Analysis of Low Mean Score Algorithms for "Sixers" Dice Game

Walden Marshall
Thayer School of Engineering
Dartmouth College, Hanover, NH, USA
Email: walden.marshall.th@dartmouth.edu

I. Introduction

"Sixers" is a dice game played by young adults in rural western Massachusetts, though its origins may trace back to Brooklyn, New York. "Sixers" is so named because it is played with six dice. While, like all dice games, there is an element of luck, there is also an element of choice. Therefore, understanding the probability theory behind the game may give players a distinct advantage, and since the game almost always involves gambling, if leveraged correctly, this advantage could translate to financial gains. In this work, we perform some simple experimental statistical analyses with the hope of finding a high performing algorithm. While we find an algorithm that produces the best mean score, we show that this algorithm will likely not produce the winningest record.

II. Gameplay and Rules

A. Objective

Each player's goal is simple: score a lower number of points than all other players.

B. Scoring

The number of points assigned to a player is the sum of the points corresponding to each die. The number of points a die is worth is the value shown on the die with one exception: a die showing three is worth zero points.

C. Starting Out

The players take turns according to counterclockwise order around the table. Each player rolls a single die and the player with the highest value goes first.

D. A Single Turn

Each player's turn is the same. The player starts by rolling all six dice, and after each roll must choose which dice to reroll. Those not rerolled are put to the side and will count towards that players final score. The player must keep at least one die after each roll. Players are allowed up to two rerolls (three total rolls).

E. Ties

In the event of a tie for lowest score, all players not involved in the tie are eliminated and the players involved in the tie play a round by themselves to determine a winner. The same holds for deciding who starts the rolling, except in this case it is the highest value of the single die rather than the lowest score of all six dice that is the determining factor.

F. Example

Here we illustrate an example turn to further reiterate the mechanics and basic strategy. First the player rolls all six dice.
Roll 1: $\{1, 3, 4, 4, 6, 6\}$

The player chooses to keep 1 and 3 because these have favorable scores. They reroll the other four dice which have unfavorable scores.
Roll 2: $\{4, 5, 5, 6\}$

All dice correspond to relatively bad scores, but the player must keep one die, so they keep the 4 and reroll the other three dice.
Roll 3: $\{1, 3, 4\}$

This is the third roll, so the player is forced to keep all three. They kept 1 and 3 on the first roll, 4 on the second roll, and 1, 3, and 4 on the second roll. So their total score is $1 + 0 + 4 + 1 + 0 + 4 = 10$.

III. Basic Theoretical Calculations

A. Simple Combinatorial Analysis

Here we evaluate the number of possible trajectories given this set of rules. Note that we are calculating the number of trajectories, not outcomes (final scores).

The trajectory is comprised of the results of the (up to) three dice rolls. For a roll of n dice, the number of possible dice combinations is the number of multisets of size n chosen from a set of size six (the number of faces on a die). In general, the number of multisets of size n chosen from a set of size k is given by the binomial coefficient $\binom{n+k-1}{k-1}$ [1]. The problem of determining the number of possible trajectories of dice rolls reduces to the problem of determining the number of possible trajectories of n, the number of dice rolled. **Figure 1** shows the possible trajectories of n.

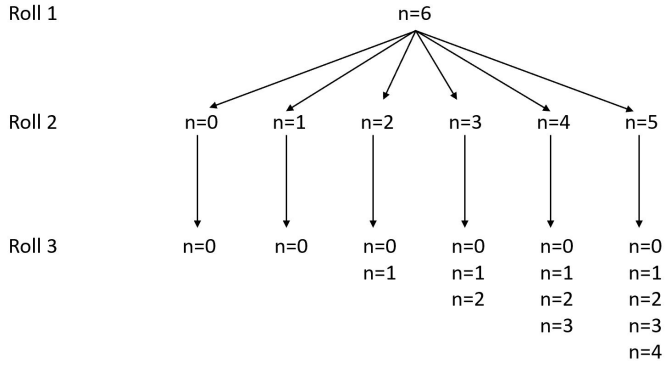

Fig. 1. Possible trajectories for n, the number of dice rolled over the three rolling cycles

At each roll, the number of possible dice combinations is $\binom{n+5}{5}$. Therefore, the total number of possible trajectories N is given as

$$N = \binom{11}{5} \cdot \left[\sum_{i=0}^{5} \left[\binom{i+5}{5} \sum_{j=0}^{i} \binom{j+5}{5} \right] \right]$$

Using Matlab [2] to perform the summations, we find the number of possible trajectories is $N = 68,477,178$.

We perform this calculation mainly to show that it may be difficult to analytically determine the final score probability distributions. Since we end up using simulation to estimate the probability distributions, this calculation also shows that we should use very large sample sizes in simulations if we expect convergence to the true distribution.

B. Mean Contribution to Score

For a single roll of a single die, we can calculate the expected contribution to final score $E(S)$ based on the bijection between value X shown on die and corresponding score $S(X)$.

$$\begin{aligned} E\big(S(X)\big) &= \sum_{X} \big(P(X)E\big(S(X)\big)\big) \\ &= \frac{1}{6}\big(1+2+0+4+5+6\big) \\ &= 3 \end{aligned}$$

We perform this calculation to inform potential strategies players could use.

IV. Simulation

Here we automate the player's decision making with algorithms so that we can find the score distributions achieved over a large number of rolls. Using Matlab code I have written [2], we are able to test algorithms in hope of finding the highest performing one.

The first step in implementing these algorithms is to recognize that we may define D, the six element multiset of dice scores from $S \in \{1,2,0,4,5,6\}$, as a function f of ordered pairs of multisets R_i and K_i where R_i and K_i are the scores of the dice rolled in the i^{th} roll and kept after the i^{th} roll respectively.

$$D = f\big((R_1,K_1),(R_2,K_2),(R_3,K_3)\big)$$

Note that the rules of the game require $R_3 = K_3$, while we have freedom to choose the functions from R_1 to K_1 and from R_2 to K_2. Defining those functions as $K_1 = f_1(R_1)$ and $K_2 = f_2(R_2)$, we may then rewrite D as a function of random variables (roll outcomes) and deterministic parameters (functions).

$$D = f\big(R_1,R_2,R_3;f_1,f_2\big)$$

Any algorithm f can then be defined in terms of two separate algorithms f_1 and f_2 which represent the choice of which dice to keep after the first and second roll respectively. Let $R_i = \{r_i^1, r_i^2, ...r_i^n\}$ where n is the number of dice rolled and i is the roll number. The function f_i is one which selects a subset of R_i.

In this work, we will implement three algorithms: Dice Value threshold, f^{dv}, Dice Number Kept, f^{dn}, and Turn Score Threshold, f^{ts}.

A. Algorithm A: Dice Value Threshold

The first algorithm we implement, f^{dv}, is one which, after each turn, keeps all dice with score below a certain threshold value. That is

$$D = f^{dv}\big(R_1,R_2,R_3;f_1^{dv},f_2^{dv},s_1,s_2\big)$$

where

$$f_i^{dv}(R_i) = \{r_i^j | r_i^j \leq s_i\}$$

where s_i is the threshold for the i^{th} role.

Since we have calculated the expected value for a single roll of a single die as 3, this seems the natural first choice for the threshold for both turns. Since no die roll results in a score of 3, we can lower the threshold to $s_1 = s_2 = 2$ without loss of generality. The second choice is to play conservatively and let $s_1 = s_2 = 4$. The third choice is to be play greedily by letting $s_1 = s_2 = 1$. Finally, we consider the choice to be greedy at first and less greedy second, i.e. $s_1 = 1, s_2 = 2$. The rationale here is that we are able to reroll a die twice more after the first roll, so it is likely we able to improve upon the score of 2. In fact, the probability of rolling a score less than 2 on either of two rolls is $(1-\frac{2}{6})^2 = \frac{16}{36} = 0.\bar{4}$ as opposed to only $0.\bar{3}$ on a single roll. Simulating f^{dv} for these choices of score thresholds, we find the results shown in **Figure 2**.

B. Algorithm B: Dice Number Kept

The next algorithm, f^{dn}, is one which, after each turn, sorts the dice scores in ascending order and keeps some number of the lowest scoring dice specified by a number. That is

$$D = f^{dn}\big(R_1,R_2,R_3;f_1^{dn},f_2^{dn},n_1,n_2\big)$$

where

$$f_i^{dn}(R_i) = \{r_i^j\} \text{ for } j \in [n_i]$$

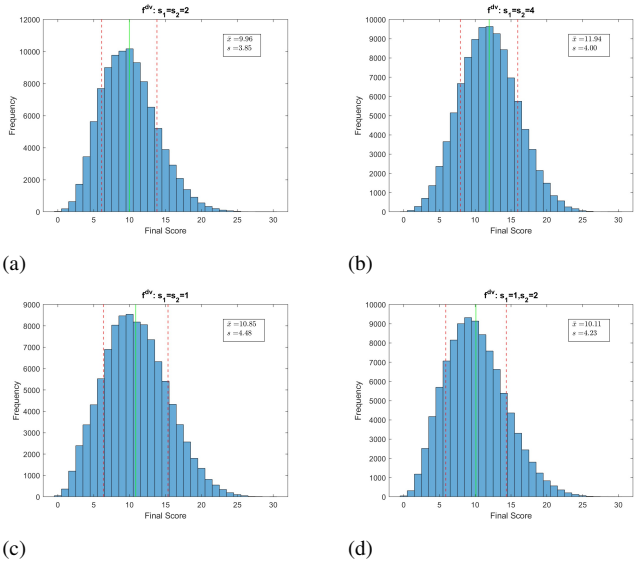

Fig. 2. Score distributions of 100000 trials for f^{dv} with a) $s_1 = s_2 = 2$, b) $s_1 = s_2 = 4$, c) $s_1 = s_2 = 1$, and d) $s_1 = 1, s_2 = 2$

where the elements of R_i are sorted in ascending order and n_i is the number of dice kept after the i^{th} role. Note that the rules of the game require $n_i \geq 1$.

Since we are permitted up to three roles and have six dice, the most natural first choice is to keep 2 dice each turn, i.e. $n_1 = n_2 = 2$. Another choice is to keep more early on, since with a larger number rolled, there are likely more low scoring dice. Therefore, let the second choice be $n_1 = 3, n_2 = 2$, and the third choice be $n_1 = 3, n_2 = 1$. Let the final choice considered be $n_1 = 2, n_2 = 3$. Simulating f^{dn} for these numbers of dice kept, we find the results in **Figure 3**.

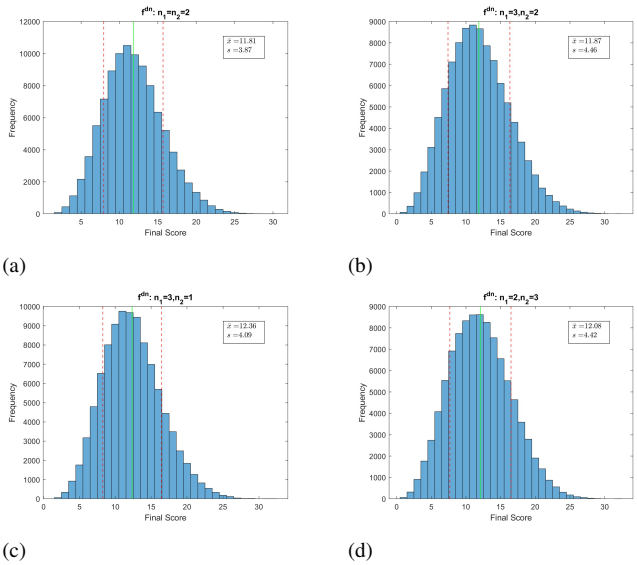

Fig. 3. Score distributions of 100000 trials for f^{dn} with a) $n_1 = n_2 = 2$, b) $n_1 = 3, n_2 = 2$, c) $n_1 = 3, n_2 = 1$, and d) $n_1 = 2, n_2 = 3$

C. Algorithm C: Turn Score Threshold

The final algorithm we consider, f^{ts}, is one which keeps the score of the dice kept after each turn under a certain threshold. That is

$$D = f^{ts}\big(R_1, R_2, R_3; f_1^{ts}, f_2^{ts}, \sigma_1, \sigma_2\big)$$

where

$$f_i^{ts}(R_i) = \{r_i^{[j]} | \sum_{k=1}^{j} r_i^k \leq \sigma_i\}$$

where σ_i is the sum of dice kept on the i^{th} role that must not be exceeded.

Choose four combinations of σ_1, σ_2 to be $(\sigma_1 = \sigma_2 = 5)$, $(\sigma_1 = 4, \sigma_2 = 6)$, $(\sigma_1 = 6, \sigma_2 = 4)$ and $(\sigma_1 = \sigma_2 = 6)$. For these turn score thresholds, simulation of f^{ts} produces the results shown in **Figure 4**.

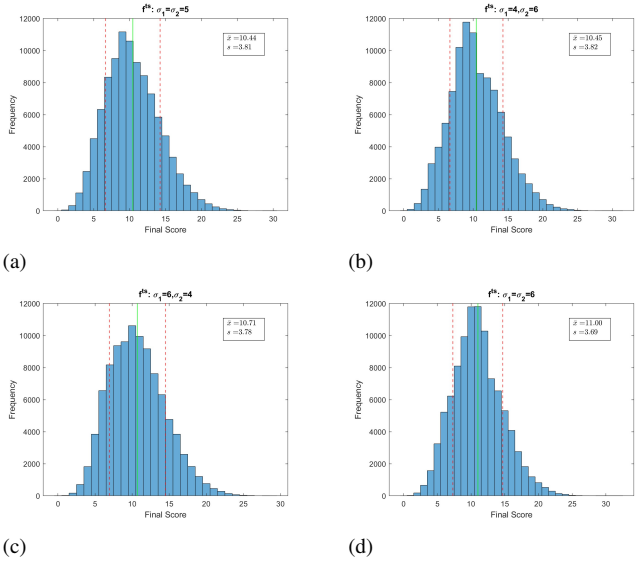

Fig. 4. Score distributions of 100000 trials for f^{ts} with a) $\sigma_1 = \sigma_2 = 5$, b) $\sigma_1 = 4, \sigma_2 = 6$, c) $\sigma_1 = 6, \sigma_2 = 4$, and d) $\sigma_1 = \sigma_2 = 6$

V. Analysis of Simulation Results

Examining the plots from the various simulations, we see that there are a few standouts that have means much lower than the others: $f^{dv}(;2,2)$, $f^{dv}(;1,2)$, $f^{ts}(;5,5)$, and $f^{ts}(;4,6)$. In our analysis, we aim to assess whether one of these algorithm/parameter combinations is inherently better than the others.

A. Normality of Distributions

While we know the data cannot actually be normally distributed, since the scores are discrete and limited to the range $[0, 36]$, let us examine their distributions' normal probability plots to determine whether we can assume normal distributions in data analysis.

In **Figure 5** we see that the distributions are not exactly normal but are fairly close. Given that we have known sources of deviation from normality (functions are discrete and bandlimited), it is reasonable to estimate the distributions as normal for analysis.

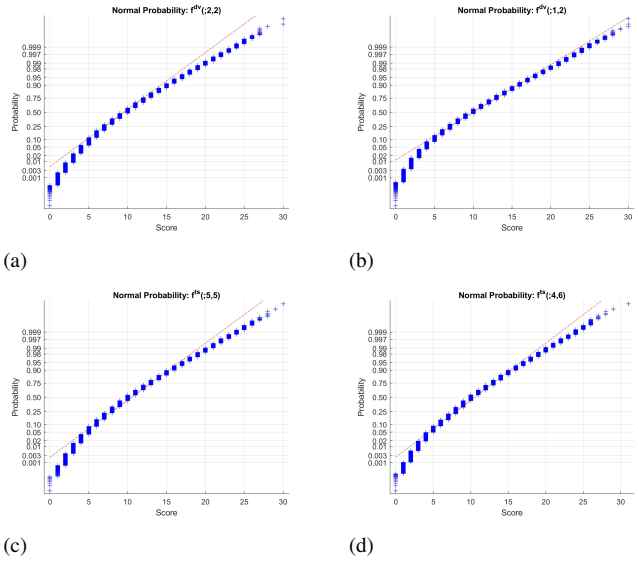

Fig. 5. Normal probability plots for a) $f^{dv}(;2,2)$, b) $f^{dv}(;1,2)$, c) $f^{ts}(;5,5)$, and d) $f^{ts}(;4,6)$

B. Comparing Means

In order to determine whether one of these strategies is better than the others we will compare the means with hypothesis tests. Since our sample size, 100,000, is much larger than the maximum extent of ν for most t-tables, we will assume that our sample distributions are equal to the population distributions. Therefore, we will use the statistic $z_{\alpha/2}$ instead of $t_{\alpha/2,\nu}$ in our hypothesis tests.

The hypothesis tests we hope to perform will take the following form:

$H_0 : \mu_1 - \mu_2 = 0$

$H_1 : \mu_1 - \mu_2 \neq 0$

For $z_0 = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(s_1^2+s_2^2)/n}}$, we find the z_0 values for each pair of sample means as shown in **Table I**.

TABLE I
Z-SCORES OF DIFFERENCES BETWEEN SAMPLE MEANS OF HIGH PERFORMING ALGORITHMS

$\bar{x}_1 \downarrow, \bar{x}_2 \rightarrow$	$f^{dv}(;,2,2)$	$f^{dv}(;,1,2)$	$f^{ts}(;,5,5)$	$f^{ts}(;,4,6)$
$f^{dv}(;,2,2)$	0	-7.77	-27.79	-28.21
$f^{dv}(;,1,2)$	7.77	0	-18.61	-19.02
$f^{ts}(;,5,5)$	27.79	18.61	0	-0.46
$f^{dv}(;,4,6)$	28.21	19.02	0.46	0

We test the hypothesis for each combination of $\bar{x}_1$ and $\bar{x}_2$ from the four highest performing functions at $\alpha = 0.001$, where $z_{\alpha/2} = 3.29$. Therefore, for all pairs of distinct means except $f^{ts}(;5,5)$ and $f^{ts}(;4,6)$, we strongly reject the null hypothesis in favor of the alternative hypothesis. This means we can conclude that $f^{dv}(;2,2)$ has the lowest mean of algorithms tested. We do not list a p-value here because all $|z_0|$ values are much larger than those included on any standard normal distribution table.

We may also construct a confidence interval for the true mean of $f^{dv}(;2,2)$. The 99.9% confidence interval takes the following form:

$$\bar{x} + z_{\alpha/2}\sigma/\sqrt{n} < \mu < \bar{x} + z_{\alpha/2}\sigma/\sqrt{n}$$
$$9.92 < \mu < 10.00$$

It is important to note that while we can show that there is a significant statistical difference between means since simulation allows for very large sample sizes, there may not be a practical difference. In a game where all scores are integers, a mean difference of $\mu_1 - \mu_2 \approx 0.2$ when the standard deviations are $\sigma \approx 4$ may have little to no effect on the number of wins a player achieves. This is investigated more deeply in the next section.

C. Comparing Wins

The bulk of the analysis thus far has focused on finding algorithms which produce the lowest means, because the mean is the best metric to measure the performance of a single algorithm's distribution. Single distribution analyses are fairly simple, since we have well-established techniques like confidence intervals and hypothesis tests on the mean.

However, in this game, the players' goals are not to achieve the best average score over a series of games, but to have the lowest score of all players in as many games as possible. It is possible that the algorithm that produces the lowest mean score is not necessarily the algorithm that beats other players at the highest rate. For instance, we would expect that for two distributions with the same mean score, an algorithm with a strong positive skew, or even just a larger standard deviation, would be able to beat an unskewed or more narrow distribution at a high rate.

We can easily evaluate this possibility by comparing head-to-head all 100,000 trials of multiple different algorithms and finding which wins the most rounds. For the four algorithms with lowest means, the win distribution is displayed in **Figure 6**. In order to use the data gathered in the initial set of simulations, we remove cases where there is a tie for first place rather than having the players in the tie play another round.

Indeed, we find in competition between the four lowest mean algorithms that $f^{dv}(;1,2)$ wins the most rounds even though it does not have the lowest mean score. However, when we let $f^{dv}(;1,2)$ and $f^{dv}(;2,2)$ compete one-on-one, we obtain the results shown in **Figure 7**.

This shows $f^{dv}(;2,2)$ winning slightly more games than $f^{dv}(;1,2)$ in a one-on-one competition. From these examples, we gather that an algorithm's actual performance may be correlated with having a low mean, but there are other factors that come into play, which require knowledge of other players' strategies to rigorously analyze.

VI. Conclusion

In this work, we propose, simulate, and analyze three basic algorithms for "Sixers". Simulation of a large number of trials is possible to achieve very quickly with the code base

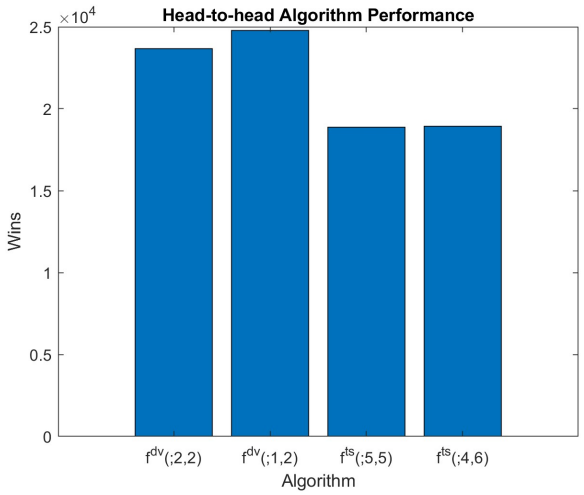

Fig. 6. Number of wins in a single round by each of the four lowest mean algorithms

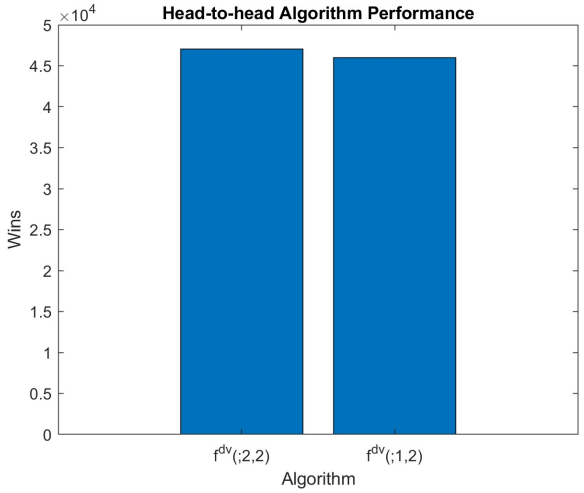

Fig. 7. Number of wins in a single round by the two lowest mean algorithms

developed, and its architecture allows other users to easily develop and test their own simple algorithms.

In this work we find that of all strategies tested, the best average score is achieved with $f^{dv}(;2,2)$, where we always keep any die that scores two or less. However, we validate experimentally that best average score is not the metric that matters most when competing with other players.

While future work could focus on performing a more exhaustive search of algorithms that aim only to achieve the lowest average score, we believe this is not the most useful path in order to achieve the optimal "Sixers" strategy. Instead, future work should focus on addressing information asymmetry, which arises from players taking turns rolling. An algorithm that wins the most games will likely take calculated risks based on three pieces of information: the current score to beat, the number of players that have yet to roll, and the likely behaviors of those players. Incorporating the score to beat into an algorithm is fairly simple and could likely be done with methods similar to those in this work. Incorporating the number of players that have yet to roll is more complicated, and would likely require more sophisticated probability theory. Finally, incorporating the likely behavior of other players based on knowledge of their past behavior is a much harder problem which likely requires artificial intelligence based approaches to system identification.

Another avenue for future work would be to analytically derive the score distributions rather than to experimentally estimate them. We showed that there are roughly 68 million possible trajectories, though most of these need not be considered, since they may include decisions to keep high scoring dice and reroll low scoring dice. Any good algorithm will not take a trajectory that makes decisions this poor, so we need not even consider these trajectories in a theoretical analysis. While it would be useful to validate experimental results with the true theoretical distribution, it may not be practical, since it would be a very tedious theoretical analysis and we can achieve near perfect convergence by simply using very large sample sizes.

Ultimately, we have shown that having a good strategy provides at best a slight advantage. Therefore, over any reasonable time scale of actually playing "Sixers", financial gains are much more likely to be dictated by variability in dice rolls than in strategy.

REFERENCES

[1] https://en.wikipedia.org/wiki/Multinomial_theorem
[2] Walden Marshall (2022). Sixers Dice Project (https://www.mathworks.com/matlabcentral/fileexchange/120133-sixers-dice-project), MATLAB Central File Exchange. Retrieved November 7, 2022.