CO759: Algorithmic Game Theory — Fall 2010

Instructor: Chaitanya Swamy

Assignment 1

Due: By Nov 1, 2010

You may use anything proved in class directly. I will maintain a FAQ about the assignment on the course webpage. *Acknowledge all collaborators and any external sources of help or reference.* All questions carry equal weightage.

Quick Primer on Linear Programming: A *linear program* (LP) is a problem of the following form(s)

$$\begin{array}{lll} \max & c^T x & \text{(P)} \\ \text{s.t.} & Ax \le b & \\ & x \ge 0 & \end{array} \qquad \begin{array}{lll} \min & b^T y & \text{(D)} \\ \text{s.t.} & A^T y \ge c & \\ & y \ge 0 & \end{array}$$

where we seek to optimize a linear function of a finite number of variables subject to a finite number of linear constraints. Any such LP may either be infeasible, or have an optimal solution, or be unbounded. Given a maximization primal linear program of the form (P) above, one can construct a *dual* linear program (D) that provides a tight upper bound on the optimal value of the primal LP. The rationale behind the construction of the dual is as follows. Let $A = (a_{ij})$ above be an $m \times n$ matrix, with a_i^T denoting the i-th row of A and A_j denoting the j-th column of A. Each primal constraint is associated with a nonnegative dual variable y_i. Now if we multiply each primal constraint with its corresponding dual variable y_i and sum the resulting inequalities, we obtain the compound inequality $\sum_{j=1}^{n}\big(\sum_{i=1}^{m} y_i a_{ij}\big)x_j \le b^T y$. Thus, if we enforce the constraints $\sum_{i=1}^{m} y_i a_{ij} \ge c_j$ for each variable x_j, then since $x \ge 0$, we obtain that $c^T x \le y^T A x \le b^T y$. Notice that these constraints are precisely the constraints of the dual LP, and so we have proved that the value of *any feasible solution* y to (D) provides an upper bound on the optimal value of the primal LP. This statement is often known as *weak duality*. (Also, observe that the primal LP (P) corresponds to the dual of the minimization problem (D).) The central theorem of linear programming is a much stronger theorem, often called *strong duality*.

(Strong Duality): Let (P) be a pair of primal and dual LPs, with (P) being a maximization LP (as above). Then, the following hold.

(i) (P) has an optimal solution iff (D) has an optimal solution;

(ii) The optimal values of (P) and (D) (if they exist) are equal;

(iii) If x^* and y^* are respectively optimal solutions to (P) and (D) respectively, then (part (ii) implies that) they must satisfy the following *complementary slackness* conditions: (a) $x_j^* > 0 \implies A_j^T y^* = c_j$; and (b) $y_i^* > 0 \implies a_i^T x^* = b_i$.

Q0: (Do Not Hand This In) Consider a (simultaneous-move) two-person game described by two $m \times n$ matrices R and C that specify respectively, the payoffs to the row player and the column player. That is, (R, C) describes a two-person game where the row and column players have m and n strategies respectively, and R_{ij} and C_{ij} are the payoffs to the row and column players respectively when the row player plays strategy i and the column player plays strategy j. Given a mixed-strategy

profile (x, y) where $x \in \Delta_m$ and $y \in \Delta_n$ are respectively the distributions of the row and column players, the expected payoff to the row and column players can thus be conveniently expressed as $x^T Ry$ and $x^T Cy$ respectively.

Prove that (x, y) is a *mixed Nash equilibrium* (NE) iff for every i and j,

$$x_i > 0 \implies (Ry)_i = \max_{i' \in \{1,\ldots,m\}} (Ry)_{i'} = x^T Ry; \qquad \text{and}$$
$$y_j > 0 \implies (x^T C)_j = \max_{j' \in \{1,\ldots,n\}} (x^T C)_{j'} = x^T Cy.$$

(This says that in a mixed equilibrium a player's distribution is supported on the set of pure strategies that constitute a *best response* to the other player's mixed strategy.)

Q1: A *symmetric game* is a game where all players are identical in the sense that all players have the same strategy-set S, and each player i's payoff under a strategy profile (s_i, s_{-i}) is a function of her strategy s_i and only the *number* of other players who play a given strategy in S, where this function is the same for all players. For a two-person symmetric game (R, C), this amounts to saying that $C = R^T$.

Show that the problem of finding a Nash equilibrium in an arbitrary two-person game can be reduced to that of finding a Nash equilibrium in a symmetric two-person game. That is, given an arbitrary game (R, C) construct a symmetric game such that given any mixed equilibrium of the symmetric game, one can obtain (efficiently) a mixed equilibrium of the original game. You may assume that $R_{ij}, C_{ij} \geq 0$ for all i, j if necessary.

(**Hint:** Construct a game that encodes the following. Each player chooses whether she wants to be the row player or column player and a corresponding strategy; if both decide to be the same (i.e., row, or column) player then they both get 0 payoff, otherwise each player gets the payoff that she would get in the game (R, C) according to her chosen role of a row or column player.)

Q2: Recall the set-cover problem discussed in class in a mechanism-design setting. We have a collection $\mathcal{S}$ of m subsets of a ground-set (or universe) U of n elements. Each set $S \in \mathcal{S}$ is a player whose private value is the weight of the set (which is nonnegative), and the goal is to find a minimum-weight collection of sets (according to the true weights) that covers U (the set-system $(U, \mathcal{S})$ is common knowledge). So $A = \{\mathcal{S}' \subseteq \mathcal{S} : \mathcal{S}' \text{ is a set cover for } U\}$, and for each player S, $V_S = \{-w_S \alpha_S : w_S \geq 0\}$ is a single-dimensional domain, where $\alpha_S(\mathcal{S}') = 1$ if $S \in \mathcal{S}'$ and 0 otherwise. The target function $g : V \mapsto A$ which maps a weight-vector to a minimum-weight set-cover is *NP*-hard to compute, so we consider the implementation of an approximation algorithm for the set-cover problem. Let $B = \max_e |\{S \in \mathcal{S} : e \in S\}|$.

In class, we showed that "the greedy" algorithm for set cover is an $O(\log n)$-approximation algorithm that is implementable. One widely-used paradigm in approximation-algorithm design is to consider a *linear-programming (LP) relaxation* of the problem whose optimum provides a bound on the value of an optimal solution to the problem, and use this LP to guide the design and analysis of the algorithm. Here we investigate the implementability of some approximation algorithms for set cover designed using this approach. Let $OPT(w)$ denote the optimal value of the following LP:

$$\min \sum_S w_S x_S \quad \text{subject to} \quad \sum_{S: e \in S} x_S \geq 1 \quad \forall e; \qquad x_S \geq 0 \quad \forall S. \tag{P}$$

(Throughout S indexes the sets in $\mathcal{S}$, and e the elements in U.) Clearly, $OPT(w)$ is a *lower bound* on the weight of a minimum-weight set cover since any set cover $\mathcal{S}'$ yields a $\{0, 1\}$ solution to (P), where $x_S = 1$ if $S \in \mathcal{S}'$ and 0 otherwise.

(a) Consider any algorithm of the following form: fix a threshold $t \geq 0$ that does not depend on the input w. We solve the LP to obtain an optimal solution x^*, and set $\mathcal{S}' = \{S : x^*_S \geq t\}$. We call this algorithm the *LP-rounding algorithm with threshold t*. Since there may be multiple optimal solutions, to be precise, fix a w-independent ordering $S_1, \ldots, S_m$ of the sets in $\mathcal{S}$, and let x^* be a *lexicographically maximal* optimal solution, that is, x^* is such that for any other optimal solution $\hat{x} \neq x^*$, there is some set S_i such that $x^*_{S_i} > \hat{x}_{S_i}$ and $x^*_{S_j} = \hat{x}_{S_j}$ for all $j < i$. (Such an optimal solution can be computed in polynomial time.) Assuming that this algorithm always returns a set cover, prove that the algorithm is implementable.

(b) Prove that for any weight-vector w, the LP-rounding algorithm with threshold $\frac{1}{B}$ returns a set cover of weight at most $B \cdot OPT(w)$.

(c) Prove that for any weight-vector w, the LP-rounding algorithm with threshold 0^+ (i.e., $\mathcal{S}' = \{S : x^*_S > 0\}$) also returns a set cover of weight at most $B \cdot OPT(w)$.

(d) Thus, in both parts (b) and (c) we obtain a B-approximation algorithm that is implementable. It is not clear however that prices (or rather payments) implementing the algorithm can be computed efficiently. Consider the the LP-rounding algorithm with threshold t, and assume that the algorithm always returns a set cover. Let $\{p_S(.)\}_S$ be the payment scheme that implements this algorithm. (Recall that the $p_S(w)$ is the maximum value v such that the algorithm selects S when run on the input (v, w_{-S}).) To avoid infinite payments or problems with individual rationality, assume that $\mathcal{S} \setminus \{S\}$ is a set cover, for all $S \in \mathcal{S}$. Prove that for any $\epsilon > 0$, and any (rational) input w, one can compute *random* payments $\{q_S(w)\}_S$ in time polynomial in the input size and $\log(1/\epsilon)$ such that $q_S(w) \in [p_S(w) - \epsilon, p_S(w) + \epsilon]$ and $\mathrm{E}\big[q_S(w)\big] = p_S(w)$. *You may assume that you are provided an oracle that can solve LPs of the form* (P) *(of size* poly*(input size)); each call to this oracle counts as one operation.*

So $M = \big(\text{LP-rounding algo. with threshold } t, \{q_S(.)\}_S\big)$ is a truthful-in-expectation mechanism.

(e) Consider the greedy algorithm for set cover discussed in class (which can also be viewed as an LP-based algorithm; see the remark below). Prove that payments implementing the algorithm can be computed in polynomial time.

Remark: The greedy algorithm discussed in class can be interpreted as constructing simultaneously a $\{0,1\}$ solution to (P) and a dual solution $y = (y_e)_e$ with the same objective value, where $y_e = \frac{w_{S_e}}{|\text{uncovered elements covered by } S_e|}$, with S_e being the first set chosen that covers e. One can then argue that $\big(y_e/O(\log n)\big)_e$ is a feasible dual solution, thus showing that the algorithm returns a set cover of weight at most $O(\log n) \cdot OPT(w)$, which is a stronger statement than what we proved in class.

Q3: In the context of truthfulness, prices are used as a means to an end, namely, to incentivize players to declare their true valuations. But in various situations, one can assign a natural meaning to "money" and prices, and we may require that the prices charged (or payments made) satisfy certain additional properties (beyond ensuring truthfulness). One such setting is where the mechanism-designer incurs a certain cost in constructing a solution for the players, and we would like the prices charged to recover (approximately) the cost incurred. This property is called *budget balance*.

We abstract and formalize this as follows. Let A be an alternative-set, and $V_1, \ldots, V_n \subseteq \mathbb{R}^A$ be the valuation domains of n players. We also have a cost function $c \in \mathbb{R}^A$, where $c(a)$ gives the cost incurred by the mechanism in executing alternative $a \in A$. We assume that $c(a) \geq 0$ for all

a. The cost function c is *common knowledge*. In this setting, social-welfare maximization (SWM), which is also termed (economic) efficiency, translates to computing the alternative that maximizes $\sum_i v_i(a) - c(a)$. (Observe that this is *precisely* the SWM-objective *if one treats the mechanism designer also as a player* whose valuation set consists of the single valuation $-c$.) Ideally, we would like to implement the function g that computes an optimal solution to the SWM problem, using prices p_i such that $\sum_i p_i(v)$ is "roughly" equal to $c\big(g(v)\big)$ for every $v \in V$. But the above SWM problem is notoriously intractable for many problems, and even obtaining good approximations is intractable (the difficulty arises due to the $-c(a)$ term in the objective function). To remedy this, we change the form of the objective function. For simplicity, we restrict ourselves to the setting where each V_i is single-dimensional and of the form $V_i = \{v_i\alpha_i : v_i \geq 0\}$, with $\alpha_i \in \{0,1\}^A$ (the α_is are common knowledge). An alternative a can then equivalently be viewed as a set of players, namely, the set $\{i : \alpha_i(a) = 1\}$ of winners; α_i is then simply the indicator set-function $\alpha_i(S) = 1$ if $i \in S$, 0 otherwise. So A is now a subset of 2^N, where $N = \{1, \ldots, n\}$. For an alternative $a \equiv S \subseteq N$, we can write $\sum_i v_i(a) - c(a) = \sum_i v_i - \big(\sum_{i \notin S} v_i + c(S)\big)$. We consider the goal of *minimizing* $\mathsf{SC}(S) := c(S) + \sum_{i \notin S} v_i$ over all $S \in A$, which is called the *social-cost minimization* problem. Minimizing the social cost is also often an *NP*-hard problem, so we seek an approximation algorithm for this problem that is implementable using prices, where the total price charged "roughly" recovers the cost incurred by the mechanism. More precisely, we want to design a computationally efficient mechanism $M = \big(f, \{p_i\}\big)$ such that

(i) f is a β-approximation algorithm for the social-cost objective: i.e., for every input $v = (v_1, \ldots, v_n)$, $f(v)$ is efficiently computable, and $f(v) = S^*$ such that $\mathsf{SC}(S^*) \leq \beta \cdot \big(\min_{S \in A} \mathsf{SC}(S)\big)$.

(ii) M is truthful and individually rational, i.e., $p_i(v) \leq v_i\alpha_i(f(v))$ for every v, player i.

(iii) The prices recover at least a γ-fraction of the mechanism-designer's cost: i.e., for every input v, we have $\sum_i p_i(v) \geq c(f(v))/\gamma$.

We call such a mechanism a *β-approximation, γ-budget-balanced, truthful mechanism*. Suppose that we have a β-approximation algorithm f for the social-cost objective such that f has the monotonicity property that for every i, and every v_{-i}, $\alpha_i\big(f(v, v_{-i})\big)$ is a nondecreasing function of v. Notice that this means that there exist prices that implement f, but these prices need not necessarily satisfy the budget-balance condition (iii). Suppose also that f has the following *"no-bossiness"* property: for every i, every v_{-i}, every $v_i, v_i' \in V_i$, if $\alpha_i\big(f(v_i, v_{-i})\big) = 1 = \alpha_i\big(f(v_i', v_{-i})\big)$ then $f(v_i, v_{-i}) = f(v_i', v_{-i})$. This says that if i is a winner, and fixing the others' inputs and changing only i's input leaves i unaffected, then the alternative computed must also be unaffected. Suppose also that A is downwards closed: if $T \in A$ and $S \subseteq T$ then $S \in A$; and c is an increasing function: $c(S) \leq c(T)$ if $S \subseteq T$. We will show that under these conditions, one can use f (in a black-box fashion) to devise an $O(\beta \log n)$-approximation, 1-budget-balanced truthful mechanism.

Given an input v, the set of winners is computed as follows. Fix an ordering of the players. We compute $S_1 = f(v)$. If for every player $i \in S_1$, we have $v_i \geq \frac{c(S_1)}{|S_1|}$, we return S_1. Otherwise, we drop the first player (according to the ordering) in S_1 for which $v_i < \frac{c(S_1)}{|S_1|}$. Let $S_2 \subseteq S_1$ be the remaining set of winners. We now check if there is some $i \in S_2$ such that $v_i < \frac{c(S_2)}{|S_2|}$; if so, we drop the first such player in S_2 and if not, we return S_2. We repeat this process, each time dropping the first player in the current set whose value is less than the average cost of the current set if such a player exists. Note that the process must terminate since we will eventually reach the empty set.

(a) Prove that the above algorithm satisfies the monotonicity property.

(b) Prove that the above algorithm is an $O(\beta \log n)$-approximation algorithm for the social-cost problem.

(c) Prove that if T is the set returned by the algorithm (which could be empty), then for every $i \in T$, the price that i pays is at least $\frac{c(T)}{|T|}$. Recall that the price that i pays is his threshold value, i.e., the value at which i ceases to be a winner given the others' current bids. (Also, every non-winner pays 0, so we get individual rationality.)

Parts (a), (b) and (c) show that the above algorithm along with the prices that implement it, give an $O(\beta \log n)$-approximation, 1-budget-balanced, truthful mechanism.

(d) Consider the setting of the set-cover problem and the LP (P) described earlier, but now suppose that the set-system $(U, \mathcal{S})$, and the (nonnegative) weights (or costs) of the sets in $\mathcal{S}$ are common knowledge. Each *element* e of the universe U is now a player, whose private value v_e is the value that e obtains if he is covered by some set. Thus, the social-cost minimization problem is to find a collection of sets $\mathcal{S}' \subseteq \mathcal{S}$ that minimizes $\sum_{S \in \mathcal{S}'} w_S + \sum_{e \text{ not covered by } \mathcal{S}'} v_e$.

Suppose you are given an LP-based ρ-approximation algorithm $\mathcal{A}$ for the set-cover problem; that is, $\mathcal{A}$ returns a set cover of weight at most $\rho \cdot OPT(w)$ for any input $w = (w_S)_S$. Show that $\mathcal{A}$ can be used to obtain a monotone, $(\rho + 1)$-approximation algorithm for the social-cost problem that satisfies the no-bossiness property.

Remark: Thus, parts (a)–(d) show that an LP-based ρ-approximation algorithm for the min-cost problem, yields an $O\big((\rho+1)\log n\big)$-approximation, 1-budget balanced, truthful mechanism. In fact, the above reduction is quite generic and applies to a variety of problems including the vertex cover, facility-location, and Steiner-tree (or forest) problems.

Q4: In this question, we will derive a characterization of implementability for social choice functions over multidimensional domains, and see some applications. You may use the following fact:

> Let $D = (N, A)$ be a directed graph with node-set N, arc-set A, and weights $\{w_a\}_{a \in A}$ on the arcs, such that there is a directed path from u to v for any two nodes $u, v \in N$. Then, for any $u, v \in N$, D has shortest path from u to v iff D has no negative-weight cycles.

(a) Let $A, V_1 \subseteq \mathbb{R}^A, \ldots, V_n \subseteq \mathbb{R}^A$ be a mechanism-design setup, and $f : V \mapsto A$ be a given function. Fix a player i, and $v_{-i} \in V_{-i}$, and assume (for notational simplicity) that $\{f(v_i, v_{-i}) : v_i \in V_i\} = A$. For any $a, b \in A$, define $\delta_{ab} = \inf\{v_i(a) - v_i(b) : v_i \in V_i, f(v_i, v_{-i}) = a\}$. Prove that f is implementable iff for every subset $\{a_1, \ldots, a_k\} \subseteq A$, we have $\delta_{a_1 a_2} + \delta_{a_2 a_3} + \ldots + \delta_{a_{k-1} a_k} + \delta_{a_k a_1} \geq 0$. Specify how prices implementing f can be computed from the δ-values.

(b) Consider the SWM problem for combinatorial auctions (CAs) with *unknown* single-minded players. Recall that we have a set U of m non-identical indivisible items, and n players. Each player i's private information is a tuple (w_i, S_i) specifying the set he is interested in, and his value for obtaining that set (or any superset) of items. We say that an algorithm f (which is required to output an allocation) is *exact* if on every input bid $\{(w_i, S_i)\}_{i=1}^n$, the set of items allotted to each player i is either $\emptyset$ or S_i; equivalently, every winning player (according to the input bids) is allotted exactly his set (and not any superset).

Show that if f is an exact algorithm, then f is implementable iff for every player i and every $v_{-i} \in V_{-i}$, we have: (i) For every set $S \subseteq U$, there is a threshold $t = t(S)$ such that for all

$w < t$, the allocation $f\big((w,S),v_{-i}\big)$ assigns the set $\emptyset$ to i, and for all $w > t$, $f\big((w,S),v_{-i}\big)$ assigns the set S to i. Define $t(S) = 0$ if $f\big((w,S),v_{-i}\big)$ assigns S to i for all $w \geq 0$; for $S \neq \emptyset$, define $t(S) = \infty$ if $f\big((w,S),v_{-i}\big)$ assigns $\emptyset$ to i for all $w \geq 0$. (Note that $t(\emptyset) = 0$.) (ii) If $S \subseteq T$, then $t(S) \leq t(T)$.

(c) Prove that the ordering algorithm discussed in class (where we order sets in decreasing order of $\frac{w_i}{\sqrt{|S_i|}}$) is exact and satisfies properties (i) and (ii) above, and hence is implementable.

(d) Recall the makespan-minimization problem on *unrelated* machines: there are m machines and n jobs. Each machine i is a player whose private information is a vector $(t_{ij})_j$ specifying the time taken to process each job j. We consider the restricted setting where each $t_{ij} \in \{L_j, H_j\}$, $L_j \leq H_j$ (that is, each job j has either "low" (L_j) or "high" (H_j) processing time on each machine), and the fractional scheduling problem, where we seek to compute a *fractional* allocation of jobs to machines with (near-) optimum makespan. So $A = \{x \in \mathbb{R}^{mn} : \sum_i x_{ij} = 1 \text{ for all } j, \quad x_{ij} \geq 0 \text{ for all } i,j\}$, and a vector $t_i = (t_{ij} \in \{L_j, H_j\})_j$ of processing times yields the function $t_i : A \mapsto \mathbb{R}$, where $t_i(x) = \sum_j t_{ij}x_{ij}$ for $x \in A$. Let $V_i \subseteq \mathbb{R}^A$ denote the set of all such functions. Consider a social choice function $f : V \mapsto A$ where, letting $x = f(t_1, \ldots, t_m)$, we have $x_{ij} \geq \frac{1}{m}$ if $t_{ij} = L_j$ and $x_{ij} \leq \frac{1}{m}$ otherwise. Prove that any such function f is implementable.

(e) (Bonus part) Let $\mathcal{A}$ be a ρ-approximation algorithm for the makespan-minimization problem on unrelated machines in the above restricted setting. Show that $\mathcal{A}$ can be used to obtain an implementable 2ρ-approximation algorithm for the fractional scheduling problem.

Q5: In this problem, we consider *multi-unit combinatorial auctions* (MUCAs), which are CAs where the m items are all *identical*. Thus, each player i's valuation function v_i can be represented as a nondecreasing function $v_i : \{0, 1, \ldots, m\} \mapsto \mathbb{R}_+$ where $v_i(x)$ specifies the value that i receives when he is allotted x items. The SWM problem of finding x_i's in $\{0, 1 \ldots, m\}$ with $\sum_i x_i \leq m$ that maximize $\sum_i v_i(x_i)$ is solvable in time polynomial in m and n. So if the input is specified by listing $v_i(x)$ for each x separately, then the running time is polynomial in the input-size, and one can implement the VCG mechanism efficiently. We are interested in settings where the v_i's are specified more succinctly and the input-length is polynomial in $\log m$; for example, if each $v_i(.)$ is piecewise linear with a constant number of breakpoints then one only needs to list $(x, v_i(x))$ for each breakpoint x. The SWM problem is often *NP*-hard in such settings.

Let $A = \{(x_1, \ldots, x_n) : x_i \in \{0, \ldots, m\}, \ \sum_i x_i \leq m\}$ denote the set of all allocations, and let $A' = \{(x_1, \ldots, x_n) : x_i \in \{0, \ldots, m\}, \ \sum_i x_i \leq m, \ x_i = m \text{ or } x_i \text{ is a multiple of } \lfloor \frac{m}{n^2} \rfloor\}$. Consider the function f that returns the allocation in A' that maximizes the social welfare among all allocations in A'.

(a) Prove that f is a 0.5-approximation algorithm, i.e., show that $\max_{(x_1,\ldots,x_n)\in A'} \sum_i v_i(x_i) \geq 0.5 \cdot \max_{(x_1,\ldots,x_n)\in A} \sum_i v_i(x_i)$.

(b) Briefly argue why f is implementable.

(c) Assuming that for every i and every integer $x \in [0, m]$ one can compute $v_i(x)$ efficiently, show that f can be computed efficiently.

(**Hint:** Use dynamic-programming.)