Yahtzee Algorithms

Trish Beeksma, Patch Kenny, and Matt Renner

University of North Carolina Wilmington

Abstract
This paper discusses four algorithms that were designed and analyzed to produce the highest average score for a game of Yahtzee. The algorithms tested are Lower Section Strategy, Upper Section Algorithm, Naive Algorithm, and Exhaustive Search. The average score and point distributions were collected and compared after running 1,000,000 games of Yahtzee with independently random rolls. The complexity of each algorithm was analyzed. The algorithms were also compared by using controlled rolling that disregarded the score card completely.

1. Introduction

Yahtzee is a game of chance and strategy played with five dice and thirteen different scoring categories. The goal of Yahtzee is to obtain the highest score. Each game of Yahtzee has thirteen turns. During each turn, the player has three rolls. On the first roll, the player rolls all five dice. After the first roll, the player may roll a subset of the dice up to two times before choosing a scoring category from one of the remaining unused categories. By the end of the game, each scoring category will have been assigned one score. The score for any of the categories can equal zero if a category is not accomplished. The final score is the sum of all the categories.

The thirteen different scoring categories and the possible scores are listed below.

1. 1’s – sum of all ones
2. 2’s – sum of all twos
3. 3’s – sum of all threes
4. 4’s – sum of all fours
5. 5’s – sum of all fives
6. 6’s – sum of all sixes
7. 3 of a kind – sum of all dice
8. 4 of a kind – sum of all dice
9. Full House (3 of one number, 2 of another) – 25 points
10. Small Straight (four sequential dice) – 30 points
11. Large Straight (four sequential dice) – 40 points
12. Yahtzee (5 of a kind) -50 points
13. Chance – sum of all dice

An additional rule of Yahtzee states that if the sum of the upper section is at least 63, then 35 bonus points are added to the total. The upper section categories include 1’s, 2’s, 3’s, 4’s, 5’s, and 6’s. Also, for each additional Yahtzee obtained after the initial Yahtzee, 100 bonus points will be added to the total.

2. Problem Statement

Informally stated the problem is to find an algorithm that will determine which dice to keep on each roll and what category to fill after each turn for sufficiently large number of games that returns the highest average score in a game of Yahtzee. The algorithms used are the Lower Section Strategy, Upper Section Algorithm, Naive Algorithm, and Exhaustive Search.

Formally, the final score of a game of Yahtzee, F, is given by the sum of the upper section score, upper, and the sum of the lower section score, lower.

\[ F = \text{upper} + \text{lower} \]

Where \( \text{upper} = \sum_{i=1}^{6} s_i \) and \( \text{lower} = \sum_{i=7}^{13} s_i \)
And where \( N \) is the set of scores, \( s_i \), from each of the 13 categories in Yahtzee such that

\[
N = \{s_1, s_2 \ldots s_{13}\}
\]

Note: In accordance with the rules of Yahtzee, if upper is greater than or equal to 63, upper = upper + 35.

Constraints: Each category is assigned one score and only one score by the end of the game.

Scoring Constraints:

1. If \( 1 \leq i \leq 6 \) then \( s_i \in \{0, i, 2i, 3i, 4i, 5i\} \)
2. If \( i \in \{7, 8, 13\} \) then \( 0 \leq s_i \leq 30 \)
3. If \( i = 9 \) then \( s_i \in \{0, 25\} \)
4. If \( i = 10 \) then \( s_i \in \{0, 30\} \)
5. If \( i = 11 \) then \( s_i \in \{0, 40\} \)
6. If \( i = 12 \) then \( s_i \in \{0, 50\} \)

3. Lower Section Strategy

The Lower Section Strategy works by prioritizing the categories in the lower section of the sheet and taking the upper section of the sheet when the algorithm fails to acquire a result in the lower section. The complexity of the implementation of this algorithm is \( O(n^2) \), where \( n \) is the number of dice needed to reroll and ran 1,000,000 games in about 27 minutes. The Lower Section Strategy evaluates every category that has not been filled using the equation:

\[
P_i = v_i \times (\%)^n
\]

Where \( P \) is the current category, \( i \) is the number of the category, \( v \) is the point value of the category, and \( n \) is the number of dice required to be rolled to meet the requirements of that category.. Then chooses the category with the maximum value from the set: \( \{P_1, P_2, P_3 \ldots P_{13}\} \)

It is important to note, while the Lower Section Strategy does evaluate the categories in the upper section, if there is a possible outcome in the lower section that is not 0, then the algorithm will favor that outcome over the upper section possibility.

4. Lower Section Strategy Analysis

The Lower Section Strategy does have its merits on paper. All of the high scoring categories are in the lower section and while the upper section does give 35 bonus points for having over 63 points in it, most of the lower section point values are around 30 and there are more of them. However, in practice the Lower Section Strategy performed below par. Looking at the histogram of 1,000,000 games shows a skewed distribution toward lower point values (values from 120 points to 145 points) rather than the more optimal distribution which would have been skewed toward the opposite end. The average score for this algorithm was 154.3 and the maximum score was 332, which (as shown later) is not an optimal result compared to the other algorithms.

The main reason why the Lower Section Strategy fails to be an optimal algorithm for Yahtzee is due to a leftovers effect on the upper section. By leftovers it is meant that when a category in the lower section fails to be rolled, rather than putting a 0 in that category, the algorithm will choose the highest point value in the upper section and fill that in instead. This leads to having a very poor upper section, with only 1 or 2 of the same number being recorded. This happens especially in cases where the algorithm is rolling for straights or is rolling for three, four, or five of a kind and the number being rolled for is already filled in the upper section.
So, it can be concluded from this data and from the runtime that the Lower Section Strategy is not an optimal algorithm in order to achieve high point values in a game of Yahtzee.

Figure 1- Lower Section Strategy Data

![Histogram Lower Section Strategy](image)

5. Upper Section Algorithm Design

The Upper Section Algorithm focuses on obtaining the 35 extra bonus points. This happens when the sum of the upper section categories (1’s, 2’s, 3’s, 4’s, 5’s, 6’s) is at least 63 points. Until the upper section is completely filled, this algorithm will go for the remaining available upper section categories when determining which dice to reroll. It will pick the highest most recurring dice to keep from the available upper section categories. After all the upper sections are filled, the algorithm will continue with the Lower Section Strategy when determining which dice to reroll.

Let \( U \) be the scoring categories of the upper section such that \( U = \{s_1, s_2 \ldots s_6\} \) then

\[
\sum_{i=1}^{6} 3s_i = 63
\]

This shows that if each upper section category is three of a kind, the upper section total will equal 63. Because this is the case, this algorithm will assign a score to the upper section if there are at least three recurring dice. It will also fill an upper section category if the sum of that section and the current upper section total equals at least 63. If all the categories in the upper section are filled, the algorithm will fill the lower section that yields the highest score. If all lower section categories return a score of 0, the section with the lowest probability of being obtained will be filled.

6. Upper Section Algorithm Analysis

The Upper Section Algorithm performed well in relation to both the total score and the time taken for the algorithm to run. The algorithm did what it was designed to as it obtained the bonus 35 points 94.66% of the games played.

For each of the 1,000,000 games that were played using the Upper Section Algorithm, the total score was determined. The point distribution is shown in Figure 2. The histogram shown below is more skewed to the right in comparison to the Lower Section Algorithm. The average score was 192.8. The maximum score was 343. The histogram is irregularly shaped because the upper section is prioritized. This made the scoring in the lower section not consistent. The big jumps show that the larger scoring sections were not regularly rolled.

Figure 2 - Upper Section Strategy Data

![Histogram Upper Section Strategy](image)

Figure 3 below is the average non zero marks from the Upper Section Algorithm. The histogram had a consistent floor of around 110. A local maxima occurs when the average non zero marks is an integer.
The time complexity of the Upper Section Algorithm is $O(n^2)$ and $\Omega(n)$ where $n$ is the number of dice rolled. It is bounded above by the time complexity of the Lower Section Strategy. It is bounded below by $\Omega(n)$. This is the best case scenario and would occur if the lower section algorithm was not implemented at all during the game.

The time taken to run 1,000,000 games was 14 minutes and 30 seconds. On average, the Lower Section Algorithm was used to determine what dice to reroll for 4.5 turns out of the 13 turns in Yahtzee. This is the case because the algorithm does not fill a score in the upper section unless it is beneficial toward the 63 points.

7. Naive Algorithm

In yahtzee, the single highest possible score is marking $s_{12}$ 50, achieved by having all 5 dice show the same integer value. A naive algorithm for this game would thus simply try every turn to obtain the requirements necessary to achieve that mark of 50. If, at the end of the round, it failed to get all the dice to have the same value, it would then mark whichever $s_i$ would yield the highest point total. The pseudoCode for choosing which dice to reroll is displayed below.

```
Funct rerollLogic(roll):
    dieCount = [0]*6;
    for dice in roll:
        dieCount[dice-1]++;
    modeDie = max(dieCount)+1;
    reroll = [ ];
    i = 0;
    while i < len(roll):
        if roll[i] == modeDie:
            reroll.add[i]
    return reroll
```

The complexity for this algorithm is $O(n)$, where $n$ is the amount of dice in the roll. This is true for both time and space. For space, it is $O(n)$ because in the worst-case 5/6ths of the dice would be chosen to be rerolled, which would make the reroll contain as almost as many dice as the roll did. For time, it is $O(n)$ because the algorithm simply iterates through the roll on 2 separate occasions. To prove the linear complexity, a graph is displayed below of the amount of steps taken by this algorithm on rolls of length 5 to 100 (each roll was tested 10,000 times). A step for this purpose is defined as the amount of times the reroll function looped. The dice rolls were controlled by having an even distribution of values or having all the values the same.

After analysis was done on some preliminary testing, it was found that the algorithm unsurprisingly obtained very few points in scores $s_{10}$ and $s_{11}$, with an average of 7.288 out
of a maximum of 70. A slight alteration was made after this observation to drastically improve performance while still maintaining the simplicity of the original algorithm. The alteration was an if statement added at the beginning of the algorithm which returned true if, after the initial roll, the condition of 4 sequential values being contained in the dice roll was met. If the condition was met, it would then reroll any dice either not contained in that sequence, or any dice with a duplicate value (i.e. if a roll was \{1,2,3,4,4\} it would reroll the second 4).

9. Naive Algorithm Analysis

The Naive Algorithm performed relatively well in terms of score given its simplistic nature. It scored 50 for yahtzee in 42.3% of the games. The slight adjustment made concerning straights made a huge difference, as the average points obtained from \(s_{10}\) to \(s_{11}\) increased to 50.6. The runtime of 1,000,000 was 11 minutes 4 seconds, and yielded an average score of 184.4 (max 339) with a standard deviation of 37.8 points. Below is a histogram showing the distribution of scores across the games.

The graph has a pretty normal distribution with a slight bump around 240. This bump can be attributed to the all or nothing nature of the scores in the lower section.

9. Conclusions

The histogram comparing the point distribution of the Upper Section Algorithm, Lower Section Strategy, and Naive Algorithm is shown below.

It was hypothesized that the Lower Section Strategy would produce the best results in regards to the score and the Naive Algorithm would do the worst. The test resulted in the Upper Section Strategy giving the highest average score, followed by the Naive, then the Lower Section. In regards to run time, the Naive Algorithm ran the fastest, followed by the Upper Section, then the Lower Section.

For the 1,000,000 games tested, each algorithm had the rolls randomly generated for each turn, for each game. Thus, to compare score performance when exposed to the same rolls, they were given a controlled set of rolls and rerolls. This was accomplished by generating a stack of 15 randomly rolled dice, and assigning a copy of the stack to each algorithm. The first 5 values in the stack were then popped and used as the initial roll for the algorithm. The dice that were chosen to be rerolled were assigned a new value given by popping the top of the stack. The process then repeated with the previous roll. That roll then found which \(s_i\) would return the maximum point total and data was recorded for what the point total was along with the corresponding \(i\). For \(s_1\)
to $s_6$, the total was added to by $\text{total}/63+35$. This allowed the obtainable 35 point bonus to be factored in to a method that removed the context of the game.

This allowed the algorithms to not only compare their score performance directly with each other, but also allowed their performance to be compared with the best solution, which was found by exhaustively searching through all possible combinations of rerolls. The graph below shows the average point total by each algorithm after 100,000 rounds, and the appendix contains the data for how prevalent each $s_i$ was for each algorithm.

The score performance for this was reversed, with lower section strategy having the highest average roll score and upper section strategy the lowest. This suggests that the upper section strategy logic towards marking the scores and making sure to obtain the 35 point bonus yielded better results in the context of the game than the other algorithms more simplistic marking logic. The exhaustive search vastly outperformed the others, which was to be expected of an algorithm that had perfect information of all future rolls which the others lacked.

10. Future Work

For the Lower Section Strategy, modifying it so it takes more consideration in the upper section while still prioritizing the lower, could make it outperform other strategies used in Yahtzee. The Upper Section Strategy could be altered in a similar way by taking into account the lower section even when the upper section is not full. If a roll was close to a category that yielded a larger score that 35, it could choose to go for that category and stray from strictly trying to obtain the upper section.
Appendix