COMPSCI 223 - Computational Microeconomics

# Programming Assignment 2: combinatorial auctions and expressive securities markets (due February 28 before class)

Please read the rules for assignments on the course web page (http://www.cs.duke.edu/courses/spring18/compsci223/). Use Piazza (preferred) or directly contact Harsh (harsh.parikh@duke.edu), Hanrui (hrzhang@cs.duke.edu), or Vince (conitzer@cs.duke.edu) with any questions. Please use clear variable names and write comments in your code where appropriate (you can put comments between `/*` and `*/`, or start a line with `#`).

Please see Homework 1 for details about getting set up with GLPK, making a directory for this homework, etc.

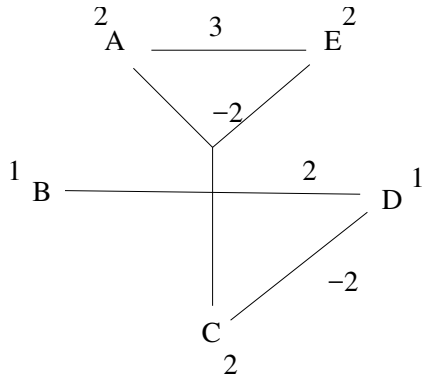
Note: in both of these questions, there is an example instance that you are asked to solve. However, just getting this example right is *not* enough to get full credit: your formulation should work on all instances. The example is just there to give you something to test your formulation on.

There are some parenthetical questions in the below; you do not need to turn in answers to these, they are just there to help you think about the questions.

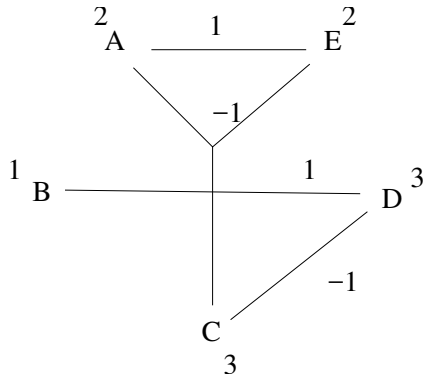
**1. (50 points.) A different bidding language for combinatorial auctions.**

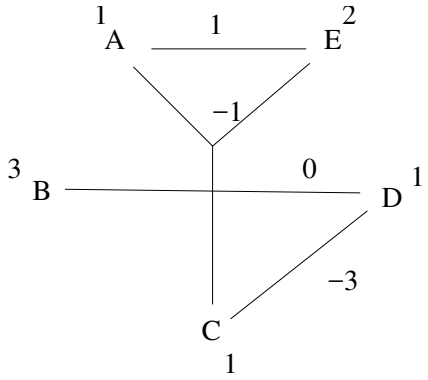
In this question, we will study a bidding language that is different from the OR and XOR bidding languages that we saw in class. A "bid" in this language consists of a graph, whose vertices are the items in the auction. Each vertex has a number associated with it, which is the bidder's value for receiving the item by itself. To represent complementarities and substitutabilities, the bidder places values on the edges of the graph. If the bidder receives *both* items on the edge, then the bidder's valuation is the sum of the values of the individual items on the edge, *plus* the value on the edge. Thus, if the value on the edge is positive, the items are complements; if it is negative, they are substitutes. We also allow for *hyperedges*, which are edges that can link 3 or more vertices together. In this case, the value on the hyperedge is added to the bidder's valuation if and only if the bidder receives *all* the vertices that the hyperedge connects.

Here is an example bid:




For example, this bidder has a value of $2+2$ for the bundle $\{A, C\}$, a value of $2+2+3$ for $\{A, E\}$, a value of $2+1+2+2+3-2$ for $\{A, B, C, E\}$, and a value of $2+1-2$ for $\{C, D\}$. (This bid does not satisfy the free disposal assumption—can you see why?)
Here are two more bids:



(What is strange about this last bid?) (Let's assume we have the option to keep some items (not allocate them to anyone); would we want to make use of this option? What about if we change the values?)

You will solve the winner determination problem for these three bids. Most of the file for doing this is already complete, but you will have to add some lines to make it work (indicated by "you need to fill this in"). You can get the file by typing

```
wget http://www.cs.duke.edu/courses/spring18/compsci223/graphbidding.mod
```

The following describes the formulation.

Conceptually, we can think of regular edges (that connect only two vertices) as a special case of a hyperedge; moreover, we can even think of the individual items as a hyperedge—one that connects only one vertex. So, *every* value in the bid graph corresponds to a hyperedge.

Note that at least in this example, the bidders agree on whether each hyperedge has a negative or positive (to be precise, nonnegative) value. So, we can classify the hyperedges as "positive" or "negative". Each hyperedge consisting of a single item is positive.

The graphbidding.mod file has the following parameters: num_items, the number of items; positive_valuation[b,e], bidder b's valuation for positive hyperedge e; occurs_in_positive[i,e], whether item (vertex) i is adjacent to positive hyperedge e; and similarly for the negative hyperedges (since we already know that the negative hyperedges have negative values, we will drop the minus symbol for those values).

The file has the following binary variables: assigned[b,i] (set to 1 if b receives i), positive_applies[b,e] (set to 1 if b receives all items in positive hyperedge e), and negative_applies[b,e] (set to 1 if b receives all items in negative hyperedge e).

We need to have some constraints on these variables. For example, the solver will always "want" to set positive_applies[b,e] to 1 to maximize welfare, but this is only legitimate if all of the items in positive hyperedge e are in fact assigned to b. So we add the constraint:

```
num_items * positive_applies[b,e] + sum{i in ITEMS} occurs_in_positive[i,e]
- sum{i in ITEMS} occurs_in_positive[i,e] * assigned[b,i] <= num_items;
```

Here's why this constraint works. If b in fact receives all items in e, then

```
sum{i in ITEMS} occurs_in_positive[i,e] - sum{i in ITEMS}
occurs_in_positive[i,e] * assigned[b,i]
```

will equal 0, hence positive_applies[b,e] can be set to 1 without violating the constraint. However, if b does not receive all items in e, then

```
sum{i in ITEMS} occurs_in_positive[i,e] - sum{i in ITEMS}
occurs_in_positive[i,e] * assigned[b,i]
```

will be positive, hence positive_applies[b,e] cannot be set to 1 without violating the constraint (but setting it to 0 will always satisfy the constraint (why?)). Note that this constraint does *not* work for the negative hyperedges, because there the solver "wants" to set the variable negative_applies[b,e] to 0. So a different trick is necessary here, one that only lets the solver set negative_applies[b,e] to 0 if there is at least one item in e that b does *not* get. You need to come up with such a trick, as well as fill in the other missing parts of the file. Check whether the answer you get makes sense.

**2. (50 points.) Solving the auctioneer's problem for an expressive securities market.**

In this question, you will write (from scratch) a .mod file for solving the auctioneer's (winner determination) problem for an expressive securities market. Remember that the auctioneer's problem is to maximize her profit in the *worst* case (state).

You can write this .mod file in any way you like, except for the following requirements:

- You should have a set STATES (of states that the world can be in) and a set of BIDS;
- For every bid b, there should be a parameter v[b], the amount that the bidder is bidding (is willing to pay);
- For every bid b, for every state s, there should be a parameter p[b,s], which is the amount that the bidder must receive in that state if the bid is accepted;
- For every bid b, there should be a variable accepted[b] that is 1 if the bid is accepted, 0 if it is not accepted (and, perhaps, something inbetween if it is partially accepted, IF you are allowing for this);
- There should be a variable called worst_case_profit (which is your objective); the auctioneer's profit in any particular state should be no less than this variable.

Now, use your formulation to solve the "A bigger instance" example from the course slides. Solve it both with and without partially acceptable bids. Make sure that the solution makes sense.