Homework #2
CS599 Fall 2012

Due Friday 11/9

General Instructions The following problems are meant to be challenging. Feel free to discuss with fellow students, though please write up your solutions independently and acknowledge everyone you discussed the homework with on your writeup. Additionally, please provide mathematical proofs of all claims you make in your solutions.

Problem 1. Payment Computation in Single-Parameter Problems.
In this problem, we consider general techniques for payment computation in single parameter mechanism design problems. We will show that, under mild conditions, a generic procedure converts a monotone allocation rule to a truthful mechanism with an arbitrarily small degradation in the approximation ratio. We then apply these ideas to the knapsack allocation problem considered in class.

We consider single parameter problems with n players and a set of allocations $\Omega \subseteq \mathbb{R}_+^n$. We assume each player i's typespace is $[0, \infty]$, and that a player i with type t_i has utility $t_i x_i - p_i$ for allocation x and payments p. Though we only consider problems where players have nonnegative values for allocations, our arguments apply equally well to cost problems where player typespaces are $[-\infty, 0]$.

We begin with the observation that any monotone allocation rule can be approximated via another monotone allocation rule which depends only high order bits of each player's reported type. Formally, fix some $\epsilon > 0$, and for each $t \in \mathbb{R}$ let $\lfloor t \rfloor_\epsilon$ denote the largest integer power of $(1+\epsilon)$ not exceeding t — specifically, $\lfloor t \rfloor_\epsilon = \max\left\{(1+\epsilon)^k : k \in \mathbb{Z}, (1+\epsilon)^k \leq t\right\}$. For each allocation rule $f : \mathbb{R}_+^n \to \Omega$, we consider the allocation rule f^ϵ which rounds the input types using the map $t \to \lfloor t \rfloor_\epsilon$ and then runs f.

$$f^\epsilon(t_1, \ldots, t_n) = f(\lfloor t_1 \rfloor_\epsilon, \ldots, \lfloor t_n \rfloor_\epsilon))$$

a. (2 points). Show that if f is a monotone allocation rule, then f^ϵ is a monotone allocation rule for every $\epsilon > 0$.

b. (3 points). Show that if f is an α-approximation algorithm for welfare maximization over Ω, then f^ϵ is a $(1 + O(\epsilon))\alpha$-approximation algorithm for welfare maximization over Ω.
Note: It is worth noting that there is little special about welfare here — any objective function which is *lipschitz continuous* with respect to player types, such as makespan or most notions of "fairness" — would yield a similar guarantee.

Next we argue that, for monotone allocation rules f satisfying a mild condition, truth-telling payments $p^\epsilon(t)$ for the rounded allocation rule f^ϵ can be computed efficiently. We say an allocation rule $f : \mathbb{R}_+^n \to \Omega$ has *B-bounded dependence* if, for every type profile t, if player i is such that $t_i \leq \frac{\min_{j \neq i} t_j}{B}$ then $f(t) = f(0, t_{-i})$. In other words, if a player i's value per unit good is much smaller, by a factor of at least B, than that of every other player, then it is as if player i's value is 0.

c. (4 points). Show that if allocation rule f has B-bounded dependence, then Myerson payments $p^\epsilon(t)$ for the rounded allocation rule f^ϵ can be computed in time polynomial in n, $\frac{1}{\epsilon}$, $\log B$, and the number of bits used to describe the types t — namely $\log \frac{\max_i t_i}{\min_{i:t_i>0} t_i}$. You may assume black-box access to f, with calls to f taking unit time.

d. (3 points). Recall the monotone allocation rule for the knapsack allocation problem considered in class, which we showed was a 2-approximation algorithm for the social welfare. Our allocation rule f simply sorted players' jobs in decreasing order of density, and fractionally allocated jobs to the knapsack in this order until the knapsack overflowed. Then, either the overflow job or all other jobs were discarded, whichever was better.

Show that this allocation rule has B-bounded dependence, for some B at most most exponential in the number of bits used to describe the knapsack allocation instance. Conclude from parts a, b, and c that, for each $\epsilon > 0$, there is a truthful, polynomial time, $(2+\epsilon)$-approximation mechanism for knapsack allocation, which runs in time polynomial in the description of its input and $\frac{1}{\epsilon}$.

Note: Not all monotone allocation rules have bounded dependence. For example, our allocation rule for the makespan problem always assigned a non-zero amount of work to every machine, and this amount of work depended nontrivially on the machine's reported speed no matter how slow. In such cases, however, it is often the case that we can modify an allocation rule to one with exponentially-bounded dependence without degrading its approximation ratio by much, and preserving monotonicity. For example, our allocation rule for the makespan problem can be "tweaked" as follows: if a machine much slower than any other machine is assigned any work, this work is simply moved to the fastest machine. If you are interested, I invite you to verify that this preserves monotonicity, and does not degrade the approximation ratio by much if B is chosen appropriately.

Problem 2. Characterizations of Truthfulness in Multi-parameter Problems.
In this problem, we will examine three characterizations of incentive compatibility in general multi-parameter domains. The first characterization, known as the *taxation principle*, is a direct characterization of dominant-strategy truthfulness that references both the allocation rule and the payment rule. The other two, *cycle monotonicity* and the *matching property*, are equivalent characterizations of dominant-strategy implementable allocation rules,[1] and do not reference payments directly.

Setup: Recall that a mechanism design problem is given by a set of allocations Ω, and for each player $i \in \{1, \ldots, n\}$ a typespace T_i, and a valuation map $v_i : T_i \times \Omega \to \mathbb{R}$. We consider deterministic mechanisms in this problem for simplicity, though our characterizations apply equally well to randomized mechanisms with essentially no modification. We also assume Ω is a finite set. Recall that a deterministic mechanism is a pair (f, p), where $f : T_1 \times \ldots \times T_n \to \Omega$ is an *allocation*

[1] Recall that an allocation rule f is *dominant-strategy implementable* if there exists a payment rule p such that (f, p) is a dominant-strategy truthful mechanism.

rule and $p : T_1 \times \ldots \times T_n \to \mathbb{R}^n$ is a *payment rule*. Also recall that such a mechanism is dominant strategy truthful if, for each player i, types $t_i, t_i' \in T_i$ of player i, and type profiles $t_{-i} \in T_{-i}$ of players other than i, the following inequality holds

$$v_i(t_i, f(t_i, t_{-i})) - p_i(t_i, t_{-i}) \geq v_i(t_i, f(t_i', t_{-i})) - p_i(t_i', t_{-i}).$$

a. (4 points) . Prove the *taxation principle*, which is formally the following statement.

Fact (Taxation Principle)**.** *A deterministic mechanism (f, p) is dominant-strategy truthful if and only if for each player i and type profile $t_{-i} \in T_{-i}$ of players other than i, there is a menu $M_i(t_{-i}) \subseteq \Omega \times \mathbb{R}$ of allocation/price pairs, where each allocation appears at most once in the menu, such that the following holds for every $t_i \in T_i$*

$$(f(t_i, t_{-i}), p_i(t_i, t_{-i})) \in \operatorname*{argmax}_{(\omega,p)\in M_i(t_{-i})} v_i(t_i, \omega) - p$$

In other words, the taxation principle states the following necessary and sufficient condition for truthfulness of a mechanism: for every player i and fixed reports of other players, the mechanism presents to i a menu containing a subset of all allocations, each of which is associated with a price. Then, once the player reports his type, the mechanism chooses the allocation/price pair in the menu maximizing i's utility.

b. (8 points) . Show that an allocation rule $f : T_1 \times \ldots \times T_n \to \Omega$ is dominant-strategy implementable if and only if it satisfies a property known as *cycle monotonicity*, defined as follows

Definition (Cycle Monotonicity)**.** *An allocation rule f is cycle monotone if for every player i, every type profile t_{-i} of other players, every integer $k \geq 0$, and every sequence $t_i^1, \ldots, t_i^k \in T_i$ of k types for player i, the following holds*

$$\sum_{j=1}^{k} \Big[v_i(t_i^j, \omega_j) - v_i(t_i^j, \omega_{j+1}) \Big] \geq 0$$

when ω_j denotes $f(t_i^j, t_{-i})$ for all $j \in \{1, \ldots, k\}$, and $\omega_{k+1} = \omega_1$.

Hints: After fixing player i and the reports t_{-i} of others, consider a complete directed weighted graph G with nodes corresponding to allocations Ω. Let the weight of edge (ω, ω') correspond the minimum amount by which i prefers ω to ω' when his type t_i is such that $f(t_i, t_{-i}) = \omega$ — i.e. the minimum amount by which i prefers truth-telling over mis-reporting so as to manipulate the allocation rule r into choosing ω', when truth-telling results in ω. Show that f is implementable if and only if the resulting graph has no negative cycles, and show that this is equivalent to cycle monotonicity. To do so, you will have to appeal to the dual linear programming formulation of the shortest path problem on weighted directed graphs, which you can find on Wikipedia or in any combinatorial optimization textbook.
Notes: The special case of cycle monotonicity when $k = 2$ is known as *weak monotonicity*. However, it is worth nothing that when the type spaces of players are convex — in the sense that the valuation space of each player is a convex set of functions mapping allocations to the real numbers — weak monotonicity and cycle monotonicity are equivalent. However, the proof of this fact is very

intricate, and beyond the scope of this homework problem. Also interesting is the following fact: weak monotonicity, when stated for single-parameter problems, can easily be seen to be equivalent to *monotonicity* — this is something you should be able to verify yourself in a few minutes, though you do not have to hand it in.

c. (6 points). Show that an allocation rule $f : T_1 \times \ldots \times T_n \to \Omega$ is dominant-strategy implementable if and only if it satisfies what we will refer to as the *matching property*. To describe the matching property, we define for each player i, types $t_{-i} \in T_{-i}$ of players other than i, and finite subset $S_i \subseteq T_i$ of player i's types, a weighted bipartite graph $G = G_i(t_{-i}, S_i)$ as follows. The left hand side of G will consist of S_i, and the right hand side will be the *multiset* of allocations $\{f(t_i, t_{-i}) : t_i \in S_i\}$. The weight of an edge (t_i, ω) in this bipartite graph will be simply $v_i(t_i, \omega)$. We can now state the matching property.

Definition (Matching Property)**.** *An allocation rule f is said to satisfy the matching property if, for every player i, types $t_{-i} \in T_{-i}$ of players other than i, and finite subset $S_i \subseteq T_i$ of player i's types, the bipartite matching $\{(t_i, f(t_i, t_{-i})) : t_i \in S_i\}$ is a maximum-weight bipartite matching in $G_i(t_{-i}, S_i)$.*

In other words, an allocation rule satisfies the matching property if it can be thought of as computing, for every finite subset S_i of player i's types and fixed reports of other players, a maximum weight matching of player i's types S_i to some multiset of allocations.
Hint: Show that the matching property is equivalent to cycle monotonicity.

Problem 3. An Application of Maximal in Range Algorithms.
We consider a multi-parameter generalization of the knapsack allocation problem known as *multi-minded multi-unit auctions*. In this problem, there are n players and m *identical* items, and a publically known integer parameter k. An allocation is simply a partial assignment of items among players — since the items are identical, such an assignment can be thought of as a tuple of nonnegative integers $(m_1, \ldots, m_n)$ such that $\sum_{i=1}^{n} m_i \leq m$, where m_i denotes the number of items assigned to player i. Each player i's type is a list of k tuples $(s_{i1}, v_{i1}), \ldots, (s_{ik}, v_{ik})$ where $s_{i\ell} \in [m]$ and $v_{i\ell} \in \mathbb{R}_+$ for each $\ell \in \{1, \ldots, k\}$, and both $s_{i\ell}$ and $v_{i\ell}$ are non-decreasing in ℓ . A tuple $(s_{i\ell}, v_{i\ell})$ can be thought of as player i's value for receiving anywhere between $s_{i\ell}$ and $s_{i,\ell+1} - 1$ items. Formally, player i's value for receiving m_i items is $\max\{v_{i\ell} : \ell \in \{1, \ldots, k\}, s_{i\ell} \leq m_i\}$.

Observe that, when $k = 1$, this is a restatement of the knapsack allocation problem we have looked at in class (albeit where the size of a player's job is private data). In this problem, we will use the maximal in range paradigm to design a 2-approximate, truthful mechanism. Crucially, since the items are identical, and integers in $\{1, \ldots, m\}$ can be represented using $\log m$ bits in the problem input, we will require our mechanism to run in time polynomial in $\log m$, n, and k.[2]

a. (8 points). The range of our maximal in range allocation rule will be the family $\mathcal{R}$ of allocations $(m_1, \ldots, m_n)$ where, for each i, either m_i is an integer multiple of $\lceil \frac{m}{n^2} \rceil$ or $m_i = m$. Show that a maximal in range algorithm with range $\mathcal{R}$ is a 2-approximation algorithm for multi-minded multi-unit auctions.

[2]It is worth noting that, if we only required runtime to be polynomial in m,n, and k, the welfare maximization problem can be solved exactly using dynamic programming.

b. (7 points). Show that a maximal in range algorithm with range $\mathcal{R}$ can be implemented in time polynomial in n, $\log m$, and k.

Conclusion. Parts a and b, combined with the fact that truth-telling payments can be computed for a maximal-in-range allocation rule with poly(n) overhead in runtime, yield a 2-approximate truthful mechanism for multi-minded multi-unit auctions that runs in time poly($n, k, \log m$).

Problem 4. An application of the Lavi-Swamy Technique.
We consider a generalization of combinatorial allocation, incorporating constraints on the assignment of items to players. We refer to this generalization as *graph-constrained combinatorial allocation (GCCA)*. Specifically, there are n players and m items, where items correspond to the edges E of a publically known undirected graph $G = (V, E)$. As usual in combinatorial auctions, each allocation is a (partial) mapping from items to players. However, we incorporate an additional constraint here: no two items (i.e. edges) assigned to different players may share a vertex $v \in V$. This models, for example, situations where selling an item $e = (v_1, v_2)$ to a player i requires commiting two physical resources (v_1, v_2) — say servers, routers, etc — exclusively to player i.

It remains to describe the type spaces and valuations of players. We consider graph-constrained combinatorial allocation when players have *additive valuations* over items. Specifically, each player i's type is a vector $v_i \in \mathbb{R}_+^E$, and his value for an allocation in which he receives a bundle $S \subseteq E$ is simply $\sum_{e \in S} v_i(e)$.

In this problem, you will design an approximation mechanism for welfare maximization in GCCA by applying the Lavi-Swamy linear programming technique.

a. (7 points). Write an integer linear program (ILP) encoding welfare maximization in GCCA when players have additive valuations. Your ILP should use a variable $x_{ie} \in \{0, 1\}$ for each player i and item $e \in E$, as well as a variable $y_{iv} \in \{0, 1\}$ for each player i and vertex $v \in V$. Recall that an integer linear program is a linear program constrained to its integer solutions.

b. (3 points). As stated in class, the Lavi-Swamy technique required the design of a packing integer linear program PILP encoding feasible allocations. Recall that a packing integer linear program is an integer linear program whose linear programming relaxation has a feasible set $P \subseteq \mathbb{R}_+^d$ that is *downwards closed* — formally, whenever $x \in P$ and $0 \leq y \leq x$ (coordinate-wise) then $y \in P$. Assuming you did part a above correctly, the feasible set of your LP will not be downwards closed. However, it is easy to verify that the Lavi-Swamy technique applies essentially unchanged so long as the projection of the feasible set of your linear program onto the variables that actually appear in the objective — namely the x_{ie} variables in our case — is downwards closed. Prove that the projection of your linear program onto the x_{ie} variables is indeed downwards closed.

c. (10 points). The Lavi-Swamy technique, as stated in class, requires a bound on the *integrality gap*[3] of the linear programming relaxation of the PILP and an approximation algorithm which *shows* that bound. Since your LP contains auxiliary variables y_{iv} which do not appear in the objective, it is easy to verify that it is sufficient to exhibit a bound on the integrality gap of the projection of your LP onto the relevant variables — namely the x_{ie}'s.

[3]Recall that a downwards-closed polytope $P \subseteq \mathbb{R}_+^d$ has integrality gap at most $\alpha \geq 1$ if, for all $v \in \mathbb{R}_+^d$, we have $\max\{v^T x : x \in P\} \leq \alpha \max\{v^T x : x \in P, x \in \mathbb{Z}^d\}$.

Show such a bound by exhibiting a polynomial time algorithm that *shows an integrality gap* of 2 for the projection of your LP onto the relevant variables. Such an algorithm must take as input an instance of GCCA with (arbitrary) additive valuations, and produce a (possibly random) feasible allocation whose (expected) welfare is at least half the optimal value of your linear programming relaxation.

Hints: One such algorithm solves your LP relaxation, and then rounds the fractional solution (x, y) to get a feasible allocation. The rounding procedure proceeds as follows: while there is an un-assigned item, choose a player i uniformly at random, and award i a random subset of remaining items using some distribution over bundles that depends on the variables $\{y_{iv}\}_{v \in V}$. If you can't prove a bound of 2, it is much easier to get constants worse than 2 — you will receive partial credit for such constants.

Conclusion. Parts a,b, and c, taken together, provide all the ingredients needed for application of the Lavi-Swamy theorem. This yields a polynomial time, dominant-strategy truthful, 2-approximation mechanism for GCCA with additive valuations.