1 Combinatorial Auctions

We’re finished with single-minded CAs.

Obviously specifying the $V_i$’s explicitly is too big. We want to be able to give that information to the "Deciding Unit”, in reasonable size. There are languages that can do that.

Now, obviously no matter what the language is, the real input is really large. But for reasonable valuations, there are languages that can specify them in reasonable size. For most purposes, one only wants a small number of groups of items, and this can be specified without exponential input. We’re not going to talk about this.

1.1 Find an alternative

We can design an iterative kind of auctions, where in the end of the iterations, we get to a good result. We call this a protocol.

The idea is that the mechanism is not of direct revelation, but of going ahead a bit at the time.

For starters, we’re not going to talk about "honesty” of the mechanism, but just look for something that works.

1.2 The economic aspect

Let’s present a reasonable option:

**Definition 1** The Demand for player $i$ with prices $p_1, ..., p_m$ is the group $S$ which maximizes $V_i(S) - \sum_{j \in S} p_j$.

For the Valresian equilibrium (defined previously), we may allow some items not to be distributed, if their price is 0.
We will now show an algorithm which finds the Valresian equilibrium given our set of definitions:

\[ \forall i : S[i] = \phi \]
\[ \forall j : P[j] = 0 \]

Repeat:

// In words: find players who want to buy an additional item for a price slightly higher than its current going price.

Find \(i\) s.t. \(S[i]\) is not his demand at prices:

\[ \forall j \in \text{demand}(i) - S[i] : p_j = p_j + \delta \]

\[ S[i] = \text{demand}(i) \]

\[ \forall k \neq i : S[k] = S[k] - S[i] \]

If no such \(i\) exists - end.

The algorithm halts after \(\frac{V_{\text{max}} \times m}{\delta}\) steps at most. \(V_{\text{max}} = \max_{i,S}(V_i(S))\) (because at each step, some product’s price changes by \(\delta\). Each product needs at most \(\frac{V_{\text{max}}}{\delta}\) raises to get to the highest price that anybody’s willing to get to it (which is \(\leq V_{\text{max}}\)). Doing that \(m\) times means that all products can get to the highest price anybody’s willing to pay for them.

**Definition 2** The Group \(S_1, \ldots, S_n\) is called a “\(\delta\)-Valresian Equilibrium” if \(\forall i\) it holds that \(S_i\) is: demand\(_i\)(\(p_j\) for \(j \in S_i\), \(p_j + \delta\) for \(j \notin S_i\))

**Theorem 3** When the algorithm halts, there’s almost a Valresian equilibrium (a \(\delta\)-Valresian equilibrium).

**Proof:** This is concluded instantly from the way we constructed our algorithm. The algorithm won’t stop until we get a \(\delta\)-Valresian equilibrium as defined above.

1.3 Problems with the Algorithm

We now notice that there’s a problem with this proof. We showed in the previous lesson that there’s not always a Valresian Equilibrium to be found. We can also conclude that if this algorithm always finds the Valresian Equilibrium, we can solve the NP-hard linear problem.

So what is the problem here? Let’s show an example which might clarify the problem:

We have two products \(a, b\).

\(V(a) = 0, V(b) = 0, V(ab) = 10.\)
For the given prices: \( p(a)=3, p(b)=3 \), our preferred division will be \( a, b \).

The same if the price of \( a \) is 4, 5, 6 or 7. However, if the price of \( a \) is 8, we will not want anything, since the price is bigger than the value (11; 10).

What actually happened here is that as soon as the price of \( a \) went up, we didn’t care for \( b \) anymore.

So, let’s amend our theorem: we say that the theorem holds if all \( V_i \)'s are “Substitutes”.

**Definition 4** \( V \) is called “Substitutes” if for every price vectors \( p \geq q \) (meaning \( \forall j : p_j \geq q_j \)) it holds that: \( \text{Demand}(p) \subseteq \{ \text{Demand}(q) \cup \{ j | p_j = q_j \} \} \) (we assume \( \text{Demand}(p) \) to give a single group of items, not a group of groups).

In words: If some prices go up, this will not cause the demand for other items (whose prices didn’t change) to drop.

1.4 When will the algorithm be valid?

- Linear functions (because my internal price for an item does not depend on other items).
- Unit demand, the player is only interested in one product, if he gets a few he wishes only for the best (lunch, workplace)
- Downward Sloping Valuation, I only care about the number of items, not which items they are, and each additional item is worth less than the previous one.

Now assume we don’t have the substitutes condition. Can we improve on the algorithm?

The answer is yes, there are such protocols, but they’re exponential. (They don’t find Valresian equilibrium, but some other equilibrium). We know that the problem is NP-complete, but we care about the number of iterations, and not about the calculation of demand (that each player does by himself). So the answer is no: It’s not possible to find the optimal allocation this without transferring an exponential amount of information.

The problem is this: Find the maximal value \( t \) such that there exist \( S, S^c \) such that \( t = V_a(S) + V_b(S^c) \).

**Lemma 5** This requires communication in size \( \geq 2^m/\sqrt{m} \). (Even if \( V_i : S \to \{0, 1\} \)).

Suppose Alice and Bob communicate between themselves, and eventually find the partition; or even just the value of the optimal allocation. The protocol is a function from the bidding history and the private information to the next bid, and says when the bidding stops and what the final answer is. Again, we’re concerned with the number of iterations, and we don’t care how hard it is for each side to calculate its next step (i.e. calculate the result of their end of the protocol).
The number of Boolean valuation functions is $2^m$. The number of Boolean monotonic valuation functions is less than this. At least $2^{2m}/\sqrt{m}$ (which is roughly the number of such functions which give 0 to $|S| < m/2$, 1 to $|S| > m/2$ and anything for $|S| = m/2$).

**Definition 6** Fouling set is defined as $V^*(S) = 1 - V(S^c)$
This means that for $\forall S : V(S) + V^*(S) \leq 2$, i.e. no both of them are happy. It is worth nothing that these are legal monotonic valuations.

**Lemma 7** Let $u \neq v$ be Boolean monotonic valuations. Then every protocol that always finds the optimal allocation must utilize different communications on the input $(V, V^*)$ and $(U, U^*)$.

Conclusion from the lemma: The number of possible communications is at least the number of possible valuations, i.e. $\geq 2^{2m}/\sqrt{m}$, and this means that for some valuations, the length is at least $2^m/\sqrt{m}$.

If the same communications take place for $(V, V^*)$ and $(U, U^*)$, then the same communications will also take place for $(V, U^*)$ and $(U, V^*)$.

Now, for $(V, V^*)$, the optimal output is 1 (give one of them what they want). However, for at least one of $(V, U^*), (U, V^*)$ the optimal output is at least 2.

**Proof:** Let’s consider the input for $(V, U^*)$. Alice cannot see the difference between the inputs $(V, V^*)$ and $(V, U^*)$, so she will say the same things. Likewise Bob from $(U, U^*)$ to $(V, U^*)$. So on each step, each of them will do the same as he or she have done in $(V, V^*)/(U, U^*)$. Therefore, a correct protocol cannot give the same communications for two different pairs of valuation functions. ■