Self-Correcting Sampling-Based Dynamic Multi-Unit Auctions

Florin Constantin
SEAS, Harvard University
33 Oxford St., Cambridge, MA 02138
florin@eecs.harvard.edu

David C. Parkes
SEAS, Harvard University
33 Oxford St., Cambridge, MA 02138
parkes@eecs.harvard.edu

ABSTRACT

We exploit methods of sample-based stochastic optimization for the purpose of strategyproof dynamic, multi-unit auctions. There are no analytic characterizations of optimal policies for this domain and thus a heuristic approach, such as that proposed here, seems necessary in practice. Following the suggestion of Parkes and Duong [17], we perform sensitivity analysis on the allocation decisions of an online algorithm for stochastic optimization, and correct the decisions to enable a strategyproof auction. In applying this approach to the allocation of non-expiring goods, the technical problem that we must address is related to achieving strategyproofness for reports of departure. This cannot be achieved through self-correction without canceling many allocation decisions, and must instead be achieved by first modifying the underlying algorithm. We introduce the `NowWait` method for this purpose, prove its successful interfacing with sensitivity analysis and demonstrate good empirical performance. Our method is quite general, requiring a technical property of *uncertainty independence*, and that values are not too positively correlated with agent patience. We also show how to incorporate "virtual valuations" in order to increase the seller's revenue.

Categories and Subject Descriptors

J.4 [**Social and behavioral sciences**]: Economics; G.4 [**Mathematical software**]: Algorithm design and analysis

General Terms

Algorithms, Economics, Theory

Keywords

Ironing, dynamic auctions, online stochastic optimization

1. INTRODUCTION

Mechanism design addresses the problem of private information in economic environments and seeks to implement desirable outcomes despite the willingness of agents to misreport this information. Auctions present a canonical problem of mechanism design. Many important mechanism design problems are in fact dynamic, for example with a dynamic agent population and new bids arriving online [11, 15]. Consider selling theater tickets, airline seats, or banner advertisements, where bidders may arrive over time and may require a response before all bids have been received.

We consider a natural instance of this problem, with C units of an identical item for sale, to be sold in the course of T time periods. Each bidder (or agent) i has an arrival time $a_i \in 1..T$, departure time $d_i \in 1..T$, and value $r_i \in \mathbb{R}_{\geq 0}$ for $q_i \in \mathbb{Z}_{>0}$ units of the item. The semantics of the bidder's type, $\theta_i = (a_i, d_i, r_i, q_i)$, are that the bidder has value r_i for receiving q_i units in some period $t \in a_i..d_i$. The arrival time models the moment at which the agent realizes her demand or learns about the existence of the auction while the departure models the latest moment at which the agent still has value for receiving the items. Agent types are drawn iid from probability density function $f(\theta_i)$, and n^t agents arrive each period, with associated density function $g(t)$.

The design goal may be alternatively one of efficiency (i.e., maximizing expected value) or revenue (i.e., maximizing expected payments). We will consider both objectives in this paper. If the goal was efficiency, then one might consider adopting the online Vickrey-Clarke-Groves mechanism [18, 19]. But this mechanism requires an optimal (or ϵ-optimal) decision policy, which is not computationally feasible in this domain. Our dynamic allocation problem is inherently combinatorial because of the multi-unit demand of agents, and optimal decision policies cannot be computed (or even represented) offline because of the size of the state space, which needs to include all possible sets of undominated agent bids that can be received in a single period. Online, sample-based algorithms to compute ϵ-optimal policies [10], also quickly become intractable because the sample-tree scales exponentially in the look-ahead horizon.

Other prior work in the probabilistic, dynamic framework considers only domains that facilitate analytic characterizations of optimal policies; e.g., domains with commonly-ranked items [6], unit-demand bidders with "regular" valuation distributions [14], smoothly changing types [20] or unit-demand bidders with time-discounting [4]. We are not aware of any prior work that is able to scalably address the multi-unit, dynamic auction problem in this paper.

In the absence of computational methods to compute optimal policies, or analytic characterizations of optimal polices, it seems necessary to adopt a heuristic approach. Parkes

EC'09, July 6–10, 2009, Stanford, California, USA.

and Duong [17] propose "output-ironing" as a methodology to transform heuristic, online algorithms for stochastic optimization into *strategyproof* dynamic auction protocols. A strategyproof dynamic auction is one in which truthful, immediate bidding is a dominant-strategy equilibrium. It is known that strategyproof dynamic auctions in this environment require that the allocation policy is *monotonic* [8, 15]. Loosely, a monotonic policy is one in which an agent continues to be allocated as it improves its bid (appropriately defined for arrival, departure, value and quantity). The idea behind output-ironing is to verify the monotonicity of an allocation policy online, as decisions are made, and correct such decisions as necessary to make the policy monotonic.

Parkes and Duong [17] successfully apply output-ironing to environments with *expiring goods*, where one or more units must be allocated in each period; e.g., the allocation of compute time on a shared computer. But this self-correcting approach is difficult to apply in environments with non-expiring goods, such as that considered in the current paper. The problem is with regard to establishing departure monotonicity, which requires that an agent allocated for some bid (or reported type) is also allocated for a bid with the same arrival, value, and quantity but a later departure. For example, suppose an agent i with reported type $(1, 5, \$10, 2)$ is allocated in period 5. It must be verified that the decision policy will continue to allocate i (in some period) for all reports of type $(1, d', \$10, 2)$ with $d' > 5$. But with non-expiring goods, any reasonable policy will wait until i's departure to decide whether to allocate i so that it maximizes the available information about other bids. This in turn makes it impossible to verify departure monotonicity with respect to later reports of departure because the bids that will arrive and affect allocation decisions are as yet unknown (e.g., at time period 5 it is unknown whether or not bids with later reported departures will be allocated). Thus, output ironing would need to cancel all allocations except those to bids for which there are no possible types with later departures (i.e., maximally-patient agents). This would result in an implemented policy with very low efficiency.

Our contribution. We design `NowWait`, a heuristic modification of the Consensus algorithm [21] for online stochastic optimization, that is provably departure monotonic and thus precludes the need for output ironing with respect to departure. When coupled with output ironing in the other dimensions of a bidder's type, it provides a strategyproof and scalable dynamic multi-unit auction. We also establish that a simplified form of output ironing, referred to here as *adjacency ironing* is sufficient to establish monotonicity. This significantly improves the scalability of the methodology. `NowWait` balances the immediate reward from accepting a bid with the estimated opportunity cost from waiting to a future period. Empirical analysis demonstrate higher than 90% efficiency with respect to the offline optimum, and higher than 98% efficiency with respect to the online optimum when this benchmark is available. This is achieved with around a 20x slow down due to using computational ironing, over-and-above the underlying method of stochastic optimization, and a per-period solve time of approximately 40 seconds in instances with more than 100 arrivals and 100 goods to allocate. The approach is very flexible, and can be applied to inputs that are first transformed to "virtual valuations" as in Myerson's revenue-optimal offline auction [13]. Our experimental results show that this can boost revenue significantly in environments with low demand relative to supply, as would be expected. The approach can also be combined with learning of the underlying distribution on agent types, because the incentive properties do not rely on having a correct probabilistic model.[1]

Other related work. Boutilier et al. [1] apply online stochastic combinatorial optimization to clearing expressive banner ad auctions, but without consideration of incentive issues. Hajiaghayi et al. [9] adopt a dynamic-programming approach to design simple dynamic auctions for settings with unit-demand bidders, with values drawn from a known prior but whose number may not be known in advance. Parkes [15] provides a general survey of online mechanisms, including both probabilistic and prior-free approaches.

The agenda of *automated mechanism design* (AMD) [2] shares the goal of creating a mechanism automatically, but differs from the approach adopted here in that it adopts optimization to design a functional description of all decisions that will be made by a mechanism, rather than seeking to adapt an existing decision algorithm, such as in the approach adopted here. This makes AMD very difficult to scale.

The work presented here conforms to the agenda of *heuristic mechanism design*, recently advocated by Parkes [16]. This stipulates that a problem in computational mechanism design be considered solved when a state-of-the-art algorithm for solving a problem with cooperative agents can be adopted, "with small modification" to solve the problem with self-interested agents. Output ironing is a small modification to Consensus in this sense, because it retains the vast majority of the decisions and also the same underlying computational approach in making decisions.

Incremental mechanism design [3] also modifies the rules of a mechanism to remove opportunities for manipulation. Unlike this paper, it requires an iterative approach because fixing one opportunity may introduce another, and is described only for offline mechanisms where the complete type profile is known. The approach is also not computational in our sense, in that it deals with functional representations of social choice functions defined on discretized type spaces. Lavi and Swamy [12] provide a general procedure to transform approximate VCG mechanisms into truthful-in-expectation mechanisms for static environments. But it is not apparent how to apply the approach to dynamic policies, where only part of an allocation is available at any point.

Outline. We first define more formally the set-up and the Consensus algorithm for online stochastic combinatorial optimization. In Section 2, we also define output ironing and establish a result about the sufficiency of adjacency-ironing. We continue by defining the important property of *departure obliviousness*, and noting the flexibility of our approach in reference to virtual valuations and learning. Sections 3 and 4 define variants on Consensus that are adapted to our problem and explain how to perform sensitivity analysis. In Section 5 we present experimental results, before concluding. All proofs are deferred to the full version, available from first author's website.

[1]The incentive problems that can occur because of informational externalities when learning in the context of dynamic auctions (see Gershkov and Moldovanu [5]) can be avoided by precluding the use of a new report of a type θ by bidder i for the purpose of updating the center's model about future reports until the agent has departed.

2. CONSENSUS AND IRONING

Recall that in our model, the type of an agent (a_i, d_i, r_i, q_i) specifies a value r_i for an allocation of q_i units in some period $\{a_i, \ldots, d_i\}$. We adopt the standard quasi-linear utility model, in which an agent that pays p for q_i units has utility $r_i - p$. We refer to $d_i - a_i$ as an agent's *patience*, and assume this is bounded with $d_i - a_i \leq \overline{\Delta}$. In the auctions we consider, an agent can make to the auctioneer a single claim about its type. All misreports of type are possible except for reports of *early arrivals*, i.e., we assume that an agent cannot claim to have an earlier arrival than its true arrival. To motivate this, recall that the arrival period models the period at which an agent learns of its own demand or learns about the existence of the mechanism. It is reasonable to restrict an agent's strategy from bidding before this period.[2]

The efficient policy maximizes total expected value:

$$V^*(h^1) = \max_{k^1 \in \mathcal{K}(h^1)} \mathbb{E}_{\theta^{2..T}}\left[\max_{k^2 \in \mathcal{K}(h^2)} \mathbb{E}_{\theta^{3..T}}\left[\ldots \max_{k^T \in \mathcal{K}(h^T)} v(k,\theta)\right]\right], \quad (1)$$

where k^t is the allocation decision taken at t, state $h^t = (S^t, A^t)$ denotes the number of available items S^t and the current set of *active* agents A^t (agents with $t \in a..d$), $\mathcal{K}$ defines the set of feasible allocations in period t, θ^t is the set of types that arrive at t, and $v(k,\theta)$ is the total value to agents allocated by decisions $k = k^{1..T}$ given types $\theta = \theta^{1..T}$. We write $i \sqsubset k$ for "agent i is allocated by decision k".

A dynamic auction $M = (\pi, x)$ defines a decision policy $\pi = \{\pi^{1..T}\}$ and a payment policy $x = \{x^{1..T}\}$, which may be randomized and depend on random events $\omega = \{\omega^{1..T}\}$, for example random samples of future bids. With this, the decision policy π induces decisions $k^t := \pi^t(S^t, A^t, \omega^{1..t})$ and collects payment $x_i^t(S^t, A^t, \omega^{1..t}) \in \mathbb{R}_{\geq 0}$ from each active agent. As useful shorthand, let $\pi_i(\theta, \omega) = 1$ denote that agent i is allocated for reported types θ given uncertain events ω, with $\pi_i(\theta, \omega) = 0$ otherwise.

Define the *critical-value* for agent i given policy π and reports θ_{-i} of other agents as $v^c_{(a_i,d_i,q_i)}(\theta_{-i}, \omega) = \min\{r_i'$ s.t. $\pi_i((a_i, d_i, r_i', q_i), \theta_{-i}, \omega) = 1\}$, or ∞ if no such r_i' exists. This is the smallest bid value for which agent i would still win, all else unchanged. Define a partial order on types: $\theta_i \preceq_\theta \theta_i' \equiv (a_i \geq a_i') \wedge (d_i \leq d_i') \wedge (r_i \leq r_i') \wedge (q_i \geq q_i')$. Type θ_i' is higher than θ_i if it offers the seller more flexibility, i.e. it has higher reward, demands less units and has a larger availability interval.

Monotonicity requires that an allocated agent would still be allocated if its type were higher, all else unchanged:

DEFINITION 1. *Policy π is monotonic if $(\pi_i(\theta_i, \theta_{-i}, \omega) = 1) \wedge (r_i > v^c_{(a_i,d_i,q_i)}(\theta_{-i}, \omega)) \Rightarrow \pi_i(\theta_i', \theta_{-i}, \omega) = 1$ for all $\theta_i' \succeq_\theta \theta_i$, for all θ_{-i}, ω, and all agents i.*

Monotonicity is sufficient, and essentially necessary (if losing agents receive no payment) for strategyproofness in this environment [15]. A mechanism with a monotonic decision policy is made strategyproof by defining a payment policy that charges each allocated agent its critical value. The critical value can be computed upon the departure.

Surprisingly, optimal policies need not be monotonic:

EXAMPLE 1. *[17] There are 3 units and 2 periods. In period 1, agent 1 has type $(1, 1, \$5, 1)$ and agent 2 has type $(1, 2, \$500, 2)$. In period 2, exactly one agent will arrive having type $(2, 2, \$1000, 3)$ with probability $1 - \gamma$ and type $(2, 2, \$5000, 1)$ with probability γ for some small $\gamma > 0$. The optimal policy must make a decision in period 1 about agent 1 because agent 1 will depart in this period. Agent 1 is not allocated because this would preclude the ability to allocate to type $(2, 2, \$1000, 3)$ that will arrive with high probability in period 2. Then in period 2, suppose the unlikely event occurs and an agent with type $(2, 2, \$5000, 1)$ arrives and the optimal policy allocates to agent 2 and also this new arrival.*

Now consider what happens were agent 2 to bid \$1000 rather than \$500. The optimal decision in period 1 is now to allocate to agent 1 for \$5 and expect to allocate to agent 2 in period 2. It is better to get certain value of \$5 from agent 1 than expected value $\gamma 5000$ from the unlikely type in period 2 (for a small enough γ). But now the same unlikely event occurs, and an agent with type $(2, 2, \$5000, 1)$ arrives in period 2. The optimal policy allocates to this agent and with 1 unit left is now unable to allocate to agent 2. This is a failure of monotonicity: agent 2 increases its value but went from winning to losing, fixing the types of other agents.

This same failure of monotonicity will occur with the policies constructed using sample-based stochastic optimization algorithms such as CONSENSUS [21] (described next) and sets up the problem addressed in this paper.

2.1 The Consensus Algorithm

The CONSENSUS (**C**) algorithm, proposed by van Hentenryck and Bent [21] for online stochastic optimization, is illustrated in Figure 1, together with the additional step of output ironing in determining decision $\breve{k}^t$ in period t.

Algorithm 1 CONSENSUS algorithm with ironing at time t.

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

A *scenario* σ^j in period t is a sample of a possible future: σ^j defines the types $\theta^{[t+1,T]}$ for periods $t+1$ through T. Given a scenario σ^j, and the current state (S^t, A^t), there is a well-defined offline optimization problem $\text{Opt}(S^t, A^t, \sigma^j)$. This is a weighted knapsack problem: find the subset of bids $A^t \cup \sigma^j$ that maximize the total value allocated without exceeding the capacity S^t. The **C** algorithm constructs samples, solves this offline optimization problem for each sample, and of the agents allocated in this offline problem picks out as winning agents only the active agents (i.e., discarding future, sampled, agents). Denote this set of winning agents in scenario σ^j as α^j. This set may then be additionally "filtered" via a `Select` function to give set $\alpha^j_{\mathtt{s}}$. The set of active agents $\alpha^j_{\mathtt{s}}$ then receives one vote, the one for scenario σ^j. It is the `Select` function that we will modify to make **C** departure monotonic. We will write $\mathbf{C} \oplus$ `Select` to specify that **C** is used together with the `Select` function.

[2]This assumption is adopted in many other papers, including Hajiaghayi et al. [8] and Pai and Vohra [14].

The proposed decision k^t for current time t is the one with most votes (breaking ties at random). We also denote by $\mathbf{C}(\text{votes}(\Sigma^{\neq j}), \alpha)$ the most voted decision if scenario j votes for decision α. Thus $\mathbf{C}(\text{votes}(\Sigma^{\neq j}), \alpha^j_{\mathtt{s}}) = k^t$. The final decision $\breve{k}^t$ results from discarding all ironed agents.

The $\mathbf{C}$ algorithm is applicable in domains satisfying *uncertainty independence*, i.e. in which the distribution of future agents is independent of past and current decisions:[3] $\mathbb{P}(\theta^{t+1..T}|k^{1..t}) = \mathbb{P}(\theta^{t+1..T})$ for all t, all $k^{1..t}$. This property requires that future demand is exogenous, and independent of current and past allocation decisions. For example, when selling airline tickets it requires that bids for seats arrive irrespective of the number of seats remaining for sale. Uncertainty independence ensures that scenarios are valid for any decision path and allows for the same $|\Sigma|$ scenarios to be valid whatever the decision made now and in future periods.

2.2 Output Ironing

We now explain output ironing (the "isIroned" function). Policies π and $\breve{\pi}$, and decisions k^t and $\breve{k}^t$, denote respectively the policies and decisions before and after ironing. Let $t^\pi_i(\theta, \omega) \in T \cup \{\infty\}$ denote i's allocation time (∞ if none exists) when reported types are θ, for random events ω.

DEFINITION 2 (IRONING). *Given decision k^t, the* ironed *decision $\breve{k}^t$ only keeps those $i \sqsubset k^t$ for which*

$$t^\pi_i(\theta''_i, \theta_{-i}, \omega) \leq t^\pi_i(\theta'_i, \theta_{-i}, \omega), \tag{2}$$

for all $\theta''_i \succeq_\theta \theta'_i \succeq_\theta \theta_i$. If (2) fails, i's allocation is canceled.

The ironing step is performed in a period t in which $\mathbf{C}$ proposes to allocate an agent. Eq. (2) requires that an allocation to agent i is canceled unless an allocation to the same agent would have also been made, and in a monotonically-earlier period, for all higher reported types of agent i. When an allocation is canceled the items that were to be allocated are discarded and agent i is never allocated.[4]

Ironing requires not only that an agent is provably allocated for all higher reports, but that this occurs in *monotonically earlier* periods. With this, one can make an inductive argument to establish that the ironed policy is monotonic:

THEOREM 1. *[17] Ironed policy $\breve{\pi}$ is monotonic.*

Uncertainty-independence facilitates ironing, by enabling the simulation of counterfactual states as the type of an agent is varied without considering changes in the realization of future bids due to different earlier decisions.

It will be sufficient to perform a simplified form of ironing. Let $\theta''_i \in \theta'_i$`++` if θ''_i is a higher type than θ'_i but θ''_i strictly improves over θ'_i in at most one dimension of (a, d, q, r). Fix θ_{-i} and ω, and omit them from the t^π_i notation.

DEFINITION 3. *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*

$$t^\pi_i(\theta''_i) \leq t^\pi_i(\theta'_i),\ \forall\, \theta''_i \in \theta'_i\texttt{++} \text{ with } r''_i = r'_i \text{ and} \tag{3}$$
$$t^\pi_i(\langle a'_i, d'_i, r'''_i, q'_i\rangle) \leq t^\pi_i(\langle a'_i, d'_i, r''_i, q'_i\rangle), \forall r'''_i \geq r''_i \geq r'_i$$

If (3) fails, i's allocation is canceled.

THEOREM 2. *Adjacency-ironing is equivalent to ironing.*

Algorithm 2 performs adjacency-ironing, following the prescription of Def. 3. Algorithm 3 provides pseudo-code for isIroned_R, which checks the first condition in Eq. (3). It tracks the changes in $\mathbf{C}$ decisions for values higher than i's reported value, r_i, fixing the rest of i's type and all other agent types. For each scenario in each time period in $a_i..t^*_i$ it identifies values at which the set of agents selected to be allocated in the offline allocation in that scenario would change: these are the *scenario breakpoints*. It does so via the BrkPts function, which determines the set of all (time, scenario, value) triples at which the set of agents selected to be allocated changes. This function also determines the "before" and "after" decision as the value is increased past the scenario breakpoint, denoted respectively $\alpha^<_{\mathtt{s}}(\beta)$ and $\alpha^>_{\mathtt{s}}(\beta)$ for breakpoint β. Example 2 presents an instance of Algorithm 3 for $\mathbf{C}$ when `Select` is `IgnoDep`.

EXAMPLE 2. *Say 3 items are sold for 2 periods. Agents $X_{1,2}$ arrive at time 1: X_i has $a_i = 1, d_i = i, q_i = i, r_i = i$, $i = 1, 2$. There are 7 time 2 scenarios $\sigma^{1..7}$, each with one agent with quantity $2, 2, 3$ and value $3, 4, 10$ on scenarios $\sigma^{1,2}$, $\sigma^{3,4}$ and $\sigma^{5,6,7}$ resp. Votes are $\{X_1\}$ and $\emptyset$ for $\sigma^{1..4}$ and $\sigma^{5..7}$ resp.; hence X_1 is allocated. Agent X_3 arrives at time 2 : $a_3 = d_3 = 2, q_3 = 1, r_3 = 0.5$. X_2 is allocated at time 2: $t^*_2 = 2$.*

Let us follow value output ironing for X_2, tracking decision changes from time $1 = a_2$ to $2 = t^*_2$ as 2's value is increased. Time 1 scenario breakpoints are $3, 4, 9$ on $\sigma^{1,2}, \sigma^{3,4}$ and $\sigma^{5,6,7}$ resp. All 7 time 2 breakpoints are at 0.5.

Denote by $\mathbf{C}_t$ the $\mathbf{C}$ decision at t. If X_2 had value 3, $\mathbf{C}_1$ would change to $\emptyset$ as $\sigma^{1,2}$ votes become $\{X_1, X_2\}$. Using Algorithm 3's notation, $\{X_1\} = \mathbf{C}(\text{votes}(\Sigma^{\neq j_\beta}), \alpha^<_{\mathtt{s}}) \neq \emptyset = \mathbf{C}(\text{votes}(\Sigma^{\neq j_\beta}), \alpha^>_{\mathtt{s}})$ at $t^\beta = 1$, for $j_\beta = 1$ and 2, $r_\beta = 3$, $\alpha^<_{\mathtt{s}} = \{X_1\}$ and $\alpha^>_{\mathtt{s}} = \{X_1, X_2\}$. All 7 time 2 breakpoints are now at 0 since $X_{2,3}$ can both be allocated. X_2 is still allocated at time 2, surviving ironing so far.

If X_2 had value 4, $\mathbf{C}_1$ would change again to $\{X_1, X_2\}$ as $\sigma^{3,4}$ votes become $\{X_1, X_2\}$; $t_2 = 1$ and any time 2 breakpoint is discarded since no items are left. Last breakpoint is at 9: all votes are now for $\{X_1, X_2\}$. As allocation times never increase as X_2's value increases, isIroned returns `false`*: X_2 survives ironing and is allocated at time 2.*

In this example, t^β always equaled t_2, precluding the need for updating $\vec{S}, \vec{A}$ and breakpoints.

Consider an agent i that can be allocated, i.e. with $q_i \leq S^t$. For simple `Select` methods, such as `OnlyDep` which selects only those agents that are departing in the current period, the only breakpoint on scenario j in period t for agent i with type $\theta_i = (a_i, d_i, r_i, q_i)$ is given by:

$$r^j_o(i) = V(S^t, A^t \backslash \{i\}, \sigma^j) - V(S^t - q_i, A^t \backslash \{i\}, \sigma^j) \tag{4}$$

where by $V(S, A, \sigma^j)$ we denote the value of the solution of $\text{Opt}(S, A, \sigma^j)$. This follows from the simple combinatorics of the offline weighted knapsack problem.

We will soon introduce more nuanced `Select` methods that may have multiple scenario breakpoints, with the voted

[3] Our results remain valid if in the uncertainty independence requirement, we also condition on past and current arrivals: $\mathbb{P}(\theta^{t+1..T}|k^{1..t}, \theta^{1..t}) = \mathbb{P}(\theta^{t+1..T}|\theta^{1..t})$. However, at each time step, new scenarios would have to be generated.

[4] We discard, rather than return, the allocated goods when canceling the decision in order to prevent a knock-on effect, wherein a decision to iron the decision of one agent would have a ripple effect on the decision of the "base policy" π and thus whether another agent should be ironed.

Algorithm 2 isIroned$_{A,D,Q}(\theta_i, t_i^*(\theta_i), (S^t, A^t)_{a_i \le t \le t_i^*}, \Sigma)$: online verification of monotonicity with respect to all higher types θ_i' for an agent of type θ_i allocated in period t_i^*. New allocation time $t_i^*(\theta_i')$ is found by simulating π starting at a_i, where i's type is replaced by θ_i'. The breakpoints computed in isIroned$_R$ for each θ_i' allow determining efficiently whether $t_i^\pi(\theta_i'') > t_i^\pi(\theta_i')$ for any $\theta_i'' \in \theta_i'$++.

for each $\theta_i' = (a_i', d_i', r_i, q_i')$, $(a_i' \le a_i, d_i' \ge d_i, q_i' \le q_i)$ **do**
  if isIroned$_R(\theta_i', t_i^*(\theta_i'), (S^t, A^t)_{a_i \le t \le t_i^*(\theta_i')}, \Sigma)$ **or**
    $t_i^\pi(\theta_i'') > t_i^\pi(\theta_i')$ for any $\theta_i'' \in \theta_i'$++ with $r_i'' = r_i'$ **then**
    return `true // i ironed`
  end if
end for
return `false // i not ironed`

decision changing more than once. This renders sensitivity analysis, and thus ironing, a bit more involved.

Remark: The payments of agents must be computed as the critical value of the ironed policy. For this, the procedure outlined above for ironing is essentially reversed: one steps down lower values until the agent would be unallocated in policy π, or allocated but fail the ironing test and thus be unallocated in ironed policy $\breve{\pi}$ [17].

2.3 Departure Obliviousness

The obvious concern with ironing, which cancels decisions and discards resources, is that it may establish monotonicity at the expense of destroying the value of a policy by canceling almost all decisions.

In fact, if this was a problem of stochastic optimization with cooperative agents then it would be optimal to delay any allocation decision until an agent's departure, since this is without cost to the agent and allows the center to receive more information about agent types. This is encapsulated in the `OnlyDep` select method: only allocate to those agents that are in the majority vote decision and depart now.

But as explained earlier, this would lead to a very low quality policy when coupled with output ironing. Output ironing would fail to establish the monotonically-earlier property of Eq. (2) for any types except maximally-patient agents, and cancel most allocation decisions.

We will focus on *departure oblivious* `Select` methods, i.e. invariant to an allocated agent's delay of departure:

Definition 4. *Policy π is* departure-oblivious *if for any agent i allocated in period t_i^*, the decisions made by the policy for periods $a_i \le t \le t_i^*$ do not change for any reported departure $d_i' > d_i$, holding all other inputs unchanged.*

This property trivially implies monotonicity with respect to departure. In combination with ironing with respect to arrival, value and quantity only ("(a, v, q)-ironing"), i.e. checking Eq. (3) only for types θ_i'' and θ_i' that differ from θ_i in these attributes, this provides full monotonicity.

Proposition 1. *For a departure-oblivious policy, (a, v, q)-ironing is equivalent to ironing.*

Note the different decompositions of ironing across the dimensions of a bidder's type: Algorithms 2 and 3 separate value from the other dimensions, whereas departure obliviousness precludes the need for departure ironing.

Algorithm 3 isIroned$_R(\theta_i, t_i^*(\theta_i), (S^t, A^t)_{a_i \le t \le t_i^*}, \Sigma)$: verification of value-monotonicity for an agent i with type θ_i allocated in period t_i^*. The set of breakpoints $B = \{\langle t^\beta, \sigma^{j_\beta}, r_\beta, \alpha_{\mathbf{s}}^<(\beta), \alpha_{\mathbf{s}}^>(\beta)\rangle\}_\beta$ track the changes in scenario votes. As i's value increases past r_β, the vote on scenario σ^{j_β} at time t^β changes from $\alpha_{\mathbf{s}}^<(\beta)$ to $\alpha_{\mathbf{s}}^>(\beta)$. If changing only the vote on scenario j_β causes the **C** decision to flip, then π must be simulated to allow checking that i's allocation time (stored by t_i) for the higher (slightly above r_β) reward is no later than for its lower (below r_β) reward. Finally, as the **C** decision changes at t^β, breakpoints between periods $t^\beta + 1$ to t_i (we have $t^\beta \le t_i$) must be updated. The counterfactual sets $\{\vec{S}, \vec{A}\} = (S^t, A^t)_{a_i \le t \le t_i^*}$ are maintained, being initialized to the actual ones determined by π for i's reported value, which is lower than all r_β's.

$B := \bigcup_{t=a_i}^{t_i^*} \text{BrkPts}^R(\theta_i, t, \vec{S}, \vec{A}, \Sigma)|_{r \ge r_i}$
$t_i := t_i^*$
while $B \neq \emptyset$ **do**
  Let $\beta := \langle t^\beta, \sigma^{j_\beta}, r_\beta, \alpha_{\mathbf{s}}^<, \alpha_{\mathbf{s}}^>\rangle$ such that $r_\beta \le r_b \, \forall\, b \in B$.
  if at t^β, $\mathbf{C}(\text{votes}(\Sigma^{\neq j_\beta}), \alpha_{\mathbf{s}}^<) \neq \mathbf{C}(\text{votes}(\Sigma^{\neq j_\beta}), \alpha_{\mathbf{s}}^>)$ **then**
    `// Simulate` π `until time` $\min\{t_i,$ `time` i `wins}`
    increase i's value to slightly over r_β
    $\breve{t} := t^\beta$
    repeat
      $\pi^{\breve{t}} := \pi^{\breve{t}}(A^{\breve{t}}, S^{\breve{t}})$
      update active agents: $A^{\breve{t}+1} := \theta^{\breve{t}+1} \cup A^{\breve{t}}|_{d \ge \breve{t}+1} \setminus \pi^{\breve{t}}$
      update supply: $S^{\breve{t}+1} := S^{\breve{t}} - \#(\pi^{\breve{t}})$
      $\breve{t} := \breve{t} + 1$
    until $i \sqsubset \pi^{\breve{t}-1}$ or $\breve{t} > t_i$
    if i has not won **then**
      return `true // i ironed`
    else
      $t_i := \breve{t} - 1$ `// allocation time for new value` r_β
      $B := \big(B|_{t \le t^\beta} \setminus \{\beta\}\big) \cup \bigcup_{t=t^\beta+1..t_i} \text{BrkPts}^R(\theta_i, t, \vec{S}, \vec{A}, \Sigma)|_{r \ge r_\beta}$
    end if
  end if
end while
return `false // i not ironed`

2.4 Flexibility: Virtual Values and Learning

In the context of single item, static auctions, Myerson [13] proved the revenue-optimality of an efficient auction defined on "virtual valuations". The virtual valuation of a bidder whose bid r_i is drawn with probability density function (pdf) f and cumulative distribution function (cdf) F is:

$$w(r_i) = r - \tfrac{1-F(r_i)}{f(r_i)} \tag{5}$$

Our approach to dynamic auction design provides considerable flexibility. For example, we can apply the algorithm essentially unchanged, except for substituting valuations with virtual valuations. Each reported type is converted into a virtual valuation by retaining (a_i, d_i, q_i) but replacing r_i with $w(r_i)$. All computation is then performed with respect to virtual valuations: samples are taken from the distribution on virtual valuations induced by the distribution on valuations, ironing is performed with respect to a partial order defined on virtual valuations, and the payment is first determined as the critical "virtual value" and then transformed into the corresponding actual value.

When the distribution f has a non-decreasing hazard rate

(as required by Myerson [13]), the ironed policy remains monotonic and thus strategyproof. Without this property, it would be necessary to also adopt Myerson's notion of ironing to transform the virtual valuation function into an increasing function. We term Myerson's method "input ironing" whereas this paper's method is "output ironing."

As another indication of our approach's flexibility, the self-correcting methodology advanced here does not require that the mechanism has correct information about the underlying distribution on types. The distribution can simply be learned over time, for example through a non-parametric history sampling approach [21]. For strategyproofness, one must preclude an agent's reported type affecting the mechanism's distributional model until the agent has departed.

3. BASIC SELECT METHODS

In this section, we introduce some basic select methods that are departure oblivious and therefore useful together with output ironing for the design of strategyproof, dynamic multi-unit auctions. We have already seen

$$\texttt{OnlyDep}\text{: }\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j|_{d=t}$$

`OnlyDep` is clearly not departure oblivious.[5] For a straw man method that is departure oblivious, we use the identity `Select` method that ignores all departure information:

$$\texttt{IgnoDep} : \texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \alpha^j \quad (6)$$

`HazRate` only selects bidders somewhat likely to leave soon:

$$\texttt{HazRate} : \texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \left\{ i \sqsubset \alpha^j \text{: } \tfrac{1-F_i^D(d_i)}{f_i^D(d_i)} < c \right\} \quad (7)$$

where departures have pdf f_i^D and cdf F_i^D, and $c \in (0,1)$ is a parameter. Agent i is retained in $\alpha^j_{\mathtt{s}}$ by `HazRate` iff its reported departure d_i is late enough.

`HROrRew` also selects bidders with high value-per-item:

$$\texttt{HROrRew} : \texttt{Select}(\alpha^j, \Sigma, S^t, A^t) =$$
$$\{ i \sqsubset \alpha^j \text{: } \left(\tfrac{1-F_i^D(d_i)}{f_i^D(d_i)} < c \right) \vee \left(\mathbb{P}[\frac{R}{Q} > \frac{r_i}{q_i}] < w \right) \} \quad (8)$$

where parameters $c \in (0,1)$ and $w \in (0,1)$, and R and Q are random variables denoting an agent's value and quantity. That is, if i is "too good to miss" then it is selected even if its departure does not satisfy Eq. (7). Parameters c and w can be optimized for the distribution on agent types to maximize the performance of $\mathbf{C}\oplus$`HazRate` or $\mathbf{C}\oplus$`HROrRew`.

LEMMA 1. $\mathbf{C}\oplus$`IgnoDep` *is departure-oblivious. If* F_i^D *has a monotone non-decreasing hazard rate (i.e. it is* regular*) then* $\mathbf{C}\oplus$`HROrRew` *and* $\mathbf{C}\oplus$`HazRate` *are departure-oblivious.*

Just as with `OnlyDep`, the selected subset of agents $\alpha^j_{\mathtt{s}}$ in scenario σ^j will change at most once as the value of agent i is increased with the `HazRate` method. The change, if any, occurs if $q_i \le S^t$ and at bid value $r_o^j(i) = V(S^t, A^t \setminus \{i\}, \sigma^j) - V(S^t - q_i, A^t \setminus \{i\}, \sigma^j)$. Agent i is in $\alpha^j_{\mathtt{s}}$ for $r_i \ge r_o^j(i)$ if and only if Eq. (7) is satisfied.

When the departure condition in `HROrRew` is satisfied then this behaves as `HazRate` and there is one breakpoint at $r_o^j(i)$ for an agent that can be feasibly allocated. But otherwise, there can be two breakpoints when the value $r_j^c(i)$, at which $\mathbb{P}[\frac{R}{Q} > \frac{r_j^c(i)}{q_i}] = w$, is greater than $r_o^j(i)$. In this case, for $r_i \in [r_o^j(i), r_j^c(i))$ agent i is in α^j but not selected and then for $r_i \in [r_j^c(i), \infty)$ agent i is also selected. More importantly, within ironing, `HROrRew` only yields one breakpoint: the condition in Eq. (8) is independent of time and is satisfied by all higher types if satisfied by a certain type.

4. THE NOWWAIT HEURISTIC

In this section we describe `NowWait`, a departure-oblivious `Select` method that makes an explicit tradeoff between the value of allocating to an agent that could disappear and the likely benefit of waiting for other opportunities.

The `NowWait Select` method filters α^j down to $\alpha^j_{\mathtt{s}}$ by retaining those agents for which the estimated value from allocating now is greater than the estimated value from waiting, considering that i's value may be lost because it may depart:

$$\texttt{NowWait}\text{: }\texttt{Select}(\alpha^j, \Sigma, S^t, A^t) = \{ i \sqsubset \alpha^j \text{: } \mathrm{now}_i^t(\alpha^j, r_i) \ge \mathrm{wait}_i^t(\alpha^j, r_i) \} \quad (9)$$

For estimating the future value on a scenario j', we make the pessimistic assumption that all agents present in A^t (except i) either depart or are allocated in this period so that the future demand is represented only by that in each scenario $j' \in \Sigma$. The global estimate is simply an average over the per-scenario estimates.[6]

Let $\alpha^j_{-} = \alpha^j \setminus \{i\}$ and $v(\alpha^j_{-})$ denote the total value to the agents allocated in α^j_{-}. Let $\#(\alpha)$ denote the number of items allocated by action α. The value obtained by allocating to agent i with value r_i in period t is estimated as:

$$\mathrm{now}_i^t(\alpha^j, r_i) = r_i + v(\alpha^j_{-}) + \tfrac{1}{|\Sigma|} \textstyle\sum_{j' \in \Sigma} V(S^t - \#(\alpha^j), \emptyset, \sigma^{j'}),$$

Let $\rho = \rho_t$ 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 (to provide departure obliviousness):

$$\rho = \mathbb{P}[D > t | D \ge t, a_i, r_i, q_i] \quad (10)$$

The value for waiting to allocate i is estimated as:

$$\mathrm{wait}_i^t(\alpha^j, r_i) = v(\alpha^j_{-}) + (1-\rho) \tfrac{1}{|\Sigma|} \textstyle\sum_{j' \in \Sigma} V(S^t - \#(\alpha^j_{-}), \emptyset, \sigma^{j'}) + \rho \tfrac{1}{|\Sigma|} \textstyle\sum_{j' \in \Sigma} V(S^t - \#(\alpha^j_{-}), \{i\}, \sigma^{j'})$$

Since for any i and σ^j, α^j, $\mathrm{now}_i^t(\alpha^j, r_i)$ and $\mathrm{wait}_i^t(\alpha^j, r_i)$ are independent of i's reported departure, we get:

PROPOSITION 2. $\mathbf{C}\oplus$`NowWait` *is departure-oblivious.*

Figure 1 highlights the subsets of interest among active and future (sampled) agents on a scenario. `Select` methods

[5] In fact, $\mathbf{C}\oplus$`OnlyDep` does not satisfy a considerably weaker myopic monotonicity property. Suppose that if agent i, allocated by $\mathbf{C}\oplus$`OnlyDep` at time $t_i^*(=d_i)$, were to delay its reported departure to $d_i' > d_i$, then all $\mathbf{C}\oplus$`OnlyDep`'s decisions would be identical up to time $t_i^* - 1$. Then $\mathbf{C}\oplus$`OnlyDep` will no longer allocate i at t_i^* since i's departure is later. We can show that under our assumptions, all methods in this paper, including `OnlyDep`, satisfy similar properties with respect to arrival and value, but not with respect to quantity.

[6] If we retained an unallocated agent i' and considered its presence when computing opportunity costs for i in scenario j', then ironing for i' would also need to analyze the effect of i' raising its value on i's (and maybe other agents') allocation decision at t because of this coupling effect through the `NowWait` select rule. We wish to avoid this additional complication.

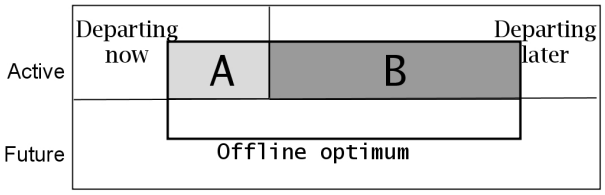

Figure 1: Subsets of agents for `Select` on a scenario.

select subsets of active agents in the offline optimum: A and B for `IgnoDep`, A (only departing now) for `OnlyDep` and subsets of A and B (*a priori* "urgent") for `NowWait`.

4.1 Sensitivity Analysis

Having defined `NowWait` we need to be able to perform sensitivity analysis of the resulting **C** algorithm. For this it is crucial to be able to compute scenario breakpoints to determine when the decision $\alpha^j_{\mathbf{s}}$ changes as agent i's value varies. There may be two breakpoints per scenario. The first occurs at $r^j_o(i)$ when agent i enters α^j and the second at $r^j_c(i)$ when agent i is retained by the `Select` method.

To better understand `NowWait`'s behavior, we use shorthand $v_{j'} = V(S^t - \#\alpha^j, \emptyset, \sigma^{j'})$ and $c_{j'} = V(S^t - \#\alpha^j_-, \emptyset, \sigma^{j'}) - V(S^t - \#\alpha^j, \emptyset, \sigma^{j'})$. $c_{j'}$ is the opportunity cost incurred by allocating to i if scenario j' was the actual future. If $\alpha^j = \{i\}$ and $j' = j$ then $c_j = r^j_o(i)$. With this, we have $\text{now}^t_i(\alpha^j, r_i) = r_i + v(\alpha^j_-) + \frac{1}{|\Sigma|}\sum_{j'\in\Sigma} v_{j'}$ and $\text{wait}^t_i(\alpha^j, r_i) = v(\alpha^j_-) + (1-\rho)\frac{1}{|\Sigma|}\sum_{j'\in\Sigma}(c_{j'}+v_{j'}) + \rho\frac{1}{|\Sigma|}\sum_{j'\in\Sigma}(\max(r_i, c_{j'}) + v_{j'})$, where the final term comes from recognizing that $V(S^t - \#(\alpha^j_-), \{i\}, \sigma^{j'}) = \max(r_i + v_{j'}, c_{j'} + v_{j'}) = \max(r_i, c_{j'}) + v_{j'}$. Simplifying, i, allocated in α^j, is retained if and only if:

$$r_i|\Sigma| \geq (1-\rho)\sum_{j'\in\Sigma} c_{j'} + \rho\sum_{j'\in\Sigma}\max(r_i, c_{j'}) \qquad (11)$$

Note that, apart from the number of items (implicit in $c_{j'}$) allocated by α^j_-, the condition in Eq. (11) is independent of scenario j. Let $\phi_\rho(r_i) = r_i|\Sigma| - (1-\rho)\sum_{j'\in\Sigma} c_{j'} - \rho\sum_{j'\in\Sigma}\max(r_i, c_{j'}) = \text{now}^t_i(\alpha^j, r_i) - \text{wait}^t_i(\alpha^j, r_i)$.

In general, the value and patience of an agent may be correlated according to type distribution $f(\theta_i)$. This will complicate sensitivity analysis for **C**⊕`NowWait`. For now we assume that patience and value are independent, and thus ρ does not depend on an agent's value r_i.

Given this independence assumption, there is a threshold $\tau^*_\rho(i)$, for i in α^j to be selected by `NowWait` such that:

$$\text{now}^t_i(\alpha^j, r_i) \geq \text{wait}^t_i(\alpha^j, r_i) \text{ if and only if } r_i \geq \tau^*_\rho.$$

Theorem 3 will prove a more general statement.

Let $r^j_c(i) = \max(r^j_o(i), \tau^*_\rho)$ denote the second scenario breakpoint. We may have $\tau^*_\rho > r^j_o(i)$ and thus $r^j_c(i) > r^j_o(i)$ if, for example, scenario j predicts less demand than the other scenarios and i is likely very patient (ρ is close to 1). For values in the range $[0, r^j_o(i))$, neither α^j nor $\alpha^j_{\mathbf{s}}$ include agent i. For values in the range $[r^j_o(i), r^j_c(i))$, i is included in α^j but it is removed by `NowWait`: $i \not\sqsubset \alpha^j_{\mathbf{s}}$. For values in the range $[r^j_c(i), \infty)$, both α^j and $\alpha^j_{\mathbf{s}}$ include i. Say that there exists a scenario j' identical to j. Then it is possible that the three possible decisions on scenario j (which coincide with the ones on j') induce three different allocations to be chosen in **C** by having the highest number of votes. In particular, it is possible that the **C** allocation is respectively with, without and then again with i for rewards respectively in $[0, r^j_o(i))$, $[r^j_o(i), r^j_c(i))$ and $[r^j_c(i), \infty)$. Decisions for such values in $[0, r^j_o(i))$ would be ironed.

Proposition 3 establishes the intuitive property that the more likely an agent i is to still be present in the next period, the higher value i needs to have in order not to be filtered from the decision by the `NowWait Select` method.

PROPOSITION 3. *As ρ increases from 0 to 1, the threshold τ^*_ρ at which an agent $i \sqsubset \alpha^j$ is selected by* `NowWait` *weakly increases from the average to the maximum of the costs $\{c_{j'} : j' \in \Sigma\}$, when agent patience is independent of value.*

Our implementation of sensitivity analysis accounts for the fact that, by varying a_i (and thus ρ) or q_i, the threshold τ^* for `NowWait` may change.

4.1.1 Correlation of Value and Patience

We now consider the consequence of allowing an agent's patience to be correlated with its value. For computational tractability, it will again be important to identify a single threshold τ^* at which the agent will pass the `NowWait Select` test. The estimated probability ρ that an agent i will still be present in the next period becomes a function $\rho(r_i)$, that depends on i's value and therefore varies as i's value is adjusted in performing sensitivity analysis.

THEOREM 3. *If $r_i\cdot(1-\rho(r_i))$ is non-decreasing in r_i, then $\exists$ threshold τ^* with:* $\text{now}^t_i(\alpha^j, r_i) \geq \text{wait}^t_i(\alpha^j, r_i) \iff r_i \geq \tau^*$.

Negative or moderately positive correlation between value and patience is thus sufficient for the existence and uniqueness of a threshold τ^*, such that $\text{now}^t_i(\alpha^j, \tau^*) = \text{wait}^t_i(\alpha^j, \tau^*)$, with $\text{now}^t_i(\alpha^j, r'_i) \geq \text{wait}^t_i(\alpha^j, r'_i)$ if and only if $r'_i > \tau^*$.[7] For $r_i(1-\rho(r_i))$ to be non-decreasing, it is sufficient for example that: (i) $\rho(r_i)$ is independent of r_i, (ii) $\rho(r_i)$ is non-increasing with r_i, or (iii) $\rho(r_i)$ is not increasing too quickly with r_i, such that $\partial\rho(r_i)/\partial r_i \leq \frac{1-\rho(r_i)}{r_i}$ for all r_i. If possible agent patiences are $\{\delta_1 < \ldots < \delta_{n_\Delta}\}$, a sufficient condition for $\rho(r_i)$ to be non-increasing in r_i (case (ii)) is that: $\frac{\mathbb{P}[\Delta=\delta_l, R=r_i]}{\mathbb{P}[\Delta=\delta_{l+1}, R=r_i]}$ is non-decreasing in r_i, $\forall\, l = 1..n_\Delta - 1$. One can show that this implies $\mathbb{E}[r|\delta_l] \geq \mathbb{E}[r|\delta_{l+1}]$, i.e. values are higher for smaller patiences.

As the smallest solution of $\phi(\cdot) = 0$ with weakly monotone ϕ (see Theorem 3's proof), τ^* can be found via binary search in $[\frac{1}{|\Sigma|}\sum c_{j'}, \max c_{j'}]$ (recall Proposition 3).

4.2 Example: One Item, Two Impatient Bidders

We illustrate `NowWait` and `IgnoDep`, in a simple optimal stopping environment with one item for sale, two periods and one impatient, unit-demand, bidder per period with value i.i.d. sampled from distribution F. Method `OnlyDep` does not remove any agent from α^j on scenario σ^j (i.e. it is identical to `IgnoDep`) since agents are completely impatient.

Denote by v^1_c the first bidder's critical value. As the number of scenarios tends to ∞, `OnlyDep`'s v^1_c is the *median* of F: half of the second period draws need to be higher for the first bid to be rejected. In comparison, `NowWait`'s v^1_c is the *mean*

[7] In a unit-demand domain, Pai and Vohra [14] also require negative correlation of value and patience, in the form of a decreasing hazard rate condition in one parameter out of value, arrival and departure when the other two are fixed.

of F: $\rho = 0$, each $c_{j'}$ is drawn from F and $\tau^* = \frac{1}{|\Sigma|}\sum_{j'} c_{j'}$. Thus, NowWait appears better placed for average-case performance. Section 5.1.1's experimental data show the effects of different critical values on allocative efficiency for two or more periods keeping the unit-supply constraint.

5. EXPERIMENTAL EVALUATION

We analyze in turn the allocative efficiency, revenue and runtime of this ironing-based approach to the design of dynamic, multi-unit auctions. Unless otherwise mentioned, the **C** algorithm uses 50 scenarios (samples of possible futures) and a bidder's quantity and patience are uniform in 1..5 and its value distribution is $Expon(0.1)$ times its quantity.

5.1 Allocative Efficiency

Each of the 124 points in Fig. 2 represents an average over at least 20 runs of NowWait and IgnoDep's relative efficiencies on a domain where a bidder's value is its quantity times an exponentially distributed variable. In such domains, we varied supply, demand, the number of time periods, the exponential parameter λ or bidders' maximum quantity or patience. IgnoDep performs at least 9% worse in about a quarter of the domains, 5% worse *on average* and never better than 9% in comparison with NowWait.

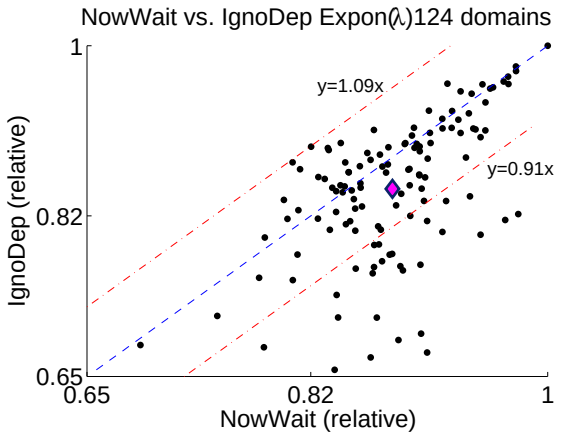

Figure 2: NowWait's versus IgnoDep's relative efficiency (offline optimum = 1) for 124 domains with exponentially distributed values with independence of value and patience. The pink diamond, at (0.882, 0.849), represents the average of all 124 points.

We go on to study more closely an auction with 10 items, 5 time periods, and 2 bidders arriving in each period. The Select method HazRate is of course dominated by the more general HROrRew, and only results for this second method are presented. The parameters c and w for the HROrRew method were optimized offline, but no setting was better than that for IgnoDep (i.e., allowing all allocated, active agents to remain in the selected set of winning agents). Table 1 presents allocative efficiencies as an average over 200 trials, divided by the average offline efficiency, i.e. the value that would be achieved if all bids were available in period 1.

Whereas ironing destroys the efficiency of OnlyDep (as expected, because all allocations except those to maximally patient agents must be canceled), NowWait still yields an efficiency of 0.895 after ironing. Note that the overall performance of NowWait with ironing is 5% better than that of both IgnoDep and HROrRew. Standard deviations for Table 1 are around 0.15 for all entries but OnlyDep with ironing, for which it is around 0.3. For all entries except OnlyDep with ironing, the 95% confidence intervals have a radius of 0.02, confirming the statistical significance of our results. The Fixed method is less sophisticated than the other methods. It optimally (offline) tunes a per-item price p and allocates any bidder whose bid amounts to at least p per item. This method's average allocative efficiency is 0.815, which further highlights the extent of NowWait's (efficiency 0.895) improvement over IgnoDep (efficiency 0.852).

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

Table 1: Allocative efficiency normalized to offline efficiency in a dynamic auction for 10 items.

Ironing	NowWait	OnlyDep	HROrRew	IgnoDep	Opt
No	0.944	0.941	0.881	0.881	1
Yes	0.918	0.245	0.874	0.874	1

Table 2: Allocative efficiency when value is distributed $Exp(0.1 \cdot (d_i - a_i))$, inducing a negative correlation between value and patience.

For comparison, NowWait's and IgnoDep's efficiencies are very similar if per-item values are U(0,1) instead. We again see an effect of each policy's approach: NowWait (resp. IgnoDep) aims for good mean (resp. median) performance (see Sec. 4.2). The mean and median are equal for the uniform, but not for the exponential distribution, as used in Table 1.

Table 2 considers the effect of allowing for negative correlation between value and patience, when the exponential distribution parameter is proportional to a bidder's patience. Before ironing, NowWait's allocative efficiency is slightly better than OnlyDep's, that has the advantage of waiting until a bidder's departure. Ironing is now even more destructive on OnlyDep. This is because high-value bidders, the ones selected by the offline knapsack problem, tend to have *small* patiences, and thus are often ironed.

For all methods except OnlyDep, ironing cancellations were very infrequent, and absent if all bidders have unit-demand. This confirms the intuition that even though possible such cancellations are quite rare and caused by combinatorial peculiarities. For NowWait, more than half the cancellations were due to ironing in value rather than arrival or quantity.

5.1.1 One Item, Impatient Bidders

We consider now the simple domain of a single unit of supply and one impatient bidder per period. This domain, considered in Section 4.2 for two periods, is appealing due to the availability of an optimal policy (Gilbert and Mosteller [7], henceforth GM), that is monotonic and thus strategyproof, providing an optimal *online* benchmark.

As bidders are impatient, there is no need for monotonicity with respect to departure. Thus, this is a setting in which the additional complexity of NowWait should *not* be expected to be worthwhile over-and-above the simplicity of OnlyDep (identical here to IgnoDep). Furthermore, unit-supply and impatience render all Select methods monotonic.

The optimal policy GM is defined by a sequence R_n of posted prices (where n bidders are yet to arrive). The criti-

Horizon	$\mathbb{E}$[NowWait]	$\mathbb{E}$[OnlyDep]	$\mathbb{E}$[GM]	$\mathbb{E}$[Opt]
2	0.911	0.897	0.911	1
4	0.871	0.867	0.873	1
8	0.855	0.859	0.863	1
16	0.854	0.863	0.867	1
32	0.858	0.871	0.877	1

Table 3: Relative (offline optimum=1) allocative efficiency in a unit-supply, impatient bidder domain. GM is the maximally-efficient, monotonic policy.

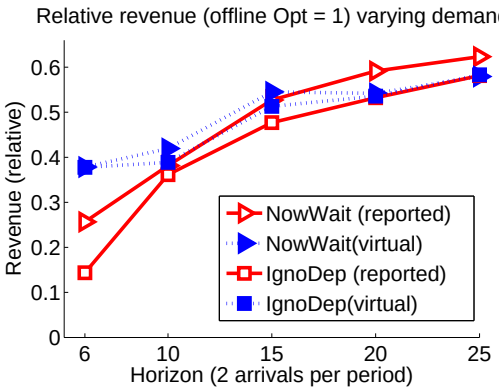

Figure 3: Revenue normalized to offline efficiency in a dynamic auction for 30 items, varying the number of time periods (and thus the level of competition). Examines the effect of adopting virtual valuations.

cal value R_n also represents the expected efficiency after the n-th remaining bidder arrives. GM is monotonic and truthful because R_n is independent of the reported values and no temporal strategies are available to impatient agents. For the Expon(λ) distribution, $R_0 = 0$ (allocate last bidder if no earlier winner) and $R_{n+1} = R_n + \frac{1}{\lambda}e^{-\lambda R_n}$ [7].

In Table 3 we compare the efficiency of NowWait and OnlyDep (averaged over 100,000 trials) with GM and the offline optimum. For small horizons (and hence small numbers of agents), the NowWait method actually outperforms OnlyDep. But the simpler, OnlyDep method does better for larger horizons and more agents. We explain this by noting that if n bidders are yet to arrive, NowWait (resp. OnlyDep) sets as critical values the mean (resp. median) of the highest order statistic of n iid $Exp(0.1)$ variables.

In summary, it is encouraging to us that the sample-based stochastic optimization methods can come within 97.8% of the value of the optimal online policy in this environment (this is the relative performance of **C** with NowWait at a horizon of 32), while being flexible and general enough to extend to multi-unit demand environments.

5.2 Boosting Revenue

Our experiments with virtual valuations in place of agent values show that our approach can increase seller's revenue.

Fig. 3 plots the revenue with and without virtual valuations. The revenue metric is normalized with respect to the total value from the efficient offline allocation. In this auction, we have 30 items available and vary the number of periods from 6 to 25, with 2 bidders arriving per period. As the number of periods increases the competition increases: the expected demand ranges from low-demand (36) to high-demand (150), i.e. the demand:supply ratio trends from 1.2x to 5x. We observe that virtual valuations have a significant positive effect on revenue for low demand environments. For example, in the case of 6 periods (and thus a low demand:supply ratio of 1.2x), adopting virtual valuations provides a boost of as much as 169% for IgnoDep and 49% for NowWait. On the other hand, we also see that virtual valuations can also be detrimental to revenue properties for high demand environments (for a demand:supply ratio of 4 or more. Thus, it would be important for a designer to understand the type of environment before adopting virtual valuations. We also see that the revenue properties of NowWait generally dominate those of IgnoDep, both with and without ironing. OnlyDep's revenue (not shown in Fig. 3) is always significantly below that of other methods due to extensive cancellations by ironing.

5.3 Computational Scalability

We run all experiments on a CentOS 8-node Pentium 4 at 3GHz cluster with 512 MB of RAM. Fig. 4 summarizes the results, averaged over 50 trials, for a dynamic auction with 2 bidders arriving each period. In the set of experiments for the left-hand side and the center plot in Fig. 4, there are 20 items and 10 periods and we increase the number of scenarios sampled within **C**. For the right-hand side plot in Fig. 4, we increase the supply of items while holding the expected demand:supply ratio constant at 3, by increasing the time horizon and thus the total expected demand as the supply increases. In addition to looking at the scalability of the system, we are also interested in the overhead that is imposed by the need to perform computational ironing.

All three plots show that NowWait's computational overhead when compared to the other Select methods is limited and reasonable. For example, as the number of scenarios increases, NowWait's overhead grows sub-linearly. This is despite NowWait's theoretical quadratic (as opposed to linear) dependence on the number of scenarios (for all $j, j' \in \Sigma$, the $c_{j'}$ costs must be computed for each scenario (σ^j) by solving two offline optimization problems). We believe that this is due in part to some NowWait-specific improvements that we have made, for example caching the offline optimization results for the $c_{j'}$ costs (see Sec. 4.1). For most experimental settings that we considered, on average, **C** and ironing take around 15 times more than **C** alone for all methods except OnlyDep, for which the overhead is around 40x.[8]

The right hand side experiment in Figure 4 measures our methods' *absolute, per period* runtime (in seconds) of CONSENSUS and ironing, as the relevant characteristics (number of time periods, supply and expected demand) of the domain are scaled proportionally. It is quite encouraging, though perhaps surprising, that NowWait's per-period runtime is decreasing for the longer horizons.

[8]We notice OnlyDep's higher overhead of ironing from Fig. 4: the ratio of the runtimes of OnlyDep and IgnoDep is close to 1 without ironing, but about 2.5 with ironing. There exists (a slightly smaller) difference in ratios even if a, d and q are kept constant in isIroned$_{A,D,Q}$ (see Algorithm 2). The reason for this discrepancy is revealed by recalling that isIroned$_R$ (see Algorithm 3) computes breakpoints at all times between a bidder's arrival and time of allocation. OnlyDep allocates bidders at their departure, significantly later on average than the other methods. Hence breakpoints for significantly more time periods are computed in isIroned$_R$ for OnlyDep.

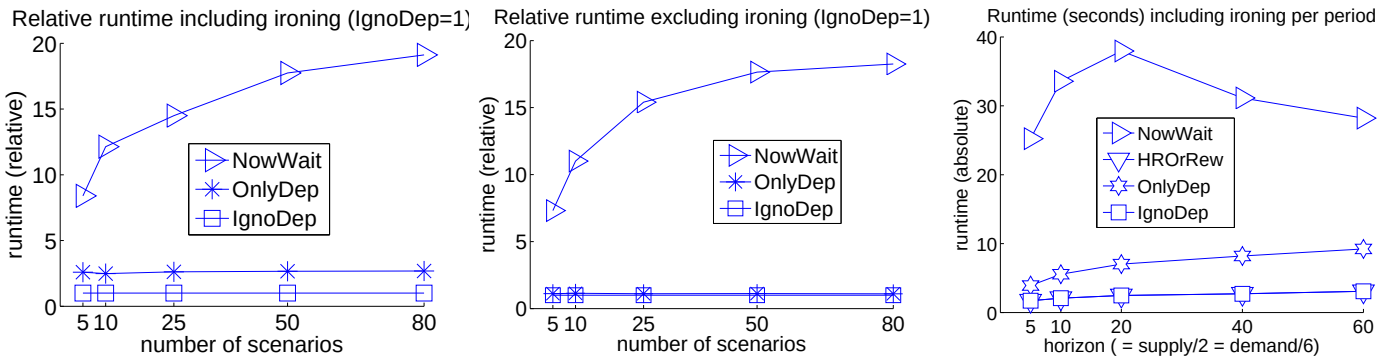

Figure 4: Runtimes in a dynamic auction. The left-hand side, resp. center, experiment shows relative runtime including, resp. excluding, ironing for an auction with 20 items and 10 time periods. The x-axis varies the number of scenarios. In the right-hand side experiment, showing seconds of runtime per time period, the expected demand:supply ratio is kept constant at 3 by adjusting them together with the number of time periods, as shown on the x-axis.

6. CONCLUSIONS

We presented the first application of stochastic optimization to dynamic, incentive-compatible multi-unit auctions with patient bidders that demand multiple units of an item. Method `NowWait` is used to modify the CONSENSUS algorithm [21] and evaluate opportunity costs when deciding whether to retain a vote to allocate to a particular agent given a particular scenario. Self correction by output ironing yields truthfulness, and we can aim for either good efficiency or revenue. The results show excellent efficiency and scalability, with a sub-linear computational overhead of `NowWait` with respect to CONSENSUS, and also demonstrate revenue boosting via the use of virtual valuations.

That `NowWait` actually has better performance than `OnlyDep` on some distributions suggests that better efficiency still may be attainable using more sophisticated online stochastic combinatorial optimization methods, such as the EXPECTATION algorithm [21]. Better estimates of opportunity costs for `NowWait` may also lead to better performance. But, either extension will require care in coupling with output-ironing. It also remains of interest to study the "first best" solution, i.e. the value and revenue-maximizing policies among monotonic policies, although this seems to us to present a significant technical and computational challenge because monotonicity constraints break the "principle of optimality" that underlies many computational approaches.

Acknowledgements. We would like to thank Quang Duong for his early involvement in this work.

7. REFERENCES

[1] C. Boutilier, D. C. Parkes, T. Sandholm, and W. E. Walsh. Expressive banner ad auctions and model-based online optimization for clearing. In *Proc. 23rd National Conf. on Artificial Intelligence (AAAI)*, pages 30–37, 2008.

[2] V. Conitzer and T. Sandholm. Complexity of mechanism design. In *Proc. 18th Conference on Uncertainty in Artificial Intelligence*, pages 103–110, 2002.

[3] V. Conitzer and T. Sandholm. Incremental mechanism design. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2007.

[4] J. Gallien. Dynamic mechanism design for online commerce. *Operations Research*, 54(2):291–310, 2006.

[5] A. Gershkov and B. Moldovanu. Learning about the future and dynamic efficiency. Forthcoming, Amer. Econ. Review.

[6] A. Gershkov and B. Moldovanu. Dynamic revenue maximization with heterogeneous objects: a mechanism design approach. `www.econ2.uni-bonn.de/gershkov/pdf/dynamicrev.pdf`, February 2008.

[7] J. Gilbert and F. Mosteller. Recognizing the maximum of a sequence. *Journal of the American Statistical Association*, 61(313):35–73, March 1966.

[8] M. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. Parkes. Online auctions with re-usable goods. In *ACM EC*, 2005.

[9] M. Hajiaghayi, R. Kleinberg, and T. Sandholm. Automated online mechanism design and prophet inequalities. In *AAAI 2007*, pages 58–65, 2007.

[10] M. J. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. In *IJCAI*, 1999.

[11] R. Lavi and N. Nisan. Competitive analysis of incentive compatible on-line auctions. In *ACM Conf. on Electronic Commerce (EC)*, pages 233–241, 2000.

[12] R. Lavi and C. Swamy. Truthful and near-optimal mechanism design via programming. In *FOCS*, 2005.

[13] R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, VI:58–73, 1981.

[14] M. Pai and R. Vohra. Optimal dynamic auctions. `http://www.kellogg.northwestern.edu/faculty/vohra/ftp/vohra91.pdf`, March 2008.

[15] D. C. Parkes. Online mechanisms. In N. Nisan, E. Tardos, T. Roughgarden, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge Univ. Press, 2007.

[16] D. C. Parkes. When analysis fails: Heuristic mechanism design via self-correcting procedures. In *Proc. 35th Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, 2009.

[17] D. C. Parkes and Q. Duong. An ironing-based approach to adaptive online mechanism design in single-valued domains. In *AAAI*, 2007.

[18] D. C. Parkes and S. Singh. An MDP-Based approach to Online Mechanism Design. In *Proc. NIPS*, 2003.

[19] D. C. Parkes, S. Singh, and D. Yanovsky. Approximately efficient online mechanism design. In *Proc. 18th Conf. on Neural Information Processing Systems (NIPS)*, 2004.

[20] A. Pavan, I. Segal, and J. Toikka. Dynamic mechanism design: Revenue equivalence, profit maximization and information disclosure. `www.stanford.edu/~isegal/dmd.pdf`, June 2008.

[21] P. van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. MIT Press, 2006.