Combinatorial Auctions - Fall 2014
Assignment #6: Combinatorial Auctions

Professor Sven Seuken
Department of Informatics, University of Zurich
Out Tuesday, November 11, 2014
Due **13:30** sharp: **Tuesday, November 18, 2014**
Submissions should be made to `dmitry.moor@ifi.uzh.ch`

[**Total: 100 Points**] This assignment can (but need not) be solved in groups of **up to two students**. Note that if you are working in a group of two, we expect that both students contribute roughly equally to the assignment and both understand all parts of the solution.

Points will be awarded for clarity, correctness and completeness of the answers. Reasoning must be provided with every answer, i.e., please show your work. You get most of the credit for showing the way in which you arrived at the solution, not for the final answer. You are free to discuss the assignment with other students. However, you are not allowed to share (even partial) answers and source code with each other, and **copying will be penalized**.

Every group has to hand in their write-up (PDF) as well as their source code. In this assignment you need to hand in **only 1 write-up per group**. Please send the source code in a zip-file.

Getting Started

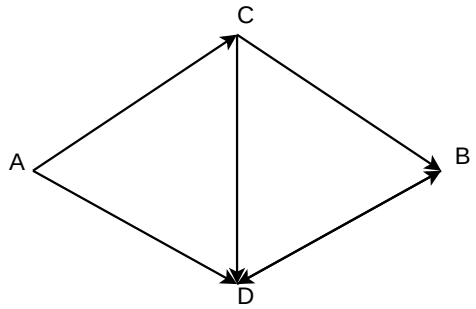
In this assignment, you will implement a combinatorial auction in Java/JOpt and test your code with the Combinatorial Auctions Testing Suite (CATS) that generates test instances for combinatorial auctions. For all programming exercises, we provide you with a code skeleton. To set everything up, you need to first download the source code from our course homepage and import it into Eclipse. Then you need to set up CATS. To do so, follow these steps.

1. Download the Windows executable CATS from `http://www.cs.ubc.ca/~kevinlb/CATS/`.
2. Extract the program into a folder.
3. Open up your command line and navigate to the folder.
4. Create your first CATS instance by typing (on Windows) *cats.exe -goods 4 -bids 10 -d arbitrary*.
5. The program should create a file called 0000.txt

Unfortunately, CATS currently only works on Windows. If you run into any problems while using CATS, please send us an email.

1. **[25 Points]** Combinatorial Auctions: Bidding Languages

   (a) **[10 Points]**



   Consider the following setting: two agents can bid for computer network links. Agent 1 wishes to send a message from A to B and Agent 2 is interested in passing a message from C to D. All links can carry one package of data in the direction specified by the arrows. In particular, link $\overline{BD}$ can carry one package in one of the directions.. Agents valuations of different paths are inverse proportional to the number of hops the message takes, i.e., $v_i(P) = \frac{1}{Number\ of\ hops\ in\ P} \cdot \$30$, $i = 1, 2$. Every agent needs to send a single packet and thus can use only a single path.

   Formulate XOR bids enumerating all possible paths both agents would submit in the auction, assuming they bid truthfully. Do the same for the OR* bidding language. Is it possible to express these bids in the OR bidding language? Why?

   (b) **[15 Points]** Suppose a general combinatorial auction problem in which a set A of objects must be allocated to two agents $i = 1, 2$. Suppose further that both agents submit their bids in the XOR bidding language. Formulate the auctioneer's winner determination problem as an interger program, i.e., formalize all variables, constraints, and the objective.

2. **[25 Points]** Winner Determination

   In this task you need to implement a solution to the winner determination problem in a combinatorial auction (see lecture notes, Definition 11.4), assuming that bids are submitted using the XOR-bidding language. Use JOpt to formulate and solve the resulting integer program.

   In the package `src/main/java/.../winnerdetermination`, you can find a code skeleton. The classes and methods that need to be implemented/altered are marked with a TODO marker.

   Verify your implementation with the tests that can be found in the package `src/test/java/.../winnerdetermination` using the data provided by CATS (in `src/test/resources`).

3. **[50 Points]** VCG Mechanism

   In this task you need to implement the VCG mechanism for combinatorial auctions. Using your code for the winner determination problem from task 2., you only need to implement the VCG payment rule.

(a) **[20 Points]** Implement the VCG mechanism. A code skeleton is provided in the class `VCGAuction.java`. You only need to implement the payment rule. For the allocation, re-use the code from task 2.

Verify your implementation with the tests that can be found in the package `src/test/java/.../vcg` using the data provided by CATS (in `src/test/resources`).

(b) **[12 Points]** Now implement the VCG mechanism with reserve prices: suppose there is a per-item reserve price $r_a > 0$ for each item $a \in A$, and the reserve price for any bundle is equal to the sum of the reserve prices of the items in the bundle, i.e., $r_B = \sum_{a \in B} r_a$ for all bundles $B \subseteq A$. To enforce the reserve prices, all atomic bids (from the set of bids specified in the XOR bidding language) that violate the reserve price constaints are eliminated. The payment rule is then adjusted such that winning bidders pay the maximum of their actual VCG payment and the reserve price for the bundle they receive.

First, implement VCG with reserve prices using the template provided in the class `ReservePriceVCGAuction.java`. Second, compare the VCG mechanism with and without reserve prices on the CATS instance `hard0000.txt` that is provided in the resources folder, using a per-good reserve price of 50. Report the total revenue and welfare for both mechanisms.

(c) **[3 Points]** Describe a different way to implement reserve prices and think about what consequences this might have?

(d) **[15 Points]** In this task you will compare the VCG mechanism with/without reserve prices on a larger sample of CATS instances to get a more solid basis for studying the effects of introducing reserve prices. To do so, you need to create 50 CATS instances. These instances should use the distribution *arbitrary* (`-d arbitrary`), have 8 goods (`-goods 8`) and 20 bids (`-bids 20`). You can create multiple instances at once using the -n option (e.g., `-n 50`). A parser and converter for CATS auction is already provided. Take a look at the tests in `VCGFromCATSTest.java` to see how you can use them.

Now run the VCG with and without reserve prices on these 50 instances, using a per-good reserve prices of 50. For both mechanisms, report the average total revenue and welfare plus standard deviations. Plot your data and explain your results.

4. **[Bonus Assignment]** Core-selecting auctions.

Core-selecting auctions are those for which the set of payments is feasible for the group of bidders as a whole and cannot be blocked by any coalition (see Chapter 11 for more details).

Consider the following setting with three bidders and two items:

	A	B	AB
Agent 1	0	1	10
Agent 2	6	1	7
Agent 3	1	6	7

(a) Provide a solution of a winner-determination problem and compute VCG payments of allocated bidders.

(b) Find a blocking coalition in this auction, i.e., a coalition of agents (and possibly a seller) who could re-assign the items among themselves so that the resulting assignment would provide a higher revenue for the seller and higher utilities for bidders in the coalition.

(c) Graphically illustrate the core and VCG payments.

(d) **VCG-Nearest** payment rule corresponds to the bidder-optimal core payment that minimizes L_2 distance to VCG payments. Find the **VCG-Nearest** payments for the problem and illustrate them.

(e) Compare the outcome of VCG to the outcome of **VCG-Nearest** in terms of efficiency, revenue and incentives for truthful bidding.