Gross Substitutes Tutorial

Part I: Combinatorial structure and algorithms
(Renato Paes Leme, Google)

Part II: Economics and the boundaries of substitutability
(Inbal Talgam-Cohen, Hebrew University)
Three seemingly-independent problems
Three seemingly-independent problems

[Kelso-Crawford ’82]
necessary /“sufficient” condition for price adjustment to converge
gross substitutes
Three seemingly-independent problems

[Kelso-Crawford '82] necessary /“sufficient” condition for price adjustment to converge gross substitutes

[Dress-Wenzel '91] generalize Grassmann-Plucker relations valuated matroids matroidal maps
Three seemingly-independent problems

[Kelso-Crawford '82]
necessary /“sufficient”
condition for price
adjustment to converge
gross substitutes

[Dress-Wenzel '91]
generealize
Grassmann-Plucker
relations
valuated matroids
matroidal maps

[Murota-Shioura '99]
generealize convexity
to discrete domains
M-discrete concave
Three seemingly-independent problems

[Kelso-Crawford '82] necessary /“sufficient” condition for price adjustment to converge
gross substitutes

[Discrete Convex Analysis]

[Dress-Wenzel '91] generalize Grassmann-Plucker relations
valuated matroids
matroidal maps

[Murota-Shioura '99] generalize convexity to discrete domains
M-discrete concave
Some notation to start

• Discrete sets of goods: \([n] = \{1, \ldots, n\}\)

• Valuation function \(v : 2^n \to \mathbb{R}\)

• Given prices \(p \in \mathbb{R}^n\) define \(v_p(S) = v(S) - p(S)\)

• Demand correspondence \(D(v; p) = \arg\max_S v_p(S)\)

• Demand oracle \(O_D(v, p) \in D(v; p)\)

• Value oracle \(O_V(v, S) = v(S)\)

• Marginals \(v(S|T) = v(S \cup T) - v(T)\)
Walrasian equilibrium

$n$ goods

$m$ buyers
Walrasian equilibrium

- Valuations $v_i : 2^N \rightarrow \mathbb{R}$

$m$ buyers

$n$ goods
Walrasian equilibrium

$n$ goods

$p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6$

$m$ buyers

$v_1 \quad v_2 \quad v_3 \quad v_4$

• Valuations $v_i : 2^N \rightarrow \mathbb{R}$
Walrasian equilibrium

Valuations

\[ v_i : 2^N \rightarrow \mathbb{R} \]

Demands

\[ D(v_i, p) = \arg\max_{S \subseteq N} [v_i(S) - \sum_{i \in S} p_i] \]
Walrasian equilibrium

- Valuations \( v_i : 2^N \to \mathbb{R} \)
- Demands \( D(v_i, p) = \arg\max_{S \subseteq N} [v_i(S) - \sum_{i \in S} p_i] \)
Walrasian equilibrium

Valuations $v_i : 2^N \rightarrow \mathbb{R}$

Demands $D(v_i, p) = \operatorname{argmax}_{S \subseteq N} [v_i(S) - \sum_{i \in S} p_i]$
Walrasian equilibrium

- **Valuations** $v_i : 2^N \rightarrow \mathbb{R}$
- **Demands** $D(v_i, p) = \arg\max_{S \subseteq N} [v_i(S) - \sum_{i \in S} p_i]$
Valuations

\[ v_i : 2^N \rightarrow \mathbb{R} \]

Demands

\[ D(v_i, p) = \arg\max_{S \subseteq N} [v_i(S) - \sum_{i \in S} p_i] \]
Walrasian equilibrium

- Market equilibrium: prices \( p \in \mathbb{R}^n \) s.t. \( S_i \in D(v_i, p) \)
i.e. each good is demanded by exactly one buyer.

**First Welfare Theorem:** in equilibrium the welfare
\[ \sum_i v_i(S_i) \] is maximized.
(proof: LP duality)

When do equilibria exist?
How do markets converge to equilibrium prices?
How to compute a Walrasian equilibrium?
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \not\in D(v_i, p^i)$ assign $X_i \in D(v_i; p_i^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$. 
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \notin D(v_i, p^i)$ assign $X_i \in D(v_i; p^i_i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$.
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \notin D(v_i, p^i)$ assign $X_i \in D(v_i; p^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$.
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \notin D(v_i, p^i)$ assign $X_i \in D(v_i; p^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$.
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \not\subset D(v_i, p^i)$ assign $X_i \in D(v_i; p^i_i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$.
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$.
- While there is $S_i \notin D(v_i, p^i)$ assign $X_i \in D(v_i; p^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$. 

$n$ goods

$m$ buyers
Walrasian tatonnement

- Initialize $S_1 = [n], S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \notin D(v_i, p^i)$ assign $X_i \in D(v_i; p^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\varepsilon$. 
• Initialize \( S_1 = [n], \) \( S_i = \emptyset \) and prices \( p_j = 0 \)
• While there is \( S_i \notin D(v_i, p^i) \) assign \( X_i \in D(v_i; p^i) \) to \( i \) and increase the prices in \( X_i \setminus S_i \) by \( \epsilon \).
Walrasian tatonnement

- Initialize $S_1 = [n], S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \notin D(v_i, p^i) \quad \text{assign} \quad X_i \in D(v_i; p^i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$. 

$n$ goods

$m$ buyers

$v_1 \quad v_2 \quad v_3 \quad v_4$
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \not\in D(v_i, p_i)$ assign $X_i \in D(v_i; p_i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$. 

$n$ goods

$m$ buyers

\[ S_1 = [n], \quad S_i = \emptyset \quad \text{and prices} \quad p_j = 0 \]

\[ p^i_j = p_j \quad \text{if} \quad j \in S_i \]

\[ p^i_j = p_j + \epsilon \quad \text{if} \quad j \not\in S_i \]
Walrasian tatonnement

- Initialize $S_1 = [n]$, $S_i = \emptyset$ and prices $p_j = 0$
- While there is $S_i \not\subseteq D(v_i, p^i)$ assign $X_i \in D(v_i; p^i_i)$ to $i$ and increase the prices in $X_i \setminus S_i$ by $\epsilon$. 
Walrasian tatonnement

• This process always ends, otherwise prices go to infinity.
• When it ends $S_i \in D(v_i; p^i)$
Walrasian tatonnement

- This process always ends, otherwise prices go to infinity.
- When it ends \( S_i \in D(v_i; p) \) in the limit \( \epsilon \to 0 \)
Walrasian tatonnement

- This process always ends, otherwise prices go to infinity.
- When it ends $S_i \in D(v_i; p)$ in the limit $\epsilon \to 0$
- What else?
Walrasian tatonnement

• This process always ends, otherwise prices go to infinity.
• When it ends \( S_i \in D(v_i; p) \) in the limit \( \epsilon \to 0 \)
• What else?
• The only condition left is that \( \bigcup_i S_i = [n] \)
• For that we need: \( S_i \subseteq X_i \in D(v_i; p^i) \)
Walrasian tatonnement

- This process always ends, otherwise prices go to infinity.
- When it ends $S_i \in D(v_i; p)$ in the limit $\epsilon \to 0$
- What else?
- The only condition left is that $\cup_i S_i = [n]$
- For that we need: $S_i \subseteq X_i \in D(v_i; p^i)$
- Definition: A valuation satisfied gross substitutes if for all prices $p \leq p'$ and $S \in D(v; p)$ there is $X \in D(v; p')$ s.t. $S \cap \{i; p_i = p'_i\} \subseteq X$
Walrasian tatonnement

• This process always ends, otherwise prices go to infinity.

• When it ends \( S_i \in D(v_i; p) \) in the limit \( \epsilon \to 0 \)

• What else?

• The only condition left is that \( \bigcup_i S_i = [n] \)

• For that we need: \( S_i \subseteq X_i \in D(v_i; p^i) \)

• Definition: A valuation satisfied gross substitutes if for all prices \( p \leq p' \) and \( S \in D(v; p) \) there is \( X \in D(v; p') \)
  s.t. \( S \cap \{i; p_i = p'_i\} \subseteq X \)

• With the new definition, the algorithm always keeps a partition.
Walrasian equilibrium

- Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.
Walrasian equilibrium

• Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

• Some examples of GS:
  • additive functions \( v(S) = \sum_{i \in S} v(i) \)
  • unit-demand \( v(S') = \max_{i \in S} v(i) \)
  • matching valuations \( v(S') = \max \) matching from \( S \)
Walrasian equilibrium

• Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

• Some examples of GS:
  • additive functions \( v(S) = \sum_{i \in S} v(i) \)
  • unit-demand \( v(S') = \max_{i \in S} v(i) \)
  • matching valuations \( v(S') = \max \) matching from S
Walrasian equilibrium

- Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

- Some examples of GS:
  - additive functions $v(S) = \sum_{i \in S} v(i)$
  - unit-demand $v(S') = \max_{i \in S} v(i)$
  - matching valuations $v(S') = \max$ matching from $S$
Walrasian equilibrium

- Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

- Some examples of GS:
  - additive functions \( v(S) = \sum_{i \in S} v(i) \)
  - unit-demand \( v(S) = \max_{i \in S} v(i) \)
  - matching valuations \( v(S) = \max \) matching from \( S \)
Walrasian equilibrium

- Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

- Some examples of GS:
  - additive functions \( v(S) = \sum_{i \in S} v(i) \)
  - unit-demand \( v(S) = \max_{i \in S} v(i) \)
  - matching valuations \( v(S) = \max \) matching from \( S \)
Walrasian equilibrium

- Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

- Some examples of GS:
  - additive functions $v(S) = \sum_{i \in S} v(i)$
  - unit-demand $v(S') = \max_{i \in S} v(i)$
  - matching valuations $v(S') = \max$ matching from $S$
  - matroid-matching
Walrasian equilibrium

• Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

• Some examples of GS:
  • additive functions \( v(S) = \sum_{i \in S} v(i) \)
  • unit-demand \( v(S) = \max_{i \in S} v(i) \)
  • matching valuations \( v(S) = \max \) matching from \( S \)
  • matroid-matching

\[
v(S) = \max_{i \in S} v(i)
\]
Walrasian equilibrium

• Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

• Some examples of GS:
  • additive functions $v(S) = \sum_{i \in S} v(i)$
  • unit-demand $v(S) = \max_{i \in S} v(i)$
  • matching valuations $v(S) = \max \text{ matching from } S$
  • matroid-matching Open: GS $\neq$ matroid-matching
Walrasian equilibrium

• Theorem [Kelso-Crawford]: If all agents have GS valuations, then Walrasian equilibrium always exists.

• Theorem [Gul-Stachetti]: If a class $C$ of valuations contains all unit-demand valuations and Walrasian equilibrium always exists then $C \subseteq GS$
Valuated Matroids

• Given vectors \( v_1, \ldots, v_m \in \mathbb{Q}^n \) define

\[
\psi_p(v_1, \ldots, v_n) = n \text{ if } \det(v_1, \ldots, v_n) = p^{-n} \cdot a/b
\]

for \( p \) prime \( a, b, p \in \mathbb{Z} \)

• Question in algebra:

\[
\min_{v_i \in V} \psi_p(v_1, \ldots, v_n) \text{ s.t. } \det(v_1, \ldots, v_n) \neq 0
\]

• Solution is a greedy algorithm: start with any non-degenerate set and go over each items and replace it by the one that minimizes \( \psi_p(v_1, \ldots, v_n) \).

• [DW]: Grassmann-Plucker relations look like matroid cond
Valuated Matroids

• Definition: a function \( v : \binom{[n]}{k} \rightarrow \mathbb{R} \) is a **valuated matroid** if the “Greedy is optimal”.
Matroidal maps

• Definition: a function $\nu : 2^{[n]} \to \mathbb{R}$ is a matroidal map if for every $p \in \mathbb{R}^n$ a set in $D(\nu; p)$ can be obtained by the greedy algorithm: $S_0 = \emptyset$ and

$$S_t = S_{t-1} \cup \{i_t\} \text{ for } i_t \in \text{argmax}_i \nu_p(i|S_t)$$
Matroidal maps

• Definition: a function \( \nu : 2^{[n]} \to \mathbb{R} \) is a matroidal map if for every \( p \in \mathbb{R}^n \) a set in \( D(\nu; p) \) can be obtained by the greedy algorithm: \( S_0 = \emptyset \) and

\[
S_t = S_{t-1} \cup \{i_t\} \quad \text{for} \quad i_t \in \text{argmax}_i \nu_p(i|S_t)
\]

• Definition: a subset system \( \mathcal{M} \subseteq 2^{[n]} \) is a matroid if for every \( p \in \mathbb{R}^n \) the problem \( \max_{S \in \mathcal{M}} p(S) \) can be solved by the greedy algorithm.
Discrete Concavity

- A function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex if for all \( p \in \mathbb{R}^n \), a local minimum of \( f_p(x) = f(x) - \langle p, x \rangle \) is a global minimum.

- Also, gradient descent converges for convex functions.

- We want to extend this notion to function in the hypercube: \( v : 2^{[n]} \rightarrow \mathbb{R} \) (or lattice \( v : \mathbb{Z}^{[n]} \rightarrow \mathbb{R} \) or other discrete sets such as the basis of a matroid)
Discrete Concavity

- A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if for all $p \in \mathbb{R}^n$, a local minimum of $f_p(x) = f(x) - \langle p, x \rangle$ is a global minimum.

- Also, gradient descent converges for convex functions.

- We want to extend this notion to function in the hypercube: $\nu : 2^{[n]} \rightarrow \mathbb{R}$ (or lattice $\nu : \mathbb{Z}^{[n]} \rightarrow \mathbb{R}$ or other discrete sets such as the basis of a matroid)
Discrete Concavity

• A function \( f : \mathbb{R}^n \to \mathbb{R} \) is convex if for all \( p \in \mathbb{R}^n \), a local minimum of \( f_p(x) = f(x) - \langle p, x \rangle \) is a global minimum.

• Also, gradient descent converges for convex functions.

• We want to extend this notion to function in the hypercube: \( v : 2^n \to \mathbb{R} \) (or lattice \( v : \mathbb{Z}^n \to \mathbb{R} \) or other discrete sets such as the basis of a matroid)
Discrete Concavity

• A function $v : 2^{[n]} \rightarrow \mathbb{R}$ is discrete concave if for all $p \in \mathbb{R}^n$ all local minima of $v_p$ are global minima. I.e.

\[
\begin{align*}
    v_p(S) &\geq v_p(S \cup i), \forall i \notin S \\
    v_p(S) &\geq v_p(S \setminus j), \forall j \in S \\
    v_p(S) &\geq v_p(S \cup i \setminus j), \forall i \notin S, j \in S
\end{align*}
\]

then $v_p(S) \geq v_p(T), \forall T \subseteq [n]$. In particular local search always converges.

• [Murota ’96] M-concave (generalize valuated matroids)
[Murota-Shioura ’99] $M^h$-concave functions
Equivalence

- [Fujishige-Yang] A function \( \nu : 2^n \to \mathbb{R} \) is gross substitutes iff it is a matroidal map iff it is discrete concave.

- [Kelso-Crawford '82] necessary /“sufficient” condition for price adjustment to converge gross substitutes

- [Murota-Shioura '99] generalize convexity to discrete domains M-discrete concave

- [Dress-Wenzel '91] generalize Grassmann-Plucker relations valuated matroids matroidal maps
Equivalence

- [Fujishige-Yang] A function \( \nu: 2^n \rightarrow \mathbb{R} \) is gross substitutes iff it is a matroidal map iff it is discrete concave.

- [Kelso-Crawford '82] necessary /“sufficient” condition for price adjustment to converge gross substitutes

- [Murota-Shioura '99] generalize convexity to discrete domains M-discrete concave

- [Dress-Wenzel '91] generalize Grassmann-Plucker relations valuated matroids matroidal maps

- In particular \( S \in D(\nu; p) \) in poly-time.
Equivalence

- [Fujishige-Yang] A function \( v : 2^{[n]} \rightarrow \mathbb{R} \) is gross substitutes iff it is a matroidal map iff it is discrete concave.

- In particular, \( S \in D(v; p) \) in poly-time.
- Proof through discrete differential equations.

- [Kelso-Crawford '82] necessary /“sufficient” condition for price adjustment to converge gross substitutes.
- [Dress-Wenzel '91] generalize Grassmann-Plucker relations valuated matroids matroidal maps.
Discrete Differential Equations

- Given a function $v : 2^{[n]} \rightarrow \mathbb{R}$ we define the discrete derivative with respect to $i \in [n]$ as the function $\partial_i v : 2^{[n]} \setminus i \rightarrow \mathbb{R}$ which is given by:

  $$\partial_i v(S) = v(S \cup i) - v(S)$$

  (another name for the marginal)
Discrete Differential Equations

• Given a function \( v : 2^[[n]] \rightarrow \mathbb{R} \) we define the discrete derivative with respect to \( i \in [n] \) as the function \( \partial_i v : 2^{[n] \setminus i} \rightarrow \mathbb{R} \) which is given by:

\[
\partial_i v(S) = v(S \cup i) - v(S)
\]

(another name for the marginal)

• If we apply it twice we get:

\[
\partial_{ij} v(S) := \partial_j \partial_i v(S) = v(S \cup ij) - v(S \cup i) - v(S \cup j) + v(S)
\]

• Submodularity: \( \partial_{ij} v(S) \leq 0 \)
Discrete Differential Equations

- [Reijnierse, Gellekom, Potters] A function $v : 2^{[n]} \rightarrow \mathbb{R}$ is in gross substitutes iff it satisfies:

$$\partial_{ij} v(S) \leq \max(\partial_{ik} v(S), \partial_{kj} v(S)) \leq 0$$

condition on the discrete Hessian.

- Idea: A function is in GS iff there is not price such that:

$$D(v; p) = \{S, S \cup ij\} \text{ or } D(v; p) = \{S \cup k, S \cup ij\}$$

If $v$ is not submodular, we can construct a price of the first type. If $\partial_{ij} v(S) > \max(\partial_{ik} v(S), \partial_{kj} v(S))$ then we can find a certificate of the second type.
Algorithmic Problems

• Welfare problem: given m agents with \( v_1, \ldots, v_m : 2^{[n]} \to \mathbb{R} \) find a partition \( S_1, \ldots, S_m \) of \([n]\) maximizing \( \sum_i v_i(S_i) \)

• Verification problem: given a partition \( S_1, \ldots, S_m \) find whether it is optimal.

• Walrasian prices: given the optimal partition \( (S_1^*, \ldots, S_m^*) \) find a price such that \( S_i^* \in \operatorname{argmax}_S v_i(S) - p(S) \)
Algorithmic Problems

- Techniques:
  - Tatonnement
  - Linear Programming
  - Gradient Descent
  - Cutting Plane Methods
  - Combinatorial Algorithms
Linear Programming

• [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\sum_S x_{iS} &= 1, \forall i \in [m] \\
\sum_i \sum_{S \ni j} x_{iS} &= 1, \forall j \in [n] \\
x_{iS} &\in \{0, 1\}
\end{align*}
\]
Linear Programming

- [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\text{s.t. } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1]
\end{align*}
\]
Linear Programming

- [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S)x_{iS} \\
\text{s.t. } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1]
\end{align*}
\]

primal

\[
\begin{align*}
\text{min } & \sum_i u_i + \sum_j p_j \\
\text{s.t. } & u_i \geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
& p_j \geq 0, u_i \geq 0
\end{align*}
\]
dual
Linear Programming

• [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\text{s.t. } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1]
\end{align*}
\]

primal

\[
\begin{align*}
\text{min } & \sum_i u_i + \sum_j p_j \\
\text{s.t. } & u_i \geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
p_j \geq 0, u_i \geq 0
\end{align*}
\]

dual

• For GS, the IP is integral: \( W_{IP} \leq W_{LP} = W_{D-LP} \)

• Consider a Walrasian equilibrium and \( p \) the Walrasian prices and \( u \) the agent utilities. Then it is a solution to the dual, so: \( W_{D-LP} \leq W_{eq} = W_{IP} \)
Linear Programming

- [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\text{subject to } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1]
\end{align*}
\]

primal

\[
\begin{align*}
\text{min } & \sum_i u_i + \sum_j p_j \\
\text{subject to } & u_i \geq v_i(S) - \sum_{j \ni i} p_j \forall i, S \\
& p_j \geq 0, u_i \geq 0
\end{align*}
\]

dual
Linear Programming

- [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max} \quad & \sum_i v_i(S) x_{iS} \\
\text{s.t.} \quad & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \supseteq j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1]
\end{align*}
\]

primal

\[
\begin{align*}
\text{min} \quad & \sum_i u_i + \sum_j p_j \\
\text{s.t.} \quad & u_i \geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
& p_j \geq 0, u_i \geq 0
\end{align*}
\]

dual

- In general, Walrasian equilibrium exists iff LP is integral.
Linear Programming

• [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max} & \sum_i v_i(S)x_{iS} \\
\sum_S x_{iS} &= 1, \forall i \in [m] \\
\sum_i \sum_{S \supset j} x_{iS} &= 1, \forall j \in [n] \\
x_{iS} &\in [0, 1]
\end{align*}
\]

\[
\begin{align*}
\text{min} & \sum_i u_i + \sum_j p_j \\
u_i &\geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
p_j &\geq 0, u_i \geq 0
\end{align*}
\]

primal \hspace{5cm} dual

• In general, Walrasian equilibrium exists iff LP is integral.

• Separation oracle for the dual: \( u_i \geq \max_S v_i(S) - p(S) \)

is the demand oracle problem.
[Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\text{subject to } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
x_{iS} & \in [0, 1]
\end{align*}
\]

primal

\[
\begin{align*}
\text{min } & \sum_i u_i + \sum_j p_j \\
u_i & \geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
p_j & \geq 0, u_i \geq 0
\end{align*}
\]

dual
Linear Programming

• [Nisan-Segal] Formulate this problem as an LP:

\[
\begin{align*}
\text{max } & \sum_i v_i(S) x_{iS} \\
\text{s.t. } & \sum_S x_{iS} = 1, \forall i \in [m] \\
& \sum_i \sum_{S \ni j} x_{iS} = 1, \forall j \in [n] \\
& x_{iS} \in [0, 1] \\
\end{align*}
\]

\[
\begin{align*}
\text{min } & \sum_i u_i + \sum_j p_j \\
\text{s.t. } & u_i \geq v_i(S) - \sum_{j \in S} p_j \forall i, S \\
& p_j \geq 0, u_i \geq 0 \\
\end{align*}
\]

primal \hspace{2cm} \text{dual}

• Walrasian equilibrium exists + demand oracle in poly-time
  = Welfare problem in poly-time

• [Roughgarden, Talgam-Cohen] Use complexity theory to show non-existence of equilibrium, e.g. budget additive.
Gradient Descent

- We can Lagrangify the dual constraints and obtain the following \textit{convex} potential function:

$$\phi(p) = \sum_i \max_S [v_i(S) - p(S)] + \sum_j p_j$$

- Theorem: the set of Walrasian prices (when they exist) are the set of minimizers of $\phi$.

$$\partial_j \phi(p) = 1 - \sum_i 1[j \in S_i]; S_i \in D(v_i; p)$$

- Gradient descent: increase price of over-demanded items and decrease price of over-demanded items.

- Tatonnement: $p_j \leftarrow p_j - \epsilon \cdot \text{sgn} \partial_j \phi(p)$
<table>
<thead>
<tr>
<th>method</th>
<th>oracle</th>
<th>running-time</th>
</tr>
</thead>
</table>

Comparing Methods
How to access the input
How to access the input

Value oracle:
given $i$ and $S$:
query $v_i(S)$. 
How to access the input

Value oracle:
given \( i \) and \( S \):
query \( v_i(S) \).

Demand oracle:
given \( i \) and \( p \):
query \( S \in D(v_i, p) \)
How to access the input

Value oracle:
given $i$ and $S$: query $v_i(S)$.

Demand oracle:
given $i$ and $p$: query $S \in D(v_i, p)$

Aggregate Demand:
given $p$, query.
$\sum_i S_i; S_i \in D(v_i, p)$
<table>
<thead>
<tr>
<th>Method</th>
<th>Oracle</th>
<th>Running-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
</tbody>
</table>
## Comparing Methods

<table>
<thead>
<tr>
<th>method</th>
<th>oracle</th>
<th>running-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
<tr>
<td>linear program</td>
<td>demand/value</td>
<td>weakly-poly</td>
</tr>
</tbody>
</table>
Comparing Methods

<table>
<thead>
<tr>
<th>method</th>
<th>oracle</th>
<th>running-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
<tr>
<td>linear program</td>
<td>demand/value</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>cutting plane</td>
<td>aggreg demand</td>
<td>weakly-poly</td>
</tr>
</tbody>
</table>

- [PL-Wong]: We can compute an exact equilibrium with $\tilde{O}(n)$ calls to an aggregate demand oracle.
## Comparing Methods

<table>
<thead>
<tr>
<th>method</th>
<th>oracle</th>
<th>running-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
<tr>
<td>linear program</td>
<td>demand/value</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>cutting plane</td>
<td>aggreg demand</td>
<td>weakly-poly</td>
</tr>
</tbody>
</table>
## Comparing Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Oracle</th>
<th>Running-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
<tr>
<td>linear program</td>
<td>demand/value</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>cutting plane</td>
<td>aggreg demand</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>combinatorial</td>
<td>value</td>
<td>strongly-poly</td>
</tr>
</tbody>
</table>
### Comparing Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Oracle</th>
<th>Running-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>tatonnement/GD</td>
<td>aggreg demand</td>
<td>pseudo-poly</td>
</tr>
<tr>
<td>linear program</td>
<td>demand/value</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>cutting plane</td>
<td>aggreg demand</td>
<td>weakly-poly</td>
</tr>
<tr>
<td>combinatorial</td>
<td>value</td>
<td>strongly-poly</td>
</tr>
</tbody>
</table>

- [Murota]: We can compute an exact equilibrium for gross substitutes in \( \tilde{O}((mn + n^3)TV) \) time.
Algorithmic Problems

- Welfare problem: given $m$ agents with $v_1, \ldots, v_m : 2^{[n]} \to \mathbb{R}$ find a partition $S_1, \ldots, S_m$ of $[n]$ maximizing $\sum_i v_i(S_i)$

- Verification problem: given a partition $S_1, \ldots, S_m$ find whether it is optimal.

- Walrasian prices: given the optimal partition $(S_1^*, \ldots, S_m^*)$ find a price such that $S_i^* \in \text{argmax}_S v_i(S) - p(S)$
Computing Walrasian prices

- Given a partition \( S_1, \ldots, S_m \) we want to find prices such that \( S_i \in \arg\max_S v_i(S) - p(S) \)

- For GS, we only need to check that no buyer want to add, remove or swap items.
Computing Walrasian prices

- Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \arg\max_S v_i(S) - p(S)$

- For GS, we only need to check that no buyer want to add, remove or swap items.
Computing Walrasian prices

• Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \arg\max_S v_i(S) - p(S)$

• For GS, we only need to check that no buyer want to add, remove or swap items.

$$w_{jk} = v_i(S_i) - v_i(S_i \cup k \setminus j)$$
Computing Walrasian prices

- Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \arg \max_S v_i(S) - p(S)$
- For GS, we only need to check that no buyer want to add, remove or swap items.

$$w_{\phi_i k} = v_i(S_i) - v_i(S_i \cup k)$$
Computing Walrasian prices

- Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \text{argmax}_S v_i(S) - p(S)$

- For GS, we only need to check that no buyer want to add, remove or swap items.

$$w_{j \phi_i} = v_i(S_i) - v_i(S_i \setminus j)$$
Computing Walrasian prices

- Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \text{argmax}_S v_i(S) - p(S)$

- For GS, we only need to check that no buyer want to add, remove or swap items.

\[ w_{\phi_i \phi_i'} = 0 \]
Computing Walrasian prices

• Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \arg\max_S v_i(S) - p(S)$

• For GS, we only need to check that no buyer want to add, remove or swap items.
Computing Walrasian prices

• Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \text{argmax}_S v_i(S) - p(S)$

• For GS, we only need to check that no buyer want to add, remove or swap items.
Computing Walrasian prices

- Given a partition $S_1, \ldots, S_m$ we want to find prices such that $S_i \in \arg\max_S v_i(S) - p(S)$

- For GS, we only need to check that no buyer want to add, remove or swap items.
Computing Walrasian prices

• Theorem: the allocation is optimal if the exchange graph has no negative cycle.

• Proof: if no negative cycles the distance is well defined. So let \( p_j = -\text{dist}(\phi, j) \) then:

\[
\text{dist}(\phi, k) \leq \text{dist}(\phi, j) + w_{jk}
\]

\[
v_i(S_i) \geq v_i(S_i \cup k \setminus j) - p_k + p_j
\]

And since \( S_i \) is locally-opt then it is globally opt. Conversely: Walrasian prices are a dual certificate showing that no negative cycles exist.
Computing Walrasian prices

- **Theorem:** the allocation is optimal if the exchange graph has no negative cycle.

- **Proof:** if no negative cycles the distance is well defined. So let \( p_j = -\text{dist}(\phi, j) \) then:

\[
\text{dist}(\phi, k) \leq \text{dist}(\phi, j) + w_{jk}
\]

\[
v_i(S_i) \geq v_i(S_i \cup k \setminus j) - p_k + p_j
\]

And since \( S_i \) is locally-opt then it is globally opt.

Conversely: Walrasian prices are a dual certificate showing that no negative cycles exist.

- **Nice consequence:** Walrasian prices form a lattice.
Algorithmic Problems

- Welfare problem: given $m$ agents with $v_1, \ldots, v_m : 2^{[n]} \to \mathbb{R}$ find a partition $S_1, \ldots, S_m$ of $[n]$ maximizing $\sum_i v_i(S_i)$

- Verification problem: given a partition $S_1, \ldots, S_m$ find whether it is optimal.

- Walrasian prices: given the optimal partition $(S_1^*, \ldots, S_m^*)$ find a price such that $S_i^* \in \arg\max_S v_i(S) - p(S)$
Algorithmic Problems

- Welfare problem: given \( m \) agents with \( v_1, \ldots, v_m : 2^{[n]} \to \mathbb{R} \), find a partition \( S_1, \ldots, S_m \) of \([n]\) maximizing \( \sum_i v_i(S_i) \).

- Verification problem: given a partition \( S_1, \ldots, S_m \), find whether it is optimal.

- Walrasian prices: given the optimal partition \( (S_1^*, \ldots, S_m^*) \), find a price such that \( S_i^* \in \arg\max_S v_i(S) - p(S) \).
Algorithmic Problems

• Welfare problem: given m agents with $v_1, \ldots, v_m : 2^{[n]} \to \mathbb{R}$ find a partition $S_1, \ldots, S_m$ of $[n]$ maximizing $\sum_i v_i(S_i)$

• Verification problem: given a partition $S_1, \ldots, S_m$ find whether it is optimal.

• Walrasian prices: given the optimal partition $(S_1^*, \ldots, S_m^*)$ find a price such that $S_i^* \in \text{argmax}_S v_i(S) - p(S)$
Incremental Algorithm

- For each $t = 1..n$ we will solve problem $W_t$ to find the optimal allocation of items $[t] = \{1..t\}$ to $m$ buyers.
- Problem $W_1$ is easy.
- Assume now we solved $W_t$ getting allocation $S_1, \ldots, S_m$ and a certificate $p = \text{maximal Walrasian prices}$.

\[
\begin{align*}
    w_{jk} &= v_i(S_i) - v_i(S_i \cup k \setminus j) + p_k - p_j \\
    w_{j\phi_i} &= v_i(S_i) - v_i(S_i \setminus j) - p_j \\
    w_{\phi_i k} &= v_i(S_i) - v_i(S_i \cup k) + p_k
\end{align*}
\]
Incremental Algorithm

- For each $t = 1..n$ we will solve problem $W_t$ to find the optimal allocation of items $[t] = \{1..t\}$ to $m$ buyers.
- Problem $W_1$ is easy.
- Assume now we solved $W_t$ getting allocation $S_1, \ldots, S_m$ and a certificate $p = \text{maximal Walrasian prices}$. 

[Diagram of allocation process]
Incremental Algorithm

- For each $t = 1..n$ we will solve problem $W_t$ to find the optimal allocation of items $[t] = \{1..t\}$ to $m$ buyers.
- Problem $W_1$ is easy.
- Assume now we solved $W_t$ getting allocation $S_1, \ldots, S_m$ and a certificate $p = \text{maximal Walrasian prices}$. 
Incremental Algorithm

- Algorithm: compute shortest path from $\phi$ to $t + 1$
- Update allocation by implementing path swaps
Incremental Algorithm

- Algorithm: compute shortest path from $\phi$ to $t + 1$
- Update allocation by implementing path swaps
Incremental Algorithm

- Algorithm: compute shortest path from $\phi$ to $t + 1$
- Update allocation by implementing path swaps
Incremental Algorithm

- Algorithm: compute shortest path from $\phi$ to $t + 1$
- Update allocation by implementing path swaps

- Graph has $O(t^2 + mt)$ non-negative edges
- After $n$ iterations of Dijkstra we get $\tilde{O}(n^3 + n^2 m)$
Incremental Algorithm

- Proof that new allocation \( \tilde{S}_1 \ldots \tilde{S}_m \) is optimal
- Define the new prices \( \tilde{p}_j = -\text{dist}(\phi, j) \)
  - (1) New prices are also a certificate for \( S_1 \ldots S_m \)
  - (2) \( \nu_i(S_i) - \tilde{p}(S_i) = \nu_i(\tilde{S}_i) - \tilde{p}(\tilde{S}_i) \)
- Hence, \( \tilde{S}_1 \ldots \tilde{S}_m \) and \( \tilde{p} \) are Walrasian prices.
Closure properties

- If $v_1, v_2 \in GS$ we might not have $v_1 + v_2 \in GS$
Closure properties

- If \( v_1, v_2 \in GS \) we might not have \( v_1 + v_2 \in GS \)

- Some preserving operations:
  - affine transformation \( \tilde{v}(S) = v(S) + p_0 - \sum_{i \in S} p_i \)
  - endowment \( \tilde{v}(S) = v(S|X) \)
  - convolution \( v_1 \star v_2(S) = \max_{T \subseteq S} v_1(T) + v_2(S \setminus T) \)
  - strong-quotient-sum
  - tree-concordant-sum
Closure properties

- If $v_1, v_2 \in GS$ we might not have $v_1 + v_2 \in GS$

- Some preserving operations:
  - affine transformation $\tilde{v}(S) = v(S) + p_0 - \sum_{i \in S} p_i$
  - endowment $\tilde{v}(S) = v(S|X)$
  - convolution $v_1 \ast v_2(S) = \max_{T \subseteq S} v_1(T) + v_2(S \setminus T)$
  - strong-quotient-sum
  - tree-concordant-sum

- Open question: can we construct all gross substitutes from matroid rank functions and those operations?
- Some progress: See talk by Eric Balkanski on Thu
End of Part I