1 Overview

In the previous lecture, we introduced the mechanism design problem, then proved the revelation principle showing that every environment and choice function that is DOM-implementable has a truthful implementation. We then proved the Gibbard-Satterthwate Theorem showing that in unrestricted environments, only dictatorial choice functions are DOM-implementable. In this lecture, we will explore a restricted environment, the Clarke-Groves-Vickrey environment, in which an important set of choice functions can be implemented.

2 The Clarke-Groves-Vickrey Environment

2.1 Definition

A Clarke-Groves-Vickrey (CGV) environment is one for which

1. The possible set of utilities is $\mathbb{R}^n$, all possible vectors of the form $\theta = (\theta_1, \ldots, \theta_n)$
2. The set of outcomes, $C$, is of the form $C = F \times \mathbb{R}^n$, where $n$ is the number of agents, and $F$ is a (usually finite) set of feasible solutions; each outcome is thus $(s, p)$, where $s \in F$ is a feasible solution, and $p = (p_1, \ldots, p_n)$ is a set of payments to the agents;
3. There exist functions $v_i, i = 1, \ldots, n$, so that if $u_i$ is the utility function for agent $i$, then $u_i$ can be written

$$u_i(s, p) = v_i(s, \theta_i) + p_i$$

for some $\theta_i \in \mathbb{R}$,
4. The choice function $f$ satisfies

$$f(\theta) = (s(\theta), p(\theta))$$

where $s(\theta)$ is an outcome $s \in F$ that maximizes $\sum_i v_i(s, \theta_i)$.

Momentarily we will give conditions on the payment function $p(\cdot)$ such that the above environment is implementable.

The functions $v_i$ are considered known to all, in particular to the game designer. The $\theta$ that describes a particular instance of each agent is considered private to the agent. Since in a CGV environment the utility functions are entirely parametrized by the $\theta$’s, we interchange the two when it is convenient, such as in writing $f(\theta)$.

The canonical example of a CGV environment is that of a public works project. The set $F$ gives the various project outcomes (e.g. build, do not build) and for any such outcome $s, v_i(s, \theta_i)$ describes the utility of agent $i$ if $s$ occurs and its type is $\theta_i$. The goal is to choose the outcome that maximizes these utilities using payments as a tool to encourage the agents to act truthfully.

Another example is that of an allocation of a good. Here we let $F = \{1, \ldots, n\}$, i.e. the outcome is the agent to which we give the good. We also let $v_i(s, \theta_i) = \delta_i, \theta_i$, where $\delta_i$ is the Kronecker delta. Namely, if agent $i$ has type $\theta_i$ then its utility if it receives the good is $\theta_i$, otherwise zero. One solution to this mechanism design problem is the well-known Vickrey Auction: We obtain a truthful implementation by setting

$$p_i(\theta) = \begin{cases} 0 & \text{if } i \neq i_{\max} \\ -\theta^{(2)} & \text{otherwise} \end{cases}$$

where $i_{\max} = \arg\max\{\theta_1, \ldots, \theta_n\}$ (ties can be broken arbitrarily) and $\theta^{(2)} = \max\{\theta_1, \ldots, \theta_n\} \setminus \theta_{i_{\max}}$. 

1
2.2 Implementability

The next result is a generalization of the Vickrey auction. Recall in the previous lecture we showed that if a general environment can be implemented, it can be implemented truthfully.

**Theorem 1** A choice function \( f(\theta) = (s(\theta), p(\theta)) \) for a CGV environment can be (truthfully) implemented if (NB: and under quite general conditions only if, even though this part is not proven here)

\[
p_i(\theta) = \sum_{j \neq i} v_j(s(\theta), \theta_j) + h_i(\theta_{-i})
\]

for arbitrary functions \( h_i, i \in \{1, \ldots, n\} \).

**Proof:** It suffices to show that for each agent, advertising its true type is a (weakly) dominant strategy. For fixed true types, suppose \( \theta_i \) is vector of advertised types for which agent \( i \) is telling the truth, and suppose \( \theta'_i \) is another vector of types such that \( \theta'_{-i} = \theta_{-i} \) but \( \theta'_i \neq \theta_i \), i.e. agent \( i \) lies while the other agents report the same type as before. If

\[
u_i(s(\theta'), \theta_i) > u_i(s(\theta), \theta_i),
\]

then

\[
v_i(s(\theta'), \theta_i) + \sum_{j \neq i} v_j(s(\theta'), \theta_j) + h_i(\theta_{-i}) > v_i(s(\theta), \theta_i) + \sum_{j \neq i} v_j(s(\theta), \theta_j) + h_i(\theta_{-i}),
\]

which implies

\[
\sum_{j} v_j(s(\theta'), \theta_j) > \sum_{j} v_j(s(\theta), \theta_j),
\]

which contradicts the fact that \( s(\cdot) \) by definition maximizes the right side. \( \square \)

3 An Example: The Shortest Path

Suppose we are looking for the shortest path through a network where the cost of using an edge is known only to the edge. How can we get the edges to honestly report their weights? More rigourously,

\[
v_e(s, \theta_i) = \begin{cases} -\theta_e & \text{if } e \in s \\ 0 & \text{otherwise} \end{cases}
\]

\[
\max_s \sum_{e \in s} \theta_e
\]

\[
\min_s \sum_{e \in s} \theta_e = \text{shortest path}
\]

The edges will report their true weights if the payoff is

\[
p_e = \begin{cases} 0 & \text{if } e \notin s \\ \theta_e + [\text{dist}_{e=\text{inf}} - \text{dist}] & \text{otherwise} \end{cases}
\]

\[
\sum_{j \neq i} v_j(s(\theta), \theta_j) = \theta_e - \text{dist}
\]

\[
h_i(\theta_{-i}) = \text{dist}_{e=\text{inf}}
\]

The complexity of this algorithm is an open question, maybe \( v^2e \) or \( v^3 \).
4 Task Scheduling (Nisan)

Suppose we are given a set of tasks and two processors. Each processor knows how long it will take to execute each task. That is, processor \( i \) takes \( t^i_j \) time to execute task \( j \). We wish to assign the tasks to minimize the time to completion. That is we wish to find a partition of the tasks \( X \) such that

\[
\min \max \{ \sum_{j \in X} t^i_j, \sum_{j \notin X} t^j \}
\]

Here

\[
\theta_1 = \{ t_1^1, \ldots, t_m^1 \}, \theta_2 = \{ t_1^2, \ldots, t_m^2 \}
\]

\[
v_1(X, \theta) = -\sum_{j \in X} t^i_j
\]

5 Nisan

Theorem 2 There is no (truthful, deterministic) implementation of the task scheduling environment even if all we want is to approximate the optimum within a factor better than 2. NB: approximate factor of 2 is possible:

\[
\min \{ \sum_{j \in X} t^i_j + \sum_{j \notin X} t^j \} \leq 2 \max \{ \sum_{j \in X} t^i_j, \sum_{j \notin X} t^j \}
\]

Proof: Suppose there exists a (truthful) mechanism. Take all \( t^j \), assume \( \theta^1 \) is the truth, then player 1 will lie, but this is a truthful implementation.

Game designer proposes \( \{(X_1, p_1), \ldots, (X_k, p_k)\} \) with fixed \( \theta_2 \).

Player 1 choses the one that maximizes (over \( k \)) \( v_1(X, \theta_1) + p_k \).

Now change the problem: For all tasks in \( X \), \( t^i_j \) goes from 1 \( \rightarrow \epsilon \).

For all tasks in \( X \), \( t^i_j \) goes from 1 \( \rightarrow 1 + \epsilon \).

All \( t^j \) are left unchanged.

Player 1 again chooses \( X \) (since \( \theta_2 \) didn’t change).

The optimum solution splits the 1,1 + \( \epsilon \) equally \( \rightarrow \) \( X \) is 2 \( - \epsilon \) worse.

6 Randomized Algorithm

\[
X \leftarrow \emptyset
\]

\[
\overline{X} \leftarrow \emptyset
\]

\[
p_1 \leftarrow \emptyset
\]

\[
p_2 \leftarrow \emptyset
\]

for \( j = 1, \ldots, k \) do

let \( i \) be a random agent \( \in \{1, 2\} \), \( i' \) the other (\( i' = 3 - i \)).

if \( t^i_j \leq \frac{4}{5} t^i_{j'} \) then \( X \leftarrow j, p_i \leftarrow p_i + \frac{t^i_{j'}}{3} \)

else \( X \leftarrow j, p_{i'} \leftarrow p_{i'} + \frac{t^i_{j'}}{3} \)

Theorem 3 The above randomized, truthful implementation is \( \frac{7}{4} \) approximate.

3