An Adaptive Learning Approach to Bogart

Alex Chang
Raymond Lin
Zachary Soohoo
University of California Irvine

August 11, 2013

Abstract

Bogart is a dice rolling game where players try, without rolling ones, to roll more dice than their opponents. The game has not been previously analyzed in depth nor has an optimal strategy been determined. We take the approach of using as adaptive learning AI to figure out an optimal strategy to play Bogart. Our results show that there does exist a form of strategy that may be followed to win the game of bogart on average.

1 Introduction

The dice game Bogart is a push-your-luck game, where a number of players take turns rolling dice. Each player begins their turn by rolling a die. If they roll a one, known as acing out, their turn is over; otherwise, they may either continue rolling or take the pot. Should they decide to continue rolling, they roll one extra die and, as long as they do not ace out they may take the pot or roll an additional die. Each time a die or number of dice is rolled, an equal number of chips from a bank are put into the pot. A player wins the game when they have 30 chips or they roll 5 simultaneous dice without acing out.

1.1 A Sample Game

An example of a game of Bogart is as follows: Player 1 rolls and gets a 1. (1 chip in the pot) Player 2 rolls, gets a 4, and continues rolling. (2 chips) Player 2 rolls 2 dice, neither of which are 1s, and continues rolling. (4 chips) Player 2 rolls 3 dice, none of which are 1s and continues rolling. (7 chips) Player 2 rolls 4 dice, none of which are 1s and continues rolling. (11 chips) Player 2 rolls 5 dice, none of which are 1s and wins the game.

Player Turn	Roll	Current Player's Score	Pot	Result	Roll Again?
A	1	0	1	1	N
B	1	0	2	3	Y
B	2	0	4	3,5	Y
B	3	0	7	2,4,4	Y
B	4	0	11	2,5,6,6	Y
B	5	0	16	2,4,5,5,5	N

Table 1: Game where Player B wins by rolling 5 dice without acing out

1.2 Our Hypothesis

Our hypothesis was that a good strategy for players to follow when both players have a similar score is for a player to stop rolling after their second roll. After a player has completed their first roll, assuming there were previously no chips in the pot, there will be 1 chip in the pot. The expected return of rolling the second set of dice is 1.389 (see Table 2), higher than the 1 in the pot. After the second set of dice has been rolled, the pot has 3 chips in it. The expected return of rolling the next 3 dice is 1.736, lower than the guaranteed 3 in the pot. Therefore, in an even game a player should stop rolling after 2 rolls. On the other hand, we thought that if a player was significantly behind in points, where the other play would almost certainly win on their next turn, it would be more likely to win by rolling 5 dice than to win by points. Winning by rolling this way requires a player to roll a total of 15 dice without rolling any 1's. The probability of this is .0649 (see Table 2). Though this is low, we expected that the chances of this are higher than the chances of your opponent rolling 1's enough that you can win by points.

Value	Result
Chance of rolling 15 dice without acing out	$(\frac{5}{6})^{15} = .0649$
Expected return of 2^{nd} roll	$(\frac{5}{6})^{2}(2) = 1.389$
Expected return of 3^{rd} roll	$(\frac{5}{6})^{3}(3) = 1.736$

Table 2: Table of combinatorics used in forming our hypothesis

2 The Problem

In a real game it is unlikely that a player would be able to roll the 15 dice necessary to win without acing out. Despite the uncertainty of the game, the likelihood of an optimal strategy motivates the analysis of Bogart. Unlike deterministic games, a random element is introduced by the dice, meaning that an optimal strategy consists of decisions that will likely lead to victory, rather than a set of decisions which will guarantee victory. One way to format a strategy for Bogart is as a dictionary, wherein for a specific game state the dictionary

recommends the best move. The game state for Bogart would include the number of chips in the pot, how many chips each player has, and which roll the current player is on. The dictionary would, for each game state, recommend either rolling or taking the pot.

3 Our Approach

In order to find the optimal strategy for Bogart we have used an adaptive learning algorithm. In contrast to a standard adaptive learning program, the program for Bogart rewards winning moves instead of punishing losing moves. This is because of the chance element which, if punishment were used, could cause the program to eliminate potentially good moves because of bad luck. The program increases weights of winning moves thereby increasing the chances of moves that lead to victory being selected.

3.1 Details

We programmed the adaptive learning simulation in Java. We used a hashmap to store all the gamestates the AI encountered throughout the course of one million games. The hash function used was the pot size, and we set the hashmap to be able to hold games of up to size 100. Instead of memorizing the gamestate directly or a copy of the gamestate object, we memorized each gamestate as a string containing all the relevant details of the game. For the weights of the gamestates, we stored it in a parallel hashmap, where each cell of the hashmap corresponded to the gamestate in the same spot in the first table. Every cell in the weights hashmap contained an integer array of size two where the first number represented the weight of yes and the second number represented the weight of no.

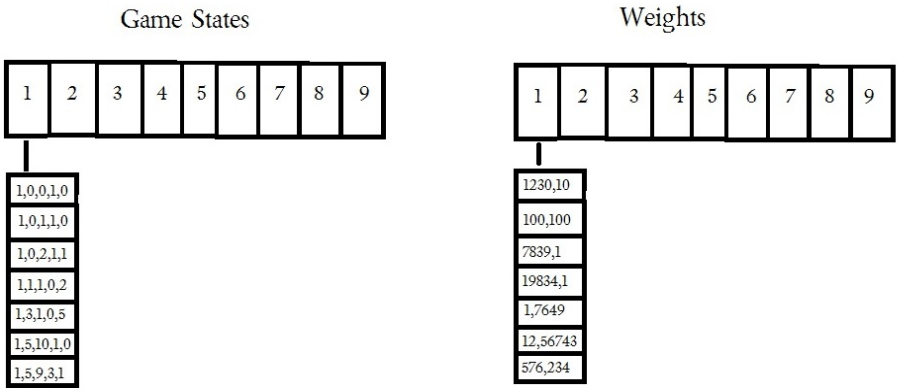

Figure 1: Parallel Hash Maps

Both adaptive learning AIs also memorized another separate array list for the current game it was playing, which was used in rewarding the AIs after a game was finished. Rewards were based on the turn the move was made, so later moves received greater weights than earlier moves. For example, the fifth move made by an AI, would receive a weight of 5. The algorithm for choosing a move was a weighted average of weights on "yes" and "no."

4 Conclusion

For a pot size of 10, it can be seen that the roll number takes a large part of whether or not the weight to roll is on yes or no. This is most likely because if the roll number is higher the player has less chance of success and leaves the opponent with a larger pot if failed (Figure 2). On a pot size of 25, if taking the pot would result in victory then the player should do so, but if the extra two points is needed on the second roll, the data (Figure 3) shows that it is not a bad idea to roll at two. Every other roll number has a high weight on no. When the score difference is 5 the weights are evenly split because there are many game states that have a score difference of 5 and the decision weight would be different on an early game state versus a late game state (Figure 4). When the score difference is 25, there is only four different score differences, and it is safe to say that one player is very behind so the data has high weights on continuation of rolling (Figure 6). One notable section is that the weights on roll 5 are almost all positive.

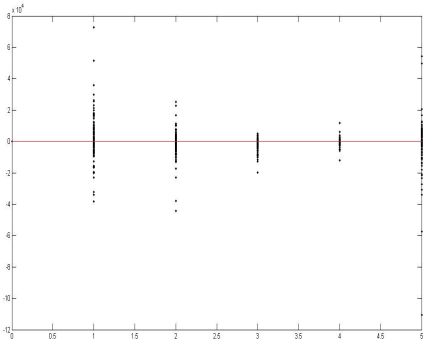

Figure 2: Pot size 10 vs. yes/no

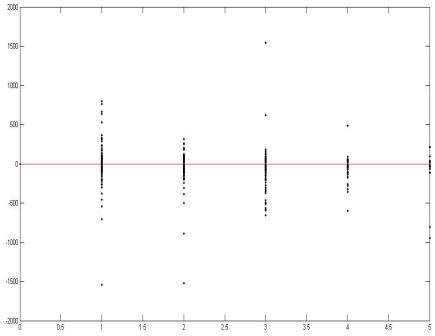

Figure 3: Pot size 25 vs. yes/no

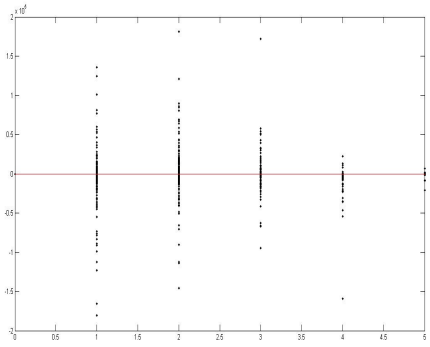

Figure 4: Score difference of 5 vs. yes/no

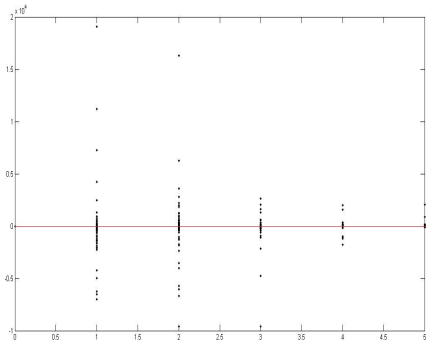

Figure 5: Score difference of 25 vs. yes/no

5 Further Work

After obtaining our results, we realized there were a number of things in our approach that could have been changed, one of the largest problems being our approach to the adaptive learning program. Instead of only hashing a gamestate by its pot size, we could have potentially hashed a gamestate by its pot size, its roll number, player scores, etc... After viewing our data, we realized the data was extremely jumbled and it was hard to tell which gamestates corresponded to each other at different points in time. This type of data made it hard to determine what types of graphs we could use and how to analyze it.

Another problem we came across was during our analysis of the data. We found that each gamestate had a total of four variables to account for: score difference, pot size, roll number, and distance to the end of the game. We were unable to plot them all on a single graph, so we had to settle for making some variables constant and graphing the other variables at the possible fixed gamestates. This resulted in data that, at times, gave consistent results, but for the most part had a large variation with no definite answer.

After analyzing the 2 player game of bogart, we feel that the game is definitely solvable at this level, and might even be solvable for cases of multiple players, such as 3 or more. We would like to remake our adaptive learning AI to hash gamestates through multiple functions, as well as generate many more graphs for analysis. Also, if we did manage to solve the 1v1 case of bogart, we would like to apply our findings to the multi-player versions of Bogart and solve them as well.