CS 186 Assignment 3: Sponsored Search Auctions

Professor David C. Parkes
School of Engineering and Applied Sciences, Harvard University
Out: Thursday, Feb 21, 2013
Due: **5pm** sharp: **Friday, March 1, 2013**
Submissions to `Dropbox for assignments in iSites`

[**Total: 135 Points**] 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 write their own code and explanations. In addition, there will be a small bonus for good performance in an in-class tournament. If you want a partner and don't have one, post to piazza as early as possible. Your submissions should be made to the assignment Dropbox on iSites, as a zip file containing code and writeup of analysis.

1 Introduction

You will program bidding agents to participate in a generalized second-price (GSP) position auction. One bidding agent follows a strategy inspired by the spiteful bidding from the chapter notes. The second agent is designed to do as well as possible in competition against other agents. A twist in the competition is that, as in the real world, your agent has a daily budget limit. In addition, you will explore the effect of payment rules on revenue properties of the GSP auction.

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 GROUPNAME`, substituting your group name for `GROUPNAME`. 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`).

The simulator: You are given an ad-auction simulator. It has the following structure:

- Time proceeds in discrete rounds $(0, 1, 2, \ldots, 47)$. Each round simulates 30 minutes, so the simulation models one full day.

- In the first round, your bid is queried through `initial_bid()` whereas for all subsequent rounds, your bid is placed through a call to `bid()`. The simulator tracks money and values in integer numbers of cents. One bid value is used for all auctions in a given round. To encourage competition, the number of available positions is one less than the number of agents in the market.
- Each round is simulated by collecting bids, assigning positions, calculating the total number of clicks received in each position, and determining payments and utilities. Sometimes there is a reserve price, and bids below the reserve price are ignored with the price to others set to be at least the reserve.
- Supply of clicks: In the simulator, a single auction is run in each round to determine the position of each bidder. Given this, let c_j^t denote the *total* number of clicks received by an ad in position j (from 1 to the number of positions) in round t (from 0 to 47). For the top position, this is given by a cosine shape:

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

  where round(x) denotes rounding to the nearest integer value. The effect is that the number of clicks starts at 80, falls to 20 by round 24, and then increases back to 80 by the end of a day. For the jth position ($j > 1$), the number of clicks is,

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

  and falling by a multiple of 0.75 from position to position.
- In the GSP auction, the price is set by the bid amount in the next lower position. Given this, the utility in round t to agent i occupying position j is calculated as

$$u_i^t = c_j^t(w_i - p_j^t) = c_j^t(w_i - b_{j+1}^t), \tag{3}$$

  where b_{j+1}^t is the amount of the next lower bid (or zero or the reserve if there is no such bid.) This is the number of clicks received multiplied by the utility (value-price) per click.
- Budget constraint: The total payment collected in round t for being in position j is $c_j^t b_{j+1}^t$. Each agent has a daily budget constraint, that for the most part we set high enough not to matter. But for the purpose of the competition it is \$600/day. An agent with budget remaining at the start of a round can bid, and may then over-spend slightly over the entire day. Once the budget has been reached, all bids in subsequent rounds in the day must be \$0 (the simulator will ensure this if you try to bid more than \$0.)
- For a single day, per-click values are determined by sampling n values (for n bidders) uniformly at random on \$0.25 to \$1.75. The simulator also runs additional permutations of value assignments to agents to improve the statistical validity of the result. If $n \leq 5$, all permutations 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 rounds, averaged over all permutations. In addition, multiple "iterations" can be run, which has the effect of repeating for a sequence of days with a new set of values sampled at the start of each day. In

this case the score is the average total utility over all permutations and all days (or iterations). For a single run over 48 rounds, the total utility to agent i with value w_i is $\sum_{t=0}^{47} u_i^t = \sum_{t=0}^{47} c_j(w_i - b_{j+1}^t)$, where j is the position assigned in round t (and $c_j = 0$ if the agent wins no position in round t.) 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:

```
> ./auction.py --h
```

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

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

The number of rounds 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 rounds will be repeated twice, with different value samples.

The `--seed INT` (where INT is any integer) command sets the seed of the random number generators in the simulator and allows for repeatability.

```
> ./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

- State: The agent you write can access history from the previous round 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!
- If you find bugs in the code, 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.
- If something is unclear about the assignment, please ask a question on Piazza. Please don't post solution code, but otherwise code snippets are fine (use your judgment).

3 The Balanced Bidding Agent

1. **[30 Points] Designing a bidding agent**

   You are given a truthful bidding agent in `truthful.py`.

   In `GROUPNAMEbb.py`, write an agent that best-responds to the bids of agents in the previous round. In particular, the agent follows "balanced bidding." Let b_{-i} denote the bids from the other agents in the last round $t-1$. In period t, balanced-bidding proceeds as follows:

   - Given bids b_{-i} by other agents, target the position k^* that maximizes
   $$\max_k s_k(w_i - p_k), \tag{4}$$
   where p_k is the price the agent would pay for position k given bids b_{-i}.
   - (Not expecting to win) IF price $p_{k^*} \geq w_i$ in this target position, then bid $b_i = w_i$ in the next round.
   - Otherwise:
     (a) (not going for top) if target position $k^* > 1$, then set the bid b_i in the next round to satisfy
     $$s_{k^*}(w_i - p_{k^*}) = s_{k^*-1}(w_i - b_i). \tag{5}$$
     (b) (going for top) if $k^* = 1$, bid $b_i = w_i$

   The balanced bidding strategy is motivated by spiteful bidding explained in the notes.[1] The file `TEAMNAMEbb.py` provides you with a skeleton of a balanced bidding agent. You will need to compute a vector of expected utilities for each position and the optimal bid using the formulas above.

2. **[20 Points] Analysis of the GSP agents**

   To answer the following questions, run the simulation with 5 agents. By default the budget is $5000 and not binding. Leave it this way.

   (a) **[10 Points]** What is the average utility of a population of only truthful agents? What is the average utility of a population of only 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.

   (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:

   ```
   > ./auction.py  --perms 10 --iters 2 Truthful,4 abxybb,1
   ```

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

[1] In fact, Cary et al. "Greedy bidding strategies for keyword auctions" *Proc. ACM EC 2007* pp 262–271, show that for a fixed set of agents with fixed values, balanced bidding leads the agents to converge on an envy-free Nash equilibrium.

4 GSP vs VCG auction

For the following questions, we introduce the idea of a *reserve price* $r > 0$. This is the minimum price at which the auctioneer is willing to sell a position. Reserve prices can be used to increase the revenue of the seller by making the auction more competitive. Only those bids weakly above r are considered. In the GSP auction, the agent in the lowest allocated position pays the maximum of the reserve price and the bid of the next lower bidder. We will also study the VCG auction, which provides a truthful alternative to GSP.

3. **[55 Points] Auction Design and Reserve Prices**

   Run all simulations with 5 agents. Leave the budget to its default of $5000.

   (a) **[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`.

   (b) **[20 Points]** Implement 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 (see the proof of Theorem 8.5 in the chapter notes). In particular, suppose there are 3 bidders with bids at least the reserve, and $b_1 \geq b_2 \geq b_3$ (and b_4 the fourth highest bid), so that they are allocated positions 1, 2 and 3. The per-impression payment for bidder 3 is $t_{vcg,3}(b) = s_3 \max(r, b_4)$, and $t_{vcg,i}(b) = (s_i - s_{i+1})b_{i+1} + t_{vcg,i+1}(b)$ for bidders $i = 2$ and $i = 1$ (in positions 2 and 1, respectively.) Make sure that the code is ultimately computing the per click price.

   (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 (a).

   Again use the `--perms`, `--seed`, and `--iters` commands, e.g. `--perms 1 --seed 2 --iters 200`.

   (d) **[10 Points]** When the reserve price is zero, explore what might happen if Google switched from GSP to VCG: run the balanced bidding agents against GSP, and at round 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`.

   (e) **[5 Points]** *Bigger picture*: In one paragraph, what did you learn from these exercises about agent design and implementation, mechanism design, and revenue? There is no specific 'right' answer here.

5 The Competition

4. **[30 Points] Budget constraints**

The balanced bidding agent does not consider that the agent has a budget. Here, you are asked to write a budget-aware agent. This agent will compete in the GSP auction with the agents submitted by other groups, so you may want to test it against a variety of strategies.

For example, you could write an adaptive agent that measures competition, or drives up the payments of other agents, or similar. Some things that you might consider are:

- Not to spend too much when the click-through rates are low (during the middle of the day)
- Try to bid in rounds where other agents are not competing very intensively (e.g., either to save their budget, or having exhausted their budget)

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 then a final. The precise structure will depend exactly on the number of agents submitted.

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

(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)