Computation and Economics - Fall 2014
Assignment #4: Strategic Bidding and Revenue in Ad-auctions

Professor Sven Seuken
Department of Informatics, University of Zurich
Out: Tuesday, October 21, 2014
Due: **12:15** sharp: **Tuesday, October 28, 2014**
Submissions should be made to `schuldenzucker@ifi.uzh.ch`.
For submission format, check description below.

[**Total: 100 Points**] You may complete this assignment either by yourself, or in a group of **up to 2 students**. Note that if you are working in a group of 2, we expect that both students contribute roughly equally to the assignment and both understand all parts of the solution and write-up. Please briefly (1-2 sentences) describe how you split up the work (e.g., "we did everything together, sitting in front of the same computer" or "student A implemented part (1) and student B implemented part (2)", etc.

While you are permitted to discuss your ideas with all students as much as you like, each group must write their own code and explanations. You are not allowed to share code/write-ups with other groups. Also, there will be a small bonus for agents that perform well in the class tournament.

If you want a partner and don't have one, write an email to the TAs as soon as possible and we will play match makers. Your submissions should be made via email to `schuldenzucker@ifi.uzh.ch` as attachments: code in .java-files and writeup of analysis as PDF. Sending .zip-archives is also fine.

Introduction

In this assignment, you are asked to write two bidding agents for an ad auction using GSP, and then do some mechanism-side experiments to explore the effects of different payment rules and reserve prices. You will find the java source code on the class webpage.

Points in this assignment will be awarded for correctness, clarity, presentation of results, and quality of the interpretations. Always document your simulation parameters and which agents are used! Present your results in an accessible way using graphs, tables, etc.

Setup

Renaming .java-files: Pick a group name `GroupName`, based on your initials, so it will be unique in the class. Change the name of the files `BBAgentABC.java` and `BudgetABC.java` to `BBAgentGroupName.java` and `BudgetGroupName.java`, respectively. For example, if your group name was `DSTA`, the files would be called `BBAgentDSTA.java` and `BudgetDSTA.java`. In each of these files, you will need to change the class names accordingly.

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

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

- Values per click, w_i, are uniformly distributed between \$0.30 and \$1.80, independently for each agent. In the first round, your bid is queried through `initialBid()` whereas for all subsequent rounds, bids are placed through a call to `bid()`. The simulator tracks money and values in integer numbers of cents.
- If a non-zero reserve price is set, bids less than the reserve will be ignored. To encourage competition, the number of available slots in any time period is one fewer than the number of active bidders (unless there is only one active bidder). An active bidder is one with a bid above the reserve.
- Click-through rate of the highest slot: The number of clicks in the *first slot* in each round, c_1^t, follows a cosine shape:

  $$c_1^t = R(35\cos(\pi t/24) + 40), \quad t = 0, 1, \ldots$$

  where $R(\cdot)$ denotes rounding to the nearest integer value. This means, if we use 48 rounds (which models a full day) for the simulation, then the clicks for the top slot start at 75 in the first round, go down to 5 in round 24, and rise again to 75 in round 48 (with an average of 40).
- Click-through rate of other slots: Let c_1^t denote the number of clicks that the first slot receives in the current round. Then, the number of clicks in the i-th slot is given by

  $$c_i^t = 0.7^{i-1} c_1^t,$$

  which closely fits the real-world click dropoff.
- Each period is simulated by collecting bids, assigning slots, calculating clicks, and determining utilities. We first consider the GSP auction. Thus, agents are allocated to slots according to their bids (in order of decreasing bids), and the price per click they have to pay is the the bid of the agent allocated to the next lower slot. For every round the utility of agent i occupying slot j is calculated as the number of clicks of that slot, c_j^t, times the difference between the agent's value per click, w_i and the price per click, p_j^t:

  $$u_i^t = c_j^t(w_i - p_j^t) = c_j^t(w_i - b_{j+1}^t).$$

- Payments and budget: In every round, the payment per allocated agent is $c_i p_i$ (i.e. the clicks of the slot it gets times the per-click price of that slot). The default budget over the entire day is \$5000. As long as you still have a positive budget, you will still be allowed to bid in the next round, even if your submitted bid will mean you will over-spend in this round. However, you can over-spend your budget in at most one round (and end up with a small deficit). If the sum of the payments from all previous rounds is greater than or equal to your budget (i.e. if you have spent or even overspent your budget), you can only bid \$0, or if you bid more than \$0, the simulation will reduce your bid to \$0.
- The per-click values w_i are drawn from a random distribution and then shuffled through the following process: first, n random values are pulled from a uniform distribution on \$0.30 to \$1.80. Note that there are $n!$ different ways of assigning these values to the n different agents. If $n! \leq 120$, simulations will be run for each of these $n!$ permutations. If $n!$ is greater than 120, then only 120 permutations (i.e., a subset of all possible permutations) will be randomly chosen. (You can change this threshold with the `maxperms` option in the parameter file – for debugging, you'll often want this to be 1).

- Winning: The winners will be determined based on cumulative utility over every round of each instance, i.e., for one instance this is $\sum_{t=1}^{48} u_i^t = \sum_{t=1}^{48} c_j(w_i - p_j^t)$.[1]

Source code: Familiarize yourself with the provided code. You will need to change the agents `BBAgentGroupName.java` and `BudgetGroupName.java` (after you've renamed them according to the instructions on page 1, i.e. Renaming .java-files), as well as the file `VCG.java`. You'll want to take a look at the simple truthful-bidding agent in `TruthfulAgent.java`, as well as the GSP implementation in `GSP.java`.

The code comes with a parameter file, called `parameters.txt`, which you can use to specify how to run the simulation. The simulator will read in those values and set the parameters accordingly. For example, you can define the number and type of the agents that compete in the auction, how many rounds the whole auction should have, and which auction mechanism to use. Have a look at `parameters.txt`, because you'll need to change the parameters later to solve the exercises.

Testing: Here are some initial parameter settings that you can use the test the program.

Setting 1 runs the auction with five truthful agents for two rounds. The `perms 1` command forces the simulator to assign only one permutation of the same values to the agents.

Setting 1

```
loglevel debug
numround 2
maxperms 1
agents TruthfulAgent 5
```

Setting 2 runs the auction with a reserve price of 40 cents. The `numiters 2` command specifies that the auction will be run twice, but with two different value draws. This can be useful in combination with the `--seed INT` (where INT is any integer) setting, which initializes the pseudo-random number generator with `INT`, which gives you repeatable value distributions. These two together can be used to compare different agent populations.

Setting 2

```
maxperms 1
numiters 2
reserve 40
seed 200
agents TruthfulAgent 5
```

Tips

- Permutations: If you're looking at a symmetric population of agents, use `maxperms 1` to make the code run faster – if the strategies are the same, who has what value does not effect the outcome.

[1] Note what this means for the role of the budget in determining your overall utility: any part of your budget that you spend will count against you, and any part of your budget that is left over will count in your favor. That's because every payment you make appears as a negative term in your utility function. In particular this also means that purposefully overspending your budget does not make sense, because that will also count against you. Really you should just focus on the sum of your utilities.

- 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.

Problem Set

1. **[35 Points]** Designing a bidding agent

   You are given a truthful bidding agent in `TruthfulAgent.java`. In `BBAgentGroupName.java`, write a best-response agent that uses the so-called *balanced-bidding algorithm*. This algorithm uses a similar idea to what we saw in the lecture notes regarding the bidding strategies that satisfy the max-spite condition (see Definition 8.3 of the lecture notes).

   The balanced bidding algorithm works very similarly. The motivation behind this algorithm is to make your bid be a best response to the other agents' bids from the previous round. This involves picking as target slot the one that optimizes utility assuming the other agents don't change their bids. There are many possible bids that would achieve the target slot, and balanced bidding says to pick the particular bid b that makes an agent indifferent between getting the targeted slot (whose price is the next lower bid) and getting the slot just above at price b (which could happen if you get 'jammed' by the bidder above.)

   So the balanced bidding strategy for player i is as follows. Fix the bids of the other players, b_{-i}, to be the ones from the previous round. Now player i

   - targets the slot i^* that maximizes his utility:

     $$i^* = \arg\max_k(q_k(w_i - p_k)),$$

     where p_k is the price he would have to pay to get slot k,
   - chooses his bid b_i for the next round so as to satisfy the following equation:

     $$q_{i^*}(w_i - p_{i^*}) = q_{i^*-1}(w_i - b_i).$$

   If $i^* = 0$ (i.e. if the target slot is the top slot), then we arbitrarily set $q_{i^*-1} = 2q_{i^*}$ to make the strategy well-defined. If the resulting bid b_i would be above the true value w_i, we bid truthfully instead.

   The file `BBAgentGroupName.java` provides you with a skeleton of a balanced bidding agent. The only two things you have to do is to compute a vector of expected utilities for each slot and the optimal bid using the formulas above.

2. **[10 Points]** Agent analysis

   To answer the following questions, run the simulation with **5 agents** and the default budget of **$5000**.

   (a) **[5 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 clever use of the `perms`, `seed`, and `numiters` commands, e.g. `perms 1 seed 2 numiters 50` would be a good starting point.

(b) **[5 Points]** Compare the performance (in terms of utility) of the truthful agent i) when all other agents are truthful and ii) when all other agents use balanced bidding. Explain your findings.

For both cases, make clever use of the `seed`, and `numiters` commands. For i) use `perms 1` additionally.

3. **[40 Points]** Mechanism Analysis

For the following questions, we introduce the idea of a *reserve price* r. This is the minimum price at which the auctioneer is willing to sell a slot (or, more generally, an item). This means, if all agents bid below the reserve price then nobody gets allocated a slot. In the case of GSP, the agent that gets the lowest of the allocated slots (there could be one or more unallocated slots) pays the maximum of the reserve price and the bid of the next lower bidder. For example, in a one slot auction with two bidders, if one bidder is above the reserve ($b_1 \geq r$) and one below ($b_2 < r$), then the first bidder pays r and not b_2. Thus, in this example, the auctioneer actually makes more revenue because of the reserve price r. In ad auctions, reserve prices are used to increase the revenue of the seller. As we have seen in the example, by setting an appropriate reserve price, the seller can indeed make more money. However, if the reserve price is set too high (e.g., $r > b_1 > b_2$ in the example), then it might happen that no bidder wins and the auctioneer makes no money. Thus, we can try to figure out the *optimal reserve price*, i.e., which reserve price maximizes the seller's revenue.

For the following questions, if answering them requires one or more simulation runs, then do that **with 5 agents** *and the default budget of* **$5000**.

(a) **[6 Points]** What is the auctioneer's revenue under GSP with no reserve price, when all the agents use the balanced bidding strategy? What happens to the revenue as the reserve price increases? Plot the auction's revenue against the reserve price and explain the behavior.

Note that 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 `numiters 50`.

(b) **[17 Points]** Implement the VCG slot auction in `VCG.java`. The file has the allocation rule already implemented. You only need to implement the payment rule according to Equation (9.13) in chapter 9 of the lecture notes.

Note: You do not have to worry about the reserve price. Just implement Equation (9.13) from the lecture notes.

(c) **[5 Points]** When the reserve price is zero, what is the revenue of VCG compared to GSP when the agents are truthful? Explain your findings.

Again, use the `perms`, `seed`, and `numiters` commands, e.g. `perms 1`, `seed 2`, and `numiters 50`.

(d) **[7 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? Compare just GSP, just VCG, and switching.

Again use the `perms`, `seed`, and `numiters` commands, e.g. `perms 1`, `seed 2`, and `numiters 50`.

(e) **[5 Points]** Bigger picture: in one paragraph, what did you learn from these exercises about agent design and implementation, mechanism design, and revenue? We aren't looking for any particular answers here, but are looking for evidence of real reflection.

4. **[15 Points]** Competition

The balanced bidding agent from task 1 does not take into account the fact that your agent has a budget. Here, you are asked to write an agent that takes this into account. This agent will compete in a GSP slot auction with the agents submitted by the other groups, so you may want to test it against a variety of strategies. For example, if everyone spends his budget early you might want to bid more in later rounds when competition is lower, or if everyone waits for the end, you might want to bid more early etc.

To make the competition more interesting, we will **reduce the total budget to $800**.

(a) **[10 Points]** In `BudgetGroupName.java`, write your class competition agent. In your writeup, 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: In order to test your agent, be sure to set the total budget to $800 (`totalbudget 80000` in the parameter file).

(b) **[5 Points]** Win the competition. We will run a GSP slot auction with all submitted agents. The auction will have a small reserve price, e.g. `reserve 10`. The agents will be ranked according to their cumulative utility over all instances of the auction. The points from the competition will be awarded as follows: the winning team gets 5 points, the second 4 points, the third 3 points and so on.

Likely parameters for the competition are:

```
numrounds 48
mechanism gsp
totalbudget 80000
numiters 50
reserve 10,
```

but they may change depending on the submitted agents, so don't fine-tune your agents to exactly these parameters.

Note: You are not expected to spend many hours on writing an optimal competition agent, unless you want to. Consider a few strategies, try them out, and pick the best one. Here are a few points to ponder on:

- Think about the click-through rates. In the middle of the auction, the click-through rate is lowest, while at the beginning and the end, it is highest. Can you find a strategy that exploits this?
- Can you attack the other bidders?
- Should your stragety depend on your value? You know that the value distribution is between $0.30 and $1.80.

5. **[Extra Credit]** In this task, we'll take a look at the winner determination problem in an auction. Assume that we do not have separability, i.e., the click-through rate $CTR_{i,j}$ cannot be simply separated into $CTR_{i,j} = e_i \cdot q_j$, but instead, for some ads it makes a big difference whether they are in slot 1 or 2, while for others, the resulting click-through rate is almost the same. More generally, we now have a unique value $CTR_{i,j}$ for every combination of bidder i and slot j.

   Assume that you have n bidders and $m = n - 1$ slots, i.e., one more bidders than slots. Remember that if advertiser i is allocated to slots j, his effective bid value is $v_{i,j} = CTR_{i,j} \cdot b_i$. Write a program that solves the winner determination problem, i.e., that determines which bidder should go into which slot such that the total effective bid value is maximized (the sum over all advertisers effective bid values). Make sure you don't allocate multiple advertisers to the same slot, and that each advertiser gets allocated to at most one slot (and one advertiser gets no slot). To test your program, do the following:

   - Assume that click-through rates are distributed uniformly at random between 0 and 1, i.e., $CTR_{i,j} \sim U[0, 1]$.
   - Assume that bids are generated uniformly at random between 0 and 1, i.e., $b_i \sim U[0, 1]$. (Note that in this task you don't need to worry about bidders' values or about their payments.)

   Here are the specific tasks:

   (a) First, for a given number of agents n and number of slots $m = n - 1$, your program should generate the random values, then figure out the allocation of bidders to slots that maximizes the total effective bid value, and then output that allocation in a suitable format together with the maximal total bid value. Test your program with a small number of agents, e.g., 5.

   (b) Next, show how your program scales as you increase the number of agents n, i.e., what happens when n is 10 or 15 or 20? Draw a graph that shows the run-time of your algorithm (i.e., n on the x-axis, run-time on the y-axis).

   (c) Formally analyze the worst-case running-time of your algorithm, in terms of n.

   (d) What does this tell you regarding the importance of the separability condition?

Comments

Cheating 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. Please don't.

Bugs 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.

Help If something is unclear about the assignment or if you need help, please ask a question on NB as early as possible. Please don't post solution code, but otherwise code snippets are fine (use your judgment).