Self-Correcting Sampling-Based Dynamic Multi-Unit Auctions
Output Ironing Demystified

Xiaoqi Zhu and Richard Liu

October 26, 2009

Dynamic Mechanism Design: An Overview

Many mechanism design problems are inherently dynamic.

- Movie tickets
- Internet advertising
- Kidney exchanges
- Airline seats

Dynamic Multi-unit Auctions

Features:

- Multi-unit supply
- Multi-unit demand
- Agents with bounded patience
- Probabilistic model of future scenarios

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability
2. Outcome efficiency

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability
2. Outcome efficiency
3. Incentive compatibility

A Simple Motivating Example

Example (Ice Cream)

Ice cream is made at one cone an hour. There are three agents.

Agent #	**Arrival**	**Departure**	**Value**	**Demand**
Agent 1	1	2	100	1
Agent 2	1	2	80	1
Agent 3	2	2	60	1

A Simple Motivating Example

Example (Ice Cream)

Ice cream is made at one cone an hour. There are three agents.

Agent #	**Arrival**	**Departure**	**Value**	**Demand**
Agent 1	1	2	100	1
Agent 2	1	2	80	1
Agent 3	2	2	60	1

Consider the naïve generalization of the Vickrey auction. If every buyer were truthful...

- Agent 1 wins in period 1 for 80; agent 2 wins in period 2 for 60.

A Simple Motivating Example

Example (Ice Cream)

Ice cream is made at one cone an hour. There are three agents.

Agent #	**Arrival**	**Departure**	**Value**	**Demand**
Agent 1	1	2	100	1
Agent 2	1	2	80	1
Agent 3	2	2	60	1

Consider the naïve generalization of the Vickrey auction. If every buyer were truthful...

- Agent 1 wins in period 1 for 80; agent 2 wins in period 2 for 60.

But, agent 1 can do better by reporting a value of 61:

- Then, agent 2 wins in period 1 for 60; agent 1 wins in period 2 for 61.

A Simple Motivating Example

- Truthfulness of the Vickrey auction no longer holds in a dynamic setting.
- The mechanism can be made strategyproof by charging agents the *critical value payment*, which we will talk more about later.

The Model

Model

C units of an identical item for sale, to be sold in T time periods.
Agent (bidder) i has type $\theta_i = (a_i, d_i, r_i, q_i)$ where

- $a_i \in \{1, ..., T\}$ is the agent's arrival time.
- $d_i \in \{1, ..., T\}$ is the agent's departure time.
- $r_i \in \mathbb{R}_{\geq 0}$ is the total value agent i is willing to pay for in some period $t \in \{a_i, ..., d_i\}$.
- $q_i \in \mathbb{Z}_{>0}$ is the number of units demanded by the agent.

The Model (cont'd)

Assumptions:

- **Limited misreports:** Agents *cannot* report **early arrivals**.
- **Single-valued preferences:** Agents are indifferent to time of allocation.

The Model (cont'd)

Inputs:

Active agents	$A^t \subseteq \theta^{1..t}$
Available items	S^t
Random events	$\omega = \{\omega^{1..t}\}$

$\downarrow$

Dynamic auction:

Decision policy	π
Payment policy	x

$\downarrow$

Outputs:

Decisions	$\pi_i^t(S^t, A^t, \omega^{1..t}) \in \{0,1\}$ for each $i \in A^t$
Payments	$x_i^t(S^t, A^t, \omega^{1..t})$ for each $i \in A^t$

Payment and Critical Value

Definition (Critical Value)

The **critical value** for agent i, given policy π and reports θ_{-i} of other agent types, is

$$v^c_{(a_i,d_i,q_i)}(\theta_{-i},\omega) = \min\{r'_i \text{ s.t. } \pi_i((a_i,d_i,r'_i,q_i),\theta_{-i},\omega) = 1\}$$

or ∞ if no such r'_i exists.

Lemma

Any truthful online mechanism that satisfies individual rationality must collect a payment equal to the critical value from each allocated agent.

Monotonicity

Definition (Partial Order on Types)

Types that offer the seller more flexibility are higher.

$$\theta_i \preceq_\theta \theta_i' \equiv (a_i \geq a_i') \wedge (d_i \leq d_i') \wedge (r_i \leq r_i') \wedge (q_i \geq q_i')$$

Definition (Monotonicity)

Policy π is monotonic if

$$(\pi_i(\theta_i, \theta_{-i}, \omega) = 1) \bigwedge (r_i > v^c_{(a_i, d_i, q_i)}(\theta_{-i}, \omega))$$

implies

$$\pi_i(\theta_i', \theta_{-i}, \omega) = 1$$

for all $\theta_i' \succeq_\theta \theta_i$, for all θ_{-i}, ω, and all agents i.

Monotonicity (cont'd)

An allocated agent must continue to be allocated if its type were higher, all else unchanged.

- Monotonicity is necessary for incentive compatibility (if losing agents receive no payment).
- Additionally, monotonicity is made sufficient for incentive compatibility by defining a payment policy that charges each allocated agent its critical value.

Bridging Algorithms and Mechanism Design

We can use an stochastic optimization algorithm to produce a decision policy π. We have seen some examples in class...

- Expectation
- Consensus
- Regret

This paper focuses on the Consensus algorithm.

Problem

- Question: Are decision policies produced by the CONSENSUS algorithm monotonic?

Problem

- Question: Are decision policies produced by the CONSENSUS algorithm monotonic?
- Answer: No...

Problem

- Question: Are decision policies produced by the CONSENSUS algorithm monotonic?
- Answer: No...but with the proper modification, yes!

Consensus Algorithm: A Refresher

Algorithm (Consensus)

votes(k) := 0 for each allocation k of up to S^t items to A^t
σ^j := GetSample(t) for each $k = 1..|\Sigma|$; $\Sigma = \{\sigma^{1..|\Sigma|}\}$
for each $j = 1..|\Sigma|$ **do**
  $\alpha^j := \text{Opt}(S^t, A^t, \sigma^t) \cap A^t$ `// active agents only`
  $\alpha^j_s := \texttt{Select}(\alpha^j, \Sigma, S^t, A^t)$
  votes(α^j_s) := votes(α^j_s) + 1
end for
$k^t := \arg\max_k \text{votes}(k)$
return k^t

Consensus Algorithm: A Refresher

Algorithm (CONSENSUS with ironing)

votes(k) $:= 0$ for each allocation k of up to S^t items to A^t
$\sigma^j :=$ GetSample(t) for each $k = 1..|\Sigma|$; $\Sigma = \{\sigma^{1..|\Sigma|}\}$
for each $j = 1..|\Sigma|$ **do**
  $\alpha^j := \mathrm{Opt}(S^t, A^t, \sigma^t) \cap A^t$ `// active agents only`
  $\alpha^j_{\mathtt{s}} := \mathtt{Select}(\alpha^j, \Sigma, S^t, A^t)$
  votes($\alpha^j_{\mathtt{s}}$) $:=$ votes($\alpha^j_{\mathtt{s}}$) $+ 1$
end for
$k^t := \arg\max_k$ votes(k)
$\breve{k}^t := \{i \sqsubset k^t : \text{not isIroned}_{A,D,Q}(\theta_i, t, (S, A)_{a_i..t}, \Sigma)\}$
return $\breve{k}^t$

Output Ironing

- "Ironed" decision discards allocations that violate monotonicity.
- Surviving allocations must allocate higher types earlier.

Output Ironing

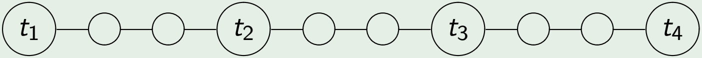
- "Ironed" decision discards allocations that violate monotonicity.
- Surviving allocations must allocate higher types *monotonically* earlier.

An Intuitive Example

Example

Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.

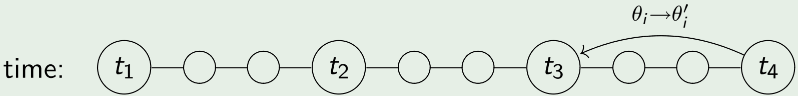
time:



An Intuitive Example

Example

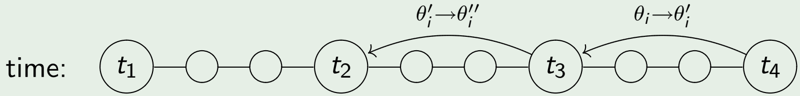
Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.



An Intuitive Example

Example

Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.

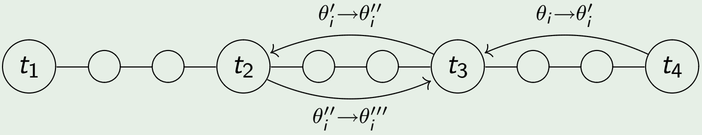


An Intuitive Example

Example

Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.

time:

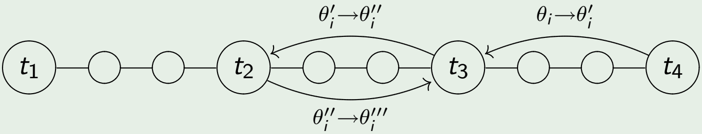


An Intuitive Example

Example

Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.

time:



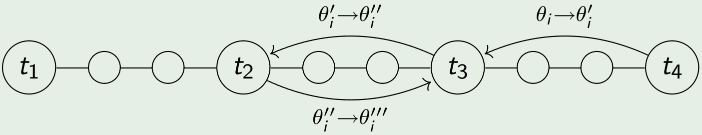
If we do not verify *monotonically*-earlier allocations, then this allocation decision would survive.

An Intuitive Example

Example

Suppose $\theta_i''' \succeq_\theta \theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. Suppose we are in period t_4 and policy π proposes to allocate to agent i, with type θ_i.

time:



If we do not verify *monotonically*-earlier allocations, then this allocation decision would survive. But **monotonicity fails** at t_2!

Output Ironing (cont'd)

Definition (Ironing)

Given decision k^t, the **ironed decision** $\breve{k}^t$ only keeps those $i \sqsubset k^t$ for which

$$t_i^\pi(\theta_i'', \theta_{-i}, \omega) \leq t_i^\pi(\theta_i', \theta_{-i}, \omega),$$

for all $\theta_i'' \succeq_\theta \theta_i' \succeq_\theta \theta_i$. If the condition fails, i's allocation is cancelled.

- $\text{isIroned}_{A,D,Q}$ checks this condition for each allocated agent i at the end of the CONSENSUS algorithm at time t.
- When an allocation is cancelled, the items are *discarded*.

Output Ironing (cont'd)

Theorem

The ironed policy $\breve{\pi}$ is monotonic.

Uncertainty Independence

Assumption:

- The distribution of future agents is **independent of past and current decisions**:

$$\Pr(\theta^{t+1..T} \mid k^{1..t}) = \Pr(\theta^{t+1..T})$$

for all t, all $k^{1..t}$.

  - However, we *can* condition on past and current arrivals.

Uncertainty Independence

Assumption:

- The distribution of future agents is **independent of past and current decisions**:

$$\Pr(\theta^{t+1..T} \mid k^{1..t}) = \Pr(\theta^{t+1..T})$$

for all t, all $k^{1..t}$.

  - However, we *can* condition on past and current arrivals.
  - This assumption render stochastic optimization feasible.
  - What types of scenarios does this preclude?

Uncertainty Independence (cont'd)

- *Uncertainty independence* facilitates ironing.
  - Enables counterfactual states to be simulated as type of an agent is varied.
  - Enables computational tractability.

A Simplification

- We only need to check monotonicity *locally*.

A Simplification

- We only need to check monotonicity *locally*.
- It turns out ironing is actually quite simple...

Adjacency Ironing

Definition (Adjacency ironing)

Given decision k^t, **adjacency-ironing** only keeps those $i \sqsubset k^t$ for which, for all $\theta'_i = (a'_i, d'_i, r'_i, q'_i) \succeq_\theta \theta_i = (a_i, d_i, r_i, q_i)$, with $r'_i = r_i$, it holds that

$$t_i^\pi(\theta''_i) \leq t_i^\pi(\theta'_i), \forall \theta''_i \in \theta'_i{++} \text{ with } r''_i = r'_i \text{ and}$$

$$t_i^\pi(\langle a'_i, d'_i, r'''_i, q'_i \rangle) \leq t_i^\pi(\langle a'_i, d'_i, r''_i, q'_i \rangle), \forall r'''_i \geq r''_i \geq r'_i.$$

If the above conditions fail, i's allocation is cancelled.

Theorem

Adjacency ironing is equivalent to ironing.

Ironing Agent Values

- When we iron agent values r_i, we run into a problem – these values are not discretized. How can we fix this?

Ironing Agent Values

- When we iron agent values r_i, we run into a problem – these values are not discretized. How can we fix this?
- Key lies in CONSENSUS algorithm – we only care when a particular vote changes

Ironing Agent Values

- When we iron agent values r_i, we run into a problem – these values are not discretized. How can we fix this?
- Key lies in CONSENSUS algorithm – we only care when a particular vote changes
- This induces particular values of r_i that we need to check called Breakpoints.

Ironing Agent Values

- When we iron agent values r_i, we run into a problem – these values are not discretized. How can we fix this?
- Key lies in CONSENSUS algorithm – we only care when a particular vote changes
- This induces particular values of r_i that we need to check called Breakpoints.

Definition (Breakpoints)

The `BrkPts` function determines the set of all (time, scenario, value) triples at which the the set of agents selected to be allocated in the offline allocation would change.

Breakpoints Example: `OnlyDep`

- The breakpoints are induced by the `Select` function that prunes the allocation results.

Breakpoints Example: `OnlyDep`

- The breakpoints are induced by the `Select` function that prunes the allocation results.
- Consider `OnlyDep`, which only allows allocation to agents departing in the current period.

Breakpoints Example: `OnlyDep`

- The breakpoints are induced by the `Select` function that prunes the allocation results.
- Consider `OnlyDep`, which only allows allocation to agents departing in the current period.
- Then, there is only one breakpoint for agent i with $\theta_i = (a_i, d_i, r_i, q_i)$ and scenario j in period t given by

$$r_o^j(i) = V(S^t, A^t, \omega \setminus \{i\}, \omega^j) - V(S^t - q_i, A^t \setminus \{i\}, \omega^j)$$

where $V(S, A, \omega^j)$ is the value of the solution of the offline optimization problem.

Using Breakpoints to Iron Agent Values

Definition (isIroned$_R$ algorithm)

- Calculates the breakpoints.
- Starting from the smallest breakpoint,
  - Simulates the decision policy to check that allocation happens monotonically earlier
  - Updates the breakpoints for the time periods after the change in allocation
- If not, allocation is ironed out.

The Efficiency Cost of Ironing

Ironing cancels decisions and discards resources. There is a clear tradeoff.

The Efficiency Cost of Ironing

Ironing cancels decisions and discards resources. There is a clear tradeoff.

Example

Suppose we restrict allocated agents only to those that depart now.

- *Rationale: maximizes information about agent types; goods are non-expiring.*

Output ironing would cancel allocation to all except maximally-patient agents. The efficiency loss is catastrophic.

The Problem Lies in Departure

- How do we establish monotonicity via ironing without considerable tradeoff in efficiency?
    - Recall the CONSENSUS algorithm:

The Problem Lies in Departure

- How do we establish monotonicity via ironing without considerable tradeoff in efficiency?
  - Recall the CONSENSUS algorithm:
    $\alpha^j := \mathsf{Opt}(S^t, A^t, \sigma^t) \cap A^t$
    $\alpha^j_{\mathtt{s}} := \mathtt{Select}(\alpha^j, \Sigma, S^t, A^t)$

The Problem Lies in Departure

- How do we establish monotonicity via ironing without considerable tradeoff in efficiency?
  - Recall the CONSENSUS algorithm:
    $\alpha^j := \mathsf{Opt}(S^t, A^t, \sigma^t) \cap A^t$
    $\alpha^j_{\mathtt{s}} := \mathtt{Select}(\alpha^j, \Sigma, S^t, A^t)$
  - We provide departure monotonicity by modifying the `Select` algorithm.

Departure Obliviousness

Definition

Policy π is **departure-oblivious** if for any agent i allocated in period t_i^*, the decisions made by the policy for periods $a_i \leq t \leq t_i^*$ do not change for any reported departure $d_i' > d_i$, holding all other inputs unchanged.

A departure-oblivious policy is trivially monotonic with respect to departure time.

Proposition

For a departure-oblivious policy, (a, v, q)-ironing is equivalent to ironing.

Departure Obliviousness

- Implement via `Select` method.

Departure Obliviousness

- Implement via `Select` method.
- Examples:
  - `OnlyDep` : $\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j|_{d=t}$
  - `IgnoDep` : $\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j$
- Which method results in a departure-oblivious algorithm?

Departure Obliviousness

- Implement via `Select` method.
- Examples:
  - `OnlyDep` : $\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j|_{d=t}$
  - `IgnoDep` : $\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j$
- Which method results in a departure-oblivious algorithm?

A More Sophisticated Algorithm

`NowWait`:

- Define $\rho = \rho_t$ to be the probability that agent i will still be present in the next period $t+1$ given type θ_i, but *ignoring* its reported departure.

A More Sophisticated Algorithm

`NowWait`:

- Define $\rho = \rho_t$ to be the probability that agent i will still be present in the next period $t+1$ given type θ_i, but *ignoring* its reported departure.
- Assume all agents present at t except for i either depart or are allocated in time t so that future demand is represented only by that in each scenario.

A More Sophisticated Algorithm

`NowWait`:

- We can then derive the expected value of allocating to agent i now ($now_i^t(\alpha^j, r_i)$) and the future value of a scenario ($wait_i^t(\alpha^j, r_i)$).
- $\text{now}_i^t(\alpha^j, r_i) = r_i + v(\alpha_-^j) + \frac{1}{|\Sigma|} \sum_{j' \in \Sigma} V(S^t - \#(\alpha^j), \emptyset, \sigma^{j'})$
- $\text{wait}_i^t(\alpha^j, r_i) =$
  $v(\alpha_-^j) + (1 - \rho) \frac{1}{|\Sigma|} \sum_{j' \in \Sigma} V(S^t - \#(\alpha_-^j), \emptyset, \sigma^{j'}) + \rho \frac{1}{|\Sigma|} \sum_{j' \in \Sigma} V(S^t - \#(\alpha_-^j), \{i\}, \sigma^{j'})$
- `NowWait`:

$$\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \{i \sqsubset \alpha^j : \text{now}_i^t(\alpha^j, r_i) \geq \text{wait}_i^t(\alpha^j, r_i)\}$$

NowWait

Proposition

$\textsc{Consensus} \oplus \texttt{NowWait}$ is departure-oblivious.

Experimental Results

Ironing	NowWait	OnlyDep	HROrRew	IgnoDep	Fixed	Opt
No	0.915	0.952	0.860	0.860	0.815	1
Yes	0.895	0.526	0.852	0.852	0.815	1

Experimental Results

Ironing	NowWait	OnlyDep	HROrRew	IgnoDep	Fixed	Opt
No	0.915	0.952	0.860	0.860	0.815	1
Yes	0.895	0.526	0.852	0.852	0.815	1

- Note the efficiency loss due to ironing when OnlyDep is used.

Experimental Results

Ironing	NowWait	OnlyDep	HROrRew	IgnoDep	Fixed	Opt
No	0.915	0.952	0.860	0.860	0.815	1
Yes	0.895	0.526	0.852	0.852	0.815	1

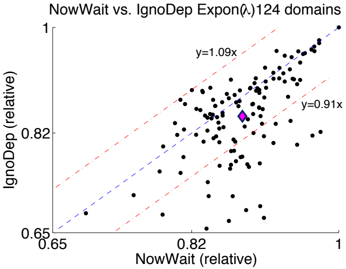
- Note the efficiency loss due to ironing when OnlyDep is used.
- By contrast, IgnoDep results in only a marginal loss of efficiency.

Experimental Results

Ironing	NowWait	OnlyDep	HROrRew	IgnoDep	Fixed	Opt
No	0.915	0.952	0.860	0.860	0.815	1
Yes	0.895	0.526	0.852	0.852	0.815	1

- Note the efficiency loss due to ironing when `OnlyDep` is used.
- By contrast, `IgnoDep` results in only a marginal loss of efficiency.
- `NowWait` further improves upon `IgnoDep`.

Experimental Results (cont'd)



Average: $(0.882, 0.849)$

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability
2. Outcome efficiency
3. Incentive compatibility

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability $\Leftarrow$ *adjacency-ironing*
2. Outcome efficiency
3. Incentive compatibility

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability $\Leftarrow$ *adjacency-ironing*
2. Outcome efficiency $\Leftarrow$ *departure obliviousness*
3. Incentive compatibility

Design Goals

Desirable properties of the dynamic auction:

1. Computational tractability $\Leftarrow$ *adjacency-ironing*
2. Outcome efficiency $\Leftarrow$ *departure obliviousness*
3. Incentive compatibility $\Leftarrow$ *monotonicity*

Concluding Remarks

- Applications of other stochastic optimization algorithms, e.g. Regret and Expectation, to dynamic multi-unit auctions
- Additional runtime improvements
- Formalize the tradeoff between monotonicity and optimality