver players use to this day. The results of these simulations would then be analyzed to reach conclusions on Mini Cactpot's long term reward returns.

Simulation Details

Random Numbers Setup

First, to set up the simulation, 15000 random numbers were chosen using a discrete uniform random distribution with the bounds [0,10000]. Then, these numbers are used to set the seeds to randomly generate 15000 boards.

Each board is generated using one of the randomly generated seeds. From this seed, nine random numbers are generated from discrete uniform distributions. The first eight are used to determine which number to slot into the next open tile slot. This uses a shrinking distribution window, as once a tile is filled with a number the pool of available tile spaces shrinks. Thus, the first random number has the bounds [0,8], but it goes down to [0,1]. Only eight random numbers are needed as the ninth number is the only remaining number left, which automatically is placed in the last remaining slot. The last random number generated determines which tile is initially revealed to the player. This also uses the bounds [0,8] as there are nine tiles to reveal.

By producing the random numbers in this way, reproduction is guaranteed, allowing anyone with access to the projects' files to reproduce the work shown later in the report.

Algorithm A - Y-algorithm

The "Y-Algorithm" is the name for the baseline created for the purposes of this report. The algorithm's strategy is as follows:

1. First, look at which tile is flipped. The value revealed is not important, only the tile location.
2. Based on the tile's location, flip three other tiles such that it forms a "Y" shape on the board.
3. Finally, consider all numbers revealed on the board and find the row with the highest expected value returns.

The tiles to flip are hard coded based on the following configurations shown in figure 1.

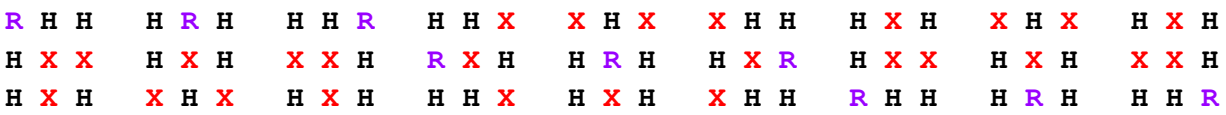

Figure 2: Tiles flipped by Y-Algorithm based on starting tile.
R is the revealed tile's position, X are the tiles to flip, H are the tiles still hidden.

The logic behind this algorithm is that this type of shape guarantees that every row, column, and diagonal has at least one tile's worth of information to use. It is entirely possible to accomplish this in more different configurations than the ones above, but only these nine are necessary to code. Since this algorithm does not check the values of the tiles until after all three tiles are flipped, variable configurations to the nine above make no difference to this algorithm.

This algorithm resembles that of a layman who is aware that creating a "Y" shape would lead to a formation that would provide information to each potential line. Of course, a layman probably cannot quickly solve all the permutation necessary to find the line with the highest expected value, but it still serves as a good comparison point to the other, more established algorithm.

Algorithm B - Aureolux's algorithm

This algorithm was created by the Reddit user Aureolux and further edited and optimized by Reddit users yuryu and super_aardvark. The algorithm works as follows:

1. Examine the input board state
2. If there are four tiles flipped, then go through every possible permutation and solve for the line with the best expected returns.
3. Else, for each empty tile and each number that could be in that tile, create a board that fills in the number in that tile. For each board created, run the algorithm on it starting from step 1. After running through all possible boards, return the tile that would result in the highest possible expected return.

In short, this algorithm uses a brute force method to go through every single possible leaf node in the decision tree that would result from this algorithm, then evaluate the best move to make based on exploring this tree.

Notably, it is hardest and takes the most time to make a decision when choosing the first tile to flip. For this reason, the creators left a hard coded dictionary that explicitly describes which tiles to flip based on what the starting position and value is.

This algorithm differs from the Y-Algorithm in several notable ways. First, it does care about both the positions and the values at said positions at every step of the process. Thus, it does play the game by first flipping one tile, then reevaluating the state of the board. Second, the algorithm was originally designed to provide the user with every possible best move - of which sometimes there are multiple. The way this paper handles this ambiguity is by assigning a priority system for which tile to pick in the case of multiple options. The priority system can be seen in Figure 3.

```
1 2 3
4 5 6
7 8 9
```

Figure 3: Priority system for choosing tile to flip.
A lower number is a higher priority for that tile.

Simulation Outputs

The outputs of interest from this simulation are the sums resulting from choosing a line as well as their corresponding rewards. These results are held in external files for future access. The rewards corresponding to sums can be found in Figure 1, on the first image. That rewards table is always consistent, meaning that every board will always refer to that rewards table.

It's important to note that, given how the rewards table is structured, the expected value would be heavily skewed by the possibility of revealing a row with the sum of 6 due to its incredibly high reward of 10,000 points relative to all other results. On a lesser scale, getting a sum of 24 may also result in some skewing of the expected value.

Algorithm Analysis

Runtime Analysis

The Y-Algorithm actually uses the same solving function used in Aureolux's algorithm. However, it only ever has to run the case where all four tiles are flipped. For this reason, the simulation is very quick, taking only a few seconds to run through all 15000 boards and return the relevant outputs.

On the other hand, Aureolux's algorithm is far more computationally intensive. Although there is no need to compute the first layer, even the second layer takes 2-3 seconds to solve for each individual board. For this reason, the machine used to run this simulation took a little over 4.5 hours to finish running the simulation.

Several steps could be taken in order to reduce this runtime. The algorithm written could require less function calls, or relied more on using special features from the dedicated numpy library. It may have also been better to use a language like C++, a language in which source code was actually already provided.

Steady State Analysis

Due to the Law of Large Numbers, the calculated mean of a simulated output would approach the actual mean as more and more trials are run. In this fashion, it is possible to find the steady state expected average by graphing the average as the number of boards played increases.

Figures 4 and 5 show line graphs plotting the average winnings every 20 games up until the final 15000th game.

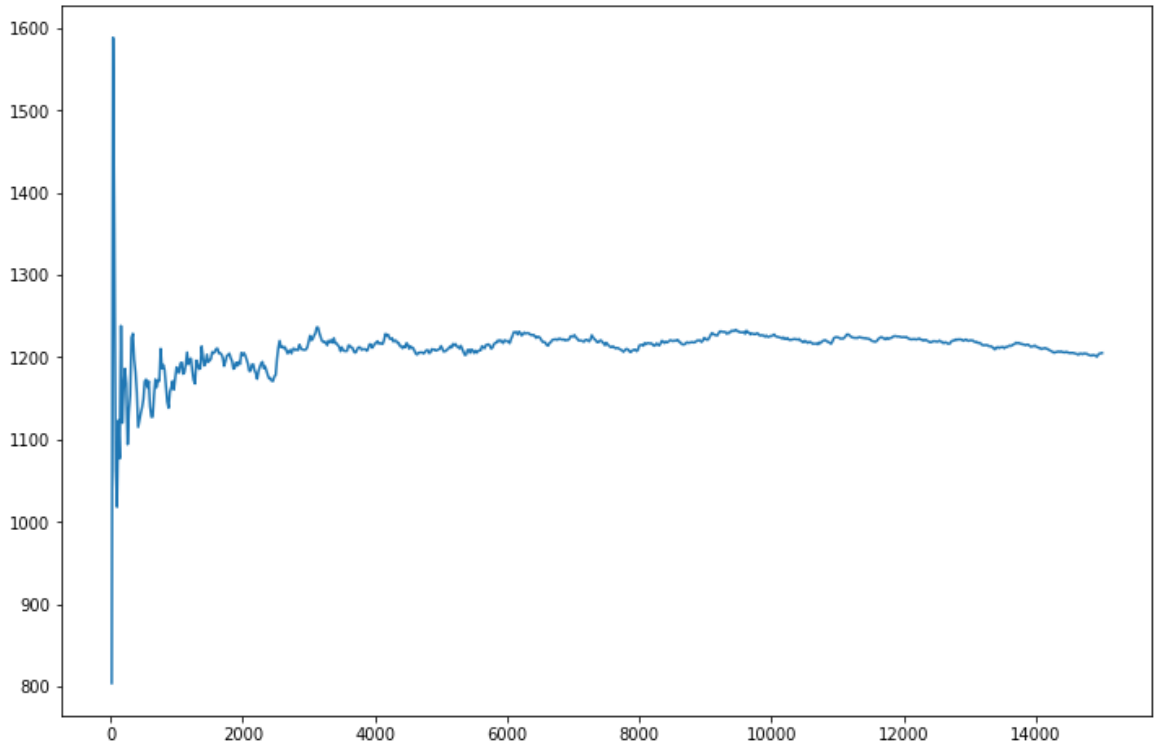

Figure 4: Average reward from the Y-Algorithm as number of boards played increases.

Several conclusions could be drawn from these graphs. First, it is apparent that both algorithms appear to be reaching some sort of steady state as the number of trials approaches 15000. Second, the steady state point of Aureolux's algorithm is notably higher, by about 250 points.

To get more exact details on the average expected results, the sample mean, sample standard deviation, and mean with 95% confidence interval were found for both algorithms. Equation 1 was used to find the mean with 95% confidence interval.

$$\overline{X}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2(n)}{n}}$$

Equation 1: Mean with α confidence interval.

The results are provided in Figure 6.

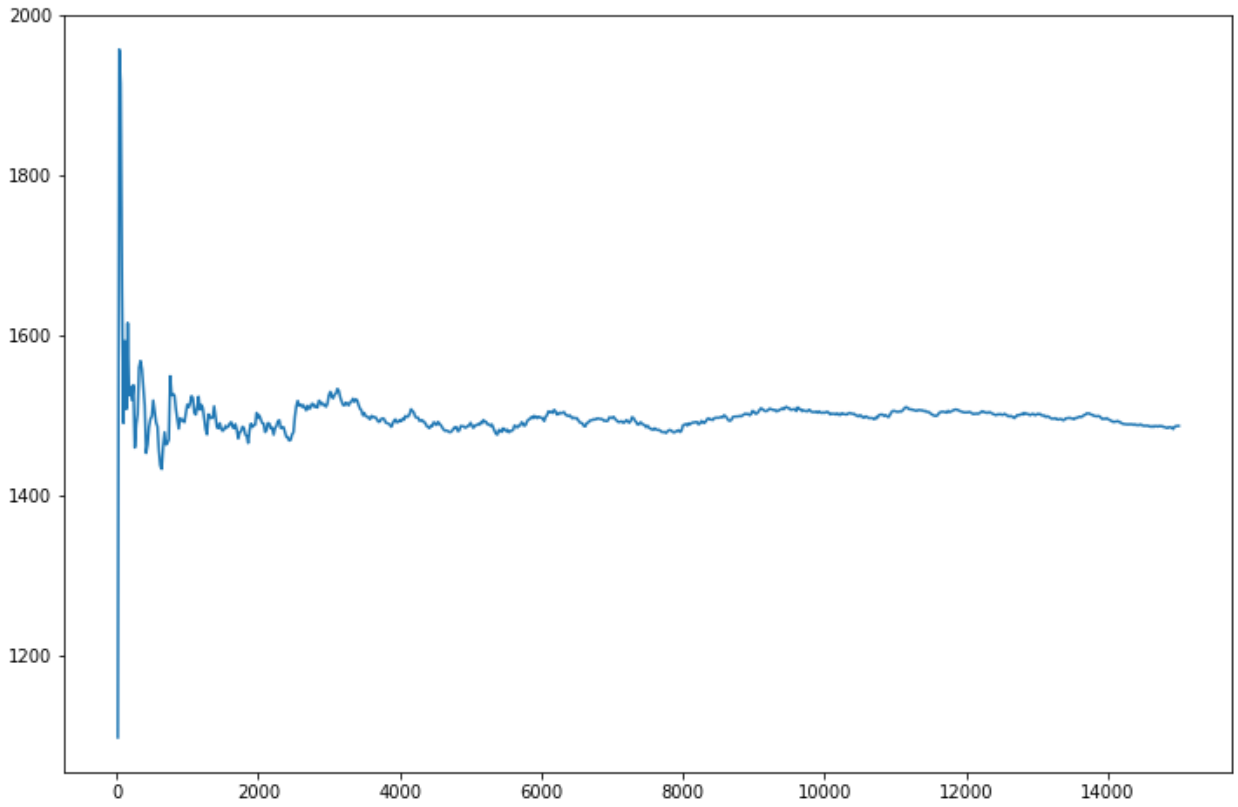

Figure 5: Average reward from Aureolux's algorithm as number of boards played increases.

	Sample Mean	Sample Std Deviation	Mean w/ 95% Confidence
Y-Algorithm	1205.22	22.76	(1160.61, 1249.82)
Aureolux	1486.93	23.39	(1441.07, 1532.79)

Figure 6: Results of both algorithms.

While the difference between the Y-Algorithm and Aureolux's algorithm does not seem particularly large, it would become apparent later that such a small disparity stacks up under longer term conditions.

Distribution Analysis

One point of interest for this project was whether or not the results gathered from these algorithms fall under any discernible distribution.

In regards to the Y-Algorithm, there appears to be a curious distribution when viewing a histogram-like bar graph in which the buckets are the possible sums of any line. This graph can be seen in Figure 7.

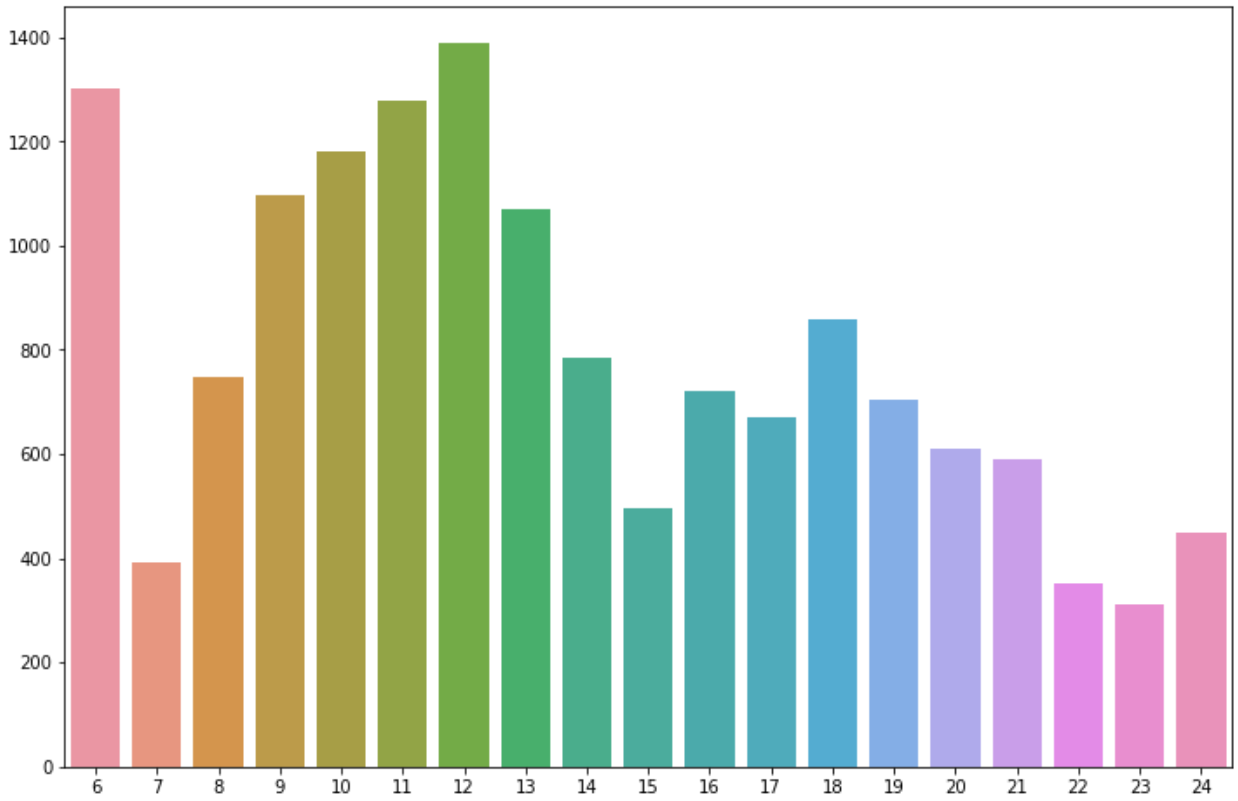

Figure 7: Number of resulting sums after running Y-Algorithm over 15000 boards.

Although the result with sum 6 is an obvious outlier, the graph otherwise suggests that there may possibly be a combination of two distributions, one with mean at around 11 to 12 and another at around 18. It is difficult to discern a mixed distribution, however, so that remains up to speculation for the purposes of this report.

Some other graphics were considered for analysis. Figure 8 displays the number of resulting rewards, ordered by increasing value. However, it fails to suggest any particular distribution. This result was also ordered in increasing frequency in Figure 9, showcasing that despite the incredibly high worth of 10,000 points, it actually was not the most frequent resulting reward when using the Y-Algorithm.

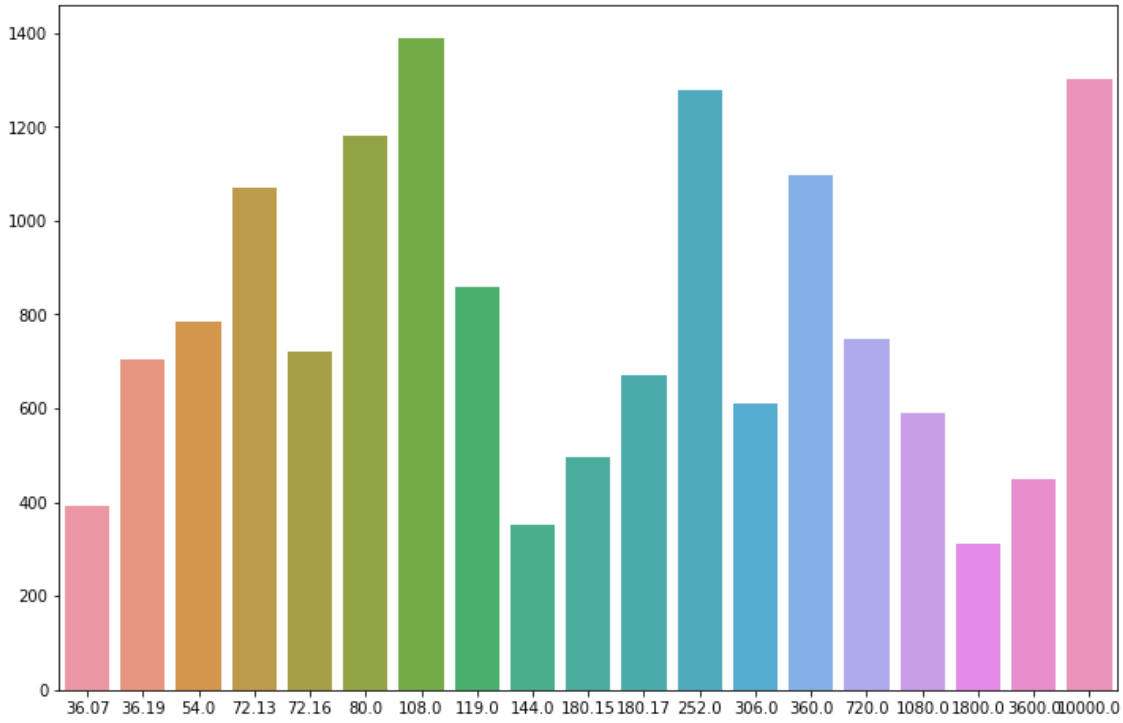

Figure 8: Number of resulting rewards after running Y-Algorithm over 15000 boards. Some different sums gave the same rewards, they are assigned as decimals to discern them.

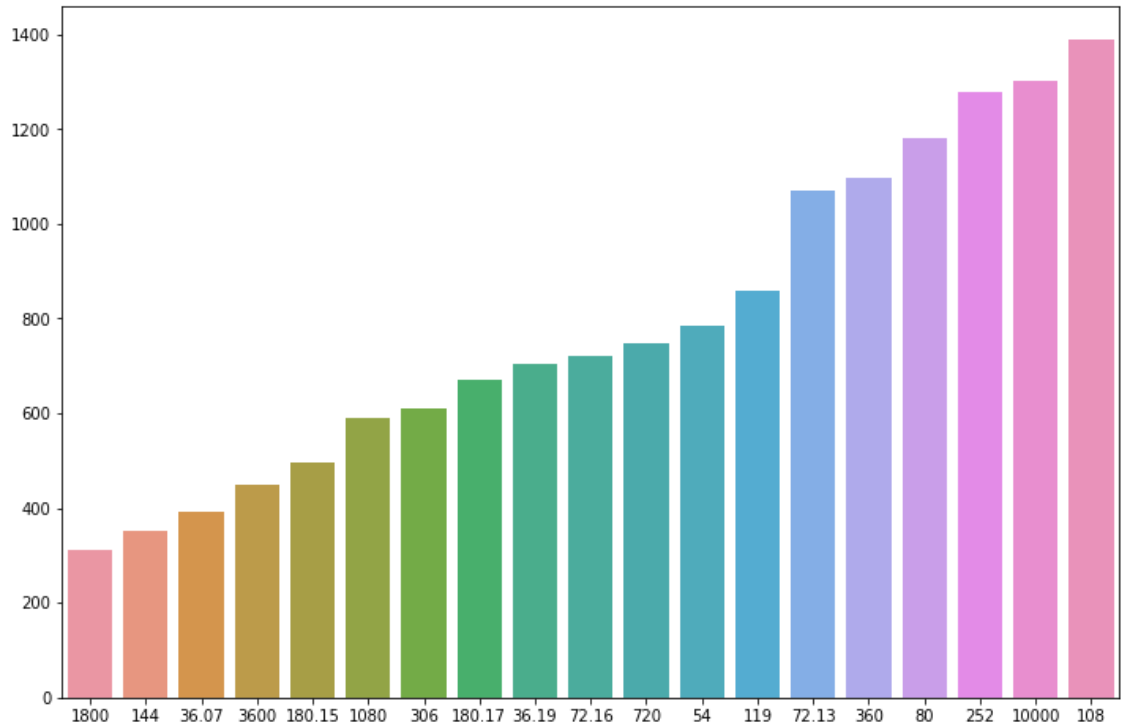

Figure 9: Resulting rewards from Y-Algorithm, ordered by frequency of occurrence.

Aureolux's algorithm actually suggested a similar situation. It also appeared to have two peaks at around 12 and 18. One thing that was notable, however, was a large spike for the result of 21. This may be because 21 is actually a sum with a notable reward, so the algorithm is willing to invest more in order to get that result. Another notable point was the much higher frequency of achieving 10,000 points as a result compared to Y-Algorithm. Given its ability to perform longer term decisions compared to the other algorithm, this result shouldn't be too much of a surprise.

Similarly to the Y-Algorithm graphics, checking distribution based on increasing rewards did not seem to bear any fruit. It may be interesting to compare the increasing frequency graphs, however. It is evident that Aureolux's algorithm is far better at choosing sums with higher point values compared to the Y-Algorithm.

The resulting graphics can be seen in Figures 10, 11, and 12.

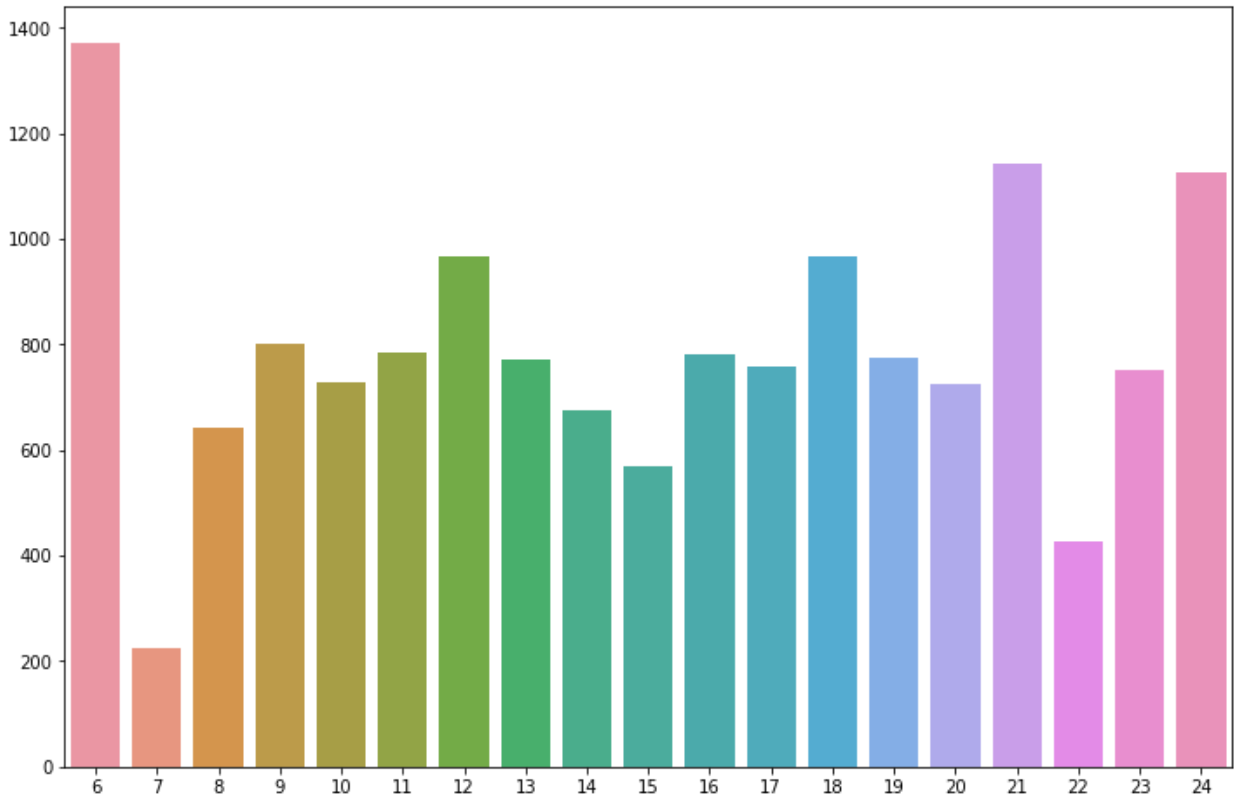

Figure 10: Number of resulting sums after running Aureolux's algorithm over 15000 boards.

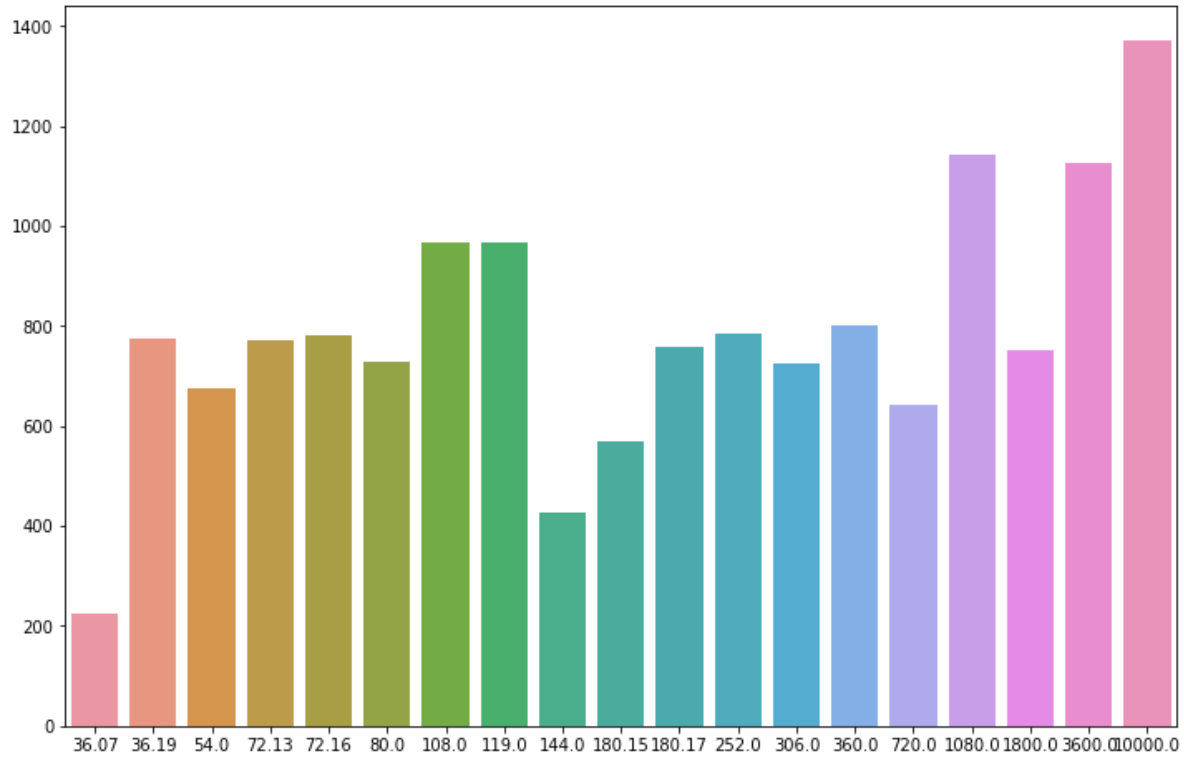

Figure 11: Number of resulting rewards after running Aureolux's algorithm over 15000 boards. Some different sums gave the same rewards, they are assigned as decimals to discern them.

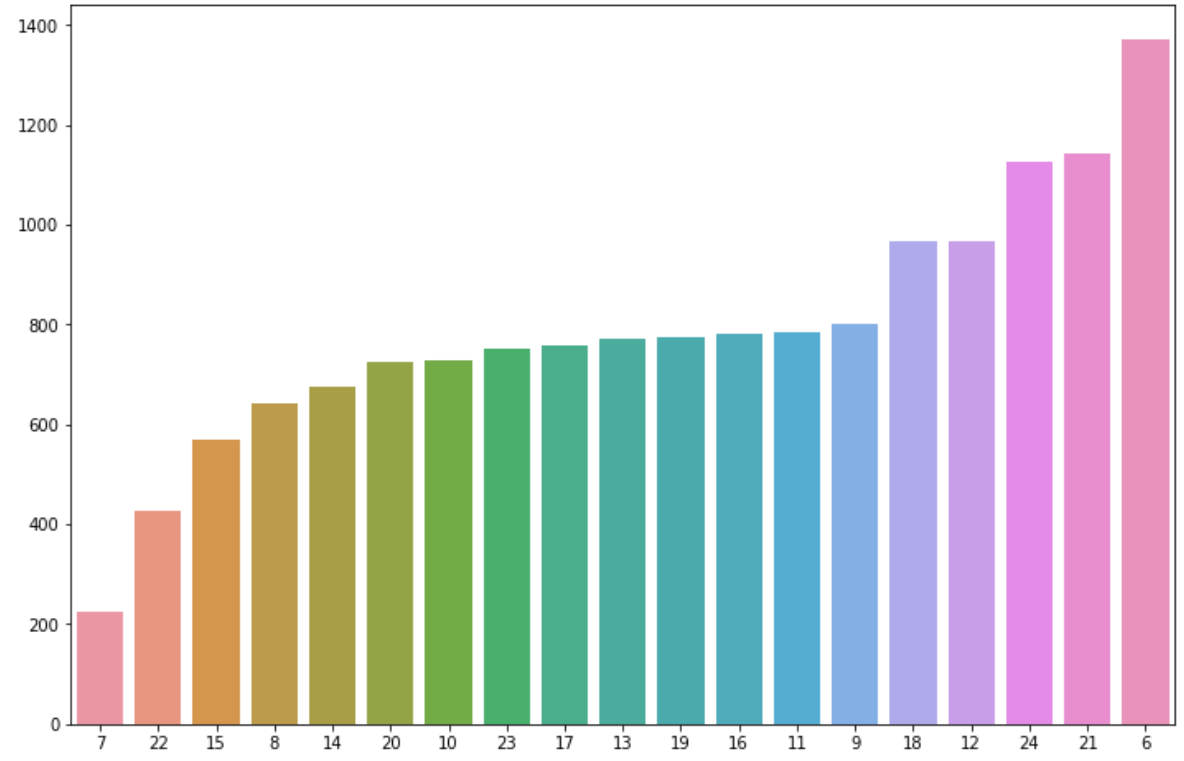

Figure 12: Resulting rewards from Aureolux's algorithm, ordered by frequency of occurrence.

Finally, it was important to consider how different the distribution of results are compared to each other. In order to accomplish this, a chi-square test was used. As both results could easily be placed into buckets that are simply just the resulting sums and these buckets qualify as classifiers, using the chi-square test is a valid and also easy way to compare the two results. Equation 2 was used to perform the Chi-Square test. As both algorithms have the same number of buckets and also the same number of total entries, both K coefficients are assumed to be 1.

$$\chi^2 = \sum_{i=1}^{k} \frac{(K_1 R_i - K_2 S_i)^2}{R_i + S_i}$$

Equation 2: Chi-Square Test Value. k is the total number of bins, i determines the specific bin. R and S are frequencies found for the bin i

After calculating this value, it is used in a specialized function in order to find the final p-value. This final p-value was evaluated to be 0.0, or in other words incredibly small. As this is definitely under the 0.05 interval threshold, this means that the two algorithms definitely does not produce results similar to each other in the slightest.

Mean/Standard Deviation For Reaching Specific Amount

As a practical exercise, it is important to see how many attempts it would take to reach a certain total value. It is possible to do this by considering the raw results of each algorithm - by iterating through the resulting rewards and adding them together, it is possible to simulate playing the game in succession until reaching a total sum.

Thus, this simulation was run by adding together successive sums and keeping track of the number of trials taken until a certain threshold was met, then resetting both the sum and trials to 0 before repeating until there were no more data points left.

The thresholds considered were 10,000 points, 50,000 points, 100,000 points, 200,000 points, and 500,000 points. The results can be found in Figure 13.

	10,000 points	50,000 points	100,000 points	200,000 points	500,000 points
Y-Algorithm	(10.37,0.22)	(44.73,0.83)	(86.14,1.58)	(169.63,2.90)	(418.51,8.10)
Aureolux	(8.45,0.14)	(35.89,0.58)	(69.63,1.08)	(136.96,2.00)	(339.25,4.74)

Figure 13: (Sample Mean, Sample Std Deviation) of rounds taken to reach total sum rewards for both algorithms

Here it becomes apparent how the seemingly miniscule difference of 250 points starts adding up. It takes 9 rounds less to reach 50,000 points, 15.5 rounds less to reach 100,000 points, 27 rounds less to reach 200,00 days, and a good 79 rounds less to reach 500,00 points. As a

player can only play 3 rounds of Mini Cactpot a day, that's almost a month saved overall. With that, it should be clear that using Aureolux's algorithm reaps quite noticeable benefits over the long term.

It is important to note that each round of Mini Cactpot actually costs any prospective player 10 points. This may seem like it could affect the given results, but the next section makes it clear that it actually factors in very little even over long periods of time.

Mean/Standard Deviation For Playing A Certain Number of Days

SImilar to the previous section, it would be practical to explore the mean and standard deviation of how many points would be accrued given a certain number of days. Specifically, in the case of Mini Cactpot, every player may only play three times per day. Thus, the amount of rounds a player can play over a period of time is 3 * (Number of Days).

The method of simulating this is similar to the previous section - the resulting rewards for both algorithms are iterated through. By keeping track of the number of boards played, it is possible to record the total score after a certain number of boards, then resetting the number of boards played and total value and continuing on.

The thresholds considered were 7 days, 14 days, 30 days, and 90 days. The results can be found in Figure 14.

	7 days	14 days	30 days	90 days
Y-Algorithm	(25317.39, 500.92)	(50634.77, 963.59)	(104368.53, 1987.23)	(314097.72, 5210.06)
Aureolux	(31236.10, 515.02)	(62472.21, 1003.43)	(133714.45, 2077.44)	(400775.60, 5618.72)

Figure 14: (Sample Mean, Sample Std Deviation) of total rewards after playing for a specified number of days for both algorithms.

Again, we start seeing the drastic effects 250 points of difference has once more and more rounds are played. Even over the course of 7 days, a difference of almost 6000 points is amassed. Over the course of 2 weeks, that doubles. It more than doubles again over the course of a month. It's also evident that there is a linear correlation between days played and the increase, making it increasingly more worthwhile to use Aureolux's algorithm if playing Mini Cactpot every day over the long term.

Given these values, it can be seen that the cost of 10 points per board played is actually very minimal to the overall results. Thus, it could more or less be ignored in the analysis without too much issue.

Conclusion

Although it was difficult to figure out the potential distribution of the results for both algorithms, all other goals mentioned in the introduction were met. Through the analysis performed, it should be clear that the Aureolux algorithm does provide a noticeable edge in performance compared to playing relatively normally. How much of an advantage and whether it is worth always using the solver to play Mini Cactpot is a decision left up to the reader.