Dart Project

Huang Sujie 2020211537

January 16, 2020

1 Background

In a dart game, a player pays no attention to the score of his opponent and is therefore non-strategic. It can be formulated as a stochastic shortest path DP and we refer to it as the non-strategic (NS) problem.

Here the goal is to minimize the number of turns where we recall that a turn consists of three throws. All three are used unless you win or go bust with an earlier throw in which case your score reverts to the score at the beginning of the turn. Working with turns is more accurate as this is how the game of darts is typically played in practice and the rules associated with turn have an important impact on the strategy of players.

When we work with turns the state space becomes more complicated and consists of (s,i,u) where s is the score at the beginning of the turn, $i \in 1,2,3$ is the number of throws remaining in the turn and u is the score-to-date within the turn so for example if i = 3 then u = 0. The state transition now satisfies $(s_{new}, i_{new}, u_{new}) = f((s,i,u),z)$ where

打到两倍得分结束游戏和打爆炸后 算新增加一轮 i都重置为3

$$f((s,i,u),z) := \begin{cases} (0,3,0), & s = u + h(z) \text{ and } z \text{ a double}, i \in \{1,2,3\} \\ (s,3,0), & s - (u + h(z)) \leq 1 \text{ and not } (s = u + h(z), z \text{ a double }) \\ (s, i-1, u + h(z)), & s > (u + h(z)) + 1, i > 1 \\ (s - u - h(z), 3, 0), & s > (u + h(z)) + 1, i = 1 \end{cases} \tag{1}$$

Note that the first two cases in (1) correspond to getting out and going bust, respectively. The third case corresponds to neither going bust nor getting out and still having at least one throw remaining in the turn. The final case corresponds to the last throw of a turn and neither getting out nor going bust. With these state dynamics the equation becomes

当一轮中 3次throw结束才结算score

$$\begin{aligned} V_{\text{NS}}(s,i,u) &= \min_{\boldsymbol{\mu}} \left\{ \text{E}\left[1_{\{i^{\text{new}}=3\}} + V_{\text{NS}}(f((s,i,u),z)) \mid (s,i,u), \boldsymbol{\mu} \right] \right\} \\ &= \min_{\boldsymbol{\mu}} \left\{ \sum_z \left[1_{\{i^{\text{new}}=3\}} + V_{\text{NS}}(f((s,i,u),z)) \right] p(z;\boldsymbol{\mu}) \right\} \end{aligned} \tag{2}$$

with $V_{NS}(0,i,u) := 0$ for any i,u. Note also that a turn is counted only when the turn has been completed. This explains why after winning in the first case of (1) we set $i_{new} = 3$ so that the final turn is counted in (2). As with the game ignoring turn feature, there is also a monotonic structure to the state dynamics and we can use this to successively solve for $V_{NS}(2,i,u), V_{NS}(3,i,u), \ldots, V_{NS}(501,i,u)$. Each of these 500 problems is more challenging than their analogs when ignoring turn feature because of the additional states here.

2 Bellman equations for the dart game with the turn feature

2.1 Bellman Equation

After the player throws a dart by targeting μ at state (s, i, u), the game may transit to the state of next throw (or next turn), reach 0, or bust depending on the score h achieved. The Bellman equations can be

reformulated as

$$
\begin{aligned}
V(s,i=3,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-2\}} p(h;\boldsymbol{\mu})\left[0+V(s,i=2,h)\right] + p_D(s/2;\boldsymbol{\mu})\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V(s,i=3,0)\right] \text{ for } u=0 \\
V(s,i=2,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu})\left[0+V(s,i=1,u+h)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V(s,i=3,0)\right] \text{ for } u=0,1,\ldots,\min\{60,s-2\} \\
V(s,i=1,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu})\left[1+V(s-u-h,i=3,0)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V(s,i=3,0)\right] \text{ for } u=0,1,\ldots,\max\{120,s-2\}
\end{aligned}
\tag{3}
$$

2.2 Value Iteration

$$
\begin{aligned}
V_{k+1}(s,i=3,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-2\}} p(h;\boldsymbol{\mu})\left[0+V_k(s,i=2,h)\right] + p_D(s/2;\boldsymbol{\mu})\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V_k(s,i=3,0)\right] \text{ for } u=0 \\
V_{k+1}(s,i=2,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu})\left[0+V_k(s,i=1,u+h)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V_k(s,i=3,0)\right] \text{ for } u=0,1,\ldots,\min\{60,s-2\} \\
V_{k+1}(s,i=1,u) = & \min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu})\left[1+V_k(s-u-h,i=3,0)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu})\left[1+V_k(s,i=3,0)\right] \text{ for } u=0,1,\ldots,\max\{120,s-2\}
\end{aligned}
\tag{4}
$$

2.3 Policy Iteration

- Solve the equation:

删掉！

$$
\begin{aligned}
V(s,i=3,u;\boldsymbol{\mu}_k) = & \textit{min} \sum_{h=0}^{\min\{60,s-2\}} p(h;\boldsymbol{\mu}_k)\left[0+V(s,i=2,h;\boldsymbol{\mu}_k)\right] + p_D(s/2;\boldsymbol{\mu}_k)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu}_k)\left[1+V(s,i=3,0;\boldsymbol{\mu}_k)\right] \text{ for } u=0 \\
V(s,i=2,u;\boldsymbol{\mu}_k) = & \textit{min} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu}_k)\left[0+V(s,i=1,u+h;\boldsymbol{\mu}_k)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}_k\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu}_k)\left[1+V(s,i=3,0;\boldsymbol{\mu}_k)\right] \text{ for } u=0,1,\ldots,\min\{60,s-2\} \\
V(s,i=1,u;\boldsymbol{\mu}_k) = & \textit{min} \sum_{h=0}^{\min\{60,s-u-2\}} p(h;\boldsymbol{\mu}_k)\left[1+V(s-u-h,i=3,h;\boldsymbol{\mu}_k)\right] + p_D\left(\frac{s-u}{2};\boldsymbol{\mu}_k\right)\left[1+V(0)\right] \\
& + p(\text{ bust };\boldsymbol{\mu}_k)\left[1+V(s,i=3,0;\boldsymbol{\mu}_k)\right] \text{ for } u=0,1,\ldots,\max\{120,s-2\}
\end{aligned}
\tag{5}
$$

- Update the optimal action

$$
\begin{aligned}
\boldsymbol{\mu}_{k+1}(s, i=3, u) &= \arg\min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60, s-2\}} p(h; \boldsymbol{\mu})\left[0 + V\left(s, i=2, h; \boldsymbol{\mu}_k\right)\right] + p_D(s/2; \boldsymbol{\mu})[1 + V(0)] \\
&\quad + p(\text{ bust }; \boldsymbol{\mu})\left[1 + V\left(s, i=3, 0; \boldsymbol{\mu}_k\right)\right] \text{ for } u = 0 \\
\boldsymbol{\mu}_{k+1}(s, i=2, u) &= \arg\min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60, s-u-2\}} p(h; \boldsymbol{\mu})\left[0 + V\left(s, i=1, h; \boldsymbol{\mu}_k\right)\right] + p_D(\frac{s-u}{2}; \boldsymbol{\mu})[1 + V(0)] \\
&\quad + p(\text{ bust }; \boldsymbol{\mu})\left[1 + V\left(s, i=3, 0; \boldsymbol{\mu}_k\right)\right] \text{ for } u = 0, 1, \ldots, \min\{60, s-u-2\} \\
\boldsymbol{\mu}_{k+1}(s, i=1, u) &= \arg\min_{\boldsymbol{\mu}} \sum_{h=0}^{\min\{60, s-u-2\}} p(h; \boldsymbol{\mu})\left[1 + V\left(s, i=3, h; \boldsymbol{\mu}_k\right)\right] + p_D(\frac{s-u}{2}; \boldsymbol{\mu})[1 + V(0)] \\
&\quad + p(\text{ bust }; \boldsymbol{\mu})\left[1 + V\left(s, i=3, 0; \boldsymbol{\mu}_k\right)\right] \text{ for } u = 0, 1, \ldots, \max\{120, s-2\}
\end{aligned} \tag{6}
$$

In practice, I utilize the package **SymPy** to write the equations explicitly. Then I use function **linalg.solve()** in **Numpy** to solve the equations.

3 Interesting behaviors in the game

3.1 why the expected turns/throws is not monotonically increasing in s?

Because the rule to end the game makes us prefer multiples of 2, especially multiples of 4.

- Ignoring the turn Feature.

总体趋势：单调

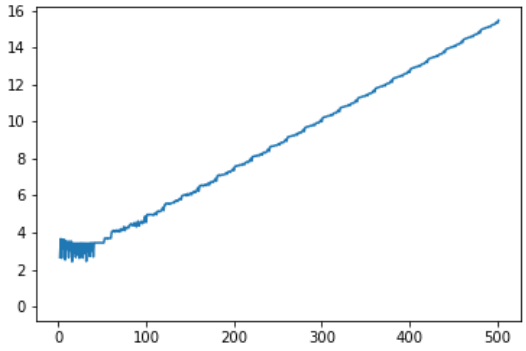
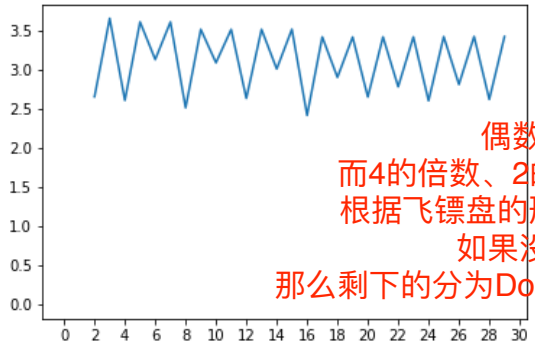


Figure 1: Trend of Optimal Values

As is shown in the graph, the expected throws or turns is generally increasing in score while the situation around 0 is complex. Then we focus on the expected throws near 0.

在40以下可以通过一步结束游戏
单调性变得不一样了

对于偶数：有概率直接结束游戏
对于奇数，至少要先throw一轮到偶数 才能结束游戏



偶数能让我们尽快结束游戏
而4的倍数、2的幂次给了我们更多机会得到偶数
根据飞镖盘的形状，当我们涉及Double x的时候
如果没射中大概率射到Single x
那么剩下的分为Double x - Single x = Single x仍然为偶数

Figure 2: Trend of Optimal Values Near Zero

The graph proves the advantage of even score especially multiples of 4. Notably, the expected throws of 2, 4, 8, 12 and 16 are at troughs of the line while 3, 5, 7, 9 and 11 are at the peaks of the line. The phenomenon is caused by the ending rule which requires that a player's final dart must be a double. As a result, we prefer even scores rather than odd ones. The optimal value of multiples of 4 is relative small because $4 = 2 * 2$ when we miss Double area for one time, we still have another chance to finish the game by hitting Double area by one more throw. For example, score = 16 is better than score = 14 or 18, because if we fail to hit Double 8, it is of great possibility that we hit Single 8, and then our score becomes $16 - 8 = 8$ which is also even. But as for score 14, if we fail to hit Double 7, it is likely that we hit Single 7 and our remaining score is $14 - 7 = 7$ which is odd, so the expected number of throws at score 14 is a little more than that of score 16. This may serve to explain why the expected turns or throws is not monotonically increasing in score.

- Considering the turn Feature.

  In the case of considering the turn feature, we get similar conclusions that the general trend of expected turns appears to be increasing. But when scores are near zero, multiples of 4 are preferred because their next states are more likely to be double which allows us to finish the game by one throw to some extent.

turns 在一般情况下 = 3*throws
turns和throws成正比 所以turns的规律也相似

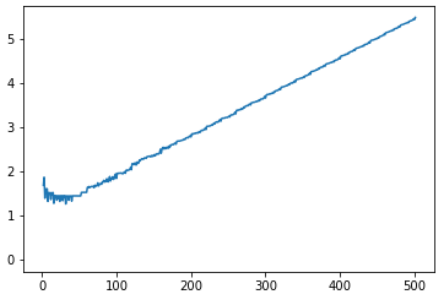


Figure 3: Trend of Optimal Values

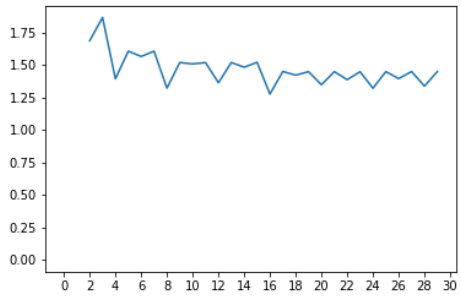


Figure 4: Trend of Optimal Values Near Zero

3.2 How is the improvement for using the full aiming location set?

We analyze this problem in the background of the game ignoring turn feature by comparing the optimal values when using different aiming location sets. And we show the differences of optimal values at score state 2-31(close the end of the game) and 472-501(at the very beginning of the game).

在不考虑turn的情况下 讨论这个问题
考虑turn的时候 结论类似，因为turn和throws成正比

影响程度：
2*x(x为奇数)>2*x(x为偶数)>x(x为奇数)

表格展示full set和part set下optimal value的差

Score state	Full set	Part set	Difference	Score state	Full set	Part set	Difference
2	2.6631	2.6631	0.00000	472	14.5826	14.5873	0.00469
3	3.6673	3.6673	0.00000	473	14.6549	14.6604	0.00553
4	2.6169	2.6169	0.00000	474	14.6902	14.6967	0.00654
5	3.6209	3.6210	0.00004	475	14.7022	14.7095	0.00727
6	3.1248	3.1399	**0.01514**	476	14.7023	14.7083	0.00605
7	3.6207	3.6207	0.00000	477	14.7306	14.7379	0.00729
8	2.5216	2.5216	0.00000	478	14.7968	14.8044	0.00760
9	3.5259	3.5259	0.00000	479	14.8502	14.8571	0.00683
10	3.0813	3.0988	**0.01754**	480	14.7843	14.7932	0.00890
11	3.5252	3.5253	0.00001	481	14.9355	14.9413	0.00579
12	2.6283	2.6424	**0.01403**	482	14.9697	14.9779	0.00825
13	3.5259	3.5259	0.00000	483	14.9906	14.9965	0.00590
14	2.9979	3.0190	**0.02113**	484	15.0147	15.0225	0.00782
15	3.5272	3.5272	0.00000	485	15.0088	15.0192	0.01040
16	2.4248	2.4248	0.00000	486	15.0478	15.0517	0.00394
17	3.4290	3.4290	0.00000	487	15.0726	15.0790	0.00645
18	2.8866	2.9117	**0.02512**	488	15.0772	15.0813	0.00407
19	3.4270	3.4275	0.00045	489	15.0798	15.0834	0.00359
20	2.6436	2.6603	**0.01672**	490	15.1264	15.1317	0.00528
21	3.4291	3.4291	0.00001	491	15.1419	15.1469	0.00508
22	2.7674	2.7911	**0.02367**	492	15.1146	15.1193	0.00473
23	3.4277	3.4290	0.00133	493	15.1851	15.1907	0.00552
24	2.6044	2.6115	**0.00707**	494	15.2204	15.2269	0.00649
25	3.4326	3.4326	0.00000	495	15.2321	15.2394	0.00724
26	2.7925	2.8205	**0.02796**	496	15.2341	15.2401	0.00604
27	3.4347	3.4347	0.00000	497	15.2616	15.2689	0.00725
28	2.6137	2.6294	**0.01566**	498	15.3269	15.3344	0.00756
29	3.4337	3.4337	0.00001	499	15.3797	15.3866	0.00687
30	2.8873	2.9123	**0.02492**	500	15.3167	15.3255	0.00886
31	3.4319	3.4319	0.00000	501	15.4631	15.4689	0.00581

在分数较大的时候对奇偶影响差不多

Table 1: Optimal Value and Improvement

1.2x(x为奇数)时：如果打中Double x则1个throw结束
如果没有打中Double x则打到single x，此时所剩分数为奇数，
则必须先打一个奇数的分数->增加1 throw
在到偶数分数争取打中double 区域->增加1 throw
至少增加2 throws

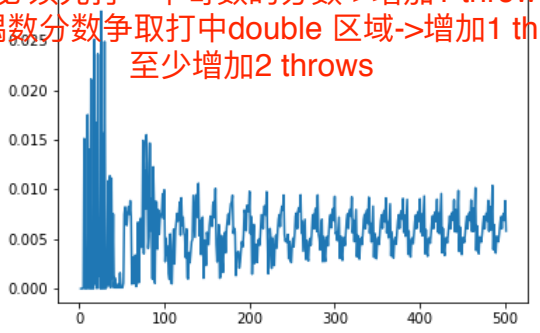


Figure 5: Improvement (Global View)

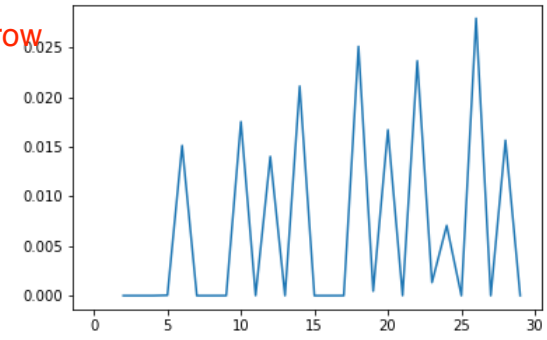


Figure 6: Improvement (Local View)

According to the table and graph, we have those findings.

- Optimal values are smaller when using full aiming location set($341 * 341$).

2.2x(x为偶数)时：
如果没打中double x则至少要多打一轮

3.x为奇数是，瞄准single/triple area打。就算没有打中可能也不会增加轮数

- As for even scores, the differences between results of the full aiming location set and part aiming location set are apparent in low-score states. When approaching the end, the choice of every step is very important in even-score status because it is likely that we hit the double and finish the game in one step.

- As for odd scores, the differences between results of the full aiming location set and part aiming location set are apparent in high-score states. The reason behind this can be cumulative effect. In every step, full aiming location set offer more choices and the best choice in the full aiming location set is not worse than the best choice in the part of aiming location set since the part of aiming location set is a subset of full aiming location set. And high-score states indicate that there are more steps to go, so the improvement in high-score states are larger than that in low-score states.

- Among low-score states, using full aiming location set leads to a great improvement when scores are multiples of 2, especially when the scores can be decomposed into the multiplication of one odd number and 2 e.g. 6, 10,14, 18, 22, 26. It is probably due to we include more points from Double x(x is an odd) area and those points increase the probabilities to finish the game at the 2*x score state. Enhancement in the probability of hitting this area is great because the difference in number of throws when we hit the double x and when we hit single x is not less than 2. For example, when the remaining score is 6, if we hit Double 3, we finish the game in one throw. But if we fail to hit Double 3, we will hit the Single 3. Then we are at 3 score, we first have to hit Single 1 and then have to hit Double 1. Therefore the difference in the expected number of throws of hitting Double 3 and hitting Single 3 when we are at score 6 is not less than 2. But in other case, the influence of enhancement in shooting accuracy is smaller.

3.3 How to make the computation more efficient/fast?

初始值两个角度选取：
1.全都赋一样的值
2.根据no turn时s-u的optimal value/alpha来赋值

3.3.1 How to choose proper initial values or policies?

Initial values represent the start point of iteration. If the initial values are close to the true values, the solutions converges quickly and we can get close to true values in a short time while it may take more than an hour if we don't choose proper initial values. In this part, we will discuss how different initial values may influence the efficiency of iteration.

- In the first case, the initial values of different states $V(s, i, u)$ are all the same. We use integers form 0 to 10 as the initial value for each state. Then we do numerical simulations for each initial value for 5 times and document the running time of each simulation. The average running time of the 5 simulations represents the efficiency of the iteration and suitability of the initial values. We simulate on the action set of 984 aiming locations. The results can be shown as below.

在984的aiming set上做value iteration数值模拟

Initial Value	Simulation1	Simulation2	Simulation3	Simulation4	Simulation5	Avg Running Time
0	60.19	63.6	61.55	61.65	63.45	62.088
1	58.3	58.93	58.65	62.14	55.07	58.618
2	51.35	49.94	52.66	50.87	55.8	52.124
3	49.03	47.87	47.41	50.36	48.26	48.586
4	45.79	45.92	45.25	46.46	46.95	46.074
5	45.35	45.62	46.08	44.74	44.64	45.286
6	46.3	48.24	45.82	45.88	45.44	46.336
7	46.66	45.49	46.04	42.5	47.2	45.578
8	45.64	45.01	47.8	48.65	45.77	46.574
9	46.75	48.78	45.93	46.03	46.94	46.886

Table 2: Running Time(s) of Each Simulation

运行时间衡量效率

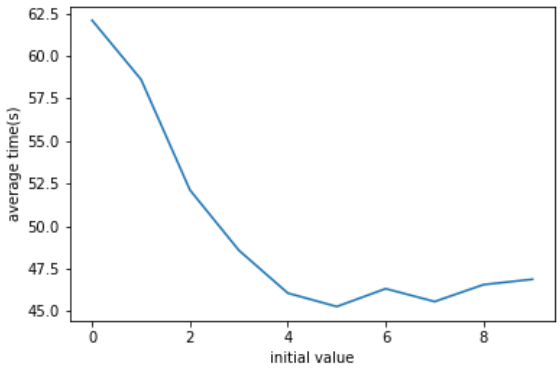


Figure 7: Trend of Optimal Values Near Zero

We found that this initialization method with my intuition is not bad and find some good initial values such as 4 and 5. The mean of true optimal values is 3.7881 and the median is 3.7291, therefore numbers near 3.8 can be good initial values.

- Next, we take the result of problem with no turn into consideration. There are at most 3 chances to throw in each turn. At most time, a turn consists of 3 throws. However, when players throw to the bust area or hit the double to finish the game, the times of throws can be less then 3. Accordingly, we estimate the true value of state $V(s,i,u)$ in the game considering turns has relations with optimal value $V^{'}(s-u)$ in the game with no turns. We describe the relationship as follows:

$$V^{'}(s-u) \leq V(s,i,u) \leq 3V^{'}(s-u).$$

And we believed that $V(s,i,u)$ is somehow positively relative to $V^{'}(s-u)$. This relationship leads us to initialize $V(s,i,u)$ by this way.

$$V(s,i,u) = \frac{1}{\alpha}V^{'}(s-u).$$

We set $\alpha = 1, 1.5, 2, 2.5, 3, 3.5, 4$ respectively and repeat the simulations for each α and show the results below.

α	Simulation1	Simulation2	Simulation3	Simulation4	Simulation5	Avg Running Time
1.5	42.73	41.12	43.19	41.96	40.55	41.910
2	40.07	40.81	41.59	40.91	42.48	41.172
2.5	39.91	40	40.34	41.08	40.3	40.326
3	40.12	41.36	39.96	40.11	40.59	40.428
3.5	43.21	44.96	43.75	40.58	44.03	43.306
4	46.51	43.62	44.31	48.44	48.17	46.210
4.5	43.07	49.6	43.71	43.49	41.39	44.252

Table 3: Running Time(s) of Each Simulation

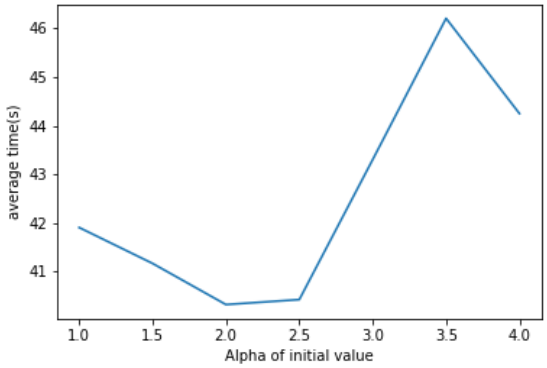


Figure 8: Trend of Optimal Values Near Zero

The graph of average running time versus α reflects that initializing $V(s,i,u)$ by $\frac{1}{2}V^{'}(s-u)$ reach the highest efficiency. The average running time of this method when $\alpha = 2$ is 40s. That is to say, utilizing optimal value in games ignoring the " Turn" feature in initialization outperforms simply giving all the state the same initial value.

Further, we explore why $\alpha \in [2, 2.5]$ have higher efficiency. The mean of optimal values(expected number of throws) at different states in the game ignoring turn feature is 8.8299, and the mean of optimal values(number of turns) at different states in the game considering turn feature is 2.3309.

$$Mean(V_{noturn}(s))/Mean(V_{turn}(s)) = 8.8299/3.7882 = 2.3309$$

Therefore it's appropriate to initialize $V_{turn}(s,i,u)$ by $\frac{1}{2.3309}V^{'}_{noturn}(s-u)$

3.3.2 How to implement the iteration computation in a better way in code?

- Use package **NumPy** to do matrix computing. 矩阵运算

  The NumPy vectorization, indexing, and matrix computing are powerful and allow us to solve the problem in a fast and versatile way.

- When using policy iteration method in the game ignoring turn feature, we calculate the probabilities of not bust for each score state in advance so that they will not be calculated repeatedly in each iteration.

- Consider the monotonic structure and solve the problem successively.

  At first, I ignore the monotonic structure and update the optimal values of all $V(s)$ or $V(s,i,u)$ in each iteration. It is very time-consuming. As for the game ignoring the turn feature, it takes 2 minutes to approaches the optimal values when ignoring the monotonic structure while 17 seconds are enough when take the monotonic structure and solve the problem successively.

  – Ignoring the 'Turn' Feature

单调结构

对于没有turn的情况：V(s,i)s关于i单调递减或者不变

计算当前状态的转移状态只会用到更小的s

$$\begin{aligned} V_{\mathrm{NS}}(s_i, i) &= \min_{\boldsymbol{\mu}_{i+1}} \mathbb{E}\left[V_{\mathrm{NS}}(s_{i+1}, i+1) \mid (s_i, i), \boldsymbol{\mu}_{i+1}\right] \\ &= \min_{\boldsymbol{\mu}_{i+1}} \sum_z V_{\mathrm{NS}}\left(f(s_i, z), i+1\right) p\left(z; \boldsymbol{\mu}_{i+1}\right) \end{aligned} \tag{7}$$

$$s_{i+1} = f(s_i, z) := \begin{cases} s_i - h(z), & s_i - h(z) > 1 \\ 0, & h(z) = s_i \text{ and } z \text{ a double} \\ s_i, & \text{otherwise} \end{cases} \tag{8}$$

对于有turn的情况

V(s,i,u) 的下一轮状态对应的s比这一轮的s小

计算当前这个V只会用到更小的s

the state s_i is non-increasing in i so that $s_{i+1} \leq s_i$ for all i. We can take advantage of this by redefining $V_{NS}(s)$ to be the minimum expected number of throws required to reach 0 starting from state $s \geq 2$. Then we have

$$\begin{aligned} V_{\mathrm{NS}}(s) &= \min_{\boldsymbol{\mu}} \left\{1 + \mathrm{E}\left[V_{\mathrm{NS}}\left(s^{\mathrm{new}}\right) \mid s, \boldsymbol{\mu}\right]\right\} \\ &= \min_{\boldsymbol{\mu}} \left\{1 + \sum_{z} V_{\mathrm{NS}}(f(s,z)) p(z;\boldsymbol{\mu})\right\} \end{aligned} \tag{9}$$

with $V_{NS}(0) := 0$. We can solve (7) successively for $s = 2, \ldots, 501$.

– Considering the 'Turn' Feature.

f(s,i,u)<s

$$\begin{aligned} V_{\mathrm{NS}}(s,i,u) &= \min_{\boldsymbol{\mu}} \left\{\mathrm{E}\left[1_{\{i^{\mathrm{new}}=3\}} + V_{\mathrm{NS}}(f((s,i,u),z)) \mid (s,i,u), \boldsymbol{\mu}\right]\right\} \\ &= \min_{\boldsymbol{\mu}} \left\{\sum_{z} \left[1_{\{i^{\mathrm{new}}=3\}} + V_{\mathrm{NS}}(f((s,i,u),z))\right] p(z;\boldsymbol{\mu})\right\} \end{aligned} \tag{10}$$

As with (9) there is also a monotonic structure to the state dynamics and we can use this to successively solve for $V_{NS}(2,i,u), V_{NS}(3,i,u), \ldots, V_{NS}(501,i,u)$.