Lecture 8

Lecturer: Michal Feldman *Scribe: P. Klein, R. Schuster, R. Yahel, T. Avni*

1 Introduction

In previous lectures we introduced the VCG mechanism which is an incentive compatible mechanism that allows us to maximize *Social Welfare*. We further introduced *Combinatorial Auctions* model, in which we have n rational players, m items in the group of items M, and for each player i, we have a valuation function

$$V_i : 2^{[M]} \to \mathbb{R}^{\geq 0}$$

We saw that in general, the computational complexity of the VCG mechanism may be exponential in n and m, however, we saw special cases in which the maximizing problem of SW by a $DSIC$ mechanism is polynomial:

1. Additive Valuations - the value of a set of items is the sum of the values of all items in the set, i.e. $\forall S \subseteq M$ and for every player i, $V_i(S) = \sum_{j \in S} V_{ij}$. In this case, we can maximize SW in polynomial time by a second price auction for each item in M.
2. Unit Demand - for each player i, and $\forall S \subseteq M : V_i(S) = \max_{j \in S} V_{ij}$. In this case, we saw that finding the maximal SW is equivalent to the problem of *Max Weight Matching* that can be solved in polynomial time in m and n.
3. Single Minded - for each player i there is a package S_i^* and a value V_i^* so that

$$\forall S \subseteq M, V_i(S) = \begin{cases} V_i^* & S_i^* \subseteq S \\ 0 & o.w. \end{cases}$$

   We saw that there is a DSIC polynomial mechanism that guarantees an approximation of $\sqrt{m}$ of the maximal SW.

We then introduced *Multi Unit Auctions*, i.e. auction of identical items in which every player has a value for k identical values, for each $k \in \{1, 2, \ldots m\}$. In this kind of auctions we demand that the mechanisms we use will be polynomial in n, $\log m$ and t, where t is the number of bits required to represent the maximum value of a player for the item. One of the challenges regarding *Multi Unit Auctions* is the representation problem of the valuation functions of the players, that may be exponential in $\log m$. One of the approaches to this challenge, which we already mentioned, is *Bidding Languages* that allow us to represent some of the valuation functions in a compact form. A second approach on which we discuss today is *Oracle Access*.

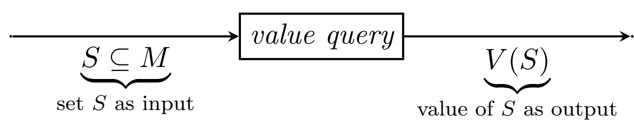
2 Oracle Access

An Oracle is a "black-box" object, that is capable of replying on a certain set of queries.

2.1 Types Of Oracles

Value Query

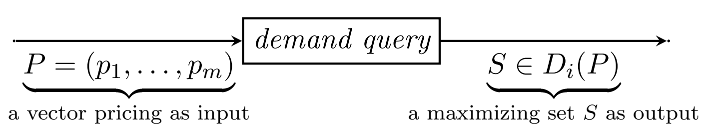
Value queries take as input a set of items $S \subseteq M$ and output the value of this set, i.e. $V_i(S)$ is given by a single oracle query.



Demand Query

Definition 1 *Given prices $P = (p_1, \ldots, p_m)$ for the items in M, the Demand of player i under pricing P is $D_i(P) = \arg\max_{S \subseteq M} \{V_i(S) - \sum_{j \in S} p_j\}$, i.e. a set S that maximizes i's utility.*

Demand Query oracle returns the *Demand* of a player i, given a pricing vector.



A *Demand Oracle* is stronger than *Value Oracle* since any query obtained by *Value Oracle* can be obtained by *Demand Oracle* using polynomial number of queries.

Claim 2 *For any valuation function V, given access to a Demand Oracle, it is possible to compute $V(s)$ using a polynomial amount of queries in m.*
Remark *This claim holds for all Combinatorial Auctions and not only for Multi Unit Auctions, so we allow our algorithms to be polynomial in m.*

Proof:

1. **Computation of the Value of a Single Item**
   Suppose we wish to compute a value of a single item $j \in M$ of a player i, i.e. $V_i(\{j\})$ using *Demand Oracle*. The following algorithm allows us to do so: For achieving the value of $V_i(\{j\})$ we may create a pricing vector P as follows:
   $\forall k \neq j$ set $p_k = \infty$ and perform binary search on $p_j \in [0, 2^t]$. We thus find the threshold p_j above which $D_i(P)$ is empty, and below it $D_i(P)$ is exactly $\{j\}$. This would be $V_i(j)$

2. **Computation of a Marginal Value of a Item**

   Definition 3 *The Marginal Value of a item j given a set of items S of player i is $V_i(j|S) = V_i(\{j\} \cup S) - V_i(S)$, i.e. it is the value change in value for the player i in adding j to the set S.*

   Suppose we wish to query for $V_i(j|S)$ with *Demand Oracle*. We compute $V_i(j|s)$ similarly to $V_i(\{j\})$.
   We set: $\forall j' \in S : p_{j'} = 0$,
   $\forall j'' \notin S, j'' \neq j : p_{j''} = \infty$.
   We now perform binary search for the threshold p_j above which j is included in $D_i(P)$.

3. **Computing the value of S**
   Let S be $\{i_1, i_2, \ldots, i_k\}$. We observe that

   $$V_i(S) = V_i(i_1) + V_i(i_2|\{i_1\}) + V_i(i_3|\{i_1, i_2\}) + ...$$

   Since $V_i(j|S) = V_i(\{j\} \cup S) - V_i(S)$, we get a telescopic series equal to $V_i(S)$, and thus prove the claim.

■

2.2 Identical Items

Input: valuations of the players $V_1, ..., V_n$.
Output: Allocations $m_1, ...m_n$
s.t. $\sum_i m_i \leq m$ (i.e. allocation of maximum $m = |M|$ items).

Dynamic Programming

We define $S(i, k)$ to be the maximal *Social Welfare* for an allocation of k items for the first i players. We define $S(0, k) = S(i, 0) = 0$ for all k and all i.

i

0	0	...	0
0		...	
$\vdots$	$\vdots$	$S(i,k)$	$\vdots$
0		...	

k

The recursive formulation would be

$$S(i,k) = \max_{0 \le j \le k} \{V_i(j) + S(i-1, k-j)\}$$

Maximal SW would be $S(n, m)$.

Reconstructing the allocation:

$k = m$
    for $i = n, ..., 1$:
        let $m_i = j$ s.t. $s(i,k) = v_i(j) + S(i-1, k-j)$
        $k = k - m_i$

Running time: $O(nm^2)$ - nm values of $S(i,k)$, each value requires up to m queries.

Remark The algorithm, as brought here, does not meet the requires complexity, since it is polynomial in m but not in $\log m$. However, it does have some interesting properties:

1. Applying some manipulations, we may achieve a Fully Polynomial Time Approximation Scheme (FPTAS) to achieve approximation of $(1-\epsilon)OPT$. The algorithm would be polynomial in the output and in $\frac{1}{\epsilon}$.
2. We will soon use a variation of this algorithm.

Descending Marginal Value Valuations

Definition 4 *We say that the players have valuations with Descending Marginal Value if*

$$\forall i, k : v_i(k+1) - v_i(k) \le v_i(k) - v_i(k-1)$$

Definition 5 *Market Equilibrium is a price p and an allocation $m_1, ..., m_n$ s.t.*

- $\sum_i m_i = M$ *i.e. all items are allocated,*

- $\forall i : V_i(m_i) - V_i(m_{i-1}) \geq p > V_i(m_{i+1}) - V_i(m_i)$ *(*)*

From (*) follows that for every player i:

$$\forall k : \underbrace{V_i(m_i) - m_i(p)}_{\text{utility of player } i \text{ for } m_i \text{ items}} \geq \underbrace{V_i(k) - kp}_{\text{utility of player } i \text{ for } k \text{ items}}$$

i.e. each player got exactly the number of items that maximizes its utility.

Theorem 6 *Market equilibrium always maximizes SW.*

Proof: For each general allocation $(k_1, \ldots, k_n)$ s.t. $\sum_i k_i \leq m$ we have that for each player i

$$v_i(m_i) - m_i p \geq v_i(k_i) - k_i p$$

If we sum over all i

$$\sum_i (V_i(m_i) - m_i p) \geq \sum_i (V_i(k_i) - k_i p)$$

$$\Downarrow$$

$$\sum_i V_i(m_i) - \underbrace{mp}_{\sum_i m_i = m} \geq \sum_i V_i(k_i) - \underbrace{mp}_{\sum_i k_i \leq k}$$

$$\Downarrow$$

$$\underbrace{\sum_i V_i(m_i)}_{\text{SW in Market Eq}} \geq \underbrace{\sum_i V_i(k_i)}_{\text{SW under general allocation}}$$

$\Rightarrow SW$ under ME is at least as good as SW under any other allocation. ■

Corollary 7 *It is enough to find a ME in order to maximize SW. Hence, it is enough to find a polynomial algorithm to find ME.*

Remark Given a price p, it is possible to use binary-search, in order to find m_i since the marginal value of the players decreases. Thus, we present the polynomial algorithm for finding *Market Equilibrium.*

Algorithm

Note: for simplicity we assume that the values of a different number of items are different for each player, and the values are different between different players, i.e. $\forall i, k, i', k'$: if $(i, k) \neq (i', k')$ then $V_i(k) \neq V_{i'}(k')$.

1. Perform binary search on $p \in [0, \max_i V_i(1)]$ for t rounds (where t is the number of bits required to represent $\max_i V_i(1)$):
   (a) Given p, for each player $1 \leq i \leq n$ perform binary search on $\{0, \ldots, m\}$ to find m_i such that $V_i(m_i) - V_i(m_i - 1) \geq p > V_i(m_i + 1) - V_i(m_i)$.
   (b) If $\sum_{i=1}^{n} m_i \geq m$, then p is too low; if $\sum_{i=1}^{n} m_i \leq m$, then p is too high; otherwise, p was found.
2. Return $(m_1, \ldots, m_n)$.

Running Time

The algorithm is polynomial in n, $\log m$ and t.

Corollary 8 *For identical items with a decreasing marginal value, there exists, as shown, a polynomial algorithm in n, $\log m$ and t that maximizes SW.*

Theorem 9 *There is not a polynomial algorithm in n, $\log m$ and t that maximizes SW for the general case of Multi Unit Auction (without assumptions such as decreasing marginal values). This holds even for two players games.*

Proof: Assume we have two players with two valuations V_1 and V_2, and that there exists a polynomial time algorithm that returns the allocation (m_1, m_2). Assume that for each query k, it holds that $V_1(k) = k$ and $V_2(k) = k$. Since the algorithm is polynomial in $\log m$, it made at most $2m - 3$ queries.

Number of Items	V_1	V_2
1	1	1
2	2	2
$\vdots$	$\vdots$	$\vdots$
m	m	m

3 values the algorithm does not query for $\Rightarrow$ there is a player with two values unknown to the algorithm

Hence, there exists a player, without loss of generality V_1, for which the algorithm does not know two of its values. Therefore, there exists a value $z \neq m_1$, that the algorithm did not query for $V_1(z)$. We observe in a slightly different input for that algorithm, V_1', V_2', which are identical to V_1 and V_2 except for the fact that $V_1'(z) = z + 1$. The algorithm on V_1',V_2' will return (m_1, m_2), since it is a deterministic algorithm and it would not query for $V_1'(z)$. Hence, it return $SW(m_1, m_2) = m$, while $OPT = m + 1$ under the allocation of $(z, m - z)$. ■

Suppose we look of an approximating algorithm for *SW*. If we used the VCG mechanism and changed the allocation so that we only get an approximation of SW it would lose its incentive compatibility.

2.3 MIR (Maximum in Range) Methodology

In this section, we will introduce the MIR methodology, which uses VCG's mechanism on a restricted range/space of allocations. In particular, we will present a MIR algorithm which reaches a 2-approximation to OPT. For simplicity, assume that the number of items m, is a multiplication of n^2.

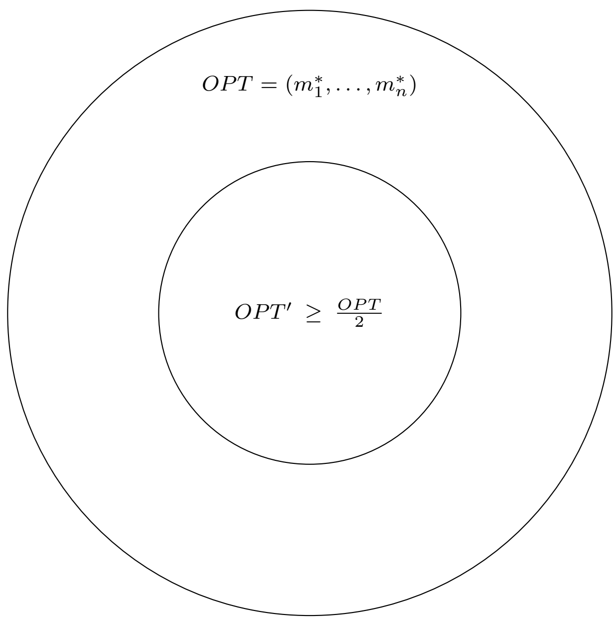


Figure 1: The allocation space - all $(m_1, \dots, m_n)$ for which $\sum_i m_i = m$

As depicted in Figure 1, our algorithm restricts the allocation space into a smaller one, in which there is an allocation OPT' that gives a SW of at least $\frac{OPT}{2}$.

Algorithm

1. Divide the m items to n^2 packages, each has the size of $\frac{m}{n^2}$ items.

2. Use VCG's mechanism on the new set of items which is the n^2 packages, using the dynamic programming algorithm shown at the previously on this lecture.

Running Time

Running time will be $O(n^5)$.
Remark The mechanism is still DSIC, since we use the VCG mechanism, though on a smaller range of the allocation space.

Claim 10 *There exists an allocation in the restricted allocations space of the n^2 packages, which yields a SW of at least $\frac{OPT}{2}$.*

Proof: Denote the optimal allocation $M = (m_1^*, ..., m_n^*)$.
Let i be the player that maximizes m_i^*, i.e. receives more items than any other player ($i = \arg\max_j m_j^*$):

1. Case 1:
$$V_i(m_i^*) \geq \sum_{i \neq j} V_j(m_j^*)$$
i.e. at least half of the optimal SW is obtained from player i. Hence, giving player i all items yields to the desired approximation. This allocation is in fact in our restricted space

2. Case 2:
$$V_i(m_i^*) < \sum_{i \neq j} V_j(m_j^*)$$
Since all m items are allocated and $m_i^* = \max_j m_j^*$, we have that $m_i^* \geq \frac{m}{n}$. We can now spread the m_i^* items of player i among the other $n-1$ player, in order to complete the number of items of each player to a multiple of $\frac{m}{n^2}$. We can do it since $m_i^* \geq \frac{m}{n}$, there are $n-1$ players besides player i, and each one of them needs at most $\frac{m}{n^2}$ items. This allocation is in fact in our restricted allocation space, and since at least half of the SW is derived from all the players that are not i, we get an approximation of $\frac{OPT}{2}$.

■

Theorem 11 *Given a Black-Box model (not necessarily a Value or Demand queries), there is no MIR algorithm that reaches a better approximation than 2-approximation in polynomial time in n and $\log m$. This holds even for two players games.*

Proof: We will use the known claim, that given 2-players game, OPT cannot be obtained in polynomial time. Let A be a MIR algorithm that reaches a better approximation than 2-approximation. If the range of allocations is full, i.e. includes all couples $(m_1, m-m_1)$ for all $m_1 \in \{0, \ldots, m\}$, we get OPT; which contradicts the above mentioned claim. Otherwise, there is at least one missing allocation in our range, let as designate it as $(m_1, m-m_1)$.

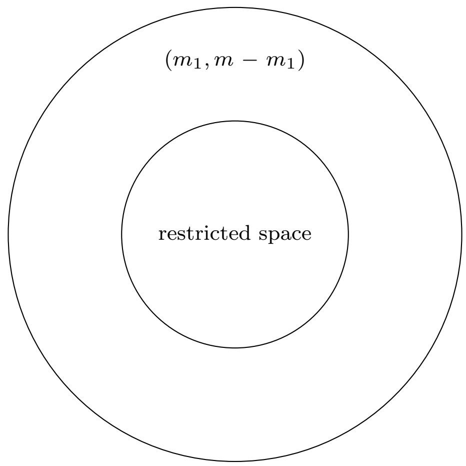


Figure 2: allocation space

Let us observe at the next input:

$$V_1(k) = \begin{cases} 1 & if\ k \geq m \\ 0 & o.w. \end{cases}$$

$$V_2(k) = \begin{cases} 1 & if\ k \geq m - m_1 \\ 0 & o.w. \end{cases}$$

Therefore, $OPT = 2$ by $(m_1, m - m_1)$, but the optimal allocation in the restricted allocations-space yields $SW = 1$. ■

Remark MIR mechanisms that are DSIC define the range independently from the input to the algorithm, otherwise the would have lost their incentive compatibility, since players could manipulate the mechanism.