

CO759: Algorithmic Game Theory — Spring 2008

Instructor: Chaitanya Swamy

Assignment 3

Due: By August 7, 2008

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.

Q1: In this question, we consider graphical games where each node has two strategies, denoted 0 and 1. Recall that p_v is the probability that node v assigns to strategy 1.

(a) Construct graphical-game gadgets that implement the following operations. Each operation has two inputs, a and b , and one output c . Correspondingly, in the gadget constructed, there should be two input nodes a and b and an output node c (and any number of intermediate nodes); the input nodes must be free to choose their strategies, that is, their payoffs should not depend on the strategy-choices of any of the nodes of the gadget.

(i) Brittle comparator: In any NE, if $p_a < p_b$ then $p_c = 1$; if $p_a > p_b$ then $p_c = 0$. (If $p_a = p_b$, p_c is allowed to be arbitrary.)

(ii) XOR: In any NE, if $p_a \in \{0, 1\}$ and $p_b \in \{0, 1\}$, then $p_c = 1$ if $p_a \neq p_b$ and 0 if $p_a = p_b$. (If either p_a or p_b is strictly between 0 and 1, then p_c is allowed to be arbitrary.)

(b) Consider a gadget with one input node a and one output node c with the functionality that $p_c = 1$ if $p_a = 0$ and $p_c = 0$ otherwise. Prove that such a gadget cannot exist by using this gadget to construct a graphical game that has no mixed NE (thus contradicting Nash's existence theorem).

(**Hint:** Construct a feedback loop that forces p_a to be nonzero iff it is zero.)

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 a $O(\log n)$ -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 two B -approximation algorithms for set cover designed using this approach. Let $OPT(w)$ denote the optimal value of the following LP:

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

(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^* (this can be done in polynomial time), and set $\mathcal{S}' = \{S : x_S^* \geq t\}$. We call this algorithm the *LP-rounding algorithm with threshold t* . 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 $B \cdot OPT(w)$.

(Hint and remarks: Thus, in both parts (b) and (c) we obtain a B -approximation algorithm that is implementable. Part (c) of course subsumes part (b), so it suffices to prove only part (c), but it may be easier to make progress on part (b). For part (c) consider an optimal solution y^* to the dual problem:

$$\max \sum_e y_e \quad \text{subject to} \quad \sum_{e \in S} y_e \leq w_S \quad \forall S; \quad y_e \geq 0 \quad \forall e. \quad (\text{D})$$

For each S such that $x_S^* > 0$, use complementary slackness to bound w_S in terms of the dual values y_e^* , and thereby bound the weight of the solution in terms of the optimal value of (D).

The greedy algorithm discussed in class can also be viewed as an “LP-based algorithm” in the following sense. One can interpret the greedy algorithm as constructing simultaneously a $\{0,1\}$ solution to (P) and a dual infeasible solution $y = (y_e)_e$ with the same objective value, such that $(y_e/O(\log n))_e$ is a feasible solution to (D); this also proves 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 $V_1, \dots, V_n \subseteq \mathbb{R}^A$ 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 $c(g(v))$ 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_i s 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 $N = \{1, \dots, n\}$. For an alternative $a \equiv S \subseteq N$, we can write $\sum_i v_i(a) - c(a) = \sum_i v_i - (\sum_{i \notin S} v_i + c(S))$. We consider the goal of *minimizing* $\text{SC}(S) := c(S) + \sum_{i \notin S} v_i$ 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 $M = (f, \{p_i\})$ such that

- (i) f is a β -approximation algorithm for the social-cost objective: i.e., for every input $v = (v_1, \dots, v_n)$, $f(v)$ is efficiently computable, and $f(v) = S^*$ such that $\text{SC}(S^*) \leq \beta \cdot (\min_{S \in A} \text{SC}(S))$.
- (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} , $\alpha_i(f(v, v_{-i}))$ 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 $\alpha_i(f(v_i, v_{-i})) = \alpha_i(f(v'_i, v_{-i}))$ then $f(v_i, v_{-i}) = f(v'_i, v_{-i})$. This says that if 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, using f , one can devise a $O(\beta \log n)$ -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. (In fact, the following *weaker* no-bossiness property suffices: if $\alpha_i(f(v_i, v_{-i})) = 1 = \alpha_i(f(v'_i, v_{-i}))$ then $f(v_i, v_{-i}) = f(v'_i, v_{-i})$.)

Q4: Consider the SWM problem for combinatorial auctions (CAs) with unknown single-minded players. 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. As usual, let $V_i = \{(w_i, S_i) : w_i \geq 0, S_i \subseteq U\}$ denote the set of all possible private valuations of player i . An algorithm f is required to output an allocation of items to players. We say that f 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).

- (a) Show that if f is an exact algorithm, if f satisfies weak-monotonicity then 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 $f((w, S), v_{-i})$ assigns the set \emptyset to i , and for all $w > t$, $f((w, S), v_{-i})$ assigns the set S to i . Define $t(S) = 0$ if $f((w, S), v_{-i})$ assigns S to i for all $w \geq 0$; for $S \neq \emptyset$, define $t(S) = \infty$ if $f((w, S), v_{-i})$ assigns \emptyset to i for all $w \geq 0$. (Note that $t(\emptyset) = 0$.) (ii) If $S \subseteq T$, then $t(S) \leq t(T)$. (In fact, weak-monotonicity is equivalent to (i) and (ii) if f is exact.)
- (b) Show that if f is exact and properties (i) and (ii) hold, then the following price-functions implement f . Suppose $f(v)$ assigns the set S_i to i . Let $p_i(v) = t(S_i)$.

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 $v_i : \{0, 1, \dots, m\} \mapsto \mathbb{R}_+$ 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 $\{0, 1, \dots, m\}$ 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 $v_i(\cdot)$ 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 $A = \{(x_1, \dots, x_n) : x_i \in \{0, \dots, m\}, \sum_i x_i \leq m\}$ denote the set of all allocations, and let $A' = \{(x_1, \dots, x_n) : x_i \in \{0, \dots, m\}, \sum_i x_i \leq m, x_i = m \text{ or } x_i \text{ is a multiple of } \lfloor \frac{m}{n^2} \rfloor\}$. Consider the function f that returns the allocation in A' maximizing the social welfare among all allocations in A' .

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