NETS 412: Algorithmic Game Theory March 28 and 30, 2017

Lecture 16+17

Lecturer: Aaron Roth *Scribe: Aaron Roth*

Approximation in Mechanism Design

In the last lecture, we asked how far we can go beyond the VCG mechanism when we want to optimize an objective other than social welfare. In this lecture, we will explore a different reason we might need to go beyond the VCG mechanism, even when we want to optimize social welfare: computational complexity. Recall, that in mechanism design problem, the VCG mechanism has to implement the following social-welfare maximizing allocation rule:

$$X(v) = \arg\max_{a\in A} \sum_i v_i(a)$$

What do we do when this problem is hard to solve – for example, when it is NP-complete? For many NP-complete problems we have good approximation algorithms, but this is not enough. Recall that for many of the properties of the VCG mechanism (including crucially truthfulness and individual rationality), we depended on the choice rule computing the *exactly* welfare maximizing outcome. If we only have the ability to find an alternative that achieves 99% of the optimal social welfare, we might lose all of the incentive properties of the VCG mechanism. As a case study, we will consider the *Knapsack auction* setting.

Definition 1 *In a* knapsack auction*:*

- *Each bidder $i \in \{1, \ldots, n\}$ has a publicly known* size $w_i \in \mathbb{R}_{\geq 0}$.
- *The mechanism has a publicly known* budget $B \in \mathbb{R}_{\geq 0}$.
- *The* feasible alternatives *are all subsets of bidders of total size $\leq B$:*

$$A = \{S \subseteq \{1, \ldots, n\} : \sum_{i\in S} w_i \leq B\}$$

- *For each $a \in A$ we write $a_i = 1$ if $i \in a$.*
- *These are single parameter domains. Each bidder i has a real value $v_i \in \mathbb{R}_{\geq 0}$, and their value for alternative a is $v_i \cdot a_i$*

This is called a knapsack auction because finding the optimal social welfare alternative, i.e.

$$\arg\max_{S\in A} \sum_{i\in S} v_i$$

is NP-hard, which means we do not believe that there is any polynomial time algorithm for this task. But note that this is a very natural problem that we would like to be able to solve – it models, for example, the problem of auctioning off seats on an airplane to people who have different sized parties, and many other discrete allocation problems in which there is some finite resource constraint. If we need to have an auction that runs in a reasonable amount of time, what can we do?

We could attempt to use a choice rule which *approximates* the social welfare objective, and to find a pricing rule (necessarily different from the VCG pricing rule) which makes this choice rule dominant strategy truthful. We know from last lecture that if this is going to succeed, we have to not only come up with an approximation algorithm for the Knapsack problem, we need to come up with one that is a monotone non-decreasing choice rule. If we can do this, we are done – we simply use the pricing rule from the last lecture to make it truthful. If we can't, we are out of luck.

First, we define what it means to approximate the optimal social welfare:

Definition 2 *For a set of values and weights $v, w \in \mathbb{R}^n_{\geq 0}$, let:*

$$OPT(v,w) = \max_{S \subseteq [n] : \sum_{i \in S} w_i \leq B} \sum_{i \in S} v_i$$

denote the optimal solution to the knapsack instance defined by v and w. We say that an algorithm A is an α-approximation algorithm for the Knapsack problem if for every $v, w \in \mathbb{R}^n_{\geq 0}$, $A(v,w) = S$ such that:

1. *S is a feasible solution: $\sum_{i \in S} w_i \leq B$*
2. *S approximates OPT to within a factor of α: $\sum_{i \in S} v_i \geq \frac{OPT(v,w)}{\alpha}$*

Recall what it means in this setting for A to be a monotone non-decreasing allocation rule: for every $v, w \in \mathbb{R}^n_{\geq 0}$, and for every i and $v'_i > v_i$, if $S = A(v,w)$ and $S' = A((v'_i, v_{-i}), w)$, then:

$$i \in S \Rightarrow i \in S'.$$

Our goal is to come up with a monotone algorithm A that is also an α-approximation algorithm for the Knapsack problem, for a reasonable value of α. Lets think about how we might do that. The first observation we make is that we can write the knapsack problem in the following integer linear optimization form. We want to solve:

$$\text{maximize} \sum_{i=1}^{n} x_i \cdot v_i$$
$$\text{such that:}$$
$$\sum_{i=1}^{n} x_i \cdot w_i \leq B$$
$$x_i \in \{0,1\} \quad \forall i$$

where here the x_i variables indicate whether agent i has been selected or not. This captures the problem we want to solve exactly; however, because solving this problem is NP-hard, it means we won't be able to reason very fruitfully about any structure that optimal solutions might have. Consider instead the following "relaxed" version of the problem, in which we let the x_i variables be fractional:

$$\text{maximize} \sum_{i=1}^{n} x_i \cdot v_i$$
$$\text{such that:}$$
$$\sum_{i=1}^{n} x_i \cdot w_i \leq B$$
$$x_i \in [0,1] \quad \forall i$$

Write $\text{OPT}_F(v,w)$ to denote the optimal value of this "fractional" version of the problem. This is simply a linear programming problem which can be solved efficiently – for our purposes, however, the benefit of this formulation is that we have some hope of understanding the structure of the solutions! We start with a simple observation:

Claim 3 *For all $v, w \in \mathbb{R}^n_{\geq 0}$:*

$$OPT_F(v,w) \geq OPT(v,w)$$

Proof This holds simply because the fractional version of the problem is simply less constrained: any optimal solution to the integer version of the problem is a *feasible* solution to the fractional version, so $\mathrm{OPT}_F(v,w)$ must always be at least as large as $\mathrm{OPT}(v,w)$ (since we can use the same optimal solution), but might be larger. ■

What this means is if we can come up with an algorithm that obtains an α-approximation to $\mathrm{OPT}_F(v,w)$ then we also obtain (at least!) an α-approximation to $\mathrm{OPT}(v,w)$, which is what we really want. It also leaves us free to consider the easier-to-understand solutions of the fractional relaxation. This relaxation has some simple structure:

Lemma 4 *Let x be a fractional solution obtaining value $OPT_F(v,w)$ in the fractional knapsack problem. Let $i,j \in [n]$ be any pair of agents such that:*

$$\frac{v_i}{w_i} > \frac{v_j}{w_j}.$$

Then $x_j > 0 \to x_i = 1$

Proof Suppose otherwise, and that for such an i,j pair, $x_j > 0$ but $x_i < 1$. Define $\delta > 0$ by $\delta = \min((1-x_i)\frac{w_i}{w_j}, x_j)$. We now define a new solution x' and argue that it is feasible, and has higher objective value, which will contradict the optimality of x. Let $x'_\ell = x_\ell$ for all $\ell \not\in \{i,j\}$, and let

$$x'_j = x_j - \delta$$

and

$$x'_i = x_i + \delta\frac{w_j}{w_i}.$$

Note that x' continues to satisfy the knapsack constraint: the change in size of the bundle was:

$$(\delta\frac{w_j}{w_i})\cdot w_i - \delta w_j = \delta w_j - \delta w_j = 0$$

Moreover, by definition of δ, we know $x'_j \geq x_j - x_j = 0$ and $x'_i \leq x_i + ((1-x_i)\frac{w_i}{w_j})\frac{w_j}{w_i} = 1$. Hence (because x was feasible) x' is feasible. The change in value of the bundle is:

$$(\delta\frac{w_j}{w_i})\cdot v_i - \delta\cdot v_j > 0$$

This follows because by assumption:

$$\frac{v_i}{w_i} > \frac{v_j}{w_j} \Rightarrow (\frac{w_j}{w_i})\cdot v_i > v_j$$

(by multiplying through by w_j). But then x' has higher objective value than x while maintaining feasibility, which contradicts the optimality of x. ■

Ok! Once we understand this structure, we can give a simple combinatorial algorithm for solving the fractional version of the knapsack problem, which will help us understand its behavior (recall we need to show that our algorithm in the end is monotone). Given our lemma, we know the algorithm given here (FractionalKnapsack) must be optimal.

Now how can we use this algorithm to get a solution to the integer knapsack problem? (We know it optimally solves the fractional version). Note that until the last step, the algorithm is constructing an integer solution. So suppose we just leave out the last step, which fills the remaining space in the knapsack with the last element fractionally? We get an integer solution – how well does it do?

Terribly! Consider the following example.

```
**FractionalKnapsack**$(v, w)$:
  Sort bidders in decreasing order by $\frac{v_i}{w_i}$ and set size $\leftarrow 0$ and $i \leftarrow 1$.
  **while** size$+w_i \leq B$ **do**
    Set $x_i \leftarrow 1$, size $\leftarrow$ size $+ w_i$, $i \leftarrow i + 1$.
  **end while**
  Set $x_i \leftarrow \frac{B-\text{size}}{w_i}$ and Set $x_j = 0$ for all $j > i$.
  Return $x$.
```

Example 1 *We have two agents with $w_1 = v_1 = 10$ and $w_2 = 1$ and $v_2 = 1.1$. Our knapsack has size $B = 10$. Note that $OPT(v, w) = 10$ – the optimal solution picks agent 1. However, $v_2/w_2 > v_1/w_1$, so the algorithm first picks agent 2, and then has no remaining space for agent 1 – so the algorithm picks a solution with value only 1.1 – about 10 times worse. We could extend this example to make the algorithm's solution arbitrarily worse!*

Ok, we've identified a problem – the fractional algorithm may be optimal, but leaving off the fractional portion of the solution may leave almost the entire knapsack empty, wasting tons of value! Lets see if we can correct that with a second attempt (Greedy2). Note that without loss of generality, we can assume that for all agents i, $w_i \leq B$. This is because we can discard any agent with $w_i > B$ without harming OPT, because such an agent cannot have appeared in the optimal solution.

```
**Greedy2**$(v, w)$:
  Sort bidders in decreasing order by $\frac{v_i}{w_i}$ and set size $\leftarrow 0$ and $i \leftarrow 1$. Set $S \leftarrow \emptyset$.
  **while** size$+w_i \leq B$ **do**
    Set $S \leftarrow S \cup \{i\}$, size $\leftarrow$ size $+ w_i$, $i \leftarrow i + 1$.
  **end while**
  **if** $\sum_{j \in S} v_j \geq v_{j+1}$ **then**
    Output $S$.
  **else**
    Output $\{i^*\}$ where $i^* = \arg\max_i v_i$.
  **end if**
```

We argue that Greedy2 is a 2-approximation to the Knapsack problem:

Theorem 5 *Greedy2 achieves a 2-approximation algorithm for the Knapsack problem.*

Proof We derived Greedy2 from the *optimal* algorithm for the fractional knapsack problem in such a way that for every agent j:

$$j \notin S \cup \{i+1\} \Rightarrow x^*_j = 0$$

where x^* is the optimal fractional solution to the corresponding fractional knapsack instance (v, w). Hence we can conclude:

$$\sum_{i \in S} v_i + v_{i+1} \geq \text{OPT}_F(v, w) \geq \text{OPT}(v, w).$$

Therefore, we know that:

$$\max(\sum_{i \in S} v_i, v_{i+1}) \geq \frac{OPT(v, w)}{2}$$

and because $v_{i^*} \geq v_{i+1}$ by definition, the solution our algorithm outputs is:

$$\max(\sum_{i \in S} v_i, v_{i^*}) \geq \frac{OPT(v, w)}{2}$$

■

It remains to show that Greedy2 is monotone – if we can do this, then we know that we can efficiently find a 2-approximation to the social welfare objective in the Knapsack problem while guaranteeing dominant strategy truthfulness.

Theorem 6 *Greedy2 is monotone non-decreasing for every agent i.*

Proof Fix any $w, v \in \mathbb{R}^n_{\geq 0}$, any agent i, and let $v'_i > v_i$. Write $v' = (v'_i, v_{-i})$. Let $T = Greedy2(v, w)$ and $T' = Greedy2(v', w)$. We need to show that $i \in T \Rightarrow i \in T'$. Write $S \doteq S(v, w)$ and $S' \doteq S(v', w)$ to denote the intermediate sets S generated by Greedy2 on each instance. First we argue:

Claim 7

$$i \in S \Rightarrow i \in S'$$

Proof Note that S and S' represent the prefix of the bidders of total size $\leq B$ when sorted in decreasing order of $\frac{v_j}{w_j}$. When agent i increases his value from v_i to v'_i he can only move himself earlier in this sorted ordering, and so if he was in the prefix represented by S he is still in the prefix represented by S'. ■

Hence we know that if $T = S$ and $T' = S'$, then on this instance, the algorithm is monotone. Note also that if $i \in S$, then $\sum_{j \in S'} v'_j \geq \sum_{j \in S} v_j$. (If $i \in S$, as we argued, $i \in S'$ and so the two sets have identical membership. S' has higher value because v'_i is increased and all other v_j remain the same). Hence, if $i \in S$, then if $T = S$, $T' = S$ and the algorithm is monotone in this instance.

The only remaining case with $i \in T$ is if $i = i^*$ and $v_i > \sum_{j \in S} v_j$. But note that in this case, we must also have $i \in T'$. i remains the highest value bidder, and so is either output as $T' = \{i^*\}$ in the last step of the algorithm, or is output as $T' = S'$ with $i \in S'$ (since in order to change the value of S', i must have raised his bid high enough to be included in it). ■

Together, we have shown that there exists a polynomial time 2-approximation for the Knapsack problem that makes truthful bidding a dominant strategy for all players (despite the fact that we cannot use the VCG mechanism to give a polynomial time solution)