**CS 357: Algorithmic Game Theory** **Spring 2022**

# Assignment 4 (due 03/03/2022)

*Instructor: Shikha Singh* *Solution Template*

**Instructions.** This is a partner assignment: up to 2 students can work together and submit a joint submission and receive the same grade. To receive the starter-code repository, fill out the following https://tinyurl.com/357partner. The total points of this assignment is 35.

You are allowed to help each other with any Python-related questions, as long as it is not directly related to the assignment code. Similarly, you are allowed to look for *python-specific* resources online, e.g. on stackexchange, pydocs, and realpython.

There are two components of the assignment—**written answers** and the **associated code**. You must submit the code through Github and PDF of LaTeX answers via Gradescope.

This assignment will be graded on correctness, clarity, presentation of results, and the quality of the interpretations made. Maintain good documentation of your simulation parameters and results. Incorporate them in your write up in a structured way using tables.

All assignments are due **at 11 pm EST** on the day of the deadline.

## First-Price Keyword Auctions (6 pts)

Read one of the earliest foundational study by Edelman and Ostrovsky: "Strategic bidder behavior in sponsored search auctions" [2], and answer the following questions briefly (around 2-3 sentences are fine; the goal is for you to read one of the classic papers in the history of sponsored search before we start our own empirical analysis):

**Problem 1.** (a) How do auto-bidders exacerbate the unstable bidding behavior in first-price keyword auctions? How does it affect the social welfare and revenue?

(b) How do the authors estimate (a) the valuations of bidders, and, (b) the stability of a keyword auction market based on past data.

(c) What conclusions do they draw from their study? What are, in your opinion, some of the shortcomings of their analysis?

## GSP with Balanced Bidding vs VCG[1]

In the rest of the assignment, you will implement the *balanced bidding agent* for a generalized-second-price (GSP) auction and the *VCG mechanism*. You will also analyze the effect of strategic bidding and reserve prices on the revenue.

[1] *Acknowledgement.* This ad auction simulator is adapted from David Parkes and Sven Seuken.

**Ad Auction Simulator** You are given an ad auction simulator. The details of the simulation are given below:

- *Rounds.* The auction will proceed over rounds $t$, where $t$ goes from from $0, 1, \ldots, 47$. Each round simulates 30 minutes, so overall we are modeling bidding over one full day.
- *Value distribution.* Value-per-click $v_i$ of the agents are uniformly distributed between \$0.25 and \$1.75. If the agents are symmetric (following the same strategy), then it does not matter how these values are assigned to them. However, if the agents are playing different strategies, we need to randomly shuffle the valuations for different iterations of the simulation. This is handled by the `--perms` argument (see the testing section.)
- *Click-through-rates.* Let $\alpha_i^t$ denote the click-through-rate received by the $i$th slot in round $t$. The click-through-rate of the first (top) slot in each round, $\alpha_1^t$, follows a cosine shape:

  $$\alpha_1^t = \text{round}(30cos(\pi t/24) + 50), \quad \text{where } t = 0, 1, \ldots, 47$$

  where round denotes rounding to the nearest integer value. This means that during the 48 rounds (thus a 24 hour day), the clicks that the top slot receive start at 80 in the first round, go down to 20 in round 24, and rises again to 80 in round 48, averaging 50 clicks through the day. The click through rates of other slots $\alpha_i^t$ for $t = 2, \ldots, k$ is given by:

  $$\alpha_i^t = 0.75^{i-1}\alpha_1^t$$

  This click-through-rate function is modeled to fit clicks observed in real-world data.
- *Bidding.* In the first round, your bid is queried through the `initalBid()` function. For all rounds after that, the bids are placed by calling the `bid()` function.
- *Reserve price.* If a non-zero reserve price is set then bids that are less than the reserve are ignored. In the GSP auction, the agent that gets the lowest of the allocated slot (note that there may be some unallocated slots) pays the maximum of the reserve price $r$ and the bid of the next-lower agent in bid-sorted order. If all bidders bid below the reserve price $r$, then no slot is allocated.
- *Number of slots.* The number of available slots in any round is set to one less than the number of *active bidders* in that round (unless there is only one active bidder). An active bidder is a bidder with a bid that is greater than or equal to the reserve price.
- *Payment and utility of agents.* In each round $t$, the total payment for slot $j$ is $c_j p_j$, where $c_j$ is the number of clicks received by slot $j$ in that round and $p_j$ is the per-click-payment of that slot determined by the auction being used. In the GSP auction, the price-per-click $p_j = b_{j+1}$ is the bid of the agent allocated to the next-lower slot. In the VCG mechanism, this price is determined by the payment rule. The utility of an agent $i$ occupying slot $j$ is calculated as $u_i = c_j(v_i - p_j)$, where $p_j$ is the price-per-click of slot $j$.
- *Simulating one round.* In each round, the simulation collects bids, assigns slots, calculates the number of clicks and payments, and determines the utilities of the bidders.

- *Budget $B$.* Each bidding agent is given a total budget $B = \$1750$ for the entire day (48 rounds). As long an agent has a positive budget, they are allowed to keep placing bids in the next round. [2] The simulation handles this part—if the sum of the payments from all previous rounds is greater than the agent's budget, then it reduces all bids to \$0.

**Python3.** Make sure you have Python 3.7 or above installed on your machine.

**Starter code.** You have been given the following python files as your stater code: `auction.py, bbagent.py, gsp.py, history.py, stats.py, truthful.py, util.py, vcg.py`. Among these, you will only be editing `bbagent.py` and `vcg.py`. But it is a good idea to familiaze yourself with the code in `truthful.py` and `gsp.py`.

**Testing.** The following command lists command-line arguments that will help in testing.

```
python3 auction.py --h
```

You will use the following most frequently.

- **Max permutations** `--perms=P`: When analyzing symmetric agents (all agents using the same strategy), you should set `P=1`. For asymmetric agent populations, it is important to make sure that the value distribution is permuted among them during multiple runs of the simulation. Note that there are $n!$ ways of assigning these values to the $n$ agents. If $n!$ is less than `P`, simulations will be run for each of these permutations, if $n!$ is greater than `P`, then `P` permutations will be randomly chosen.
- **Number of iterations** `--iters=I`: This runs the simulation `I` times. While debugging, set this to one, but when aggregating and analyzing data, you want this to be high to get good results (run it at least 50 or more times if the code is not too slow.)
- **Seed** `--seed=S`: When you are debugging and you want different iterations to have the same repeatable value distribution (and tie breaking), you can use this argument and set `S` to an integer.
- **Reserve price** `--reserve=R`: By default this is zero, but when doing tests with non-zero reserve prices, you can use this argument.
- **Mechanism** `--mech=M`: By default, set to `gsp`, but can be changed to `vcg`, or `switch`.

**Initial test.** The following command runs the simulation with five truthful agents for two rounds. The `--perms=1` argument forces the simulator to assign a single permutation of value-per-click to the agents.

```
python3 auction.py --loglevel=debug --numRound=2 --perms=1 Truthful,5
```

The `--loglevel=debug` option gives you round-by-round logs of each bidders bids, slots they received, the number of clicks, payments, etc. Finally, the average stats are listed at the end. If you would like to improve the logging and stats generated, you may do so.[3]

[2] You can, however, overspend your budget in at most one round and end up with a small deficit.

[3] Let me know if you improve any features in the code, as they will help future iterations of the course.

**Note on number of agents.** I recommend using a total of five agents for all the tests.

## Coding Tasks and Questions

**Problem 2.** (6 pts) (**Implementing the Balanced Bidding Agent**) In the file `bbagent.py` implement a bidding agent that uses the balanced-bidding strategy. This strategy was studied in detail in the following paper: Greedy Bidding Strategies for Keyword Auctions[1].

Assume that each agent knows the bids made by all other agents in the previous round. Assuming all other agents do not change their bid, a greedy balanced-bidding agent targets a slot $j$ that maximizes its utility. There may be a range of bids that would achieve the target slot $j$. Recall from lecture that a balanced-bidding strategy, picks the highest bid $b$ from the range that makes the agent indifferent between getting the current slot at the current price and the slot above at price $b$. The idea is that you want to drive up the price of the competitor occupying the slot above you without fear of retaliation.

In particular, implement a balanced-bidding agent for player $i$, whose bidding strategy is defined as follows. Fix $b_{-i}$ of all other agents.

- find a target slot $j$ that maximizes utility $\alpha_j(v_i - p_j)$
- choose a bid $b_i$ for the next round so that it satisfies the following:

$$\alpha_j(v_i - p_j) = \alpha_{j-1}(v_i - b_i)$$

  For $j = 0$ (the top slot), set $\alpha_{-1} = 2\alpha_0$ just to make the above equation well defined.

In the above calculation, you should use the last round's click through rate, rather than the "future" click through rate, because agents only have access to the historical data.

The file `bbagent.py` contains a skeleton of the code. You must complete the functions:

- `expectedUtils()`: this function computes the expected utility of the bidder from targeting each slot and returns a vector of utilities
- `targetSlot()`: this function calls `expectedUtils` and uses it to compute the target slot that gives the bidding agent the maximum utility
- `bid()`: this function computes the equilibrium bid $b_i$ using the balanced bidding technique.

You will find the `slotInfo()` function useful which has already been implemented for you. You may want to look at `gsp.py` to understand the implementation of the GSP auction, especially the `bidRangeForSlot()` function.

**Problem 3.** (6 pts) (**Analyzing GSP Bidding Strategies**) We will test the balanced bidding agent, and compare it to the truthful agent. To answer the following questions, run the simulation with 5 agents with number of iterations set to 50 or more using the `--iters` argument. Use the `--seed` argument to generate repeatable distributions. For symmetric agents (parts (a) and (b)) use `--perms=1`.

(a) What is the average utility of a population of only truthful agents?

(b) What is the average utility of a population of only balanced-bidding agents? Compare with part (a) and explain the differences you see.

(c) (Asymmetric population) Compare the average utility of one truthful agent when it is in a population of BB agents and one BB agent when it is in a population of truthful agents. An example test-run looks like:

```
python3 auction.py --perms=10 --seed=3 --iters=30 Truthful,1 BBAgent,4
```

What do you observe? Interpret and explain your findings.

For each case, present your findings in a table along with the test parameters used.

**Problem 4.** (5 pts) (**Implementing the VCG mechanism**) Complete the implementation of the VCG mechanism in `vcg.py` by implementing its payment rule.

The allocation rule has already been implemented for you. Notice that the allocation rule ignores all bids below the reserve price.

When implementing the payment rule, **take the reserve price into account.** Suppose $k$ is the last allocated slot, then the last allocated bidder pays the maximum of the next-lower bid $b_{k+1}$ or the reserve price. This payment affects the payment of the other bidders, which is easiest to see using the recursive definition of the VCG payment rule:

$$p_i = b_{i+1}(\alpha_i - \alpha_{i+1}) + p_{i+1}$$

**Problem 5.** (6 pts) (**Switching from GSP to VCG**) In this question, we will explore what happens if Google were to switch from using GSP to VCG and its effect on the short-term revenue. For the following tests use 5 agents and `--perms=1`, and other suitable arguments would be `--seed` and `--iters=50` or above.[4]

(a) What is the revenue of the GSP auction in a population of all balanced-bidding agents?

(b) What is the revenue of the VCG auction in a population of all truthful agents? Explain your findings about GSP vs VCG revenue in the context of the theoretical results discussed in lecture.

(c) What happens if we switch midway from GSP to VCG when all agents are using the balanced-bidding strategy? Run the GSP auction in a population of balanced bidding agents, and midway (at round 24) switch to VCG.

You can do this by using the `--mech=switch` argument. How does switching affect the revenue compared to (a) and (b)? Present your results in a tabular format and provide the test parameters used.

**Problem 6.** (6 pts) (**Analyzing affect of Reserve Prices on GSP and VCG**) Reserve prices are widely used as a mechanism to improve revenue in auctions. In a single-item auction, a reserve price $r$ is $r$ at which auctioneer is willing to an item. Setting a reserve price of course means that the item may sometimes not get sold.

[4]Keep the reserve price set to zero in Problem 4.

This idea generalizes to sponsored-search auctions, in which if an agent's bid-per-click is below the reserve price $r$ then their bid is ignored. In GSP, the agent that gets the lowest of the allocated slot (note that there may be some unallocated slots) pays the maximum of the reserve price $r$ and the bid of the next-lower agent in bid-sorted order.

Reserve prices can be used to increase the sellers revenue. However, we must be careful on how we select it as setting it too high may actually decrease the revenue. In this question, we will investigate the sweet spot, the *optimal* reserve price setting for both the GSP auction.

For the following tests you may use the `--reserve=R` argument to set the reserve price of the simulations. Also use `--perms=1` and `--iters=50` or above.

(a) Investigate the effect of increasing the reserve price on the revenue of the GSP auction when all agents are using the balanced-bidding strategy. Start by setting the reserve price to zero, and then increase it in small increments. Observe how the revenue changes. What is the revenue-optimal reserve price?

(b) Investigate the effect of increasing the reserve price on the revenue of the VCG auction when all agents are using the truthful strategy. Start by setting the reserve price to zero, and then increase it in small increments. Observe how the revenue changes. What is the revenue-optimal reserve price?

(c) How do the reserve prices in (a) and (b) compare to each other?

Tabulate your results and state your test parameters along with your interpretations.

## Extra Credit (Up to 5 pts)

**Problem 7.** (**Class competition**) So far in the assignment, we have not taken the budget into account in determining the bidding strategy. As (a hopefully fun) extra-credit exercise, design a balanced-bidding agent that optimizes the use of its budget.

Your agent will compete in a GSP slot auction with agents submitted by other groups. The auction will have a small reserve price (e.g. 10 cents).

To do well against a variety of strategies, your agent might want to take the competitor's strategy into account. For example, if others spend their budget early, you may want to wait to bid more in later rounds when there is less competition, etc. All the agents in the competition will be ranked based on their *average utility* per round over all iterations.

Points will be awarded as follows: if $t$ teams participate, then the winning team (ranked first) gets $f = \min\{t, 5\}$ points, the second gets $f - 1$ points and so on. [5]

For this part, you must come up with a **team name**. Say your team name is `ephs`, then create a new file `ephsbudget.py` in your repository and implement your bidding agent in it. Don't forget to add, commit and push the file to the repository.

In the write up, you must explain your bidding agent: the choices you made in designing it and how you expect it to perform against other budget-optimized bidding agents.

**Note.** Do not modify the simulation itself for the purpose of the assignment. If you find any bugs in the code, let me know. If you would like improve the logging or stats features, etc., go for it and pass it along so the assignment can be improved for the future.

[5] So if you intend to participate, you should convince at least four other teams to participate as well!

## References

[1] Matthew Cary, Aparna Das, Ben Edelman, Ioannis Giotis, Kurtis Heimerl, Anna R Karlin, Claire Mathieu, and Michael Schwarz. Greedy bidding strategies for keyword auctions. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 262–271, 2007.

[2] Benjamin Edelman and Michael Ostrovsky. Strategic bidder behavior in sponsored search auctions. *Decision support systems*, 43(1):192–198, 2007.