17.1 Introduction

Today we're going to see (for the first time) a setting where computational efficiency is a problem, and talk about some ways of dealing with it.

17.2 Knapsack Auctions

17.2.1 Basic Setup

Suppose that we're selling ad time during the Super Bowl. Different ads have different lengths, so each bidder actually wants a slightly different thing. How can we sell off all of our ad time?

We can formalize this as follows. There is some total amount of time W. Bidder i want to place an ad of length w_i, which is public information. If bidder i gets to place their ad, they get value v_i (which is private information). A bid in this setting is, like before, a single number b_i which is "supposed" to be v_i. As before, we'll assume quasilinear utilities, so the utility of player i is $v_i - p$ if it gets to place its ad for price p. Clearly the total amount of ad time we sell has to be at most W.

Clearly this can be generalized beyond ad time. We have W stuff to sell, player i wants w_i stuff, and gets value v_i if it does get the stuff that it wants. They key point here is that w_i is public but v_i is private, that values are "all-or-nothing", and that we can only sell W total stuff. These are called "knapsack auctions", for reasons that we'll see soon.

Note that this is a single-parameter environment: $X = \{x \in \{0,1\}^n : \sum_{i=1}^n w_i x_i \leq W\}$. The social surplus is $\sum_{i=1}^n v_i x_i$. This generalizes the setting of k identical goods ($w_i = 1$ for all $i \in [n]$ and $W = k$).

17.2.2 Classical Results

Can we design an "awesome auction" for these auctions? Let's go back to our two-stage design process: first figure out the allocation function assuming that bids are truthful, and then figure out the price function to make it incentive compatible. Since it's a single-parameter environment, Myerson's Lemma applies: we can find prices to make it incentive compatible if and only if the allocation function is monotone.

Since we want to maximize social surplus, let's start with the allocation rule that does that:

$$\mathbf{x}(b) = \arg\max_{x \in X} \sum_{i=1}^{n} b_i x_i.$$

This clearly maximizes the surplus, which was one property that we wanted from awesome auctions. What about being incentive compatible?

Theorem 17.2.1. $\mathbf{x}(b)$ *is monotone.*

Proof. Fix some $i \in [n]$ and other bids b_{-i}. Let $y > z > 0$. Let $x = \mathbf{x}(b_{-i}, z)$ and let $x' = \mathbf{x}(b_{-i}, y)$. So we need to show that $x_i \leq x'_i$.

Since x is the surplus-maximizing allocation for bids (b_{-i}, z), we know that

$$\sum_{j \neq i} x'_j b_j + x'_i z \leq \sum_{j \neq i} x_j b_j + x_i z.$$

On the other hand, since x' is the surplus-maximizing allocation for bids (b_{-i}, y), we know that

$$\sum_{j \neq i} x_j b_j + x_i y \leq \sum_{j \neq i} x'_j b_j + x'_i y.$$

Adding these two together, we get that

$$\sum_{j \neq i} x'_j b_j + \sum_{j \neq i} x_j b_j + x'_i z + x_i y \leq \sum_{j \neq i} x_j b_j + \sum_{j \neq i} x'_j b_j + x_i z + x'_i y.$$

Simplifying this gives

$$x'_i z + x_i y \leq x_i z + x'_i y.$$

Now simplifying further give

$$x_i(y - z) \leq x'_i(y - z).$$

Since $y - z > 0$, this implies that $x_i \leq x'_i$. Hence $\mathbf{x}$ is monotone. □

So by Myerson's Lemma and this theorem, we know that there are prices which can make the surplus-maximizing allocation function incentive compatible. These are actually pretty simple: by monotonicity and the fact that $x_i(b) \in \{0, 1\}$, there is some *critical bid* $z(b_{-i})$ which is the smallest value i could bid and still have $x_i(b) = 1$. Since the jump in the allocation is 1, by Myerson this means that the price player i should pay is $p_i(b) = z(b_{-i})$ if $x_i(b) = 1$ (and $p_i(b) = 0$ otherwise).

Note that this proof didn't use anything about knapsack auctions. We just proved that the surplus-maximizing allocation is monotone in every single-parameter environment, so by Myerson there are prices to make the surplus-maximizing allocation incentive compatible in every single-parameter environment! This was a triumph of classical mechanism design.

17.2.3 Computational Efficiency

But what about the computational efficiency requirement of awesome auctions? As we saw, computing prices is pretty easy if we can compute the allocation. But can we actually compute the surplus-maximizing allocation?

Recall the *Knapsack Problem*: given a knapsack of capacity C and n items, where item i has value a_i and size s_i, find the maximum value subset $S \subseteq [n]$ which fits in the knapsack, i.e., maximize $\sum_{i \in S} a_i$ subject to $\sum_{i \in S} s_i \leq C$. This problem is one of the classical NP-hard problems (the decision version is NP-complete), and it is easy to see that it reduces to computing the surplus-maximizing allocation in a knapsack auction: given an instance of Knapsack, we create a knapsack auction with $W = C$ and with n bidders, where $w_i = s_i$ and where $v_i = a_i$. Then the surplus-maximizing

allocation is exactly the optimal solution to the Knapsack instance. So we don't think that we can actually compute the surplus-maximizing allocation in polynomial time, since we believe that P $\neq$ NP.

How do we deal with this? One approach is to ignore computational efficiency, of course. But as computer scientists, we are not willing to give up on efficiency. So we're going to take an approach which is natural from a TCS point of view: we're going to relax the requirement that we maximize the surplus. In other words, we're going to try to design *approximation algorithms* for surplus maximization rather than computing the optimal solution. Of course, since we're doing mechanism design rather than algorithm design we care about more than just approximation and running time: we also care about incentive compatibility. So, by Myerson, what we're looking for is not just any approximation algorithm for Knapsack, but rather a *monotone* approximation algorithm.

Digression: approximation algorithms. As a small digression, we can think of this for any NP-hard problem, giving us an overarching question: for every NP-hard problem that we care about, is there a *monotone* approximation algorithm with approximation ratio the same as the *best* approximation algorithm? In other words, at least for single-parameter environments (by Myerson), can we get incentive compatibility for free? Or does incentive compatibility require us to get worse approximations than we could otherwise get? This relationship and connection to approximation algorithms is one of the main reasons (at least according to Roughgarden) that there has been such an explosion in results in algorithmic mechanism design.

It's known that *black-box* reductions can't achieve this: we can't start with an arbitrary approximation algorithm, and turn it into an incentive compatible mechanism with the same approximation ratio in a black box way [CIL12]. But perhaps for every problem we care about, we can find equivalent incentive compatible mechanisms.

Back to knapsack auctions. As some of you may know, it turns out that Knapsack is a very easy problem to approximate.

Theorem 17.2.2. *There is an FPTAS (fully polynomial time approximation scheme) for Knapsack: an algorithm with approximation ratio of* $1-\epsilon$ *and running time polynomial in* n *and* $1/\epsilon$ *for all* $\epsilon > 0$.

Unfortunately, it turns out that this approximation algorithm is not monotone! It can be tweaked to be monotone, but this requires a bit more background in approximation algorithms than we have in this course. So today we'll do a simpler 1/2-approximation which *is* monotone.

17.2.4 IC 2-approximation for Knapsack

Let's assume that $w_i \leq W$ for all $i \in [n]$, since otherwise player i can't get any utility anyway (there's not enough stuff for them to get any value). So such a player wouldn't be participating in the auction, or equivalently we can just remove them from the auction (since w_i is public).

Our mechanism works as follows. Given bids, in order to simplify notation we first sort and re-index so that

$$\frac{b_1}{w_1} \geq \frac{b_2}{w_2} \geq \cdots \geq \frac{b_n}{w_n}$$

Now we pick the bids in order until the next one doesn't fit. More formally, let i^* be the maximum i such that $\sum_{j=1}^{i} w_j \leq W$. We return either this solution ($x_i(b) = 1$ iff $i \leq i^*$) or the single highest bidder ($x_i(b) = 1$ iff $i = \arg\max_{i \in [n]} b_i$), whichever creates more surplus: if $\sum_{i=1}^{i^*} b_i \geq \max_{i \in [n]} b_i$ then return the solution $[i^*]$, and otherwise return $\arg\max_{i \in [n]} b_i$. Clearly this runs in polynomial time.

It's worth noting that we really need the final step of taking the best of these two. For example, if $b_1 = 2$ and $w_1 = 1$, and $b_2 = W$ and $w_2 = W$, then the simple sorting algorithm would choose the first bidder, but this is much worse than choosing the second bidder.

Theorem 17.2.3. *This allocation rule is a* $1/2$*-approximation.*

Proof Sketch. If $\sum_{i=1}^{i^*} b_i \geq \frac{1}{2} OPT$, then we're done: the solution we return has at least that value. So suppose that $\sum_{i=1}^{i^*} b_i < \frac{1}{2} OPT$. Clearly $OPT \leq \sum_{i=1}^{i^*+1} b_i$, since the $i^* + 1$ bid makes it no longer feasible but if we *did* have W exactly equal to $\sum_{i=1}^{i^*+1} w_i$ then the optimal solution would be $[i^* + 1]$. Thus we have

$$OPT \leq \sum_{i=1}^{i^*} b_i + b_{i^*+1} < \frac{1}{2} OPT + b_{i^*+1},$$

and hence $b_{i^*+1} \geq \frac{1}{2} OPT$. Thus $\max_{i=1}^{n} b_i \geq \frac{1}{2} OPT$. □

Theorem 17.2.4. *This allocation rule is monotone.*

Proof. Fix player $i \in [n]$, other bids b_{-i}, and $y > z \geq 0$. If $x_i(b_{-i}, z) = 0$, then clearly $x_i(b_{-i}, y) \geq x_i(b_{-i}, z)$. So suppose that $x_i(b_{-i}, z) = 1$. Then either z is the highest bid, or z is one of the i^* best bids in the ordering. In the first case, y is still the highest bid, so $x_i(b_{-i}, y) = x_i(b_{-i}, z) = 1$. In the second case, after bidding y player i will still be one of the first i^* in the ordering, so $x_i(b_{-i}, y) = x_i(b_{-i}, z) = 1$. Thus $x_i(b_{-i}, y) \geq x_i(b_{-i}, z)$ in every case, and so $\mathbf{x}$ is monotone. □

So, by Myerson, for knapsack auctions there is a mechanism $(\mathbf{x}, \mathbf{p})$ which is incentive-compatible, achieves at least half of the maximum possible social surplus, and which runs in polynomial time.

17.3 Revelation Principle

This is a bit unrelated, but it's an important concept and so now is as good a time as any to talk about it. If you remember our definition of incentive-compatibility, the main condition was that telling the truth is a dominant strategy. If you think about it, this is actually two separate things:

- Every bidder has a dominant strategy
- The dominant strategy is *direct revelation*: telling the private information to the mechanism.

As we've discussed, it's really nice to have every player have a dominant strategy. We can relax this a bit (as we'll talk about later in the class), but dominant strategies are a great feature to have since it lets us more easily reason about what's going to happen. But a pretty natural question is whether the *second* part of this is crucial. In other words: if we require there to be a dominant

strategy but don't require this to be direct revelation, does that give us any important added flexibility as the mechanism designer?

It turns out that the answer is "no", and this is known as the *revelation principle*.

Theorem 17.3.1. *For every mechanism $M = (\mathbf{x}, \mathbf{p})$ in which every bidder has a dominant strategy, there is an equivalent direct revelation mechanism $M' = (\mathbf{x}', \mathbf{p}')$.*

Proof Sketch. For each player $i \in [n]$, let $s_i(v_i)$ be the dominant strategy in M for player i with private information v_i (so direct revelation would require that $s_i(v_i) = v_i$).

Our new mechanism M' is simple: given bid vector $b = (b_1, b_2, \ldots, b_n)$, we send bids $(s_1(b_1), s_2(b_2), \ldots, s_n(b_n))$ to the mechanism M, and then use whatever allocation and pricing M decides. More formally, for a bid vector b let $s(b) = (s_1(b_1), s_2(b_2), \ldots, s_n(x_n))$. Then we set $\mathbf{x}'(b) = \mathbf{x}(s(b))$ and similarly set $\mathbf{p}'(b) = \mathbf{p}(s(b))$.

We claim that this new mechanism is direct-revelation incentive-compatible. This should be intuitively obvious: bidding b_i in M' is the exact same as bidding $s_i(b_i)$ in M. Given private information v_i, by definition $s_i(v_i)$ is a dominant strategy in M, and thus bidding v_i is a dominant strategy in M'. Thus M' has exactly the same behavior as M, just with direct revelation. □

References

[CIL12] Shuchi Chawla, Nicole Immorlica, and Brendan Lucier. On the limits of black-box reductions in mechanism design. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 435?448, New York, NY, USA, 2012. Association for Computing Machinery.