Final Project: Spectrum Auction Simulation

1 Introduction

This final project option is a Spectrum Auction, which builds on the work we did in Labs 5, 6, and 7.

2 Game Description

The model of the world, goods, bidders, and valuations is slightly different than the setup of Lab 7. In particular, whereas there we assumed the **Global Synergy Value Model** (**GSVM**), for the final project we are assuming the **Local Synergy Value Model (LSVM)**, also with national and regional bidders. This model is so-called because the regional bidders are assigned a local proximity where they prefer to operate.

Additionally, you may recall that the labs were conducted as simultaneous second-price auctions. In the final project, we will instead be running **simultaneous ascending auctions** in an auction format known as **Simultaneous Multiple-Round Auctions** (**SMRA**).

2.1 The LSVM-18 Model

LSVM-18[1] is a small model of a spectrum auction. There are 18 goods, labelled A through R. They are arranged in a grid as follows:

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R

Following Scheffel et. al, we assume one national bidder and five regional bidders. Valuations are generated according to the following procedure:

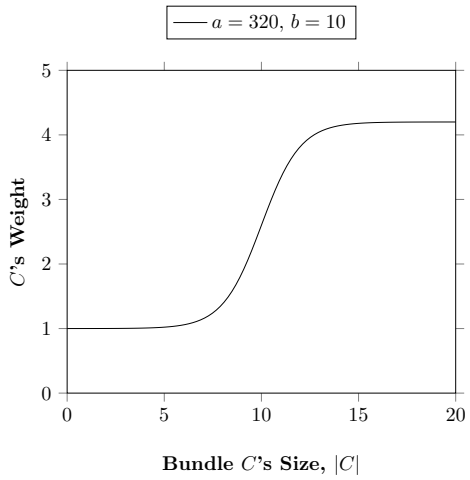
- The *national bidder* has positive base values for all 18 goods, drawn i.i.d. from $U[3, 9]$.
- Each *regional bidder* is assigned a *preferred* good drawn uniformly at random from the set of goods. The regional bidder's *proximity* consists of: their preferred good, goods that are vertically or horizontally adjacent to it, and goods that are vertically or horizontally adjacent to *those*.[2] Thus, a "preferred" good does not mean the bidder values it more than it does other goods; it is merely used to define a regional bidder's proximity. Shown below is the proximity of a regional bidder that prefers good **O**:

A	B	C	D	E	F
G	H	I	J	K	L
M	N	**O**	P	Q	R

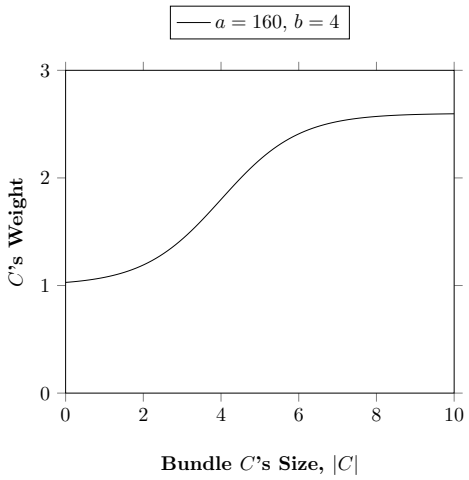
  Regional bidders have base values drawn i.i.d. from $U[3, 20]$ for goods within their proximity, and base values of 0 for goods outside it.
- Define P as a partition of the set of all goods containing maximally connected bundles. That is, for all $C \in P$, with good k in C, all goods l adjacent to k are also in C.

[1] Tobias Scheffel, Georg Ziegler, and Martin Bichler. On the impact of package selection in combinatorial auctions: an experimental study in the context of spectrum auction design. *Experimental Economics*. 15:667–692, 2012.

[2] In other words, the proximity is the set of goods with a Manhattan distance from the preferred good of at most 2.



(a) National Bidder's Bundle Valuations.



(b) Regional Bidders' Bundle Valuations.

Bundle values are computed according to the following function:[3]

$$v_i(S) = \sum_{C \in P} \left(1 + \frac{a}{100\left(1 + e^{b-|C|}\right)}\right) \sum_{k \in C} v_i(k) \tag{1}$$

In words, the goods in a bundle S are maximally partitioned in P, and then the value of S is computed as a weighted sum of the total base values of all elements C of the partition, where the weights are determined by the constants a and b and the element's size. For the national bidder, $a = 320$ and $b = 10$, while for all regional bidders, $a = 160$ and $b = 4$. Thus, for the national bidder, the weight of a bundle remains small until it is of size 6, at which point it increases rapidly, tapering off again around 12. For regional bidders, smaller bundles, say of size 2, are assigned relatively small weights; but weights increase rapidly as the bundle size approaches 6, at which point they taper off again.
N.B. Goods outside a regional bidder's proximity *can* contribute to the weighting function, if they are in the same element C of P as a good within the proximity. Thus, even though the base value of such goods is 0, regional bidders can still accrue additional value by winning goods outside their proximity.

- Bidders may bid on as many goods as they like. Indeed, regional bidders may bid outside their respective proximities, which, as already noted, can add value.
- In addition to base values, which are private information, each regional bidder's preferred good is also private information.

2.2 Simultaneous Multiple-Round Auctions (SMRA)

The mechanism your agent will bid in for the project is a variant of **Simultaneous Multi-Round Auction (SMRA)**. SMRA are simultaneous ascending auctions, which solicit bids in all the auctions from all the agents synchronously (i.e., in rounds). Furthermore, all the auctions end simultaneously, when there is no further bidding in any of them.

The ascending auction that will populate our SMRA mechanism is a variant of eBay's auction design. In this auction, the price in each auction in each round is given by the second-highest bid, floored by a fixed price increment ϵ. Between rounds, the prices across all auctions are broadcast to all agents. Moreover, an agent is informed of all goods it is tentative winning.

[3] You have access to this function in the code (see `getValuation` in Section 5); you do *not* need to implement it yourself.

The eBay auction rule allows for the possibility of **jump bidding**. Jump bidding means that prices need not ascend uniformly. Instead, they can jump by multiple increments, as dictated by the second-highest bid. Jump bidding gives agents an opportunity to strategize, as they can "stake out their territory" by bidding high in some auctions.

Finally, SMRA also generally invokes an activity rule. Activity rules in ascending auctions are designed to encourage sincere bidding. Likewise, they tend to discourage exiting the auction only to re-enter later. Our particular choice of activity rule—**revealed preference**—is described in Section 2.4. Before delving into those details, we present our version of SMRA:

Algorithm 1 The SMRA Spectrum Auction

Inputs: A set of goods G
Outputs: An allocation matrix (an assignment of goods to bidders) and pricing (one price per good)

Initialize the tentative allocation to be empty
Initialize the prices of all goods to 0

do
  Broadcast all current prices
  Send each bidder their tentative allocation
  Get bids from each bidder comprising at most one bid per good
  Prune bids that violate the **revealed preference activity rule**
  Prune all bids with any entries that do not exceed the respective good's current price by at least ϵ
  Call all remaining bids *valid*
  if the set of valid bids is non-empty **then**
    for each good $g \in G$ with at least one valid bid **do**
      Let bidder i^* be the current winner of good g, if any
      Define the set of *incremental* bids on g as the valid bids placed by bidders other than i^*
      if the set of incremental bid(s) on g is non-empty **then**
        Tentatively allocate g to a highest bidder
        if there exists a second-highest valid bid b_2 **then**
          Set the price of g to b_2
        else
          Increase the price of g by ϵ
        end if
      end if
    end for
  end if
while the set of incremental bids for at least one good is non-empty

The outcome of the auction is the final allocation and the final prices

If the highest bid for a good is not unique, a winner is chosen uniformly randomly among the highest bidders.

2.3 Examples

Below are multiple examples of auction outcomes, each from one round to the next, for an individual good. We assume for the sake of simplicity that Bidders A and B are always the two highest bidders, if they bid at all, so no other bidders come into play. Invalid bids are not shown. Incremental bids (bids by bidders other than the current winner) are highlighted in yellow.

Current Winner	Current Price*	Bid A	Bid B	New Winner	New Price
-	0	-	-	**-**	0
-	0	ϵ	-	**A**	ϵ
-	0	$c \geq \epsilon$	-	**A**	ϵ
-	0	$3+\epsilon$	$2+\epsilon$	**A**	$2+\epsilon$
A	p	-	-	**A**	p
A	p	$p+3\epsilon$	-	**A**	p
A	p	-	$p+3\epsilon$	**B**	$p+\epsilon$
A	p	$p+2\epsilon$	$p+2\epsilon$	**A or B**	$p+2\epsilon$
A	p	$p+3\epsilon$	$p+2\epsilon$	**A**	$p+2\epsilon$
A	p	$p+2\epsilon$	$p+3\epsilon$	**B**	$p+2\epsilon$

*The minimum valid bid is this value (i.e., the current price) plus ϵ. Even the current winner cannot place a bid that is not above this minimum. Once bidders beat this minimum, however, they need not bid in increments of ϵ: e.g, if the current price is 10 and the price increment is 2, any bid of 12 or greater is valid.

2.4 The Revealed Preference Rule

The **revealed preference rule** is an activity rule which we will adopt to determine the validity of a bid in our SMRA auction. Like any activity rule, its purpose is to restrain bidders to the space of "reasonable" strategies; in particular, this rule is designed to discourage bidders from re-entering the ascending market for a good after previously exiting it.

The revealed preference rule is defined as follows: A bidder is only allowed to switch their preferred demand set from S to T if, since the time when S was preferred, the price of T increased by less than the price of S. If the price of T increased by more than the price of S since any time when the bidder's demand set was S, then a bidder cannot switch from preferring S to preferring T.

To present the revealed preference rule formally, we introduce the following notation: Let m be the number of goods (i.e., ascending auctions). Let $s < t$ be two rounds in the auction. Let $\boldsymbol{p}^s$ and $\boldsymbol{p}^t$ be vectors in $\mathbb{R}^m$ describing good prices at rounds s and t, respectively. Let $\boldsymbol{x}^s$ and $\boldsymbol{x}^t$ be vectors in $\{0,1\}^m$ describing the bundles a bidder demands at rounds s and t, respectively (i.e., all goods for which it submits a valid bid).

If a bidder is bidding sincerely, then at round s, $\boldsymbol{x}^s$ is the set of goods that is utility maximizing:

$$v(\boldsymbol{x}^s) - \boldsymbol{p}^s \cdot \boldsymbol{x}^s \geq v(\boldsymbol{x}^t) - \boldsymbol{p}^s \cdot \boldsymbol{x}^t.$$

Similarly, at round t, $\boldsymbol{x}^t$ is the set of goods that is utility maximizing:

$$v(\boldsymbol{x}^t) - \boldsymbol{p}^t \cdot \boldsymbol{x}^t \geq v(\boldsymbol{x}^s) - \boldsymbol{p}^t \cdot \boldsymbol{x}^s.$$

Combining these inequalities yields the revealed preference rule:

$$(\boldsymbol{p}^t - \boldsymbol{p}^s) \cdot (\boldsymbol{x}^t - \boldsymbol{x}^s) \leq 0.$$

This rule ensures that sincere bidders will not take a break from bidding part way through the auction, because if they do (i.e., if they report the empty set as their favorite bundle), and then if prices go up on some goods, then they can no longer bid on bundles that contain those goods. *In particular, this rule discourages bidders from sitting out the very first round, as this will render all future bids invalid.*

2.5 Game Specifics

- We choose the price increment $\epsilon = 1.25$.

- To make sure our simulated auctions don't run indefinitely, we cap their number of rounds. If an auction reaches a predetermined final round without having terminated, it will end abruptly. This termination point will be drawn from an exponential distribution, and is guaranteed to be at least 35. Although this distribution is common knowledge, the draw will not be revealed to your agent or any others, as knowing this number would facilitate sniping.[4]

3 Strategy and Considerations

This is a very complex game and requires several strategic considerations beyond what was in play in lab.

First, due to the combinatorial nature of this auction, there are an exponential number of possible bundles on which your agent can bid. This number is especially large for the national bidder, so any strategy that involves enumerating them all will not be viable. How will your agent determine which bundle to bid on, and at what prices, in an efficient manner?

Second, how high above the price thresholds will your agent bid? Due to the second-price nature of these auctions, could your agent submit high bids to secure a tentative allocation, without risking having to pay too high a price? Or would doing so potentially reveal too much information to other bidders? Put another way, should your agent bid high right from the start, revealing its interests to other bidders—a form of cooperative behavior—or should it lay low at first, gauging its opponents' strategies—how cooperative are they?—before bidding aggressively: i.e., showing its cards?

Third, how will your agent's strategy differ when it is the national bidder, versus a regional bidder? There is more competition for national goods; how will this affect your agent's bidding strategy? Will your agent make use of the valuation distributions (which are public knowledge) to try to predict the bids of opposing agents, and ultimately the final auction prices?

A successful agent strategy for this game will comprise these ideas and many others. We suggest building a theoretical model of the game (consider starting your writeup!) before attempting to implement an agent strategy. This model will help you envision even more of the many important factors that your agent's strategy should try to take into account.

4 Game API

4.1 `MyLSVM18Agent` class

Your task is to fill in the following methods of the `MyLSVM18Agent` class:

- `getBids(Map<String, Double> minBids)`: this represents your strategy; it is called each round, and you must return a `Map<String, Double>` representing your bid for each item (represented as `String`s) that you would like to bid on.

In both of the above methods, `minBids` represents the **minimum allowable bid** for each respective item, *not* the current price. Specifically, `minBids` contains the current prices plus ϵ.

Be careful with this, because if any single one of your bids is too low, your entire bid bundle will be rejected. And due to the revealed preference rule, "sitting out" a round can have dire consequences.

We provide a few useful tools for checking the validity of your bids; see Section 4.2.

[4] urbandictionary.com defines sniping as the act of bidding on an item on eBay literally seconds before the auction ends. This prevents anyone from outbidding you while also ensuring that you don't bump the price up too much.

We also provide the following utility methods that you may choose to fill in, if you find them useful:

- `onAuctionStart()`: called at the beginning of each simulation of the Spectrum Auction. If you are keeping any per-simulation data, this is where you would want to initialize or reset it.
- `onAuctionEnd(allocations, payments, tradeHistory)`: at the end of an auction simulation, gives you all of the information from the game, for you to use however you want. Unlike the tentative allocations you have access to during the game, this contains information on *all agents'* bidding history and final allocations and payments. Entries in these variables that correspond with your agent in particular will have a key or agent ID of `this.getAgentBackend().getPublicID()`.

4.2 Useful inherited methods

You may find the following methods useful in the Spectrum Auctions. All are inherited by your agent.

- `Set<String> getProximity()`: returns the set of goods in your proximity. If you are the national bidder, the returned set will contain all the goods.
- `double getValuation(Collection<String> goods)`: returns your agent's valuation for a bundle of goods. For simplicity, we also provide a version of this method that takes in a single `String`, so you can easily determine your value for individual goods.
- `Set<String> getTentativeAllocation()`: gets the set of goods tentatively allocated to your agent.
- `Set<String> isNationalBidder()`: returns `true` if you are the national bidder, and `false` if you are a regional bidder.
- `boolean isValidBidBundle(Map<String, Double> myBids, Map<String, Double> minBids)`: returns `true` if `myBids` is a valid bid bundle to submit to the server, by performing the following checks:
  1. None of your bid amounts are `null`.
  2. All bids in `myBids` are at least the minimum allowable bid, as indicated in `minBids`.
  3. Your bid bundle satisfies the revealed preference rule (we check this by maintaining records of past good prices, and of your past bids).

  We *highly* recommend running this check; if your bid bundle passes, it is guaranteed to be accepted by the server.
- `double clipBid(String good, double bid, Map<String, Double> minBids)`: returns `bid` clipped to be above the minimum allowable bid for `good`. In other words, returns `max(bid, minBids[good])`.
- `int getCurrentRound()`: returns a number indicating the current round number in the ascending auctions. This value is 0-indexed.

5 Tier 1 Agent

We have provided you with access to our Tier 1 TA Bot implementation to test your own agent against. The Tier 1 bot is implemented in the `Tier1LSVM18Agent` class, and does the following:

- Identifies the set of goods for which its single-good value is above the minimum bid.
- Bids exactly the minimum allowable bid on each of those goods.

6 Testing

To test your agent *offline* against other agents, run your `MySpectrumAuctionAgent` file directly in Eclipse. The main method instantiates 10 agents, including your own, and runs a local simulation, offline, in order to avoid server delays and wait times. This simulation consists of 500 iterations of the AdX game.

As with any programming project, you should test your programs extensively. In addition to the usual unit tests, etc., you should play test games against other agents. You can test against 9 copies of your own agent, 9 copies of our Tier 1 TA Agent (see Section 5), or any combination thereof. You can also create your own dummy agents to test your own agent against; to do so, just extend `AbsSpectrumAuctionAgent`. To set the agents to test against, you can edit the agent types in `src/main/resources/offline_test_config.json`.

If you design a strategy that requires training, you can submit a pre-trained agent by which we mean an agent that reads a precomputed data file. To use a pre-trained agent that reads its input from a file, you should save the file in `src/main/resources` and read it using `getResourceAsStream()`; see this guide. By including your file in this resource directory, it will be included in the executable JAR we use to run your submission; file-reading is thus independent of the specifics of the filesystem.

To make sure that your agent will run as intended in a competition, we have provided a script that launches the Spectrum Auction server, along with your agent and 5 test bots. Please be sure to run this script before handing in your code, to check that your agent connects to the server properly and properly submits its bids.

The command is `/course/cs1440/pub/2021/SpectrumAuction/test`. Run it from your project's root directory. It will run 5 iterations of the game, which takes about one minute, or less (but it may take a bit longer the first time if Maven needs to download any packages).

7 Submission

Before submitting your code, please run `mvn clean` from your project's root directory.

In order to submit your code, please follow the instructions in the Lab Installation/Setup/Handin Guide.