Online Markets Project 1

Two-Person First Price Auction Analysis

April 2022

1 Introduction

The following study aims to analyze the patterns and strategies of players participating in a first price auction. The first price auction is an auction in which bidders place bids simultaneously, without knowledge of the other players' bids, and the highest bidder wins (and pays their bid). This auction style is commonly used in practical applications, sometimes referred to as a blind auction, and as such it is important to understand how and why bidders take the actions they do. For the purposes of this study, consider the following first price auction setup:

Let two players, P_1 and P_2, be playing a head-to-head first price auction. Each P_i draws value $v_i \in [0, 100]$ uniformly. Each P_i then places a bid $b_i \in [0, 100]$. Once both bids have been placed, the highest bidder wins and derives utility $v_i - b_i$ while the lower bidder loses and gets 0 utility. The players' goal is to win and maximize utility received. Both players are aware of the distribution from which their values have been drawn, but are unaware of each others' bid amounts.

Unlike a second price auction, a first price auction does not have a dominant strategy. This makes it hard to predict what players would (on average) bid as a function of their value. This study aims to analyze the winning probability, expected utility, and optimal bids for the authors, as well as predict the ways in which bidders will bid in a first price auction, using statistical methods with Python.

2 Preliminaries

All 54 students in our Online Markets class were told the rules of the game and were asked to play twice, assuming that, each time, a random other student in the class would compete with them. Accordingly, they were able to generate, using their net id as the seed of a hash function, two random values in $[0, 100]$. These were split arbitrarily into auction A and auction B. It was originally decreed that these draws would both be drawn uniformly from the interval $[0, 100]$, but upon assessing the data we have established that in actuality values drawn for auction A were drawn uniformly from $[0, x]$ while values drawn for auction B were drawn uniformly from $[x, 100]$, where $x \in [49, 50.8]$, but is most likely 50 or 49. While we don't know exactly which value of the cutoff x, we assume this was done to ensure an even distribution of values to make our analyses more fruitful given our relatively small data set.

After the students played the game and placed their bids, all the data was compiled into a single csv and provided to us for this study. The dataset contains player IDs, along with each individual value and bid amount for both auction A and auction B.

It is easy to deduce that any rational player with value v_i will place their bet within the interval $(0, v_i)$. If they bid anything outside this interval, they are guaranteed to get a utility of zero, while if they are in the interval, there is a possibility they get a positive utility. Accordingly, without further analysis of optimal strategies, we can notice that the interval of rational bids will increase linearly with v_i. It follows that we should attempt to predict a player's action as a function of their value.

At this point, we should clearly specify the information any player of the game has available to them. They know that their value, along with their opponents, was drawn uniformly from the interval $[0, 100]$. This allows them to deduce that $E(v_i) = 50$. In addition, because the values are drawn uniformly, they are able to derive the cumulative distribution function (cdf):

$$F_{v_i}(z) = \frac{z}{100} \tag{1}$$

and the probability density function (pdf):

$$f_{v_i}(z) = \frac{1}{100} \tag{2}$$

There are no other clear facts that they can derive from the information given. As we've mentioned, it is hard to determine an optimal strategy in this context given no other information. That said, players are still going to need to make a bid, so they must come up with some reasoning to support their decision. This reasoning could be as naive as "I like the number 33", or it could be much more complex and based on perceived thoughts of other players in the population of players they could play against (our class). It is impossible, and would be unsubstantiated, to attempt to predict these reasonings. They are ultimately psychological in nature. There are infinite possible reasonings one could apply, and we have no way of predicting which of these would be more likely. Assuming a player of this game adopts this perspective, we can imagine that they would expect the other player, with value v_j, to pick a bid uniformly within the interval $(0, v_j)$. Accordingly, we can predict that any player will assume that $E(b_j) = \frac{v_j}{2}$.

Given this assumption, what can we say about how player i will bid? Well, we know $E(v_j) = 50$, so it follows that $E(b_j) = E(\frac{v_j}{2}) = 25$. Let us assume that our player has also made this assumption. From these assumptions, we can predict the action our player will take as follows:

$$A_i(v_i) = \begin{cases} \text{draw bid uniformly from (0,25)} & \text{v}_i \leq 25 \\ \text{let } \epsilon \to 0, \text{ bid } 25 + \epsilon & \text{v}_i > 25 \end{cases} \tag{3}$$

This is a great start, but what happens if we assume that our players opponent has made similar assumptions? In that case, without loss of generality, player i knows that $E(v_j) = 50$, so they can assume that player j would bid $25 + \epsilon$. Knowing this, however, means that they should let $\epsilon' \to 0$ and bid $25 + \epsilon'$. At the same time, we could reasonably assume that player j would know they could expect player i to bid $25 + \epsilon'$, so they would bid $25 + \epsilon''$ in the same exact fashion. We can see that this creates an infinite feedback loop. Given this dynamic, we can not say anything about when this loop will stop, except that we know bidders will not bid above their value, which we can expect to be 50. Accordingly, knowing that $E(v_j) = 50$, which means we expect each player to have values above 25, we can reasonably say that, on average, players will bid uniformly in the interval $(25, 50)$. In fact, because this is an expectation, we can work backwards to predict that a player with value v_i will bid uniformly in the interval $(\frac{v_i}{2}, v_i)$. We will test this hypothesis in the following section.

3 Results

Our code for all calculations can be found in the appendix.

In accordance with our earlier assumptions, we began by removing the author's respective data and removing all bids exactly equal to zero or above the player's value. We then calculated the author's winning probability, expected utility, and optimal expected utility.

For each auction, one can calculate the winning probability as follows:

$$\text{Win\_Prob}(b_i) = \frac{1}{\text{player\_count}} \sum_{b_j \text{ s.t. } j \neq i} f_{b_i}(b_j) \tag{4}$$

where

$$f_{b_i}(b_j) = \begin{cases} 0 & b_i < b_j \\ \frac{1}{2} & b_i = b_j \\ 1 & b_i > b_j \end{cases} \tag{5}$$

For Auction A, we got winning probabilities:

Win_Prob(27.66)=0.7755 Win_Prob(30)=0.3673

For Auction B, we got winning probabilities:

Win_Prob(47.66)=0.8878 Win_Prob(51)=0.5102

Then, because utility is defined as $U(v_i, b_i) = v_i - b_i$ if we win, and zero if we don't, we can easily calculate our expected utility as follows:

$$E(U(v_i, b_i)) = \text{Win\_Prob}(b_i) * (v_i - b_i) \tag{6}$$

For Auction A, we got expected utilities:

$E(U(33, 27.66))$=4.1412 $E(U(43.7, 30))$=4.2759

For Auction B, we got expected utilities:

$E(U(59.3, 47.66))$=12.1622 $E(U(71.5, 51))$=10.4592

Calculating the bid that optimizes our expected utility given our values is less trivial. First, we note that we should not bid anything greater than or equal to our value, as we've mentioned rational bidders should do. Second, we note that, for any bid within $[0, 100]$, if we can reduce our bid without falling below any more bids than we were already below, it will be a more optimal bid. Then, we establish and prove the following lemma:

Lemma 1. *Let $v_i \in [0, h]$. Let B be the set of possible opponents bids that are below v_i with $|B| = n$ and let $b_j \in B$. Let $0 < m \leq n$ be the number of bids that are below b_j and $0 \leq k \leq n$ be the number of bids that are equal to b_j. Let $b'_j = b_j + \epsilon$ s.t. $0 < \epsilon < \frac{m(v_i - b_j)}{k+m}$. Then $E(U(v_i, b_j)) < E(U(v_i, b'_j))$ as long as no $b''_j \in B$ are in (b_j, b'_j).*

Proof. $E(U(v_i, b_j)) = (\frac{k}{n} + \frac{1}{2}\frac{m}{n})(v_i - b_j) = \frac{2k+m}{2n}(v_i - b_j) = \frac{(2k+m)(v_i-b_j)}{2n} = \frac{2kv_i - 2kb_j + mv_i - mb_j}{2n}$.
On the other side, $E(U(v_i, b'_j)) = \frac{k+m}{n}(v_i - b'_j) = \frac{kv_i - kb'_j + mv_i - mb'_j}{n} = \frac{kv_i - kb_j - k\epsilon + mv_i - mb_j - m\epsilon}{n}$.
Now we can write $E(U(v_i, b_j)) * n = kv_i - kb_j + \frac{1}{2}mv_i - \frac{1}{2}mb_j$ and $E(U(v_i, b'_j)) * n = kv_i - kb_j - k\epsilon + mv_i - mb_j - m\epsilon$.
And now, $(E(U(v_i, b_j)) * n) - kv_i + kb_j = \frac{1}{2}m(v_i - b_j)$ and $(E(U(v_i, b'_j)) * n) - kv_i + kb_j = m(v_i - b_j) - \epsilon(k + m)$.
Now, we subtract $\frac{1}{2}m(v_i - b_j)$ to get
$(E(U(v_i, b_j)) * n) - kv_i + kb_j - (\frac{1}{2}m(v_i - b_j)) = 0$
$(E(U(v_i, b'_j)) * n) - kv_i + kb_j - ((\frac{1}{2}m(v_i - b_j))) = \frac{1}{2}m(v_i - b_j) - \epsilon(k + m)$
Now, we can solve the equation $\frac{1}{2}m(v_i - b_j) > \epsilon(k + m)$ to get $\epsilon < \frac{m(v_i - b_j)}{k+m}$
Note that because $k \geq 0$, $m \geq 0$, and $b_j < v_i$, $\frac{m(v_i-b_j)}{k+m} > 0$.
This means that for all epsilon $0 < \epsilon < \frac{m(v_i-b_j)}{k+m}$ we have
$0 < \frac{1}{2}m(v_i - b_j) - \epsilon(k + m)$, or written equivalently

$(E(U(v_i, b_j)) * n) - kv_i + kb_j - (\frac{1}{2}m(v_i - b_j)) <$
$(E(U(v_i, b'_j)) * n) - kv_i + kb_j - (\frac{1}{2}m(v_i - b_j))$.
It follows that $E(U(v_i, b_j)) < E(U(v_i, b'_j)) \; \forall \; 0 < \epsilon < \frac{m(v_i - b_j)}{k+m}$. $\square$

Now that we have these two facts, we can see that the optimal bid must lie in the set $B' = \{b_j + \epsilon | b_j \in B, 0 < \epsilon < \frac{m(v_i - b_j)}{k+m}\}$. We can now loop through all the opponents bids, store the expected utility we would get from betting that bid plus some small positive value in a list, and sort that list to find the optimal bid and it's expected utility. Note that the above proof does not apply for the smallest opponent bid; this was addressed in our code. In addition, We noticed that no two bids in our data different by exactly 0.0001, and the furthest level of precision we recorded was 4, so we set $\epsilon = 0.0001$ arbitrarily. Also note that $0.0001 < \frac{m(v_i - b_j)}{k+m}$ as m is a positive integer and neither of the author's values were within 0.0001 of any other bids.

For Auction A, we got optimal bids:

Opt_Bid(33) = 14.0001 Opt_Bid(43.7) = 15.0001

For Auction B, we got optimal bids:

Opt_Bid(59.3) = 42.5501 Opt_Bid(71.5) = 50.1001

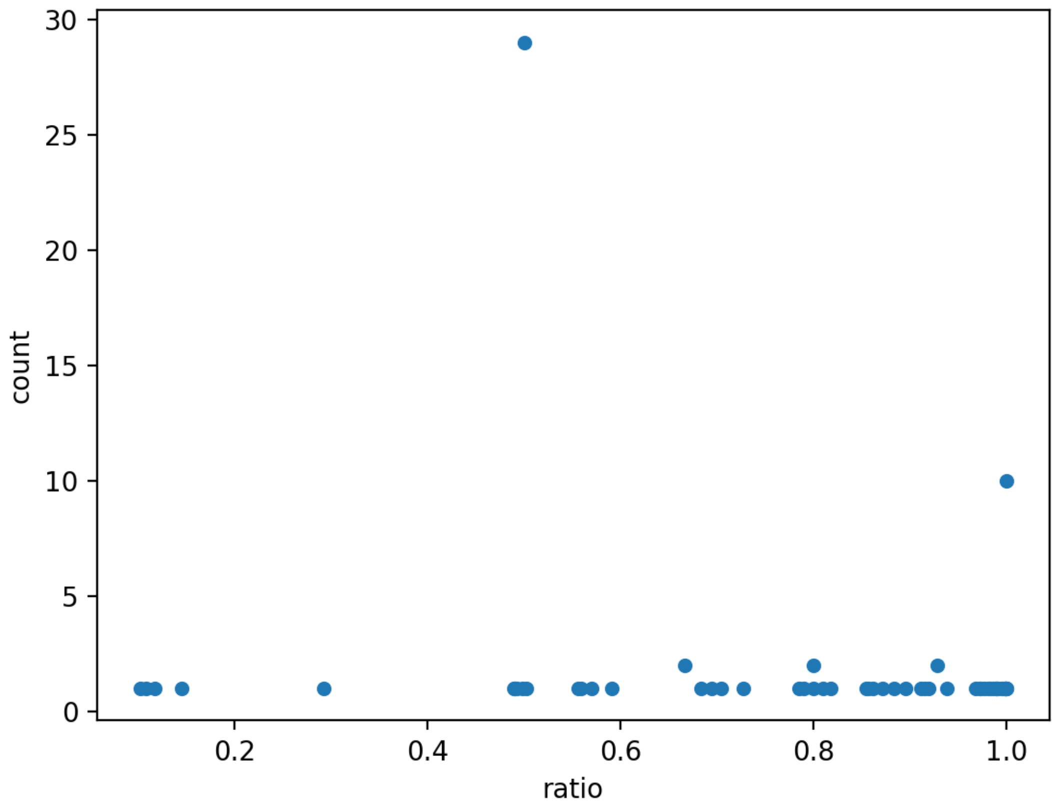
Now, to assess the validity of our hypothesis, we need to measure, for each bidder, how far their bid falls from their value relative to their value. That is, we want to test whether or not:

1. $E(b_i) = \frac{3}{4}v_i$
2. At least 95% of the data is in the interval $(\frac{v_i}{2}, v_i)$
3. The data in this interval follows the cdf/pdf of a uniform distribution on $(\frac{v_i}{2}, v_i)$

To test the first criteria, we calculated the average ratio between bids and their values. The average across our rational bidders was 0.7122, which is pretty close to the desired 0.75.

To test the second, we calculated the percentage of ratios (calculated in the last step) that were above one half. We got 60% on this measure. When we included data that was above or equal to one half, we got 89%. This suggests a significant portion (29%) of players bet exactly half their value, which goes against our prediction.

Although we have already essentially disproved our hypothesis, below is a plot of ratios on the x-axis and the count of each ratio we observed on the y-axis.



The two large values at 0.5 and 1 show the strongest evidence invalidating our hypothesis as they indicate relatively massive changes in the pdf of the underlying distribution creating this data.

4 Conclusions

In the preliminaries section of this paper we talked about the irrationality of predicting the data based on psychologically principles. At this point, we are going to temporarily dispel this notion. Given the high concentration of answers at ratio values of one half and one, we can assume a couple things about player's decision making in our sample.

First and foremost, it is simple to observe that the logic we used to get to our hypothesis is not entirely obvious and required much more time to think through than we were given to play the game. In fact, it is highly likely that most people answered with a minute or two of opening the game. In addition, there wasn't anything actually at stake here. We had a couple data points in our data that were exactly zero and some that were above their respective value. We could almost guarantee this wouldn't happen if there was real value at stake in this game.

So, while the data did not technically support our hypothesis, if you look at the scatter plot we made, it is not entirely unreasonable to assume that, given data that more accurately reflected the scenario this game attempts to model, we could see data that fits our hypothesis. After all, at a quick glance the vast majority of the data (89%) is within [0.5,1].

This study should be re-done in the following way. Give everyone dollars equal to their derived value. If they lose, they keep zero of the dollars. If they win, they keep their profit. It would be interesting to see how the data would change in this situation.

P.S. If that's too pricey, simply take away the dollars afterwards (without telling people at first of course), or just reduce the stakes to a few dollars. It is likely people would play a lot differently with time and a few dollars at stake.

5 Appendix

```
import pandas as pd
import matplotlib.pyplot as plt

# Fetch the csv with pandas and remove ourselves
def fetch_csv():
    total_bids = pd.read_csv("/Users/_name_/Desktop/bid_data.csv")
    mr_pd_netids = ["***8079", "***4122"]

    mr_pd_bids = total_bids[total_bids.ID.isin(mr_pd_netids)].reset_index(drop=T
    bid_data = total_bids[total_bids.ID.isin(mr_pd_netids) == False].reset_index
    bid_data = bid_data[bid_data['bid in Auction A'] > 0]
    bid_data = bid_data[bid_data[' bid in Auction B'] > 0]
    bid_data = bid_data[bid_data['bid in Auction A'] <= bid_data['v_A']]
    bid_data = bid_data[bid_data[' bid in Auction B'] <= bid_data['v_B']]
    bid_data = bid_data.reset_index(drop=True)
    return mr_pd_bids, bid_data

mr_pd_bids, bid_data = fetch_csv()
print(mr_pd_bids)
print(bid_data)

player_count = 0
# Experimentally, pd was first, but I presume this is because the filter above f
```

```
pd_bA = mr_pd_bids.iloc[0].loc['bid in Auction A']
pd_bB = mr_pd_bids.iloc[0].loc[' bid in Auction B']
mr_bA = mr_pd_bids.iloc[1].loc['bid in Auction A']
mr_bB = mr_pd_bids.iloc[1].loc[' bid in Auction B']
print("mr_pd bids:")
print(pd_bA)
print(pd_bB)
print(mr_bA)
print(mr_bB)
pd_vA = 33
pd_vB = 59.3
mr_vA = 43.7
mr_vB = 71.5


pd_A_win_prob = 0
pd_B_win_prob = 0
mr_A_win_prob = 0
mr_B_win_prob = 0

pdU_A = pd_vA - pd_bA
pdU_B = pd_vB - pd_bB
mrU_A = mr_vA - mr_bA
mrU_B = mr_vB - mr_bB

A_bids = []
B_bids = []
Bids_Values = []
for idx, player in bid_data.iterrows():
    player_count = player_count + 1
    playerbid_A = player.loc['bid in Auction A']
    A_bids.append(playerbid_A)
    playerbid_B = player.loc[' bid in Auction B']
    B_bids.append(playerbid_B)
    playervalue_A = player.loc['v_A']
    playervalue_B = player.loc['v_B']
    Bids_Values.append([playervalue_A, playerbid_A])
    Bids_Values.append([playervalue_B, playerbid_B])
    if pd_bA == playerbid_A:
        pd_A_win_prob = pd_A_win_prob + 0.5
    elif pd_bA > playerbid_A:
        pd_A_win_prob = pd_A_win_prob + 1
    if pd_bB == playerbid_B:
        pd_B_win_prob = pd_B_win_prob + 0.5
    elif pd_bB > playerbid_B:
        pd_B_win_prob = pd_B_win_prob + 1
    if mr_bA == playerbid_A:
        mr_A_win_prob = mr_A_win_prob + 0.5
    elif mr_bA > playerbid_A:
        mr_A_win_prob = mr_A_win_prob + 1
    if mr_bB == playerbid_B:
        mr_B_win_prob = mr_B_win_prob + 0.5
    elif mr_bB > playerbid_B:
        mr_B_win_prob = mr_B_win_prob + 1

pd_A_win_prob = pd_A_win_prob / player_count
pd_B_win_prob = pd_B_win_prob / player_count

mr_A_win_prob = mr_A_win_prob / player_count
mr_B_win_prob = mr_B_win_prob / player_count

pdex_A = pdU_A * pd_A_win_prob
pdex_B = pdU_B * pd_B_win_prob

mrex_A = mrU_A * mr_A_win_prob
mrex_B = mrU_B * mr_B_win_prob

print("mr_pd win probs:")
print(pd_A_win_prob)
print(pd_B_win_prob)
print(mr_A_win_prob)
print(mr_B_win_prob)

print("expected utility:")
print(pdex_A)
print(pdex_B)
print(mrex_A)
print(mrex_B)

A_bids.sort()
B_bids.sort()

pd_A_utilities = []
pd_B_utilities = []

mr_A_utilities = []
mr_B_utilities = []

prob_first = 1 / player_count
prev_A_bid = A_bids[0]
```

```
pd_first_A = (pd_vA-(prev_A_bid+0.0001))*prob_first
pd_A_utilities.append([prev_A_bid, pd_first_A])

mr_first_A = (mr_vA-(prev_A_bid+0.0001))*prob_first
mr_A_utilities.append([prev_A_bid, mr_first_A])

A_equal_count = 0
for i in range(1,len(A_bids)):
    bid = A_bids[i] + 0.0001
    prob = (i+1)/player_count
    pd_utility = (pd_vA-bid)*prob
    mr_utility = (mr_vA-bid)*prob
    if A_bids[i] == prev_A_bid:
        A_equal_count = A_equal_count + 1
        last_idx = i - A_equal_count
        pd_A_utilities[last_idx] = [bid, pd_utility]
        mr_A_utilities[last_idx] = [bid, mr_utility]
    else:
        pd_A_utilities.append([bid, pd_utility])
        mr_A_utilities.append([bid, mr_utility])
    prev_A_bid = A_bids[i]

prev_B_bid = B_bids[0]

pd_first_B = (pd_vB-(prev_B_bid+0.0001))*prob_first
pd_B_utilities.append([prev_B_bid, pd_first_B])

mr_first_B = (mr_vB-(prev_B_bid+0.0001))*prob_first
mr_B_utilities.append([prev_B_bid, mr_first_B])

B_equal_count = 0
for i in range(1,len(B_bids)):
    bid = B_bids[i] + 0.0001
    prob = (i+1)/player_count
    pd_utility = (pd_vB-bid)*prob
    mr_utility = (mr_vB-bid)*prob
    if B_bids[i] == prev_B_bid:
        B_equal_count = B_equal_count + 1
        last_idx = i - B_equal_count
        pd_B_utilities[last_idx] = [bid, pd_utility]
        mr_B_utilities[last_idx] = [bid, mr_utility]
    else:
        pd_B_utilities.append([bid, pd_utility])
        mr_B_utilities.append([bid, mr_utility])
    prev_B_bid = B_bids[i]
```

```
pd_A_utilities.sort(key = lambda x: x[1])
pd_B_utilities.sort(key = lambda x: x[1])

mr_A_utilities.sort(key = lambda x: x[1])
mr_B_utilities.sort(key = lambda x: x[1])

pd_A_U_optidx = len(pd_A_utilities) - 1
pd_B_U_optidx = len(pd_B_utilities) - 1

mr_A_U_optidx = len(mr_A_utilities) - 1
mr_B_U_optidx = len(mr_B_utilities) - 1

pd_opt_A = pd_A_utilities[pd_A_U_optidx]
pd_opt_B = pd_B_utilities[pd_B_U_optidx]

pd_optbid_A = pd_opt_A[0]
pd_optutility_A = pd_opt_A[1]
pd_optbid_B = pd_opt_B[0]
pd_optutility_B = pd_opt_B[1]

mr_opt_A = mr_A_utilities[mr_A_U_optidx]
mr_opt_B = mr_B_utilities[mr_B_U_optidx]

mr_optbid_A = mr_opt_A[0]
mr_optutility_A = mr_opt_A[1]
mr_optbid_B = mr_opt_B[0]
mr_optutility_B = mr_opt_B[1]

print("pd bid, utility")
print(pd_optbid_A)
print(pd_optutility_A)
print(pd_optbid_B)
print(pd_optutility_B)
print("mr bid, utility")
print(mr_optbid_A)
print(mr_optutility_A)
print(mr_optbid_B)
print(mr_optutility_B)

diff_from_mean = 0
data_included = 0
ratio_list = []
ratio_count = {}
for i in range(0,len(Bids_Values)):
    bid_val = Bids_Values[i]
    val = bid_val[0]
    bid = bid_val[1]

    ratio = bid / val
    ratio_count[ratio] = ratio_count.get(ratio, 0) + 1
    diff_from_mean = diff_from_mean + ratio
    if ratio >= 0.5:
        data_included = data_included + 1
average_diff = diff_from_mean / (player_count * 2)
data_included = data_included + 1
ratio_count_list = []
for ratio in ratio_count:
    count = ratio_count[ratio]
    ratio_list.append(ratio)
    ratio_count_list.append(count)
print(average_diff)
print(data_included)
s = pd.Series(ratio_list, ratio_count_list)
data = {'ratio': ratio_list,
        'count': ratio_count_list}
df = pd.DataFrame(data,columns=['ratio','count'])
df.plot(x ='ratio', y='count', kind = 'scatter')
plt.show()
```