Algorithmic Game Theory December 16, 2013

Lecture 9

Lecturer: Michal Feldman Scribe: Yael Amsterdamer, Shir Landau Feibish, Adi Vardi

1 Definitions and Notations

This lecture discusses mechanisms in a world of identical products, also known as *Multi-Unit Auctions*. In Multi-Unit auctions, there exist m identical products, and n buyers. Each player $i \in [n]$ has a valuation function $v_i : \{0, ..., m\} \rightarrow R^{\geq 0}$, s.t $v_i(k)$ denotes the value of player i for k such products. As always, we assume normalization ($v_i(0) = 0$) and monotonicity ($\forall i, k : v_i(k) \leq v_i(k+1)$).

2 Challenges

1. Representation: Every valuation contains m numbers. Since the products are identical, the size of the input is bound by $O(log(m))$, and we would therefore like to find algorithms which are polynomial in $log(m)$. Two main approaches which we will discuss shortly are bidding languages and black boxes.
2. Algorithmic: The main algorithmic challenge is to compute an optimal allocation. Denote m_i to be the number of products that player i receives. In any allocation vector $m = (m_1, ..., m_n)$ it must hold that $\sum_{i=1}^{n} m_i \leq m$.
3. Strategic: We would like to plan *truthful* mechanisms.

3 Black box

A black box (denoted BB) is an abstraction of the different types of interactions with the valuation function. An example of the model is depicted in Figure 1.

Examples of the main models of BB include:

1. Value queries: The black box is given a number k, and it outputs the valuation of k products. An example of this type of BB is depicted in Figure 2.
2. Interaction with the valuation function, as treated in the field of communication complexity.

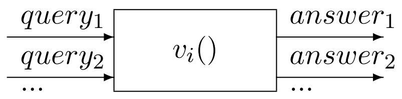


Figure 1: An example of a black box model

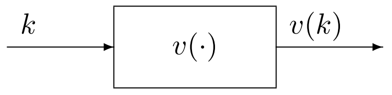


Figure 2: An example of a value query black box

We consider "good" results to be either possibility results for weak queries or impossibility results for strong queries.

4 The bidding languages

These are limited languages which can be used to describe part of the valuations, meaning they can have a compact representation but can still be used to describe certain situations. Examples of the main models of bidding languages include:

1. Single-minded: Each player i has a pair (v_i^*, k_i^*), which defines the valuation function
$$v(k) = \begin{cases} v_i^* & \text{if } k \geq k_i^* \\ 0 & \text{else} \end{cases}$$
2. Step function: The value of player i is defined by a sequence of pairs $(k_1, v_1^*), (k_2, v_2^*), ..., (k_r, v_r^*)$. For example, for the syntactics $v_i = ((2,7),(5,23))$, the semantics are depicted in Figure 3.
3. Piece-wise-linear: denoted PWL, the value of player i is defined by the marginal values which are represented by a sequence of pairs $(k_1, p_1), (k_2, p_2), ..., (k_r, p_r)$. For example, the PWL valuation $((2,7),(5,23))$ is depicted in the Figure 4

Claim 1 *Every step function valuation $(k_1, w_1), ..., (k_j, w_j), ..., (k_r, v_r)$ can be transformed into a PWL valuation while increasing of the size of the representation by a factor of 2.*

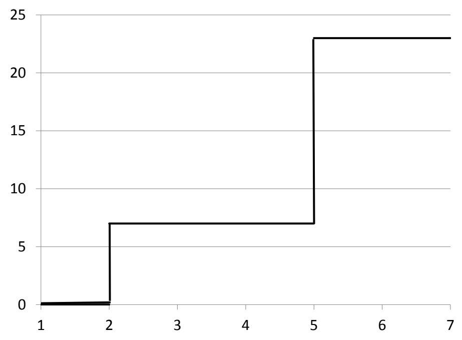


Figure 3: The semantics of the step function with the syntactics $v_i = ((2,7),(5,23))$

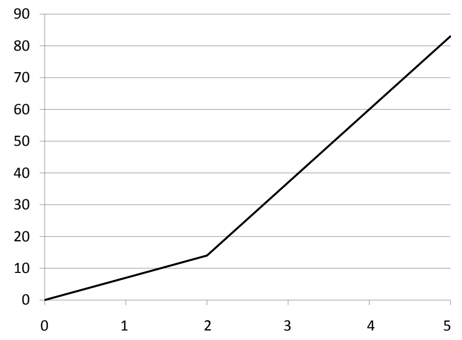


Figure 4: The depiction of the PWL valuation $((2,7),(5,23))$.

Proof Idea: The pair (k_j, w_j) can be transformed to the PWL representation $((k_j - 1, w_j - w_{j-1}), (k_{j+1} - 1, 0))$ giving a factor 2 representation. ■

Claim 2 *There exists a valuation with a compact PWL representation, such that its step function representation is larger by a factor of m.*

Proof Idea: Consider the PWL valuation $(m, 1)$, depicted in Figure 5. Since each k has a different valuation, m pairs are needed in order to represent this valuation as a step function.

Therefore, not every valuation that can have a small PWL representation will also have a small step-function representation. ■

We consider "good" results to be either possibility results for weak languages or impossibility results for strong languages.

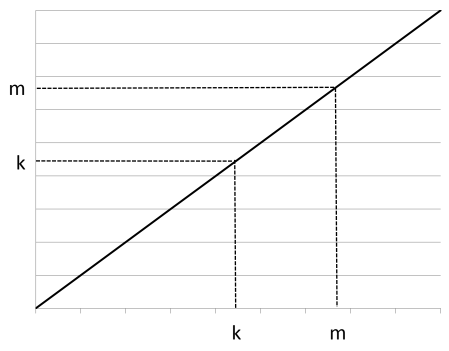


Figure 5: The PWL valuation $(m, 1)$, which gives the function $v(k) = k$.

5 Summary of upcoming topics

1. A dynamic programming algorithm for finding **OPT** in time that is polynomial in m and n. It finds an assignment $(m_1, ..., m_n)$ which
   (a) Maximizes the SW: maximizes $\sum_{i=1}^{n} v_i(m_i)$
   (b) Complies with the restriction: $\sum_{i=1}^{m} m_i \leq m$
2. We will examine a *downward-sloping* valuation, and discuss an algorithm which find **OPT** in time that is polynomial in n, $logm$ and t, where t is the number of bits required to represent a value.
3. Negative results for general valuations:
   (a) Warm up: in a model of value queries (weak queries) we need to find $2m - 2$ queries in order to find **OPT**.
   (b) We will expand the impossibility result to more general models of black-box, and we will show that any interaction with a black-box requires at least $m-1$ queries.
   (c) In a model of bidding languages there does not exist a polynomial algorithm which finds **OPT** (unless P=NP).
4. Approximation algorithms: an $FPTAS$ algorithm for any ϵ gives an approximation of $(1 - \epsilon)$ in time polynomial in n, $logm$, and $\frac{1}{\epsilon}$
5. We will require *truthful* mechanisms.

6 Finding an Optimal Allocation - First Attempt

We will try to run *VCG* with the approximation algorithm. One possible approach is maximal-in-range algorithms: in these algorithms, we can place limitations on the assignments we consider. For example, a limitation can be that player 7 can't receive more than

3 products. This limitation imposes some bounded range of possible solutions, and we can then run *VCG* on the bounded range. The payoffs will still be exactly the damage that every player inflicts on the society, with respect to the bounded range. The utility of each player will be determined by the SW, and therefore the players will have an incentive to tell the truth.

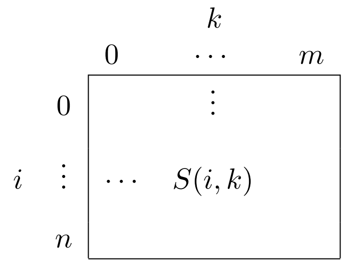
We therefore have the following tradeoff: on the one hand, we would like a wide range which will allow us to get a good approximation. On the other hand, we would like a bounded range so that we can find **OPT** in this range in polynomial time. We will show a truthful maximal-in-range (*MIR*) algorithm, which will give us a 2-approximation in polynomial time. We will show that:

1. There does not exist an *MIR* algorithm which gives a better approximation than a 2-approximation
2. For single-minded bidders there exists a truthful *FPTAS* algorithm.
3. For the general valuation problem, if all of the products are allocated, then even for 2 players we can not get an approximation that is better than a 2-approximation in the model of value queries.

7 An algorithm polynomial in m

We next show an algorithm that computes **OPT** (namely, an allocation that maximizes the SW) in time polynomial in n, the number of buyers, and m, the number of products. Note that the complexity of the algorithm is higher than we would like, as m is exponential in the algorithm input. However, this algorithm will be useful as a basis for approximation algorithms.

The algorithm uses dynamic programming: define an $(n+1) \times (m+1)$ table S, which will hold, in each cell $S(i,k)$ the optimal SW obtained by some allocation of k products to the first i buyers.



$$S(i,k) = \max_{j=0}^{k} [v_i(j) + S(i-1, k-j)]$$

We initialize $S(i,0) = 0$ for $0 \le i \le n$ and $S(0,k) = 0$ for $0 \le k \le m$. Computing the value of each cell can be done in O(m) time, and thus the algorithm finds **OPT** in time O(nm^2).

8 Downward-sloping Valuations

This family of valuations is defined as follows.

Definition 3 *Valuation v is said to be* downward-sloping *if for every buyer i and number of products k, it holds that*

$$v_i(k+1) - v_i(k) \le v_i(k) - v_i(k-1)$$

Intuitively, this definition implies that the marginal benefit from each additional product gradually decreases as the number of products increases.

Definition 4 *A* market equilibrium *is a price per product p and an allocation $(m_1, \ldots, m_n)$ of products to buyers, such that $\sum_{i=1}^n m_i = m$ and for every i,*

$$v_i(m_i+1) - v_i(m_i) < p \le v_i(m_i) - v_i(m_i-1) \tag{1}$$

Since the utility function of a buyer i is $u_i(m_i) = v_i(m_i) - p \cdot m_i$, we get that for every i, $m_i = \arg\max_{k=1}^m v_i(k) - p \cdot k$. This is true since with every product more than m_i, the player gains at most $v_i(m_i+1) - v_i(m_i)$, which is smaller than the price he pays, p; and similarly, for every product less than m_i, the player could pay p and gain at least $v_i(m_i) - v_i(m_i-1)$.

Theorem 5 *Market equilibrium maximizes the social welfare.*

Proof: For every possible allocation of products to buyers $(k_1, \ldots, k_n)$ such that $\sum_{i=1}^n k_i \le m$ it holds that

$$\begin{aligned}\left(\sum_{i=1}^n v_i(m_i)\right) - m \cdot p &= \textstyle\sum_{i=1}^n (v_i(m_i) - m_i \cdot p) \\ &\ge \textstyle\sum_{i=1}^n (v_i(k_i) - k_i \cdot p) \\ &\ge (\textstyle\sum_{i=1}^n v_i(k_i)) - m \cdot p\end{aligned}$$

Thus, $\sum_{i=1}^n v_i(m_i) \ge \sum_{i=1}^n v_i(k_i)$ and indeed, $(m_1, \ldots, m_n)$ maximizes the SW. ■

8.1 An Algorithm for Downward-sloping Valuations

The following algorithm computes and outputs a price p which is the market equilibrium price.

1. Perform a binary search over $p \in [0, V]$, where $V = \max_{i=1}^{n} v_i(1)$, which is the maximal marginal gain from any product, by the fact that the valuation functions are downward-sloping.
   (a) For every buyer i, preform a binary search over $[0, \dots, m]$ in order to find m_i for which equation 1 holds.
   (b) In this manner, a vector $(m_1, \dots, m_n)$ is obtained.
   (c) If $\sum_{i=1}^{n} m_i > m$, then p is too low, increase it.
   (d) If $\sum_{i=1}^{n} m_i < m$, then p is too high, decrease it.
   (e) If $\sum_{i=1}^{n} m_i = m$, a market equilibrium was found, return $(m_1, \dots, m_n)$

By the time complexity of the algorithm above, we prove the following theorem.

Theorem 6 *Let n be the number of buyers, m a number of identical products and t the number of bits required to represent a buyer's value for a set of products. There exists an algorithm of time polynomial in n, $\log m$ and t for finding the optimal allocation for downward-sloping valuation functions.*

Remark. In general combinatorial auctions, the parallel of downward-sloping valuations is called "gross-substitutes". In this world, there exists a market equilibrium, i.e., prices $(p_1, \dots, p_m)$ for every product and an allocation $(S_1, \dots, S_n)$ of a set of products per buyer, s.t. for every buyer i, it holds that $S_i \in \arg\max_S \Big(v_i(S) - \sum_{j \in S} p_j \Big)$.

9 Negative Results

By attempting to extend the previous results to general valuation functions, we obtain the following negative results.

Theorem 7 *Every algorithm has to use at least $2m - 2$ value queries in order to find the optimal allocation, even for two agents.*

Proof: Assume by contradiction that there exists an algorithm A, that uses less than $2m-2$ value queries. Hence, for each valuation (v_1, v_2), A calculates an allocation (m_1, m_2) that maximizes the social welfare. Let the valuation be $v_1(k) = k$, $v_2(k) = k$, for each k. Hence, the answer to a query for $v_1(k)$ or $v_2(k)$ is k for each k. After a polynomial number of queries, A has to determine (m_1, m_2). Since A used less than $2m-2$ queries, it has to determine the allocation without knowing some of the values. Assume w.l.o.g that A didn't query $v_1(z)$ for some $z \notin m_1, m_2$. We define $v_1'(k)$ s.t. $v_1'(k) = k$ for $k \neq z$ and $v_1'(z) = z + 1$. The optimal allocation for (v_1', v_2) is $(z, m-z)$ with social welfare of $m+1$ (any other allocation has social welfare of m). Since A didn't query $v_1(z)$ it can't find the optimal allocation, a contradiction. ■

Let S be an arbitrary set s.t. $S \subseteq \{1, .., m-1\}$. Let $S^* = \{0 < k \leq m \mid m-k \notin S\}$. We define $v_S(k) = k+1$ for $k \in S$ and $v_S(k) = k$ for $k \notin S$. Let the valuation be (v_S, v_{S^*}).

Observation: For each k, $v_S(k) + v_{S^*}(m-k) = m+1$.
If $k \in S$ then $v_S(k) = k+1$ and $v_{S^*}(m-k) = m-k$.
If $k \notin S$ then $v_S(k) = k$ and $v_{S^*}(m-k) = m-k+1$.

Lemma 8 *Let $S, T \subseteq \{1, .., m-1\}$ be two sets s.t. $S \neq T$. Then the answers for the valuation (v_S, v_{S^*}) will be different from the answers to the valuation (v_T, v_{T^*}).*

Proof: Assume by contradiction that the answers for (v_S, v_{S^*}) and (v_T, v_{T^*}) are identical. Then we show that the answers for (v_T, v_{S^*}) and (v_S, v_{T^*}) will be identical as well, and this is a contradiction. Lets look at the valuation (v_S, v_{T^*}). Since the answers for (v_S, v_{S^*}) and (v_T, v_{T^*}) are identical, if we query agent 1 we receive the same answers for v_S and v_T. if we query agent 2 we receive the same answers for v_S^* and v_T^*. Hence, the answers for (v_S, v_{T^*}) are identical to the answers for (v_S, v_{S^*}) and (v_T, v_{T^*}). A symmetric argument can be applied to (v_T, v_{S^*}). Therefore we showed that the answers for (v_T, v_{S^*}) and (v_S, v_{T^*}) are identical. Now we prove that this is a contradiction. Let $k \in S \backslash T$. $v_S(k) + v_{T^*}(m-k) = m+2$. From the observation, for each k, $v_S(k) + v_{S^*}(m-k) + v_T(k) + v_{T^*}(m-k) = 2m+2$. Combining the equations we have: $v_{S^*}(m-k) + v_T(k) = m$. Therefore, k is optimal for (v_S, v_{T^*}) but not optimal for (v_T, v_{S^*}), contradicting that we receive the same answers for the two pair of valuations. ■

Theorem 9 *Every algorithm has to use at least $m-1$ black-box queries in order to find the optimal allocation.*

Proof: Assume by contradiction that there exists an algorithm A, that uses less than $m-1$ black-box queries. According to Lemma 8, for each $S, T \subseteq \{1, .., m-1\}$ s.t. $S \neq T$, the answers for the valuation (v_S, v_{S^*}) will be different from the answers to the valuation (v_T, v_{T^*}). Since there are 2^{m-1} such sets, there must be at least 2^{m-1} different answer sequences. To express 2^{m-1} different answers sequences there must be at least one sequence with size of $m-1$ bits. Since each answer is a most t bits, the number of answers must be at least $\frac{m-1}{t}$. This is a lower bound for the number of black-box queries. ■

10 Bidding Languages

Now we examine bidding languages. The simplest language is single-minded.
The input: (k_i, w_i) for each player.
The output: The set of winning players – which maximizes $\sum_i w_i$, with the restriction that $\sum_i k_i \leq m$. This is exactly the knapsack problem.
Conclusion: Even with single-minded language, the optimal allocation problem is NP-hard.