

CS 136 Assignment 6: Sponsored Search Auctions

David C. Parkes

School of Engineering and Applied Sciences, Harvard University

Out: October 26, 2018

Due: **5pm sharp: Friday, November 2 2018**

Submissions to **Canvas**

Late days taken: 1 – 3/6 left.

[**Total: 137 Points**] *Submit a single zip file containing your code and your writeup.* The total number of points includes a small bonus for good performance in an in-class tournament. However, please note that the main purpose of this assignment is to learn and implement different auction mechanisms - winning the tournament is NOT the goal of this assignment! *It is advisable that you don't spend too much time tuning your agent to win the tournament.* This is a group-assignment to be completed by groups of up to 2 students. While you are permitted to discuss your agent designs with other students, each group must develop its own code and write-up its own submission. (If you need help finding a partner please post to Piazza.) Your submissions should be made to Canvas: the code must be in .py files and the writeup of the analysis must be in PDF format. Both partners must contribute to all aspects of the assignment. It should go without saying that you must write all of the code yourself, per Harvard's Academic Integrity Policy. If you use any code whatsoever from any source other than the official Python documentation, you **must** acknowledge the source in your writeup. Moreover, please follow the below instructions exactly when submitting your code:

- Submit a total of 2 files to canvas: your writeup in PDF format, and a .zip file containing only the following .py files: `TEAMNAMEbb.py`, `TEAMNAMEbudget.py`, `vcg.py`. If you're using an IDE, then please **do not** submit the entire project directory created by the IDE. Please **do not** submit .git files.
- If you want to create functions which both your agents can share, please create them in a new file named `TEAMNAMEhelper.py`, and submit that as well (zipped together with the other 3 .py files). Your code should run on an un-modified version of the handout code; in particular, you **should not** modify any of `auction.py`, `stats.py`, `gsp.py`, `history.py`, `truthful.py`, `util.py`, and you should not submit these files.
- Please **do not** use any functions not in the Python standard library. So, you should not use `pandas` or `numpy`. You can do the operations that you'll need to do using a combination of the `math` library as well as functions like `map`, `reduce`, `zip`, `filter`.

1 Introduction

You will program bidding agents to participate in a generalized second-price (GSP) auction. The first agent you will design uses a *balanced bidding* strategy. See Equation 10.16 in the reading.

You will also design an agent to do as well as possible in a competitive environment, and in the presence of daily budget constraints on how much it can spend (see, e.g. the discussion in Section 10.6.2 in the reading). You will also explore the effect that the design of payment rules has on revenue.

2 Setup

Generating .py-files: Pick a group name, perhaps based on your initials, so it will be unique in the class. Run `python start.py TEAMNAME`, substituting your group name for `TEAMNAME`. This will create appropriately named template files for your clients. For example, if your group name was “abxy”:

```
> python start.py abxy
Copying bbagent_template.py to abxybb.py...
Copying bbagent_template.py to abxybudget.py...
All done. Code away!
```

In each of these newly generated files, you will need to change the class name `BBAgent` to your teamname plus client specification (e.g. for the balanced bidding agent: `class Abxybb`). This class name should match the file name exactly, though it can have capital letters (and the filename should be exactly the one generated by `start.py`).

The simulator: You are given an ad-auction simulator. We ignore quality effects, so that in a time period, only the position matters in determining the probability of a click and thus the expected number of clicks received by an ad. The simulator works as follows:

- Time proceeds in discrete periods $(0, 1, 2, \dots, 47)$. Each period simulates 30 minutes and includes multiple auctions, and the 48 periods correspond to a day. The simulator can be used to simulate multiple days by increasing the number of iterations.
- In the first period, an agent’s bid is queried through `initial_bid()` whereas for all subsequent periods, the bid is queried through a call to `bid()`. The simulator tracks money and values in integer numbers of cents. The same bid value is used for every auction that occurs within a period. To encourage competition, the number of available positions is one less than the number of agents in the market.
- Each period is simulated by collecting bids, assigning positions, determining the expected number of clicks received by each bidder, and determining payments and utilities. Sometimes there is a *reserve price* (see Section 4). The positions are assigned in order of submitted bid.
- Number of clicks: Let c_j^t denote the number of clicks received by an ad in position j (from 1 to the number of positions) in period t . For the top position, this follows a cosine curve:

$$c_1^t = \text{round}(30 \cos(\frac{\pi t}{24}) + 50), \quad t = 0, 1, \dots, \quad (1)$$

where `round(x)` returns the nearest integer value to x . The number of clicks starts at 80, falls to 20 by period 24, and increases to 80 by the end of a day. The number of clicks

decreases according to a multiplicative position effect. The number of clicks received by an ad in position j ($j > 1$) in period t is

$$c_j^t = 0.75^{(j-1)} c_1^t, \quad (2)$$

so that it decreases by a factor of 75% from position to position.

- The price in the GSP auction depends on the next highest bid. Given this, the utility in period t to agent i occupying position j is

$$u_i^t = c_j^t (v_i - \text{pay}_{\text{gsp},j}^t) = c_j^t (v_i - b_{j+1}^t), \quad (3)$$

where v_i is the per-click value, and $\text{pay}_{\text{gsp},j}^t$ the price for position j under the GSP auction and equals b_{j+1}^t , which is the amount of the next highest bid (or zero or the reserve if there is no such bid.) Thus is the number of clicks received multiplied by the utility (= value - price) per click.

- Budget constraint: The total payment by an agent in position j in period t is $c_j^t b_{j+1}^t$. Each agent has a *daily budget constraint*, but for most of the assignment this will be high enough not to matter. For the competition this is \$600/day and it will matter. An agent with budget still available at the start of a period is still allowed to bid. For this reason, an agent is allowed to slightly over-spend its budget. Once the budget is exhausted the bid in all subsequent periods of the day must be \$0 (the simulator will ensure this if you try to bid more than \$0.)
- At the start of each day, the per-click value v_i of an agent is sampled independently from the uniform distribution on $[0.25, 1.75]$. The simulator also runs additional permutations of value assignments to agents to improve the statistical significance of the estimated utility. If the number of agents is at most 5 then all permutations of values to agents are tested. Otherwise, 120 random permutations are tested. (You can change this threshold with the `--perms` option)
- The score of an agent is the total utility over all 48 periods, and averaged over all permutations of values to agents that are used. In addition, multiple iterations can be run. This has the effect of repeating the test over multiple days. In this case the score is the average total utility over all permutations and all days. For a single day, the total utility to agent i with value v_i is $\sum_{t=0}^{47} u_i^t = \sum_{t=0}^{47} c_{x_{it}}^t (v_i - b_{x_{it}+1}^t)$, where x_{it} is the position assigned in period t to agent i (and $c_{x_{it}}^t = 0$ if the agent is not assigned to a position in period t). *Any unspent budget has no effect on utility or score.*

Source code: Familiarize yourself with the provided code. You will need to change the agent created by `start.py`, as well as the file `vcg.py`. You'll want to take a look at the simple truthful-bidding agent in `truthful.py`, as well as the the GSP implementation in `gsp.py`.

Testing: Here are some initial test commands to run. The following command gives a list of helpful command line parameters:

```
> python auction.py --h
```

The following command can be used to test the code. It runs the auction with five truthful agents for two periods (rather than the default of 48 periods). The `--perms 1` command forces the simulator to assign only one permutation of value to the agents.

```
> python auction.py --loglevel debug --num-rounds 2 --perms 1 Truthful,5
```

The number of periods defaults to 48 if this is not set. The following command runs the auction with a reserve price of 40 cents. The `--iters 2` command specifies that the 48 periods will be repeated twice, with different value samples (i.e., two days.) The `--seed INT` (where INT is any integer) command sets the seed of the random number generators in the simulator and allows for repeatability.

```
> python auction.py --perms 1 --iters 2 --reserve=40 --seed 1 Truthful,5
```

- Tips**
- *Permutations:* If you're running an experiment on a population of agents that each have the same strategy, use `--perms 1` to make the code run faster. In this case this is probably OK because you're likely comparing the average utility or revenue of the auction, and not looking at specific agents.
 - *Pseudo-random numbers:* If you're trying to track down a bug, or understand what's going on with some specific case, use `--seed INT` to fix the random seed and get repeatable value distributions and tie breaking.

- Comments**
- *Numbering convention:* The positions (slots) in this write-up are 1-indexed (top slot is slot number 1) but 0-indexed in the code. Don't be confused by this!
 - *Ties:* In the case of two tied bids one of the two is randomly chosen to be allocated the higher slot.
 - *State:* The agent you write can access history from the previous period within each day. This could allow for more sophisticated strategies.
 - *Workarounds:* The code is designed so that it's hard to mess up the main simulation accidentally, but because everything is in the same process, it is still possible to cheat by directly modifying the simulation data structures and such. Don't!
 - *Bugs:* If you find bugs in the code, please let us know. If you want to improve the logging or stats or performance or add animations or graphs, feel free :) Send those changes along too.
 - *Questions:* If something is unclear about the assignment, please ask a question on Piazza. Please don't post solution code but it is OK to post code snippets are fine (use your judgment).
 - *Debugging tips:* Be careful about division by zero errors! Some of you had these in problem set 2. Also make sure list indices are always integers!

3 The Balanced Bidding Agent

1. [2 Points] What is your team name?

Solution 1:

les_caribous

2. **[30 Points] Designing a bidding agent** You are given a truthful bidding agent in `truthful.py`.

In `TEAMNAMEbb.py`, write an agent that best-responds to the bids of agents in the previous period. In particular, the agent follows a *balanced bidding* strategy. Consider period t . Let b_{-i}^{t-1} denote the bids from the agents other than i in the period $t - 1$. Suppose there are m positions. Balanced-bidding proceeds as follows:

- Given bids b_{-i}^{t-1} , agent i targets the position j^* that maximizes

$$\max_{j \in \{1, \dots, m\}} [pos_j \cdot (v_i - t_j)], \quad (4)$$

where pos_j is the position effect (which you should estimate by using the number of clicks in the previous round) and t_j is the price the agent would pay for position j given bids b_{-i}^{t-1} . For example, in GSP auction, t_j equals the j -th highest bid in b_{-i}^{t-1} .

- (Not expecting to win) If price $t_{j^*} \geq v_i$ in this target position, then bid $b_i^t = v_i$ in period t .
- Otherwise:

- (a) (Not going for the top) If target position $j^* > 1$, then set bid b_i^t to satisfy

$$pos_{j^*}(v_i - t_{j^*}) = pos_{j^*-1}(v_i - b_i^t). \quad (5)$$

- (b) (Going for the top) If $j^* = 1$, then bid $b_i^t = v_i$.

This strategy is motivated by the balanced bidding discussion the reading (Section 10.4.1).¹ The file `TEAMNAMEbb.py` provides a skeleton of a balanced bidding agent. You will need to complete the code to compute the expected utility for each position and the optimal bid. Note that in the code the bids in the previous round can be accessed by looking at `history.round(t-1)`; the function `slot_info` does this for you.

Solution 2:

Implemented, see `les_caribousbb.py`.

3. **[20 Points] Experimental Analysis** To answer the following questions, run the simulation with 5 agents. By default the budget is \$5000, which is not binding. Leave it this way!

- (a) **[10 Points]** What is the average utility of a population of truthful agents? What is the average utility of a population of balanced bidding agents? *Compare the two cases and explain your findings.* Make use of the `--perms`, `--seed`, and `--iters` commands, e.g. `--perms 1 --seed 2 --iters 200` would be a good starting point.

Solution 3a:

First we present the results for the truthful agents:

¹Cary et al. "Greedy bidding strategies for keyword auctions" *Proc. ACM EC 2007* pp 262–271, show that this balanced bidding strategy converges to the spiteful Nash equilibrium of the GSP auction.

```
python auction.py --loglevel debug --num-rounds 48 --perms 1 --iters 200 --seed 2
Truthful,5
```

```
##### RESULTS #####
```

```
Stats for Agent 0, Truthful
Average spend $1256.30 (daily)
Average utility $372.85 (daily)
```

```
-----
Stats for Agent 1, Truthful
Average spend $1210.67 (daily)
Average utility $317.86 (daily)
```

```
-----
Stats for Agent 2, Truthful
Average spend $1298.95 (daily)
Average utility $339.20 (daily)
```

```
-----
Stats for Agent 3, Truthful
Average spend $1261.15 (daily)
Average utility $368.27 (daily)
```

```
-----
Stats for Agent 4, Truthful
Average spend $1180.72 (daily)
Average utility $329.75 (daily)
```

```
-----
Average daily revenue (stddev): $6207.79 ($1457.17)
Average daily utility over all agents:
```

$$= \frac{372.85 + 317.86 + 339.20 + 368.27 + 329.75}{5} = 345.586$$

Now the results for the balanced bidding agents:

```
python auction.py --loglevel debug --num-rounds 48 --perms 1 --iters 200 --seed 2
Les_caribousBB,5
```

```
##### RESULTS #####
```

```
Stats for Agent 0, Les_caribousBB
Average spend $884.70 (daily)
Average utility $745.14 (daily)
```

```
-----
Stats for Agent 1, Les_caribousBB
Average spend $843.88 (daily)
Average utility $682.82 (daily)
```

```
-----
Stats for Agent 2, Les_caribousBB
Average spend $908.40 (daily)
Average utility $727.15 (daily)
```

```
-----
Stats for Agent 3, Les_caribousBB
Average spend $878.43 (daily)
```

Average utility \$748.78 (daily)

Stats for Agent 4, Les_caribousBB

Average spend \$838.22 (daily)

Average utility \$670.63 (daily)

Average daily revenue (stddev): \$4353.64 (\$1373.14)

Average daily utility over all agents:

$$= \frac{745.14 + 682.82 + 727.15 + 748.78 + 670.63}{5} = 714.904$$

We notice that the average utility for the balanced bidding agents is higher than that of truthful agents. To be precise, it is about twice larger than that of the truthful agents. This is because the balanced bidding agents specifically choose those positions which are utility maximizing for them, instead of just trying to bid what their value is.

As a result, while Truthful agents bid their true value and get assigned whichever position they receive, BB agents don't base their bid simply on their value, but on their utility (maximizing their utility instead of their value). They take into account the cost of the position versus the benefits, and thus might prefer a lower position if the cost difference is larger than the loss of clicks difference, which maximizes their utility. In this way balanced bidding agents have an utility twice as big than the truthful agents.

- (b) [10 Points] In addition, what is the average utility of one balanced-bidding agent against 4 truthful agents, and one truthful agent against 4 balanced-bidding agents? For the new experiment, make use of the `--seed`, and `--iters` commands, but you will now want to run multiple permutations. Note that you can add multiple agents types using:

```
> python auction.py --perms 10 --iters 200 Truthful,4 abxybb,1
```

What does this suggest about the incentives to follow the truthful vs. the balanced bidding strategy?

Solution 3b:

First, we test one balanced-bidding agent against 4 truthful agents:

```
python auction.py --perms 10 --iters 200 Truthful,4 Les_caribousBB,1
```

```
##### RESULTS #####
```

```
Stats for Agent 0, Truthful
```

```
Average spend $1290.90 (daily)
```

```
Average utility $378.20 (daily)
```

```
-----  
Stats for Agent 1, Truthful
```

```
Average spend $1323.17 (daily)
```

```
Average utility $399.83 (daily)
```

```
-----  
Stats for Agent 2, Truthful
```

```
Average spend $1321.70 (daily)
```

Average utility \$418.95 (daily)

Stats for Agent 3, Truthful

Average spend \$1259.49 (daily)

Average utility \$367.63 (daily)

Stats for Agent 4, Les_caribousBB

Average spend \$528.78 (daily)

Average utility \$580.13 (daily)

The total average utility in this case is:

$$= \frac{378.20 + 399.83 + 418.95 + 367.63 + 580.13}{5} = 428.948$$

. The average utility of the truthful agents is:

$$= \frac{378.20 + 399.83 + 418.95 + 367.63}{4} = 391.1525$$

The utility of the balanced-bidding agent is: 580.13

Now, we test one truthful agent against 4 balanced-bidding agents:

```
python auction.py --perms 10 --iters 200 Truthful,1 Les_caribousBB,4
```

```
##### RESULTS #####
```

Stats for Agent 0, Truthful

Average spend \$1365.18 (daily)

Average utility \$678.55 (daily)

Stats for Agent 1, Les_caribousBB

Average spend \$795.99 (daily)

Average utility \$668.94 (daily)

Stats for Agent 2, Les_caribousBB

Average spend \$811.66 (daily)

Average utility \$669.19 (daily)

Stats for Agent 3, Les_caribousBB

Average spend \$800.98 (daily)

Average utility \$675.93 (daily)

Stats for Agent 4, Les_caribousBB

Average spend \$794.63 (daily)

Average utility \$648.10 (daily)

The total average utility is:

$$= \frac{678.55 + 668.94 + 669.19 + 675.93 + 648.10}{5} = 668.142$$

The average utility of the balanced-bidding agents is:

$$= \frac{668.94 + 669.19 + 675.93 + 648.10}{4} = 665.54$$

The utility of the Truthful agent is: 678.55

In short out results are:

Avg(4BBs): 665.54 and 1Truthful: 678.55

Avg(4Truthful): 391.1525 and 1BB: 580.13

When comparing the results we notice that in a population of only Truthful agents, the BB agent obtains a worse result than when the BB agents are the majority. This is explained by the fact that the utility is increased for all the BB agents if they conduct the same strategy. This is because since they all try to maximize their utility and not necessarily their position, then by all playing this strategy together then they all try to maximize their utility, while in a position in Truthful agents then because they always bid their max bet they drive the price higher. Simply put, everyone playing balanced-bidding increases performance for everyone by not necessarily shooting for the top spot, which reduces the prices for each spot by distributing the bid prices more evenly over all the positions. Even though just one BB agent has higher utility than the 4Truthful agents, that utility is lower than the utility when the 4 BB agents dominate, for the reason explained above.

An important result is that even though in conditions where the majority is truthful, BB agents have an advantage. When we have a section of truthful agents in a majority of BB agents they don't do so badly! This interesting fact explains why in real-life situations companies like Google use the GSP auction. Even though there may be many users that submit false claims about their values, it doesn't significantly affect the utility of truthful bidders. Thus the incentives to follow BB decrease until an equilibrium proportion is reached. This is ideal for companies, who want to simplify the process, and it is ideal for the auctioneer because it maximizes their revenue.

4 Experiments with Revenue: GSP vs VCG auctions

In this section we compare the revenue properties of the GSP and VCG auctions with different reserve prices. A reserve price, $r > 0$, sets a minimum price for any position. Used carefully, reserve prices can increase revenue. The reserve price in the GSP works as follows: only bids (weakly) above r can be allocated. The agent in the lowest allocated position pays the maximum of the reserve

price and the maximum bid of unallocated bidders. The VCG auction works in a similar way and is explained below.²

4. **[55 Points] Auction Design and Reserve Prices** Run all simulations with 5 agents. Leave the budget to its default of \$5000.

- (a) **[20 Points]** Complete the code that runs the VCG auction in `vcg.py`. The allocation rule is already implemented. You need to implement the payment rule. Because of the reserve price, it is most convenient to use the recursive form of the VCG payment rule (see the proof of Theorem 10.5 in the reading). In particular, suppose there are 3 bidders with bids weakly greater than the reserve price, and $b_1 \geq b_2 \geq b_3$, and say that b_4 is the fourth highest bid. Bidders 1–3 are allocated the top 3 positions. Let $t_{\text{vcg},i}(b)$ denote the expected payment by bidder i in a single auction given bid profile b . For bidder 3, this is $t_{\text{vcg},3}(b) = \text{pos}_3 \max(r, b_4)$. For bidders 1 and 2, in positions 1 and 2 respectively, this is $t_{\text{vcg},i}(b) = (\text{pos}_i - \text{pos}_{i+1})b_{i+1} + t_{\text{vcg},i+1}(b)$.

Solution 4a:

Implemented, see `vcg.py`.

- (b) **[10 Points]** What is the auctioneer’s revenue under GSP with no reserve price when all the agents use the balanced-bidding strategy? What happens as the reserve price increases? What is the revenue-optimal reserve price? You can set the reserve price in the simulation with the command line argument `--reserve INT` (where INT is the reserve price in cents). Also use `--perms 1` and `--iters 200`.

Solution 4b:

We ran the following command:

```
python2 auction.py -perms 1 -iters 200 -reserve=0 -seed 2 Les_caribousBB,5
```

No reserve price: Average daily revenue (stddev): \$4353.64 (\$1373.14)

Therefore, the auctioneer’s revenue under GSP with no reserve price and all agents balanced-bidding is about \$4300.

Now we increase reserve price to see how it behaves: Reserve price = 10: Average daily revenue (stddev): \$4353.64 (\$1373.14)

Reserve price = 20: Average daily revenue (stddev): \$4429.40 (\$1411.08)

Reserve price = 30: Average daily revenue (stddev): \$4668.67 (\$1483.59)

Reserve price = 40: Average daily revenue (stddev): \$4648.91 (\$1265.00)

Reserve price = 50: Average daily revenue (stddev): \$4908.94 (\$1403.20)

Reserve price = 60: Average daily revenue (stddev): \$5065.57 (\$1343.69)

Reserve price = 70: Average daily revenue (stddev): \$4961.31 (\$1507.40)

Reserve price = 80: Average daily revenue (stddev): \$5027.22 (\$1447.33)

Reserve price = 90: Average daily revenue (stddev): \$5239.81 (\$1574.98)

²One way to think about it is that the reserve is made operational by including a “dummy bidder” whose bid is the amount of the reserve, and ignoring all bids with value less than r .

Reserve price = 100: Average daily revenue (stddev): \$4833.19 (\$1732.41)
Reserve price = 110: Average daily revenue (stddev): \$4703.17 (\$2166.00)

From the iteration we have run here, we can see that under the current auction system, it looks like the revenue-optimal reserve price is about 60-70-80 (measured in cents). The auctioneer's revenue seem to start to plateau at this point. Naturally there is some anomaly at 90 because with such a high standard deviation, it is hard to get accurate results, but we can imagine the revenue optimal price to be about 60-70-80. This is because, as we can see, the auctioneers revenue increases as the reserve price increases, and eventually plateaus at around reserve=60, and then starts decreasing as the reserve price increases further. The reason for this is simple. The bidders value is distributed between 0 and 200. At a very low reserve price, the agents have the reserve price below their value and that gives them much more room for balanced bidding (bidding less than their value for a lower position that better optimizes their utility. However, as reserve price increases, it squeezes them and they are now force to bid closer and closer to their actual value and therefore their balanced-bidding strategy start being ineffective. This leads to higher auctioneers revenues (as we can see the revenue approaches the revenue under truthful bidders) because bidders have less control and must bid higher, resulting in higher payments (since in GSP, payment is second bid). However, when the reserve price exceeds the bidders value, then the bidder does not bid because it would derive negative utility, and so some bidders are off the market. This reduces revenues since not all bidders participate. This is why we see a slow rise, but a rapid fall in auctioneers revenues.

- (c) **[10 Points]** What is the auctioneer's revenue under VCG with no reserve price when all agents are truthful? What happens as the reserve price increases? Explain your findings and compare with the results of part (b). Again use the `--perms`, `--seed`, and `--iters` commands, e.g. `--perms 1 --seed 2 --iters 200`.

Solution 4c:

We ran the following command (we ran with -seed 1 without realizing, but the number of seeds shouldn't affect the trend that we are looking for here so we decided to keep it this way instead of re-running everything with 2 seeds). All that matters really is consistency.

```
python2 auction.py -perms 1 -iters 200 -reserve=0 -mech=vcg -seed 2 Truthful,5
```

No reserve price: Average daily revenue (stddev): \$4231.64 (\$1220.41)

Therefore, the auctioneer's revenue under VCG with no reserve price and all agents truthful is about \$4200.

Now we increase reserve price to see how it behaves:

Reserve price = 10: Average daily revenue (stddev): \$4231.64 (\$1220.41)

Reserve price = 20: Average daily revenue (stddev): \$4231.64 (\$1220.41)

Reserve price = 30: Average daily revenue (stddev): \$4270.93 (\$1307.98)

Reserve price = 40: Average daily revenue (stddev): \$4446.27 (\$1079.27)

Reserve price = 50: Average daily revenue (stddev): \$4634.40 (\$1210.92)
 Reserve price = 60: Average daily revenue (stddev): \$4661.68 (\$1106.69)
 Reserve price = 70: Average daily revenue (stddev): \$4897.41 (\$1188.77)
 Reserve price = 80: Average daily revenue (stddev): \$5115.31 (\$1422.16)
 Reserve price = 90: Average daily revenue (stddev): \$5041.38 (\$1561.15)
 Reserve price = 100: Average daily revenue (stddev): \$4997.27 (\$1795.65)
 Reserve price = 110: Average daily revenue (stddev): \$5047.89 (\$2113.02)
 Reserve price = 120: Average daily revenue (stddev): \$4630.32 (\$2122.53)

As we can see, the situation here is more or less similar to the GSP situation. As the reserve price increases, the auctioneer's revenue increases. This is logical and follows the same argument we made in 4b. However, here, the reserve price is higher, more in the 90-100 range. This makes sense, under VCG and truthful bidding, there is less room for agents to game the auction and therefore a higher reserve price leads to a higher auctioneer's revenue because the agents bid their true value all the time. This is another proof that strategy-proofness is ideal for its simplicity and revenue optimality.

- (d) **[10 Points]** Fix the reserve price to zero. Explore what might happen if a search engine switched over from the GSP to VCG design. For this, run the balanced-bidding agents in GSP, and at period 24, switch to VCG, by using the `--mech=switch` parameter. What happens to the revenue? Again use the `--perms`, `--seed`, and `--iters` commands, e.g. `--perms 1 --seed 2 --iters 200`.

Solution 4d:

We fix reserve price at 0, so we run the following:

```
python2 auction.py -perms 1 -iters 200 -reserve=0 -mech=switch -seed 2 Les_caribousBB,5
```

```
##### RESULTS #####
```

```
Stats for Agent 0, Les_caribousBB
```

```
Average spend $792.70 (daily)
```

```
Average utility $837.14 (daily)
```

```
-----
```

```
Stats for Agent 1, Les_caribousBB
```

```
Average spend $758.75 (daily)
```

```
Average utility $767.95 (daily)
```

```
-----
```

```
Stats for Agent 2, Les_caribousBB
```

```
Average spend $820.33 (daily)
```

```
Average utility $815.23 (daily)
```

```
-----
```

```
Stats for Agent 3, Les_caribousBB
```

```
Average spend $791.96 (daily)
```

```
Average utility $835.24 (daily)
```

```
-----
```

```
Stats for Agent 4, Les_caribousBB
```

```
Average spend $760.58 (daily)
```

```
Average utility $748.27 (daily)
```

Average daily revenue (stddev): \$3924.32 (\$1287.80)

This is a bit hard to evaluate since the standard deviation is so large, but what we can see as we keep increasing the reserve price is that the revenue originally is somewhat between GSP and VCG (greater than VCG but less than GSP) and slowly converges. This is not exactly represented at reserve price = 0 because of the very large standard deviation and the fact that at a reserve price of 0, the situation is much more analogous. Just as previously, as the reserve price increases, the revenues eventually converges, decreasing revenue as the reserve price increase eventually (just like previously), because the higher the reserve price, the less bidding the agents do (since at some point it is above their value). This all makes sense and is what is to be expected, although the individual results for specific reserve prices are harder to evaluate in isolation because of the large standard deviation.

- (e) **[5 Points]** In one paragraph, state what you learned from these exercises about agent design, auction design, and revenue? (There is no specific right answer).

Solution 4e:

We mainly learned a lot about behaviors of auction systems. We learned that in a non-strategy proof system, there is a lot of gain to be made from a better strategy than truthful bidding, and that makes a good case for why it is important for an auction to be strategy proof. The balanced-bidding agents performed much better and generated much more utilities for themselves and much less revenues for the auctioneer, so the auctioneer clearly prefers strategy-proof auctions. It also gives the auctioneer much more reliability on its expected revenue. As we saw, increasing reserve price slowly brought the agents closer to being truthful agents (from being forced to) and improved the auctioneers revenue quite significantly. Also, we gained a much better understanding of the way position auctions work and how the reserve price plays a role in them. As we saw, the optimal revenue reserve price was different for each auction type and it confirmed our understanding of the importance of strategy-proofness in auctions, both for the auctioneer and for the agents.

5 The Competition

5. **[30 Points] Budget constraints** The balanced-bidding agent does not consider the budget constraint when deciding how to bid. In the final part of the assignment, your task is to design a *budget-aware agent*. This agent will compete in the simulated GSP auction against the agents submitted by other groups. You might like to test your design against a variety of other strategies. For example, you could write an agent that measures competition, or tries to drive up the payments of other agents so that they pay more and exhaust their budgets! Example ideas include:

- Trying not to bid too much when prices are high.
- Try to bid when other agents are not bidding very much and the price is lower.

For the purpose of the competition, the daily budget constraint will be set to

\$600 (use the --budget flag). We will run a GSP auction with a small reserve price. Auctions will contain around 5 agents. The competition will be structured as a tournament, with agents placed into groups of 5 or 6 and the top few agents making it into a semi-final and final. The precise structure of the tournament will depend on the number of agents submitted.

- (a) **[25 Points]** In `TEAMNAMEbudget.py`, write your competition agent. Describe in a few sentences how it works, why it is designed this way, and how you expect it to perform in the class competition. *You are not expected to spend many hours on writing an optimal agent, unless you want to. Consider a few possible strategies, try them out, pick the best one.*

Solution 5a:

Agent implemented, see `les_caribousbudget.py`.

Our analysis here goes in two steps. We first describe the analysis of what we implemented for the agent, and we describe the analysis of what we wish we could have implemented more had we had more time (overly optimistic implementations that unfortunately we weren't able to do, but felt were important to our analysis):

Our Agent Analysis:

- a The tournament agent will be moderately based on the balanced bidding agent. One thing that the balanced bidding agent didn't use is assuming a structure to the clicks at each period. With this information we can make higher bids in periods where we know that we are getting a relatively higher utility.
- b Even though it's true that in the real world we are not provided information like the number of clicks per round. In real life we can still make predictions and inscribe our beliefs about the click rate through functional approximations. For our problem we will use the information provided, namely:

$$c_1^t = \text{round}(30 \cos(\frac{\pi t}{24}) + 50), \quad t = 0, 1, \dots, \quad (6)$$

As well as the fact that the number of clicks decreases according to a multiplicative position effect. The number of clicks received by an ad in position j ($j > 1$) in period t is

$$c_j^t = 0.75^{(j-1)} c_1^t, \quad (7)$$

Given this, the utility in period t to agent i occupying position j is

$$u_i^t = c_j^t (v_i - \text{pay}_{\text{gsp},j}^t) = c_j^t (v_i - b_{j+1}^t), \quad (8)$$

Let's first plot how the click rate changes for a few different positions for different periods. (This still conveys relevant information since we have the same value for a click for an entire day). We have the following functions plotted:

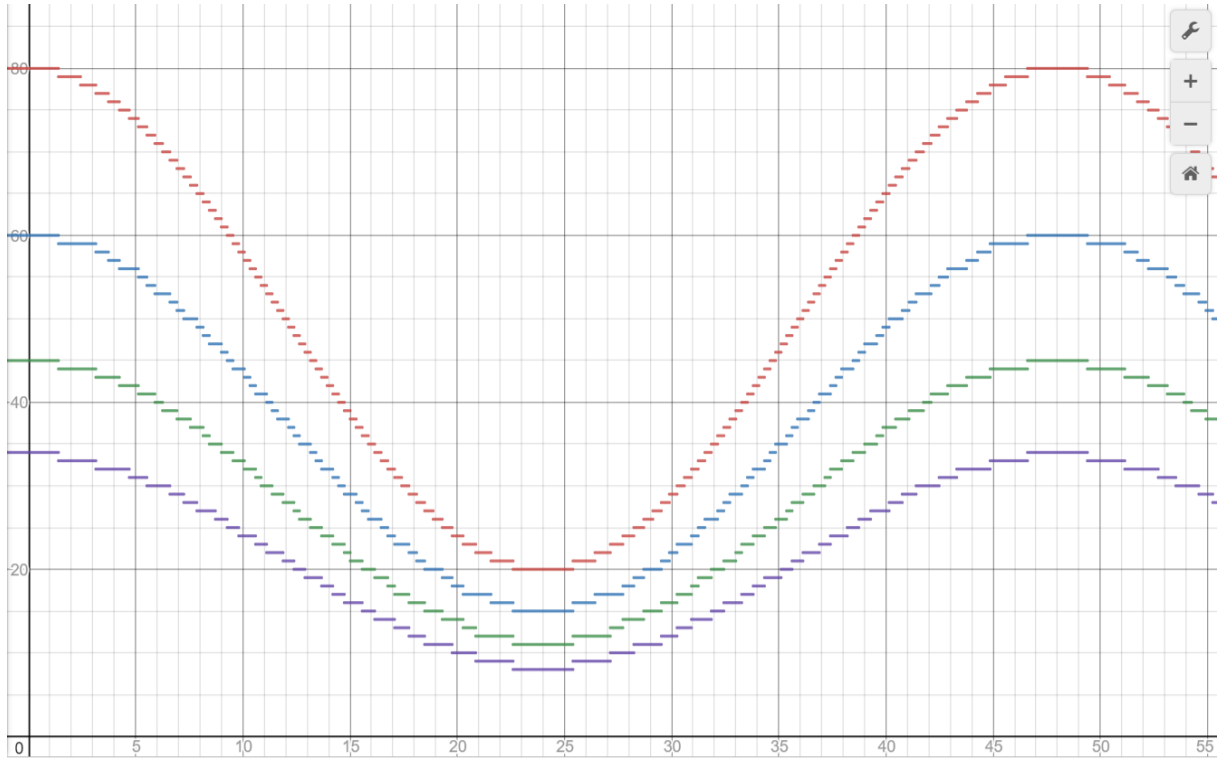


Figure 1: Click rate per period per position





1		$y = \text{round}\left(30 \cdot \cos\left(\frac{\pi}{24}x\right) + 50\right)$	×
2		$g = \text{round}\left((0.75) \cdot \left(\text{round}\left(30 \cdot \cos\left(\frac{\pi}{24}x\right)\right)\right)\right)$	×
3		$h = \text{round}\left((0.75)^2 \cdot \left(\left(\text{round}\left(30 \cdot \cos\left(\frac{\pi}{24}x\right)\right)\right)\right)\right)$	×
4		$f = \text{round}\left((0.75)^3 \cdot \left(\text{round}\left(30 \cdot \cos\left(\frac{\pi}{24}x\right)\right)\right)\right)$	×

Figure 2: The functions representing the click rate

Firstly we need to explain why we applied round a second time on the functions g,h,f. The reason is that once we multiply by the position to position factor 0.75% we may obtain total utility values which are less than cents, which we will simply round.

- c The important information to extract from the graph is that the difference between positions is not linear, nor do positions have the same importance in different parts of the day. For this reason it's clear that in an auction we should prioritize more higher positions at the beginning and at the end of the day, and not value a lot the difference between positions close to the 24th period. Even more importantly we need to make sure that we will have enough money close to the end of the day such that we can make full use of the advantage of higher positions.

These two are the main elements we implemented in our budget agent. For the expected utility computation, we implemented the cosine cyclical utility valuation to prioritize bidding on better periods (higher in the cosine), and we implemented the position discounting to further refine our play. We designed it this way to maximize utility on the cheaper periods and thus expending less of our budget for a similar utility gain. We wish we could have also added an element to try to force others to expand more of their budget unnecessarily, but unfortunately ran out of time. We expect that this will help us perform better by letting other agents outbid each other for large positions at a large cost on other periods, while we will outbid them for large positions and utility at a lower cost on cheaper periods, and thus we will gain large utility on these periods and eventually other agents will run out of budget.

The second main element we implemented is the period valuation. We know from the previous element we implemented that we should prioritize higher positions at the beginning and end of the day, so when t is small or when t is large. This is exactly what we do. For periods less than 12 (so the first 6 hours, since t goes from 0 to 47, block of 30 minutes), we bid according to the balanced-bidding agent, which performed better than truthful bidding. We try to maximize our utility during these periods. We do the same for t greater than 35 to catch the last 6 hours. For the middle 12 hours of the day, we know it won't be optimal periods. Therefore, we only bid incrementally more than the reserve price. We thought of bidding 1 cent more, but figured many other agents might do the same, so we bid 5 cents more to easily outbid them at a incrementally small cost and gain small utility incremental benefits. We expect this will help us perform better as well and make small incremental gains in periods for which we don't battle for the top spot, while capitalize in cheaper periods.

The whole gist of our strategy is to capitalize on cheaper periods and go "all-in" in a utility-maximizing fashion during these periods, and be "slightly better than nothing" in expensive periods to make a small incremental gain. One main downside of this strategy is that, depending on the way other agents play, we might never have really good periods to play for cheap and might end up playing a "defensive" strategy for more periods than we would like to play. This is hard to evaluate but definitely something that could harm us in the tournament.

Overall, we expect our agent to do fine, yet not incredible in the tournament. While we do think the two strategies we implemented provide a good "bang-for-buck" improvement upon the simple balanced-bidding, there are many other strategies we wish we could have implemented, like the one described below, and the strategy of trying to purposely "slow-down" other agents by having them expense their budget, by bidding a bidding scheme in periods in which we know we won't win so to raise the price for others. **Analysis we wish we had time to implement:**

An hypothesis we have is that realistically agents either don't have a scheme of betting that takes into account different periods, or that even if they do they are okay with spending more money in the beginning of the day. For this reason we could approximate their money spending factor per period as a monotonically decreasing function, say an exponential function with a negative coefficient. For this reason if we multiply our prioritization of positions by $(1-H(t))$ where $H(t)$ is how the relative interest of our competitors is going to change as t passes, we will have it that we are not over-competing when there are multiple people desiring a position, and we are superior when most of our competitors have spent their budget.

For this reason a simple approximation to the way we bid is bounding the bid value by the bid prescribed by balanced bidding re-weighted by $(1-H(t)) * Round(\cos(\pi * \frac{t}{24})) + C$. Let's assume for simplicity $H(t) = e^{-t*\alpha}$

Then our adjustment of the bidding value will be a normalization by:

$$(1 - e^{-t*\alpha}) * Round(\cos(\pi * \frac{t}{24})) + C$$

We also need to make sure that we are not simply the most interested once everything is almost complete, so we add an extra hyper-parameter g , which will adjust the number of days before the end when we are most interested.

We have the following function for the absolute interest (before normalization):

$$(1 - e^{-(t+g*\alpha)}) * Round(\cos(\pi * \frac{t+g}{24})) + C$$

An example plot is the following:

Here we have the following hyper-parameter values:

To use these for our tournament agent we will re-weight the bid on each period by the

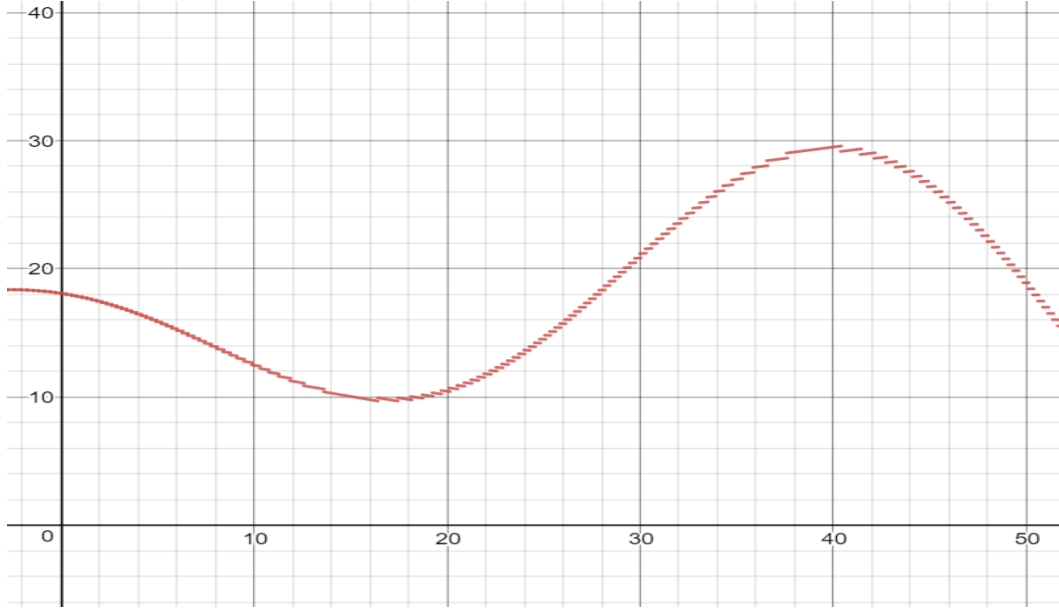













Figure 3: Absolute interest

interest on that particular period. The sum of the function of our interest is ≈ 890 , so for each each period we will use a faction of our total available money equal to:

$$\frac{1}{890} \left((1 - e^{-(t+9)*0.011}) * \text{Round}(\cos(\pi * \frac{t+9}{24})) + 17 \right)$$

1	 $y = \left(1 - e^{(-\alpha \cdot (x+g))}\right) \cdot \text{round}\left(30 \cdot \cos\left(\frac{\pi}{24}\right)\right)$ 
2	 $g = 9$  
3	 $C = 17$  
4	 $\alpha = 0.011$  

- (b) [5 Points] Win the competition (!) Likely parameters for the competition are
`./auction.py --num-rounds 48 --mech=gsp --reserve=10 --iters 200` (followed by
the list of submitted agents)

Solution 5b:

See our submitted agents, `les_caribousbudget.py`.