EECS 395/495: Algorithmic Mechanism Design	**Homework 2**
Assigned: 4/22/10	**Due: 4/29/10**

Note: Work in pairs. Turn in one assignment. You may refer to lecture notes, assigned readings, or the reference text book *Algorithmic Game Theory*. You may not look for help on the Internet or sources outside course materials. Your solutions should easily and concisely convey your complete understanding of each problem. If you cannot solve a problem, come to office hours. If your solution cannot easily be understood then it is wrong.

1. In class we showed that we can turn a non-monotone algorithm for any single-parameter agent setting into a BIC mechanism that has at least the same expected social surplus. Consider another important objective in computer systems: *makespan*. Suppose we have n machines and m jobs. Each machine i is a selfish agent with privately known slowness-parameter $|v_i|$ and each job j has an intrinsic length w_j. The time it takes i to perform job j is $|v_i w_j|$ and we view this as a cost to agent i.[1] The load on machine i for a set of jobs J is $|v_i| \sum_{j \in J} w_j$. A scheduling is a partitioning of jobs among the machines. Let J_i be the jobs assigned to machine i. The makespan of this scheduling is the maximum load of any machine, i.e., $\max_i |v_i| \sum_{j \in J_i} w_j$. The goal of a scheduling algorithm is to minimize the makespan. (Makespan is important outside of computer systems as "minimizing the maximum load" is related to fairness.)

   This is a single parameter agent setting, though the outcome for each agent i is not a binary $x_i \in \{0, 1\}$. Instead if i is allocated jobs J_i then $x_i = \sum_{j \in J_i} w_j$. Of course $x_i(v_i)$ is, as usual, $\mathbf{E}_{\mathbf{v} \sim \mathbf{F}}[x_i(\mathbf{v}) \mid v_i]$. The agent's cost for such an outcome is $|v_i x_i(v_i)|$.

   Show that our method for monotonizing a non-monotone algorithm fails to preserve the expected makespan. (Hint: all you need to do is come up with an instance and a non-monotone algorithm for which expected makespan increases when we apply the transformation from class. Do not worry about the values being negative nor about the allocation being non-binary; these aspects of the problem do not drive the non-monotonicity result.)

2. A key part of the the reduction from BIC mechanism design to Bayesian algorithm design was in identifying intervals on which to resample an agent's value. We did this by looking at the allocation rule in probability space instead of value space. We called this $g(q) = x_i(F_i^{-1}(q))$. We then looked at the cummulative allocation rule $G(q) = \int_0^q g(z)dz$. We took the convex hull of this, $G'(q)$, and identified regions for resampling as those where $G(q) \neq G'(q)$.

   Formally explain why we needed to go to probability space for this construction. I.e., if we just defined $X_i(v) = \int_0^v x_i(z)dz$ and its convex hull $X_i'(v)$, resampling in regions where $X_i'(v) \neq X_i'(v)$ would not result in a monotone allocation rule. Why?

3. Monotone algorithms can be combined.

   (a) Consider the following algorithm proposed by a student in class:
   - Simulate greedy by value (i.e., sorting by v_i).
   - Simulate greedy by value-per-item (i.e., sorting by $v_i/|S_i|$).
   - Output whichever solution has higher surplus.

[1] We are putting these quantities in absolute values because if the private value represents a cost, it is most consistent with the course notation to view v_i as negative.

Prove that the resulting algorithm is monotone.

(b) We say a deterministic algorithm is *independent of irrelevant alternatives* when

$$\mathbf{x}(\mathbf{v}_{-i}, v_i) = \mathbf{x}(\mathbf{v}_{-i}, v_i') \qquad \text{iff} \qquad x_i(\mathbf{v}_{-i}, v_i) = x_i(\mathbf{v}_{-i}, v_i'),$$

i.e., the outcome is invariant on the value of a winner or loser. Prove that the algorithm $\mathcal{A}$ that runs k deterministic monotone independent-of-irrelevant-alternatives algorithms $\mathcal{A}_1 \ldots, \mathcal{A}_k$ and then outputs the solution of the one with the highest surplus is itself monotone.

4. Consider the following knapsack problem: each agent has a private value v_i for having an object with publicly known size w_i inserted into a knapsack. The knapsack has capacity C. Any set of agents can be served if all of their objects fit simultaneously in the knapsack. We denote an instance of this problem by the tuple $(\mathbf{v}, \mathbf{w}, C)$. Notice that this is a single parameter agent setting with cost function:

$$c(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_i x_i w_i \leq C \\ \infty & \text{otherwise.} \end{cases}$$

The knapsack problem is $\mathcal{NP}$-complete; however, very good approximation algorithms exist. In fact there is a *polynomial time approximation scheme* (PTAS). A PTAS is an family of algorithms parameterized by $\epsilon > 0$ where $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation runs in polynomial time in n, the number of agents, and $1/\epsilon$.

PTASs are often constructed from pseudo-polynomial time algorithms. Let V be an upper bound on the value of any agent, i.e., $V \geq v_i$ for all i, and assume all agent values are integers. For the integer-valued knapsack problem there is an algorithm with runtime $O(n^2 V)$. (Constructing this algorithm is non-trivial. Try to do it on your own, or look it up in an algorithms text book.) Notice that this is not fully polynomial time as V is a number that is part of the input to the algorithm. The "size" of V is the amount of space it takes to write it down. We ordinarily write numbers on a computer in binary (or by hand, in decimal) which therefore has size $\log V$. An algorithm with runtime polynomial in V is exponential in $\log V$ and, therefore, not considered polynomial time. It is *pseudo-polynomial time*.

A pseudo-polynomial time algorithm, $\mathcal{A}$, for surplus maximization in the integer values setting can be turned into a PTAS $\mathcal{A}_\epsilon$ for the general values setting by rounding. The construction:

- $v_i' \leftarrow v_i$ rounded up to the nearest multiple of $V\epsilon/n$.
- $v_i'' \leftarrow v_i' n/(V\epsilon)$, an integer between 0 and n/ϵ.
- Simulate $\mathcal{A}(\mathbf{v}'', \mathbf{w}, C)$, the integer-valued pseudo-polynomial time algorithm.
- Simulate the algorithm that allocates only to the highest valued agent.
- Output whichever solution has higher surplus.

Notice the following about this algorithm. First, its runtime is $O(n^3/\epsilon)$ when applied to the $O(n^2 V)$ pseudo-polynomial time algorithm discussed above. Second, it is a $(1+\epsilon)$-approximation if we set $V = \max_i v_i$. (This second observation involves a several line argument. Work it out yourselves or look it up in an algorithms text book.)

In this question we will investigate the incentives of this problem which arise because we do not know a good choice of V in advance.

(a) Suppose we are given some V. Prove that $\mathcal{A}_\epsilon$, for any ϵ, is monotone.

(b) Suppose we do not know V in advance. A logical choice would be to choose $V = \max_i v_i$ and then run $\mathcal{A}_\epsilon$. Prove that this combined algorithm is not monotone.

(c) For any given ϵ, derive an ex post IC $(1+\epsilon)$-approximation mechanism from any integer-valued pseudo-polynomial time algorithm. Your algorithm should run in polynomial time in n and $1/\epsilon$. (Hint: the hard part is not knowing V.)