Professor Hartline
CS 332
11 Jan 2022

Bid Analysis Report

Introduction

In this report, we seek to complete analyses regarding optimality in maximizing utility for a *Discrete First Price Auction.* The exact parameters for the auction are described in the **Preliminaries** section. First Price Auctions are a subject of research, and it provides a good opportunity to learn about analyses of optimality with respect to game theory, algorithms, and computation. Our goals were twofold.

First, we analyzed the data collected. We did so by implementing a Monte Carlo simulation, and by doing an exact calculation given the rest of the data. We examined the optimality of our own bids, some ideal bids, and then compared them. In sum, we found a pattern of optimal bids depending on value, namely that the optimal bid was about half the value and tended to be close to the lower and nearest value from the distribution. In comparison, our bids were not half the respective value, but did stay near to the values from the distribution.

Second, we used what we learned from our analysis to devise a bidding algorithm. Because we are in an "Online" Markets class, we chose to view this through the lens that we are receiving samples of bid data at different times, and using the current data to make predictions. This algorithm performed much better than our original strategy, and worse than the optimal strategy given all the data, as expected.

Preliminaries

The auction setup is described below as it is given in the assignment.

You are bidding in a first-price auction.

- *Everyone in the class draws a value uniformly from {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.*
- *You bid against one other random student in the class (your value and their value drawn randomly as above).*
- *Your utility is your value minus your bid if you win and zero otherwise.*
- *(Added later) Ties are decided by coin flip.*

All participants entered their bids for each given value without knowing the bids of the other participants (barring one other participant with whom each participant was allowed to discuss). This data was collected and distributed, and analyses were conducted on it. The responses were anonymized in the data.

The first price auction mechanism which we evaluate in this report is similar to a second price auction in many ways, but unlike the second price auction, it is not clear that the dominant strategy is to bid your value. Although there exists a lot of research of dominant strategies and equilibria in these types of auctions, because it is less clear, it gives us the opportunity to investigate and learn from the mechanism, especially because of the real-world data which was collected.

Professor Hartline
CS 332
11 Jan 2022

Prior to evaluation, we hypothesized that bidding the value - 9.999 would be effective in maximizing utility.

To evaluate this hypothesis and the bids, one method employed was the Monte Carlo method, where we simulate the auction many times. We implemented this in Python (see Appendix A1). In essence, we draw a random bidder and random value for this bidder, and use the data to collect this random bidder's bid. We then simulate the auction according to the rules stated above, with this random (competing bidder) versus either our own bids, or some arbitrary bid to test for optimality in maximizing utility. In another method, we implemented exact calculation to find expected utility (Appendix A2).

Devising an Algorithm (see the Python algorithm at Appendix A3.)

After evaluating the bids, our goal was to create an algorithm that would take in data, and calculate optimal bids based on the information it gets from the marketplace. We also made our algorithm "online," meaning that the algorithm takes in data bid-by-bid, and updates its optimal bids as more and more data comes in.

As mentioned in **Conclusions**, $v/2$ where v is the value of the bidder. Due to this, we found that starting with a bid of $v/2$ for all of the possible values was a good initial value for our algorithm. However, based on our **Results**, we still wanted to fit the algorithm as new data came in, as we did not assume that every bidder would also bid optimally.

The algorithm starts off by taking in new data (2d array of bids), previous optimal bids, and a size argument. We then use the exact calculation method to find the optimal bids given the data we just received. After finding the optimal bids for each value given the current data, we then weigh our current optimal bid vs. the previous optimal bids based on how much data we are receiving, and how much data we have already received.

The weights are calculated using a proportion of the size of the incoming data to the amount of data already seen. After calculating the weights, we used a linear combination of the previous optimal bids and current optimal bids to find our new optimal bids. We then returned this and the new size argument, which was incremented by the length of the data parameter, so that the algorithm is ready for new online data.

Results

I. Analysis of Bid Data

Since we participated in the auction, we evaluated our own bids. To evaluate these bids, we implemented a Monte Carlo simulation and exact calculation, which are Appendices A1 and A2 respectively. From the Monte Carlo simulation, which we ran 100,000 times, our bids performed as notated in Figure R1.1.

Figure R1.1, Our bids in a Monte Carlo simulation, see Appendix A1

Values	10	20	30	40	50	60	70	80	90	100
Bids	0.0001	10.00	20.00	30.00	40.00	50.00	60.00	70.00	80.00	90.00

		01	01	01	01	01	01	01	01	01
Win Rate	0.0039	0.16	0.29	0.42	0.55	0.67	0.79	0.86	0.91	0.96
Average Utility	0.039	1.588	2.898	4.154	5.415	6.617	7.970	8.566	9.126	9.625

The exact calculation was computed by calculating the number of wins and expected utility against all other bids in the dataset for each of the value-bid pairs in our own bids. This data is collected in figure R1.2.

Figure R1.2, Our bids' performance by exact calculation, see Appendix A2

Values	10	20	30	40	50	60	70	80	90	100
Bids	0.0001	10.0001	20.0001	30.0001	40.0001	50.0001	60.0001	70.0001	80.0001	90.0001
Win Rate	0.0037	0.16	0.29	0.42	0.55	0.66	0.79	0.85	0.91	0.96
Expected Utility	0.037	1.57	2.89	4.16	5.45	6.65	7.94	8.55	9.11	9.62

Average expected utility over all values for our bids: 5.60

Notably, although no statistical significance test was completed, these results are very similar, which verifies the legitimacy of a Monte Carlo simulation given a large number of random samples.

The exact error for a Monte Carlo simulation can be calculated using:

$N > \frac{\ln 2\alpha^{-1}}{2\epsilon^2}$ gives error of $\pm\ \epsilon h\ w.p.\ 1 - \alpha$

Where the functions we approximate, the expected utility and win probability have a domain of [0, h], N gives the number of simulations, and alpha gives a significance value.

Arbitrarily, we select a small value of 0.001 for alpha, and calculate that $\epsilon = 0.00616$ and thus the error is within the range of 0.00616 for win probability, and within 0.616 for expected utility with 100,000 simulations, and a significance value of 0.001.

After analyzing our own bids, we calculated some optimal bids given the data, using the same two methods. Instead of finding the optimal bid for each value exactly, we started with a bid of 0 and incremented up to the value, since any higher will result in negative utility. These results were graphed for both the exact calculation and the Monte Carlo simulation (see Figures R1.5 and R1.6). The optimal bids and their expected utilities given from these calculations are given in Figures R1.3 and R1.4.

Professor Hartline
CS 332
11 Jan 2022

Figure R1.3, Optimal Bids calculated from Monte Carlo simulation, see Appendix A1

Values	10	20	30	40	50	60	70	80	90	100
Optimal Bid	5.02	10.25	11.79	21.74	20.2	31.03	31.66	41.19	41.23	51.19
Average Utility	0.274	1.95	4.01	6.88	9.95	13.67	18.47	24.13	29.90	36.02

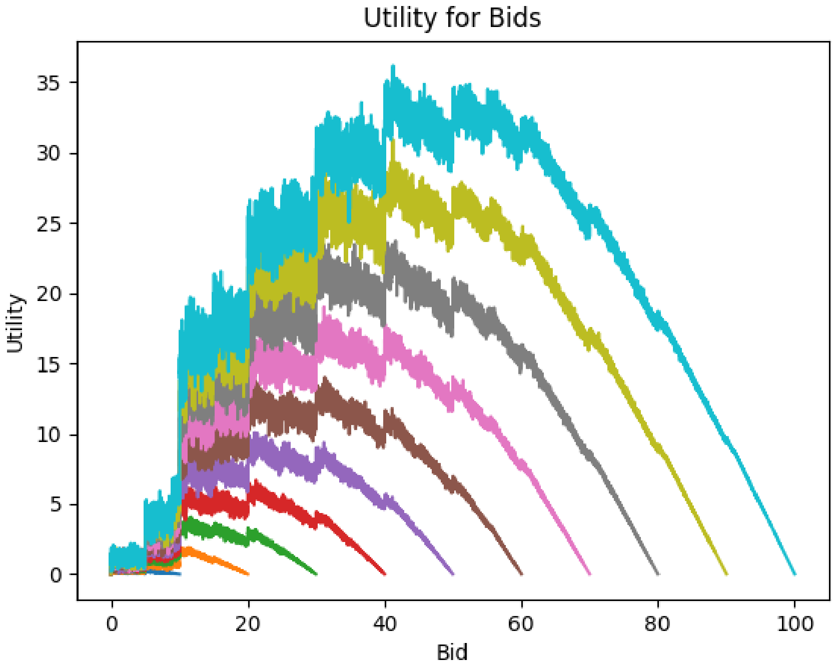
Figure R1.4, Optimal Bids calculated exactly, see Appendix A2

Values	10	20	30	40	50	60	70	80	90	100
Optimal Bid	5.01	11.01	11.01	21.01	21.01	31.01	31.01	41.01	41.01	51.01
Expected Utility	0.195	1.71	3.61	6.07	9.26	12.94	17.40	22.44	28.20	34.05

Average Expected Utility over all values for optimal bids = 13.59

Note that the results of R1.3 and R1.4 are very similar, but less so than previously, because the number of simulations was decreased to 1000 for computational feasibility, and because these errors are compounded among a distribution of errors which will take place from the many bids which were tested incrementally.

Figure R1.5, Utility for bids at various values, Monte Carlo simulation, see Appendix A1

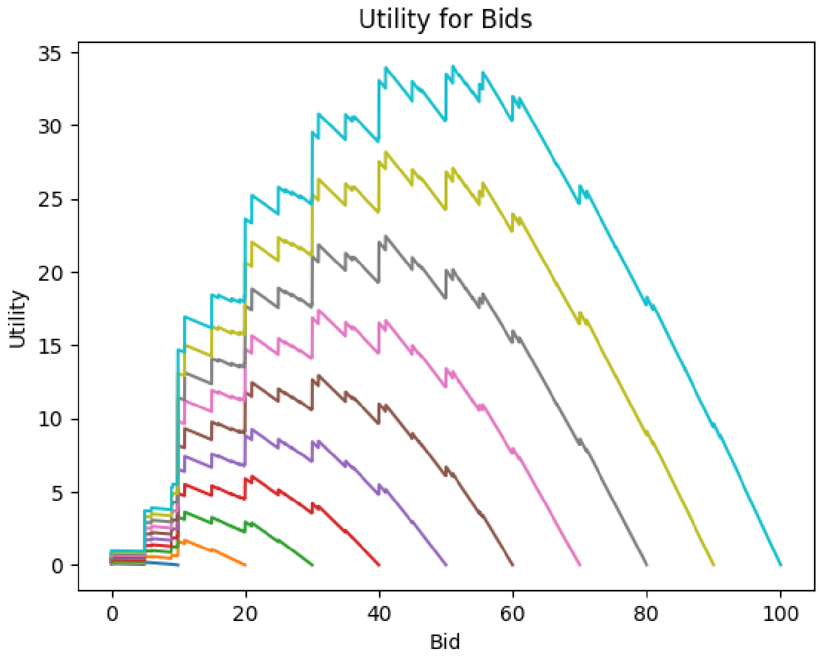


Legend

Value	Color
100	🟦
90	🟫
80	⬛
70	🟪
60	🟥
50	🟪
40	🟥

30	🟩
20	🟧
10	🟦

Figure R1.6, Utility for bids at various values, exact calculation, see Appendix A2



Legend

Value	Color
100	🟦
90	🟨
80	⬛
70	🟪
60	🟥
50	🟪
40	🟥
30	🟩
20	🟧
10	🟦

II. Algorithm for Bidding in Auction

Figure R2.1, Average Utilities of Devised Algorithm in a Monte Carlo simulation, see Appendix A3.

Samples Given	10 (online)	20 (online)	30 (online)	40 (online)	All (simultaneously)
Average Utility, Run 1	12.45	12.03	12.11	12.11	13.01

Average Utility, Run 2	11.96	11.99	12.11	12.02	12.96

After creating the algorithm and testing different initial parameters, we decided to compare our algorithm online vs. when all of the data is given at once. We also shuffled the data (Run 2) to ensure that the order of data points going in did not affect our results too drastically. Notably, we found that the shuffling did not greatly alter the results of the algorithm and that increasing sample size did not necessarily improve our algorithm. Another finding was that when the data was taken all together, the algorithm performed significantly better than when we took in bid data entry-by-entry.

Conclusions

Comparing the optimal bid data to our own bid data, it's clear that our own original bids did not maximize utility. The expected utility of our own bids was 5.60 and the expected utility of the optimal bids was 13.59. Our bids performed best in expected value in comparison to the optimal bids when the values were lower (<= 30). To understand why our bids did not perform as well as the optimal bids, but had good utility at low values, we compare the strategies of the optimal bids and our original bids.

The strategy of our own bids was to bid our value - 9.9999. The idea behind these bids was to bid the minimum amount (to maximize utility) while still winning against anyone who had lower value, assuming that anyone with lower value would bid at least our value - 10 in order to avoid incurring a negative utility. The performance of our strategy is consistent with Figure R1.6. Figure R1.6 gives the utility for different bids for each value. The pattern for each value approximates a step function with a sharp increase near each multiple of 10, and an overall positive slope until the bid reaches half the value, where the graph then slopes negatively and steeply.

For low values, our strategy of our value - 9.9999 approximates the optimality of using half the value of the bid. However, as the values increase, our utility begins to suffer compared to the maximum utilities, as our strategy deviates from the more optimal strategy.

We interpret the "step function property" of the optimal bids in Figure R1.6 to be explained by the fact that a high density of the bids in the data were near a multiple of 10, and so the optimal bids take advantage of this fact by beating many of the bids which are near multiples of 10, while maintaining low cost of bid. Our own strategy attempted to implement this part of the strategy, however, it missed out on the key idea that the bids should be near half the value. This idea is confirmed by the Bayes-Nash Equilibrium of a FPA of value*(n-1/n), where n is the number of bidders, 2. These two main ideas of using the Bayes-Nash equilibrium along with fitting the bids to the tendencies of the bidder data gives way to our devised algorithm.

Our devised algorithm started with an initial weight of these equilibrium values, and then in an online manner, collected new bids, and updated the optimal bids returned to reflect this data.

From our bidding algorithm, we found some disadvantages to using the online approach, as our results were significantly worse when given data row by row than when we gave the aggregated data. This is a result of the fact that the pieces of data are not all weighted the same, as the algorithm is unsure of the total amount of data that will be received, and it is simply a result of the fact that at lower sample size, less data is given. When calculating the optimal bids for each row and adjusting iteratively, the algorithm will only remember past data in terms of the current model of optimal bids. As noted in **Results**, however, shuffling the data did not seem to largely affect the utility, as calculated in a Monte Carlo simulation, which is surprising, given this online idea. Additionally, as the number of samples increased, the maximal utility did not change greatly. Ideally, the algorithm would fit stronger to the data, but we also did not want to overfit the data.

Another challenge we found was how much we should weigh samples vs. our equilibrium bids. Without additional data, it was much more difficult to answer this question, as weighing the samples more would almost always result in a better utility, but we could not discern whether the algorithm was getting better or overfitting. In the future, we could have generated our own data (which could be inaccurate/unrealistic), or gathered more samples of bidding data to further fine-tune our weighting system.

In general, it is valuable to use the mathematical methods of game theory and economics to have a basis for understanding optimality in such auctions, and it is also valuable to use data to better inform decisions, especially if bidders do not act optimally. We attempted to make a combination of these methods in our algorithm. A few questions to drive further research may include: How can we model sample bids for a human population? What is the best method of determining weights in an online algorithm without knowing the entire size of the dataset beforehand? And finally, in what contexts does the bidding done by some population approximate the Bayes-Nash equilibrium?

Professor Hartline
CS 332
11 Jan 2022

APPENDIX

A1, Bid Evaluation: Monte Carlo Simulation script

```
import csv
import random
import matplotlib.pyplot as plt
import numpy as np
def winning_sim(my_value_index):
   with open('bid_data.csv', newline='') as f:
       reader = csv.reader(f)
       data = list(reader)
   wins = 0
   ties = 0
   total = 0
   sims = 1000
   for i in range (sims):
       person = random.randint(1, 42)
       person_value = random.randint(0, 9)
       bid = float(data[person][person_value])
       if person == 29:
           continue
       my_value = my_value_index
       my_bid = float(data[29][my_value])
       if my_bid > bid:
           wins +=1
       if my_bid == bid:
           ties +=1
       total += 1
   return(wins, ties, total)
for i in range(10):
   print(winning_sim(i))
def expected_utility(this_person_number, my_value):
   with open('bid_data.csv', newline='') as f:
       reader = csv.reader(f)
       data = list(reader)
   total_utility = 0
   total_runs = 0
   sims = 100000
   for i in range (sims):
       person = random.randint(1, 42)
       person_value = random.randint(0, 9)
       bid = float(data[person][person_value])
       if person == this_person_number:
           continue
       my_bid = float(data[this_person_number][my_value])
       if my_bid > bid:
           total_utility += (my_value*10 + 10 - float(my_bid))
       if my_bid == bid:
           coin = random.randint(1,2)
           if coin == 1:
               total_utility += (my_value*10 + 10 - float(my_bid))
       total_runs += 1
   return total_utility/total_runs
for i in range(10):
   print(expected_utility(29, i))
def find_best(own_value):
   with open('bid_data.csv', newline='') as f:
       reader = csv.reader(f)
       data = list(reader)
   total_utility = 0
   sims = 1000
   best_bid = -1
   best_utility = 0
   steps = 100
   x_bids = []
   y_utility = []
   for my_bid in range (0,own_value*steps):
       my_bid = float(my_bid/steps)
       total_utility = 0
       for i in range (sims):
           person = random.randint(1, 42)
           person_value = random.randint(0, 9)
           bid = float(data[person][person_value])
           if person == -1:
               continue
           my_value = own_value
           if my_bid > bid:
               total_utility += (own_value - float(my_bid))
           if my_bid == bid:
               coin = random.randint(1,2)
```

```
               if coin == 1:
                   total_utility += (own_value - float(my_bid))
       x_bids.append(my_bid)
       y_utility.append(total_utility/sims)
       if total_utility > best_utility:
           best_utility = total_utility
           best_bid = my_bid
   max_utility, max_bid = max(y_utility), x_bids[np.argmax(y_utility)]
   plt.plot(x_bids, y_utility)
   plt.xlabel ('Bid')
   plt.ylabel ('Utility')
   plt.title ('Utility for Bids at Value {}'.format(own_value))
   plt.savefig('bid_a_{}'.format(own_value))
   return best_bid, best_utility/sims
for i in range(10, 110, 10):
   print(find_best(i))
```

A2, Bid Evaluation: Exact Calculation Script

```
import csv
import matplotlib.pyplot as plt
with open('bid_data.csv', newline ='') as csv_file:
   csv_reader = csv.reader(csv_file, delimiter=',')
   data = list(csv_reader)

   my_bids = data[29]
   del data[29]
   del data[0]
   for i in range(len(my_bids)):
       my_bids[i] = float(my_bids[i])
   for i in range(len(data)):
       for j in range(len(data[i])):
           data[i][j] = float(data[i][j])
   # 10 possible values (for me) * 41 entries * 10 possible bids = 4100 possible bids
   def exactCalculation(bid, value):
       wins = 0
       utility = 0
       for i in range(len(data)):
           for j in range(len(data[i])):
               if bid > data[i][j]:
                   wins += 1
                   utility += (value - bid)
               elif bid == data[i][j]:
                   wins += 0.5
                   utility += (value - bid) / 2
       return [wins / 410, utility / 410]
   win_p = []
   e_util = []
   for bid in my_bids:
       ans = exactCalculation(bid, bid + 9.9999)
       win_p.append(ans[0])
       e_util.append(ans[1])
   print("My win p: " + str(win_p))
   print("My expected utility: " + str(e_util) + "\n")


   opt_bids = []
   opt_e_util = []
   opt_win_p = []
   for val in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]:
       bestBid = 0
       bestUtil = 0
       bestWinP = 0

       utilities = []
       r = [x * 0.01 for x in range(1, val*100)]
       for j in r:
           ans = exactCalculation(j, val)
           utilities.append(ans[1])
           if ans[1] > bestUtil:
               bestUtil = ans[1]
               bestBid = j
               bestWinP = ans[0]
       opt_bids.append(bestBid)
       opt_e_util.append(bestUtil)
       opt_win_p.append(bestWinP)
       plt.clf()
       plt.plot(r, utilities)
       plt.xlabel ('Bid')
       plt.ylabel ('Utility')
       plt.title ('Utility for Bids at Value {}'.format(val))
```

```
        plt.savefig('exact_bid_a_{}'.format(val))

    print("My bids: " + str(my_bids))
    print("Optimal bids: " + str(opt_bids) + "\n")
    print(str(opt_win_p))
    print(str(opt_e_util) + "\n")
    print(sum(e_util) / 10)
    print(sum(opt_e_util)/ 10)
```

A3, Custom Bidding Algorithm with Monte Carlo evaluation script

```
import csv
import math
import numpy as np
import random

def exactCalculation(bid, value, data):
        wins = 0
        utility = 0
        for i in range(len(data)):
            for j in range(len(data[i])):
                if bid > data[i][j]:
                    wins += 1
                    utility += (value - bid)
                elif bid == data[i][j]:
                    wins += 0.5
                    utility += (value - bid) / 2
        return [wins / 410, utility / 410]

def bidAlgo(data, orig, s = 50):

        s += len(data)
        opt = []
        for val in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]:
            bestBid = 0
            bestUtil = 0
            bestWinP = 0
```

```
           for i in range(10 * val):
               ans = exactCalculation(i * 0.1, val, data)
               if ans[1] > bestUtil:
                   bestUtil = ans[1]
                   bestBid = i * 0.1
                   bestWinP = ans[0]
           opt.append(bestBid)
       w1 = len(data) / s
       w2 = 1 - w1
       ans = []
       for i in range(len(opt)):
           ans.append(opt[i] * w1 + orig[i] * w2)
       return [ans, s]

with open('bid_data.csv', newline ='') as csv_file:
   csv_reader = csv.reader(csv_file, delimiter=',')
   my_data = list(csv_reader)
   my_bids = my_data[29]
   del my_data[29]
   del my_data[0]
   for i in range(len(my_bids)):
       my_bids[i] = float(my_bids[i])
   for i in range(len(my_data)):
       for j in range(len(my_data[i])):
           my_data[i][j] = float(my_data[i][j])

   # 10 possible values (for me) * 41 entries * 10 possible bids = 4100 possible bids



   win_p = []
   e_util = []

   for bid in my_bids:
       ans = exactCalculation(bid, bid + 9.9999, my_data)
       win_p.append(ans[0])
       e_util.append(ans[1])
   print("My win p: " + str(win_p))
   print("My expected utility: " + str(e_util) + "\n")
   opt_bids = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
   cur_s = 0
```

Professor Hartline
CS 332
11 Jan 2022

```
## giving all data at once

[opt_bids, cur_s] = bidAlgo(my_data, opt_bids)
print(opt_bids)
sims = 1000
total_utility = 0

for i in range (sims):

   person = random.randint(0, 40)
   person_value = random.randint(0, 9)
   my_value = random.randint(0, 9)
   bid = float(my_data[person][person_value])

   my_bid = float(opt_bids[my_value])
   if my_bid > bid:
       total_utility += (my_value*10 + 10 - float(my_bid))
   if my_bid == bid:
       coin = random.randint(1,2)
       if coin == 1:
           total_utility += (my_value*10 + 10 - float(my_bid))

print(total_utility / sims)


opt_bids = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

cur_s = 0

## online iterations (shuffle if needed)

## np.random.shuffle(my_data)

for i in range(30):
    if cur_s == 0:
        [opt_bids, cur_s] = bidAlgo([my_data[i]], opt_bids)
    else:
        [opt_bids, cur_s] = bidAlgo([my_data[i]], opt_bids, cur_s)

print(opt_bids)
```

Professor Hartline
CS 332
11 Jan 2022

```
sims = 1000
total_utility = 0

for i in range (sims):

   person = random.randint(0, 40)
   person_value = random.randint(0, 9)
   my_value = random.randint(0, 9)
   bid = float(my_data[person][person_value])

   my_bid = float(opt_bids[my_value])
   if my_bid > bid:
       total_utility += (my_value*10 + 10 - float(my_bid))
   if my_bid == bid:
       coin = random.randint(1,2)
       if coin == 1:
           total_utility += (my_value*10 + 10 - float(my_bid))

print(total_utility / sims)
```