gops_writeup

July 18, 2021

1 A Breif Computational Analyisis of Goofspiel Strategy

Nick Holden

Goofspiel, aka the *Game of Pure Strategy* is a zero-sum card game between two players with symmetric information. The game was invented by mathematician Meril Flood in the 1930s. This game is an example of repeated simultaneous decision-making on imperfect information.

The gameplay is rather simple. A standard deck of 52 cards is divided into four suits. One suit is given to each player, one is discarded, and the last is shuffled randomly and placed face down between the two players, this suit serves as the draw pile. At the start of each turn, a "value" card ($A = 1, 2 = 2, \dots Q = 12, K = 13$) is flipped face-up. Then each player simultaneously "bids" on the value card by placing a card from their hand face down, then both players reveal their cards. Whichever player plays the higher card claims the value card, and both bid cards are discarded. After the conclusion of the 13th round, the value cards claimed by each player are summed up and whichever player with the highest score wins.

Many strategies leverage random, deterministic, or learning strategies.

1.1 Basic Strategies

1.1.1 Random Strategy

Random play is exactly what it sounds like. A player running a random strategy will play their bid cards randomly, with no regard for the value card at play.

1.1.2 Deterministic Strategy

A deterministic strategy uses a static set of rules for play. A player running a basic deterministic strategy will always bid the same card in response to a certain value card. One common deterministic strategy is the matching strategy, in which a player will simply match the value card each time they bid. For example, if the current upturned value card is a 7, a player running a matching strategy will bid a 7. Other common deterministic strategies are varients on the matching strategy. One example is the value card + n strategy, in which a player will play a card n higher than a value card. If the value card + n is greater than 13, the player running upcard + n will then throw off their lowest card. (Note the matching strategy is equivalent to upcard + 0).

Deterministic strategies are very easy to play against. If you know your opponent is running the matching strategy, you can simply play the upcard + 1 strategy to win every card except the king. Giving a final score of 78 to 13. Two players running deterministic strategies are never in Nash Equilibrium. If both players are aware the other is running a deterministic strategy, each player will continue up-bidding their strategy until they eventually realize they can improve their results by ducking certain tricks. However, if both players realize this, the game eventually devolves into random play. Each player trying to get the edge on the other by playing unpredictably.

The matching strategy has been proven (Ross, Sheldon M. (September 1971). *Goofspiel – The Game of Pure Strategy*) to be the optimal strategy into random play. In general, upcard + n strategies perform better into random play the closer n is to 0.

1.1.3 Learning Strategies

Learning strategies observe their opponent's play, and each play is determined by the opponent's previous play. Naturally, we observe a strong performance by learning strategies into simple deterministic strategies, they are consistently able to predict the opponent's play, and up-bid their opponent by exactly one card, and thus have very high win rates into deterministic strategies. Learning strategies perform less admirably against random strategies — however — as there is nothing to learn from the opponent's play.

1.2 Empirical Analysis

We compare the performance of these strategies to one another by observing their win rates. We can compare two strategies' relative performances by either looking at their average scores, or their win rates. Since I am more interested in winning and losing, I will look at win rates.

```
[84]: strats = ['Matching Strategy','Upcard + 1','Upcard + 4',"Upcard +␣
       ↪7",'Learning','Random']
      strat_functions = [det_0,det_1,det_4,det_7,learning,random_strat]
      df = winrate_table(strats,strat_functions)
      df
```

```
[84]:                      Matching Strategy  Upcard + 1  Upcard + 4  Upcard + 7  \
      Matching Strategy                0.500       0.000      1.0000       1.000
      Upcard + 1                       1.000       0.500      0.0000       1.000
      Upcard + 4                       0.000       1.000      0.5000       0.040
      Upcard + 7                       0.000       0.000      0.9680       0.500
      Learning                         1.000       1.000      0.9565       0.980
      Random                           0.042       0.128      0.6990       0.849

                         Learning  Random
      Matching Strategy    0.0000  0.9655
      Upcard + 1           0.0020  0.8795
      Upcard + 4           0.0340  0.2475
      Upcard + 7           0.0235  0.1105
      Learning             0.5000  0.6500
      Random               0.3490  0.5060
```

We see that the matching strategy indeed performs well into random play. Additionally, we see that as n increases, the upcard + n strategy performs worse and worse against the random strategy. We also see the learning strategy crushes simple deterministic strategies, but loses most of its edge into random play.

We now try to develop a collection of more advanced strategies and compare their performance to matching, upcard + 1, learning, and random play.

1.2.1 Deterministic Random

This strategy is a hybrid of random play and deterministic play. On each turn, this strategy will either play upcard + 2, matching or duck the trick with equal probability. The goal of the randomization is so the strategy still performs reasonably well into random play, while also making it harder for learning strategies to counter it.

[85]:
```
strategy = ['Deterministic Random']
function = [determ_random]
df = base_winrate_table(strategy,function)
df
```

[85]:
```
                          Matching  Upcard + 1  Basic Learning  Random
Deterministic Random         0.472       0.422           0.416   0.713
```

We see this strategy performs much better into learning than other deterministic strategies (40% winrate vs ~1% winrate). This is at the direct cost of being worse into other deterministic strategies, and a mediocre winrate into random play.

1.2.2 Robust Learning

Our basic learning strategy works as follows. This strategy will keep track of what the villain played on the previous upcard n. Let's say the villain played k, then our strategy will consider the residual k - n. Then on the next turn, the basic learning strategy will play upcard + (k - n + 1).

We observed high win rates by basic learning into simple upcard + n strategies, but weaker performance into random play. We attempt to generate a robust learning strategy that maintains a high win rate into upcard + n strategies but is less sensitive to the noise of random play. Our proposed robust learning strategy will determine the average residual but throw out outliers (ie residuals with absolute value > 3), and then play upcard + residual + 1. I expect similar performance into simple upcard + n strategies and better performance into random play.

[96]:
```
strategy = ["Basic Learning",'Robust Learning']
function = [learning,robust_learning]
df = base_winrate_table(strategy,function)
df
```

[96]:
```
                  Matching  Upcard + 1  Basic Learning  Random
Basic Learning       1.000       1.000          0.5000  0.6405
Robust Learning      0.997       0.998          0.3455  0.7205
```

We see very similar win rates into the matching and upcard + 1 strategies, and an 8% jump in win rate into random play. Since this strategy looks at the mean of residuals, its expected residual is 0. So after the first few cards are played, robust learning, on average, will converge to upcard + 1. This is why we see a higher win rate into random play, but this win rate is lower than upcard + 1 due to the wasted early turns and since this strategy is not guaranteed to converge in time.

This strategy is also rather consistent, hence the high win rate of basic learning into robust learning.

1.2.3 Correlation Strategy

This strategy assumes that the villain is employing one of two strategies: random or deterministic. This strategy attempts to quickly determine which of the two the villain is running, and counter it. If the villain is employing random play, then we run the matching strategy, if the villain is deterministic, we run basic learning.

The core of the strategy is detecting random play. Generally, deterministic strategies will bid higher cards in response to higher-value cards. So this strategy will run a Pearson correlation on the villain's cards played and the corresponding value cards. If they are correlated ($\rho > 0.2$) then we assume the strategy is deterministic and employ basic learning. Otherwise, we assume the strategy is random and employ the matching strategy.

[97]:
```
strategy = ["Correlation Strategy"]
function = [pearson_strategy]
df = base_winrate_table(strategy,function)
df
```

[97]:
```
                      Matching  Upcard + 1  Basic Learning  Random
Correlation Strategy       1.0       0.912          0.3315   0.807
```

We see this strategy is very good at detecting the matching strategy, and has a little difficulty detecting the upcard + 1 strategy. This is likley because the upcard + 1 strategy ducks the highest value trick, whcih throws off the correlation. If I were to go back and further improve this strategy, I would make the correlations more robust by throwing out large outliers in attempt to prevent planned ducks from hiding deterministic play.

1.2.4 Learning Counter

Many of the above strategies perform quite poorly into basic learning. This is not because basic learning is a good strategy per-say because any strategy in Goofspeil has a counter. Since we know the mechanics of basic learning, we can easily generate a strategy that counters those mechanics. this strategy is not particularly interesting or intellegent, it will simply predict what card the learning strategy will play, and play one higher.

[98]:
```
strategy = ["Learning Counter"]
function = [learning_counter]
df = base_winrate_table(strategy,function)
df
```

[98]:
```
                  Matching  Upcard + 1  Basic Learning  Random
Learning Counter     0.152      0.4965            0.92  0.5135
```

1.2.5 Hybrid Strategy

We now have strategies that perform well into each of the 4 basic strategies. We can now synthisize them into a hybrid strategy that detects the opponents play, and then plays the optimal strategy against it. We already have a way of distinguishing between random and determinisitc play, now we just have to detect a learning strategy. The hybrid strategy determines learning play by examining the previous turns, and examines how the opponent's play changes based on the hero's play. If it seems like the opponent's play is dependent on the hero's play, then the hybrid strategy will run the learning counter strategy.

[99]:
```
strategy = ["Hybrid"]
function = [hybrid]
df = base_winrate_table(strategy,function)
df
```

[99]:
```
        Matching  Upcard + 1  Basic Learning  Random
Hybrid    0.9995       0.858           0.852   0.815
```

This strategy out performs the basic strategies across the board. But lets see how our more advanced strategies perform into eachother

[100]:
```
strats =["Matching", "Basic Learning","Deterministic Random", "Robust␣
 ↪Learning", "Correlation","Learning Counter","Hybrid"]
strat_functions = [det_0,learning,determ_random,␣
 ↪robust_learning,pearson_strategy,learning_counter,hybrid]
df = winrate_table(strats,strat_functions)
df
```

[100]:
```
                      Matching  Basic Learning  Deterministic Random  \
Matching                0.5000          0.0000                0.5305
Basic Learning          1.0000          0.5000                0.5630
Deterministic Random    0.4595          0.4305                0.5140
Robust Learning         1.0000          0.3330                0.4725
Correlation             1.0000          0.3140                0.6340
Learning Counter        0.1610          0.9190                0.6060
Hybrid                  0.9990          0.8440                0.6280

                      Robust Learning  Correlation  Learning Counter  Hybrid
Matching                       0.0010       0.0000            0.8200  0.0020
Basic Learning                 0.6970       0.6675            0.0875  0.1405
Deterministic Random           0.5065       0.3735            0.3575  0.3880
Robust Learning                0.5195       0.5050            0.2580  0.4370
Correlation                    0.4965       0.5000            0.5495  0.5265
Learning Counter               0.7710       0.4535            0.5000  0.4680
Hybrid                         0.5565       0.4935            0.5835  0.5000
```

No strategy on this table has a >50 win rate against all other strategies. Each strategy has its set of strategies it dominates and a set of strategies that dominate it. This is because Goofspiel has no Nash equilibrium.

However, the study of different strategies and their pros and cons is still worthwhile. In the real world, we are often faced with complex decision problems where we have to bid on finite resources.

Like in goofspiel, we have to not let our strategy become too predictable, or else we may be taken advantage of.

[ ]:
```

```