Reinforcement Learning for Indirect Mechanism Design

A THESIS PRESENTED
BY
DUNCAN RHEINGANS-YOO
TO
THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF ARTS (HONORS)
IN THE SUBJECT OF
COMPUTER SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
APRIL 2020

Reinforcement Learning for Indirect Mechanism Design

Abstract

Incentive mechanisms such as auctions are tools commonly used for resource allocation. Most of the mechanism design literature concerns itself with direct mechanisms, where agents report their preference type. However, in settings where it is infeasible for agents to communicate their full preference type, we require indirect mechanisms equipped with a simpler message space. We introduce and formalize the Generalized Eating Mechanism (GEM), a large parametric class of indirect mechanisms. We also formulate the mechanism design problem as a Markov Decision Process and use reinforcement learning (RL) algorithms to train good mechanisms within a subclass of GEM. Our RL mechanisms are able to achieve optimal or almost optimal performance in static serial dictatorship, dynamic serial dictatorship, and static price settings.

Contents

To M. H. Yoo:
Scholar, Scientist, Teacher, Grandfather.

Acknowledgments

I am beyond grateful for the wealth of support that my network of friends and mentors has provided me on my academic journey.

My longtime advisor David Parkes took me under his wing when I was a clueless freshman and nurtured my interest in research from a spark into a small but steady flame. His infectious enthusiasm and generosity with his time are a large part of why I have come to love academic research and EconCS.

Equally important to my undergraduate journey has been Scott Kominers, who has been invaluable as both academic advisor and personal life coach. Whenever I needed help with some decision, I knew I should go to Scott, and every time he would make himself available. He never told me what he thought the right choice was, instead giving me the tools to think about the problem and determine the right choice for myself.

I want to thank those who have contributed directly to this thesis. My research partner Alon Eden has been by my side throughout; without his cutting insight, this project would not be what it is now. I am also grateful for the helpful input of Zhe Feng, Matthias Gerstgrasser, and Paul Tylkin. A special thanks goes to those who took the time to offer me feedback on my writing—in alphabetical order Gianluca Brero, Alon Eden, Scott Kominers, Charlie O'Mara, David Parkes, Mary Jane Porzenheim, and Penny Rheingans.

I also want to thank those who have supported me in other ways. Vincent James believed in me like no teacher had before, encouraging me to strive for excellence in everything I do. Hongyao Ma showed me how to navigate the nitty gritty of academic research and writing, the things "the adults" don't tell you. My older brother Ross Rheingans-Yoo has served as my inspiration these 22 years, and though I try to catch him, he keeps moving the goalposts. My parents, Penny Rheingans and Terry Yoo, have been my unshakeable foundation. More of my success than anyone will ever know comes from the work you put into our family and into helping me succeed. And finally, I wish to thank my grandparents, Mary Rheingans, the late Robert Rheingans, Sung Ja Yoo, and the late Man Hyong Yoo, for their never-ending love and support.

On two occasions I have been asked, 'If you put into the machine wrong figures, will the right answers come out?' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage

1 Introduction

Much of the world's resources are allocated in public markets where agents exchange goods for money. In some settings however, public markets are an infeasible system of resource allocation. If the number of buyers or the number of items being sold is small, for example when selling a piece of art, it may be difficult for a seller to predict market-clearing prices. In some settings, it may be repugnant to allow transfers of money at all [27]. In public school choice, the idea of allocating slots to those who can pay the most violates the egalitarian principles of public education. The field of mechanism design has developed in part to address the problem of resource allocation in settings where public markets are infeasible.

Mechanism design (MD) is the problem of "inverse game theory," namely how to design the rules of the game to promote a desired outcome. The mechanism designer commits to a set of rules called a mechanism, then agents play the game, inducing some outcome such as an allocation of resources. The single-item auction is a canonical example of a class of mechanisms: agents submit bids for the item, then depending on the bids, the item may be allocated to some agent or none of them, and agents may make some payments.

Different mechanisms have different incentive properties for the participating agents. For example, the Second Price Auction allocates the item to the highest bidder and charges them the *second highest* bid [33]. In a Second Price Auction, it is a dominant strategy for an agent to bid her true value v for the item. If the highest other bid is larger than v, she cannot win the item except by paying more than v for it, so bidding v and losing is the best she can do. If the highest other bid is smaller than v, bidding v lets her win the item; importantly, bidding lower than v will not result in a lower payment unless it means she does not win the

item. By contrast, the First Price Auction allocates the item to the highest bidder and charges them their bid. In a First Price Auction, an agent is incentivized to bid lower than her true value, so that if she win she pays less than her value for the item.

Beyond just auctions, MD has a host of applications including matching, social choice, and bilateral trade. Recent advances in Artificial Intelligence have sparked renewed interest in MD. One of the historical limitations of MD in practice is that it assumes agents are rational, which is not always the case with humans. However, economic transactions are increasingly handled by AI systems; for example, automated trading algorithms execute over 70% of trades on U.S. stock markets [18]. Varian [32] has written that the assumption of rationality may be most reasonable for computerized agents, making MD all the more relevant in a time where AI plays an unprecedented role in the economy. Parkes and Wellman [22] have also discussed the importance and challenges of economic design in the age of AI. However, despite the rich history of and renewed interest in MD, many problems are intractable by current analytic methods. Further, there is minimal understanding of MD in realistic settings including rich preferences and indirect mechanisms where participants do something other than report their preferences.

Much of the MD literature focuses on direct mechanisms, where the space of agent actions is the same as the space of agent preference types. In a single-item auction setting, an agent's preference type is their valuation for the item, so the space of preference types is the positive real line. Thus, a direct mechanism in this setting is one where each agent reports one positive real number as a bid. A direct mechanism is called "incentive compatible" if agents are incentivized to report their type. The Second Price Auction is incentive compatible, while the First Price Auction is not. In fact, the Second Price Auction is the *only* single-item auction that is both efficient (maximal social welfare) and incentive compatible [17].

The revelation principle states that any social choice function implemented by the equilibrium of an arbitrary mechanism can be implemented by an incentive compatible direct mechanism. In other words, absent constraints on computation or communication complexity, any social choice problem that can be solved by a mechanism can be solved by a direct mechanism. This has led researchers to focus the majority of their effort on studying direct mechanisms.

On the other hand, in some settings it is difficult for agents to communicate their full preference type. These domains require mechanisms that are indirect (i.e., mechanisms with simpler message spaces). Consider the combinatorial assignment problem, where the task is to allocate some number of items to some number of agents. In the fully general combinatorial assignment setting, an agent's preference type is their valuation for each subset of items. In the worst case, any sufficiently flexible representation of this type will have exponential size in the number of items. This necessitates the use of a simpler, indirect mechanism. Unfortunately, much less is known about indirect mechanisms than direct mechanisms. One challenge to studying indirect mechanisms is that the notion of incentive compatibility is not well defined because agents are not reporting their preference type. This means that rather than simply imposing incentive compatibility as a constraint on the mechanism space, the mechanism designer must instead study the equilib-

rium play induced by candidate mechanisms.

Our work uses the tools of machine learning to train socially efficient indirect mechanisms in simulated economic settings.

1.1 Automated Mechanism Design

Automated Mechanism Design is the agenda of leveraging algorithms for the design of mechanisms [8]. It stands in contrast to manual mechanism design, where the designer uses analytic methods to find a good mechanism and then proves it has desirable properties.

Recent research has leveraged advances in machine learning for the automated design of direct mechanisms. Dütting et al. [13] use deep neural networks to encode auction mechanisms, with the input being bidder reports and the output being allocation and payments. Fundamentally, an auction is just a function from reports to allocation and payments, and neural networks provide a way of parameterizing this function. The mechanism designer can then optimize the mechanism parameters, the weights of the neural net, for the chosen design objective. Dütting et al. [13] train the auction networks on samples from the value distributions, with the objective of maximizing expected revenue subject to dominant strategy incentive compatibility constraints. They were able to reproduce the optimal auction in some settings where the optimal auction is known, but more importantly they were able to improve on the best-known auctions in settings where current theory has been unable to discover the optimal design.

Subsequent research by Feng et al. [14] extends the neural architecture of Dütting et al. [13] to handle budget-constrained bidders. Tacchetti et al. [30] use neural networks to design efficient, budget-balanced auctions where the payment collected is minimized. Golowich et al. [16] apply neural networks to the multi-facility social choice problem. In each of these cases, the researchers were designing direct mechanisms, which meant they could achieve equilibrium agent behavior by imposing incentive compatibility as a constraint on the optimization of their neural networks. One difficulty of using these methods for automated indirect mechanism design is that incentive compatibility is not well-defined, so it is unclear how to ensure the mechanism is optimized for performance under equilibrium agent behavior. For the purposes of this thesis, we will focus on mechanisms where agents have simple, dominant strategies. We leave work on more complicated settings and mechanisms, along with the equilibria they induce, for future research.

1.2 Contributions

We propose and formalize the *Generalized Eating Mechanism*, a large parametric class of indirect mechanisms that combines elements of *Posted Price Mechanisms* and the *Classical Eating Mechanism* [3]. This parametric characterization enables us to learn, in principle, socially efficient mechanisms within the Generalized Eating class, similar to how Dütting et al. [13] learn revenue-optimal auction designs within their

parameterized auction space.[1] Many well-known indirect mechanisms such as *Random Serial Dictatorship* (RSD) [1], *Ascending Auctions* [21], and the Classical Eating Mechanism exist as particular parameter realizations, but the continuous nature of our parametric class enables us to learn a host of other similar mechanisms. To our knowledge, this is by far the broadest parametric class of indirect mechanisms in the MD literature to date.

We apply reinforcement learning techniques to simulated economic settings involving subclasses of the Generalized Eating Mechanism. Our computational work focuses on two particular subclasses: *Serial Dictatorship* and *Dynamic Price*. To our knowledge, this is the first use of machine learning for indirect mechanism design, at least for something more complex than setting posted prices in a market.[2]

Serial Dictatorship (SD) mechanisms visit agents in some order, allowing each agent to select one item. For our purposes, we allow the order to be updated dynamically at each time step based upon which agents and items are left. For example, the mechanism visits agent 1 first, then depending on which item they select, the mechanism visits agent 2 or agent 3 next. SD mechanisms that update the order dynamically are called *Dynamic Serial Dictatorships*, while those that set a fixed order once and follow it are called *Static Serial Dictatorships*. RSD is a Static Serial Dictatorship where the priority order is randomly determined. For the most part, references in the literature to "Serial Dictatorship" mean Static Serial Dictatorship, but Dynamic Serial Dictatorship is of particular interest to us.

On an agent's turn, it is a dominant strategy for them to select their favorite remaining item. SD mechanisms, along with other mechanisms with simple dominant strategy equilibria, can in theory be implemented as direct mechanisms. For SD, each agent submits an ordinal ranking of the items, then the mechanism visits the agents in the same order; on each agent's turn, the mechanism assigns them the highest remaining item in their submitted ranking. However, these mechanisms can also be implemented as indirect mechanisms, and in some situations it is this very indirectness that matters. For example, it is computationally easier for agents to select their favorite item from a set than it is to produce an ordinal ranking of the items. The point of this research is to show that reinforcement learning can be used for indirect mechanism design because in some settings direct mechanisms are infeasible.

The first setting we test is a Serial Dictatorship setting where there is persistent asymmetry between agents' valuations. The learning task is to learn a good policy for picking the next agent to visit, and the optimal policy visits agents earlier in the order if they have higher expected valuations. In this setting, optimal performance can be achieved with a Static Serial Dictatorship. Our reinforcement learning mechanism is able to achieve $> 99\%$ of the optimal expected social welfare, significantly outperforming our implementation of RSD, which achieves $\approx 52\%$ of optimal.

[1] We chose efficiency as our objective, but our methods would also apply to the design of revenue-optimal indirect mechanisms.

[2] It is common practice for suppliers to use regression models to predict price elasticity of demand in order to set better prices for their products. However, to our knowledge, machine learning has not been applied to problems more complex than this, and reinforcement learning has not been applied at all to indirect mechanism design.

We also test a Serial Dictatorship setting where the optimal policy must be dynamic. There are two types of items: red and yellow, with n items of each type. There are three types of agents: red, yellow, and blue, with n agents of each type. Each blue agent has a high valuation for either red or yellow items but not both. Each red agent has a medium valuation for red items and low valuation for yellow items. Yellow agents are the reverse of red agents, with medium valuations for yellow items and low valuations for red items. The optimal policy is to visit the blue agents first because they have the highest valuations, then to visit red and yellow agents such that every remaining item ends up with an agent of its color. Unlike the previous setting, this requires a policy that will act differently depending on how many red vs. yellow items are left after the blue agents have gone. Our reinforcement learning mechanism is able to learn this optimal dynamic behavior, achieving 100% of the optimal expected social welfare. By comparison, the best static serial dictatorship achieves $\approx 89\%$ of the optimal welfare and RSD achieves $\approx 75\%$.

A Dynamic Price Mechanism also visits agents in a serial manner, but it can also require that an agent pay some price to take an item. These prices are allowed to be dynamic, i.e. responsive to which agents and items remain. We study a setting where agents independently either have a specific high or specific low valuation for items. Agent value distributions are symmetric, so order cannot be used to outperform RSD. Instead, the optimal mechanism sets prices between the agents' high valuation and the low valuation, so that an agent only buys the item if they have a high valuation. Our reinforcement learning mechanism is able to learn these optimal prices, achieving 100% of the optimal expected social welfare, compared to $\approx 66\%$ for our implementation of RSD.

1.3 Related Work

The idea of Automated Mechanism Design was originally formalized by Conitzer and Sandholm [8, 9]. Early work by Cai et al. [5–7] resulted in polynomial time algorithms for the construction of Bayesian incentive compatible optimal auctions. The first use of machine learning for auction design was done by Dütting et al. [12], but they only dealt with learning payment rules, not allocation rules. The previously discussed work by Dütting et al. [13] was the first to use deep neural nets to learn dominant strategy incentive compatible optimal auctions, spurring the previously discussed subsequent research by Feng et al. [14], Tacchetti et al. [30], and Golowich et al. [16].

Shen et al. [28] use neural networks for the design of revenue optimal auctions in a single-bidder setting. Unlike other work on automated auction design, which encode auctions as neural networks and enforce incentive compatibility as a constraint on the optimization, Shen et al. [28] use a neural network to output a set of menu items, which encodes a mechanism. Simultaneously, they train a "buyer network" which maps these menu items to buyer strategies. This approach allows them to simulate different types of buyers, not just economically rational ones. Shen et al. [28] only study a single-bidder setting, so it is unclear how well buyer networks can scale, but the underlying idea offers a promising direction for future work studying the

equilibria of indirect mechanisms.

1.4 Broader Context

This thesis is part of a broader project studying the automated design of incentive mechanisms in dynamic environments where agents may have varying levels of sophistication (e.g., a mix of human and AI actors). Our project leverages recent work by Tylkin et al. [31] developing a research platform around Atari 2600 games. The platform has been modified, making the games multiplayer and allowing for resource constraints, a mix of human and AI agents, and elements of both cooperation and competition. Originally, the platform was developed to study the design of Helper-AIs, AI agents that can assist the player, for example by shooting aliens in Space Invaders. However, the platform is also well-suited for the study of incentive mechanisms.

We envision a mechanism design loop that leverages crowdsourcing and imitation learning to model human behavior:

1. Humans play a game such as Space Invaders, modified to include some incentive mechanism. For example, the game could be linked with another player's game, with a limited supply of bullets between the two games and some mechanism for distributing them.

2. We collect trajectories of human play, including game play and interaction with the mechanism, and use these trajectories to train imitation learning agents, AIs who emulate the behavior of the humans in the data set.

3. We use machine learning to learn a mechanism optimized for the behavior exhibited by the imitation learned AIs, as well as possibly some very skilled AIs.

4. We roll out this mechanism to the human players, and the cycle repeats.

This thesis is concerned with the third step of the loop, leveraging machine learning to design mechanisms. We do not yet have imitation learning agents, so for this thesis we eschew some of the more involved elements such as dynamic environments and varying levels of sophistication among agents. We focus instead on simple environments that illustrate the potential of using machine learning for indirect mechanism design.

1.5 Outline

The remainder of the thesis is structured as follows. In Chapter 2, we examine a number of indirect mechanisms that exist in the literature. In Chapter 3, we present the Generalized Eating Mechanism, which

generalizes those mechanisms and contains many more. In Chapter 4, we present our reinforcement learning methods. In Chapter 5, we present the results of our reinforcement learning tests. In Chapter 6, we provide some directions for future work.

Somebody has to, and no one else will.

The Comet King–*Unsong*

2

Examples of Indirect Mechanisms

In this chapter, we present five indirect mechanisms whose features form the basis of the Generalized Eating Mechanism. We operate in a combinatorial assignment setting where the mechanism designer has m items to allocate to n agents. In this setting, a mechanism is some procedure that interacts with the agents to produce some allocation of items to agents. Let X be the set of allocations, then each agent i has a cardinal value function $u_i : X \to \mathbb{R}$, so agent i's value for allocation $x \in X$ is $u_i(x)$. Each u_i is drawn randomly from a distribution known to the mechanism designer, but the specific realization is known only to agent i.

Agents are economically rational, which means that they act to maximize their expected value minus their expected payment (if any). If agent strategies are such that no single agent can profitably change their strategy, we say the game is in *equilibrium*. The mechanism designer aims to optimize some objective of the equilibrium allocation; we will consider the objective to be social welfare (total value), though in other settings revenue (total payment) might make sense as the objective.

To illustrate how each mechanism works, we will explain how it runs on a small setting. There are three agents—Alice, Bob, and Charlie—and three items—1, 2, and 3. The bidders have additive valuations, which means they have some value for each item and their value for a set of items is the sum of the values for each of the items in the set. Let Alice have realized values $(15, 10, 5)$ for items 1-3 respectively, and let Bob and Charlie have values $(2, 6, 4)$. The agent behavior will illustrate the functionality of the mechanism and not necessarily the equilibrium strategies.

2.1 Serial Dictatorship

Serial Dictatorship (SD) is a class of mechanisms that visit agents in some order, letting each agent take one item on their turn. As described in Section **??**, some SD mechanisms are static; they set a fixed order once and follow it. Others are dynamic; they can update the order as they go depending on which agents and items remain.

Example 2.1.1 (Serial Dictatorship)**.** A Serial Dictatorship mechanism has the following structure:

- In each round $t > 0$, the mechanism visits an agent it has not visited before. The decision of which agent to visit depends on the specifics of the SD mechanism.
- When an agent is visited, they are allowed to select one item. If no agents or items remain, terminate.

SD mechanisms have simple, truthful strategies; on an agent's turn, it is a dominant strategy for them to pick their favorite remaining item. The most common implementation of an SD is with a static random priority order:

Definition 2.1.1 (Random Serial Dictatorship)**.** Assign items in the following manner:

- In round 0, choose a random priority order over agents.
- In rounds $t > 0$, the mechanism visits the agent in t^{th} position in the priority order. This agent is allowed to select one item. If no items or agents remain, terminate.

Example 2.1.2 (Random Serial Dictatorship)**.** Alice's values are $(15, 10, 5)$ and Bob/Charlie's are $(2, 6, 4)$. The priority is randomly determined to be Bob first, then Alice, then Charlie. Bob selects item 2, then Alice selects item 1, then Charlie selects item 3.

Random Serial Dictatorship (RSD) is *symmetric* in that agents who have the same valuations for items obtain the same expected value. It also does not involve transfers, which makes it desirable for assignment settings without money.[1] However, RSD is not ordinally efficient; in fact, there does not exist a truthful, symmetric, ordinally efficient mechanism.[2] [35]

2.2 Single-Bid Auction

One drawback of serial dictatorship mechanisms is that they do not incorporate information about agents' valuations. While a lack of reporting makes agent strategies simpler, it can negatively affect the efficiency of the mechanism.

[1] Harvard College employs a variant of RSD to allocate slots in its General Education Courses

[2] A random allocation X is said to ordinally dominate another Y if for every agent a and item i, the probability a receives an item they weakly prefer to i is weakly higher under X than Y, and strictly higher for some agent a' and item i'. A mechanism is ordinally efficient if the random allocations it produces are not ordinally dominated.

Take as an example a setting where agents have a high probability of having low valuations for all items and a low probability of having high valuations for all items. A mechanism designer whose objective is social welfare would prefer to put the agents with high valuations early in the order and those with low valuations late in the order. Any serial dictatorship mechanism, on the other hand, simply assigns the priority order agnostic of agent valuations. Thus, we desire some limited reporting mechanism where agents can communicate some summary of their valuations, for example a single real number.

However, agents are self-interested, so without payments they are always incentivized to represent that they have high valuations. Devanur et al.'s [11] *Single-Bid Auction* addresses this concern by making agents pay for items and coupling an agent's report with the prices they face:

Definition 2.2.1 (Single-Bid Auction)**.** Assign items in the following manner:

- In round 0, each agent i submits a bid $r_i \in \mathbb{R}$.
- In rounds $t > 0$, the mechanism visits the agent with the t-highest bid. This agent i is allowed to purchase any number of remaining items, each for price r_i.

Example 2.2.1 (Single-Bid Auction)**.** Alice's values are $(15, 10, 5)$ and Bob/Charlie's are $(2, 6, 4)$. Alice bids 8, Bob bids 5, and Charlie bids 3. Alice has the highest bid, so she goes first and chooses to buy items 1 and 2 for 8 each, paying a total of 16 for a set with value 25. Bob goes next, but his value for item 3 is 4 and he would have to pay 5, so he does not buy it. Charlie goes next and buys item 3, paying 3 for an item he values at 4.

The optimal bidding strategy for an agent is not obvious, but the strategy space is small enough that agents can efficiently learn their optimal strategy in a repeated version of the game. Previous work studying the Single-Bid Auction has shown that it achieves a $\frac{\log m}{m}$ approximation of the social welfare (m items), and this cannot be improved by setting different prices for different items, as long as prices only depend on the initial bids [4]. It remains an open question whether making prices dynamic, i.e. responsive to the purchasing decisions of agents, can improve performance.

2.3 Ascending Auction

While the Single-Bid Auction uses one initial report by each agent to set prices, other mechanisms update prices over time. One such class of mechanisms is the Ascending Auction class. Ascending Auctions exist in many forms, but they universally involve monotonically non-decreasing item prices and typically involve agent actions in each time step. Ascending Auctions have been used in a variety of single-item and combinatorial settings, most notably to sell radio spectrum licenses in the United States [21]. We present a simple version of a multi-item ascending auction:

Definition 2.3.1 (Ascending Auction)**.** Assign items in the following manner:

- In round 0, set the price of each item to be 0.
- In rounds $t > 0$, each agent can point at one item or no items. If no items have more than one agent pointing to it, every agent purchases the item they are pointing to at its current price, and the mechanism terminates. Otherwise, for each item with multiple agents pointing to it, increment its price by a small amount ε.

In contrast to the Single-Bid Auction, in which agents only take one action in round 0, agents take actions in each round of the ascending auction.

Example 2.3.1 (Ascending Auction)**.** Let $\varepsilon = 1$. Alice's values are $(15, 10, 5)$ and Bob/Charlie's are $(2, 6, 4)$. Prices start at $(0, 0, 0)$ for items 1-3 respectively. The rounds proceed as follows:

1. Alice points at item 1, Bob and Charlie point at item 2. Prices change to $(0, 1, 0)$.
2. Alice points at item 1, Bob and Charlie point at item 2. Prices change to $(0, 2, 0)$.
3. Alice points at item 1, Bob points at 2, and Charlie points at 3. They each purchase their items for prices 0, 2, and 0 respectively.

In the last round, Bob and Charlie prefer item 2 to item 3 by an amount of 2, but the price for item 2 is 2 higher than the price for item 3, so they are indifferent between the items.

2.4 Descending Auction

Descending Auctions are the natural complement to Ascending Auctions, with monotone non-increasing (rather than non-decreasing) item prices. Descending Auctions are used in many settings, from the sale of flowers at the Aalsmeer Flower Auction to the sale of U.S. Treasury bills [21]. There are many variants of descending auctions; we present a slightly non-standard one for a multi-item setting:

Definition 2.4.1 (Descending Auction)**.** Assign items in the following manner:

- In round 0, set prices large enough such that no agents want to purchase items.
- In rounds $t > 0$, each agent can choose to buy or not. If no agents buy, decrement the price of every item by ε. If one agent buys, let them buy any number of items, each for their current prices. This agent can no longer buy in future periods. If more than one agents buy, break the tie randomly.

Example 2.4.1 (Descending Auction)**.** Let $\varepsilon = 1$. Alice's values are $(15, 10, 5)$ and Bob/Charlie's are $(2, 6, 4)$. Prices start at $(15, 15, 15)$ for items 1-3 respectively. The rounds proceed as follows:

- Rounds 1-6: No agents choose to buy, and each of the prices decrement by 1 each round until they are $(9, 9, 9)$ entering round 7.
- Round 7: Alice chooses to buy, and buys items 1 and 2 for 9 each. The price for item 3 remains 9.
- Rounds 8-13: No agents choose to buy, and the price of item 3 decrements by 1 each round until it is 3 entering round 14.
- Round 14: Bob and Charlie both choose to buy, and the tie is randomly broken in favor of Charlie. He buys item 3 for 3.

2.5 Classical Eating Mechanism

The Classical Eating Mechanism (or Probabilistic Serial Mechanism) is a procedure first proposed by Bogolmonaia and Moulin [3] that produces a probabilistic allocation of items. The mechanism creates a "loaf" of bread for each item and assigns an "eating speed" to each agent. Agents eat continuously from the loaves of their favorite items; when a loaf is fully consumed, the agents eating it move on to their favorite remaining item. The mechanism terminates when no loaves remain. Then, if an agent consumed a proportion q of a loaf, he is assigned the corresponding item with probability q. Typically, the Classical Eating Mechanism is implemented by querying agents for an ordinal ranking of items in order to execute the eating. We present a slightly modified version here, where agents make eating decisions dynamically, rather than by submitting a ranking beforehand.

Definition 2.5.1 (Classical Eating Mechanism)**.** Assign items in the following manner:

- In rounds $t > 0$, assign each agent an eating speed. Each agent chooses an item. Agents consume probability mass of receiving their chosen items at their given eating speeds. The round ends when an item runs out of probability mass, and that item is removed.
- When all items are removed, for each (agent, item) pairing (i, j), the probability agent i is assigned item j is the probability mass of item j that agent i consumed.

Example 2.5.1 (Classical Eating Mechanism)**.** Alice's values are $(15, 10, 5)$ and Bob/Charlie's are $(2, 6, 4)$. Each agent is given eating speed 1. The rounds proceed as follows:

1. Alice eats from loaf 1, while Bob and Charlie eat from loaf 2. The agents eat at the same speed, so when Bob and Charlie have finished consuming loaf 2, Alice is halfway through loaf 1. Alice gets $\frac{1}{2}$ the probability mass of receiving item 1, while Bob and Charlie each get $\frac{1}{2}$ the probability mass of receiving item 2.

2. Alice eats from loaf 1, while Bob and Charlie eat from loaf 3. The agents eat at the same speed, so Alice finished consuming the remaining half of loaf 1 as Bob and Charlie consume all of loaf 3. Alice gets $\frac{1}{2}$ the probability mass of receiving item 1, while Bob and Charlie each get $\frac{1}{2}$ the probability mass of receiving item 3. All loaves are consumed, so the eating terminates.

Alice consumed all of loaf 1, so she gets item 1 deterministically. Bob and Charlie each consumed half of loaves 2 and 3, so each of them owns a $\frac{1}{2}$ chance at each item.

An important parameter of the Classical Eating Mechanism is the eating speed. Agents are allowed to have different eating speeds, and these can even change dynamically. Unlike RSD, the Classical Eating Mechanism is ordinally efficient. When agents have equal eating speeds, it is also envy-free, which means that no agent desires to swap their random allocation with another agent.[3] However, it is not truthful. For example, if agent *i*'s favorite item is all other agents' least favorite item, agent *i* might not eat that item first, in order to spend their time in the early rounds consuming items that all the agents desire and thus are scarce.

[3] Envy-freeness implies symmetry

Do what I do. Hold tight and pretend it's a plan.

The Doctor–*Doctor Who*

3

General Framework

We now build towards formalizing our parametric class: the Generalized Eating Mechanism (GEM). The purpose of having such a parametric class is that we can use the methods of machine learning to optimize over mechanisms in the class. We operate in the combinatorial assignment setting of Chapter 2 where we have m items that we wish to assign to n agents. Each agent's cardinal value function depends only on what items they are allocated. We introduce a fair bit of notation in this chapter, so we include a reference table in Figure 3.0.1.

In round 0 of the mechanism, each agent makes some limited report and the mechanism creates a loaf for each item. In each subsequent round, each agent points at one item or no items. The mechanism then determines eating speeds and prices for each (agent, item) pairing and budgets for each agent. Agents each choose a subset of items to eat from and then eat from those loaves at their assigned speeds. Agents pay the mechanism in proportion to the prices and amounts of the loaves they consume. Agents are only allowed to choose sets of items for which they can consume what remains of those loaves and stay under budget. A round ends when a loaf is consumed. When all loaves have been consumed, no agents remain, or prices have been constant across consecutive periods with no consumption by agents, the mechanism terminates. As in the Classical Eating Mechanism, items are randomly assigned to agents based upon the proportion of the corresponding loaves the agents consumed.

In this chapter, we will formalize GEM, along with a couple of natural subclasses, and show that all of

Symbol	Meaning
n	Number of agents
m	Number of items
r	Agent report vector
$c^{(t)}$	Agent action vector in round t
$v^{(t)}$	Speed matrix in round t
$p^{(t)}$	Price matrix in round t
$b^{(t)}$	Budget vector in round t
$x^{(t)}$	Partial allocation matrix in round t
$a^{(t)}$	Agents remaining vector in round t
$s^{(t)}$	Mechanism state in round t
$h^{(t)}$	Mechanism history in round t

Figure 3.0.1: Glossary of notation

the indirect mechanisms from Chapter 2 exist as special cases of GEM.

3.1 Mechanism State and History

At each step t of the mechanism, we keep track of the following aspects of the state of the mechanism:

- Initial agent reports $r \in \mathbb{R}^n$
- The partial random allocation $x^{(t)} = [0,1]^{n \times m}$
- The remaining agents $a^{(t)} \in \{0,1\}^n$, where $a_i = 1$ iff. agent i remains

We define the *state* of the mechanism at time t as $s^{(t)} = (r, x^{(t)}, a^{(t)})$, and thus the space of possible states is $S := \mathbb{R}^m \times [0,1]^{n \times m} \times \{0,1\}^n$. We also keep track of the following aspects of the history of the mechanism:

- The price matrix from the previous round $p^{(t-1)} \in \mathbb{R}^{n \times m}$
- The agent actions in the current round $c^{(t)} \in [m+1]^n$

We define the *history* of the mechanism at time t as $h^{(t)} = (p^{(t-1)}, c^{(t)})$, and thus the space of possible histories is $H := \mathbb{R}^{n \times m} \times [m+1]^n$

3.2 Generalized Eating Mechanism

Definition 3.2.1 (Generalized Eating Mechanism)**.** Let $speed : S \times H \to \mathbb{R}^{n \times m} \times \{0,1\}^n$, $price : S \times H \to \mathbb{R}^{n \times m}$, and let $budget : S \times H \to \mathbb{R}^n$. Then $(speed, price, budget)$ defines a *Generalized Eating Mechanism* where:

- Each agent submits a report $r_i \in \mathbb{R}$
- The initial state is $s^{(1)} = (r, \{1\}^n, \{0\}^{n \times m})$, and in round t is set as $s^{(t)} = (r, a^{(t)}, x^{(t)})$. Prices are initialized to some matrix $p^{(0)}$.
- In rounds $t > 0$:
    1. Each agent takes in-round action $c_i^{(t)} \in [m + 1]$.
    2. The history is defined as $h^{(t)} = (p^{(t-1)}, c^{(t)})$.
    3. Speeds for this round and agents left for the next round are computed as $(v^{(t)}, a^{(t+1)}) = \mathit{speed}(s^{(t)}, h^{(t)})$.
    4. Prices for each agent are computed as $p^{(t)} = \mathit{price}(s^{(t)}, h^{(t)})$. Budgets for each agent are computed as $b^{(t)} = \mathit{budget}(s^{(t)}, h^{(t)})$.
    5. Each agent i chooses a set of items to consume. If each agent only chooses items for which they have zero eating speed, proceed to the next round. Otherwise, all agents eat from the items in their set at their assigned speeds. The round ends when an item is consumed.
    6. Agents are charged their price for the item times the proportion of the item they consume. This means an agent is only allowed to choose a set of items if they could consume what remains of those items and stay under budget. The partial allocation $x^{(t+1)}$ is updated to reflect the partial/full consumptions. Each fully consumed item is assigned randomly to an agent based on the partial allocation vector.
- If in round t all items have been consumed, no agents remain, or $p^{(t)} = p^{(t-1)}$ and no items are consumed in round t, the mechanism terminates.

An agent i's initial report r_i is just a single real number, as in the Single-Bid Auction. This allows the agent to communicate some summary of their preferences but nothing near their whole preference type. Unlike the Single-Bid Auction, the prices agent i faces are not necessarily r_i for each item, although they could be. Some of the mechanisms in GEM, such as Random Serial Dictatorship and the Ascending Auction, ignore the reports entirely.

An agent i's in-round action $c_i^{(t)}$ is to point at a single item or no items. This allows agents to communicate something about their preferences given the state of the mechanism, for example a preferred remaining item. Next, the mechanism computes speeds, prices, and budgets. Then, the agents make a second action which is choosing a set of items to consume. This means an agent strategy has three components: initial reporting, in-round pointing, and in-round consumption.

A round ends when an item is fully consumed. Agents pay in proportion to the amounts of each item they consumed, as illustrated below:

Example 3.2.1 (GEM)**.** There are two items and two agents: Alice and Bob. For now, we leave aside reports, in-round actions, and budgets. In round 1, Alice is given eating speed 2 for both items and price 2 for both items. Bob is given eating speed 1 for both items and price 1 for both items. Alice chooses to eat from only item 1, and Bob chooses to eat from both items. Because item 1 is being consumed faster than 2, the round ends when 1 is fully consumed and 2 is only partially consumed. $\frac{2}{3}$ of item 1 is consumed by Alice, $\frac{1}{3}$ of item 1 is consumed by Bob, and $\frac{1}{3}$ of item 2 is consumed by Bob. Alice pays $2 \times \frac{2}{3} = \frac{4}{3}$ and Bob pays $1 \times \frac{1}{3} + 1 \times \frac{1}{3} = \frac{2}{3}$. Item 1 is fully consumed and assigned with $\frac{2}{3}$ probability to Alice and $\frac{1}{3}$ probability to Bob. $\frac{2}{3}$ of item 2 remains for the next round, with the other $\frac{1}{3}$ belonging to Bob.

We only allow an agent to consume a set of items if they could consume all of what remains of those items and stay under budget. Budgets are useful primarily in settings without transfers; if the payments are made with fake money, budgets are necessary to stop agents from eating from all the items. Even in settings with transfers, it might be undesirable for an agent to consume every item, even if they are willing to pay for it. Another way of implementing budgets is instead of constraining agent consumption choices, we simply end the round when an agent runs out of budget. However, this alternate formulation suffers from an inability to force a deterministic allocation, which is necessary to produce many of the mechanisms we are interested in.

For learning purposes, the *speed*, *price*, and *budget* functions can be implemented by neural networks. The parameters of the mechanism are then the weights of those neural networks, which can be learned via reinforcement learning algorithms.

Figure 3.2.1 illustrates how GEM functions. The red boxes denote agent decisions, the blue boxes denote mechanism decisions, and the yellow circle denotes the mechanism state and history. The agents make their initial reports r and then their first round actions $c^{(1)}$. This determines the initial state-history tuple $(s^{(1)}, h^{(1)})$, which feeds into the *speed*, *price*, and *budget* functions. The outputs of these functions, together with $(s^{(1)}, h^{(1)})$, feed into the agent consumption. The output of the agent consumption combines with the next round's agent action $c^{(2)}$ and the initial reports r to form the next state-history tuple $(s^{(2)}, h^{(2)})$, and the loop repeats.

The *speed* function has two components: one which determines eating speeds $v^{(t)}$, and one which determines which agents remain for the next round $a^{(t+1)}$. Some mechanisms within GEM remove an agent once they have a round with positive eating speed; some never remove agents. Other mechanisms might have some other criteria for keeping or removing agents.

3.3 GEM Examples and Subclasses

The GEM family has a number of natural subclasses, as illustrated in Figure 3.3.1:

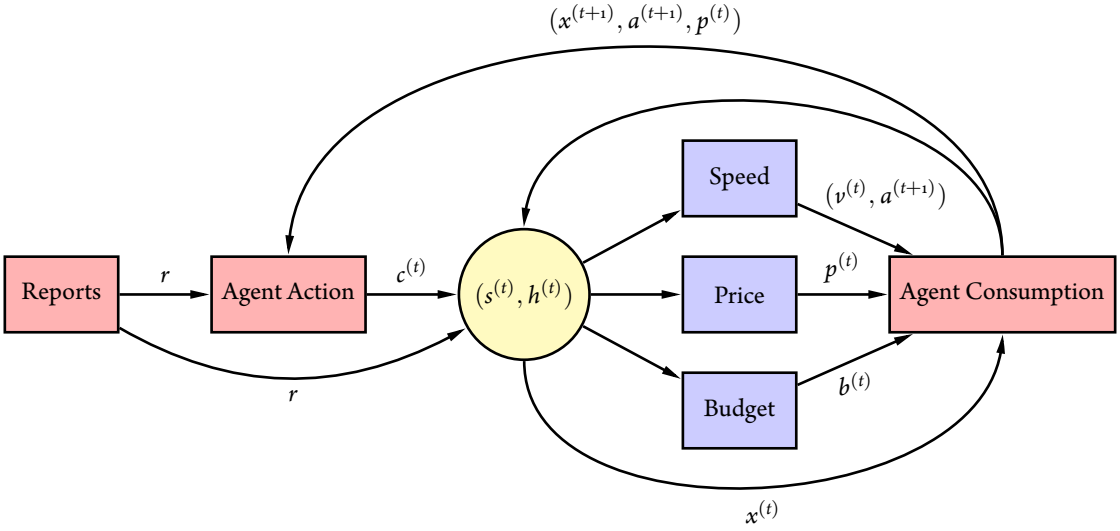


Figure 3.2.1: GEM Architecture

- SD–Serial Dictatorship. The mechanism visits agents in some (possibly dynamic) order, letting each agent take on item on their turn.
- SPP–*Static Posted Price*. The mechanism sets a static order and prices, then visits the agents in that order, letting an agent buy some number of items on their turn for the set prices.
- The intersection of SPP and SD is the family of Static Serial Dictatorships, to which Random Serial Dictatorship (RSD) belongs. The Single-Bid Auction (SBA) is an example of an SPP mechanism that is not a Serial Dictatorship. There also exist Dynamic Serial Dictatorships, which can update the order based upon the remaining items and agents.
- IDPP–*Interactive Dynamic Posted Price*, the natural generalization of SPP to have dynamic order and prices. The Descending Price Action is an example of IDPP that is not captured by SPP or SD.
- MDP–*Markov Decision Process*. Characterizing a subclass of IDPP as an MDP allows for the use of reinforcement learning algorithms for the efficient learning of good mechanisms. This is the subject of Chapter 4.
- GEM–the Generalized Eating Mechanism itself. The Classical Eating Mechanism and Ascending Price Auction belong to GEM but not IDPP.

The remainder of this chapter is dedicated to formalizing the IDPP and SPP subclasses, as well as showing that each of the mechanisms from Chapter 2 is a special case of GEM. First and foremost, the Classical

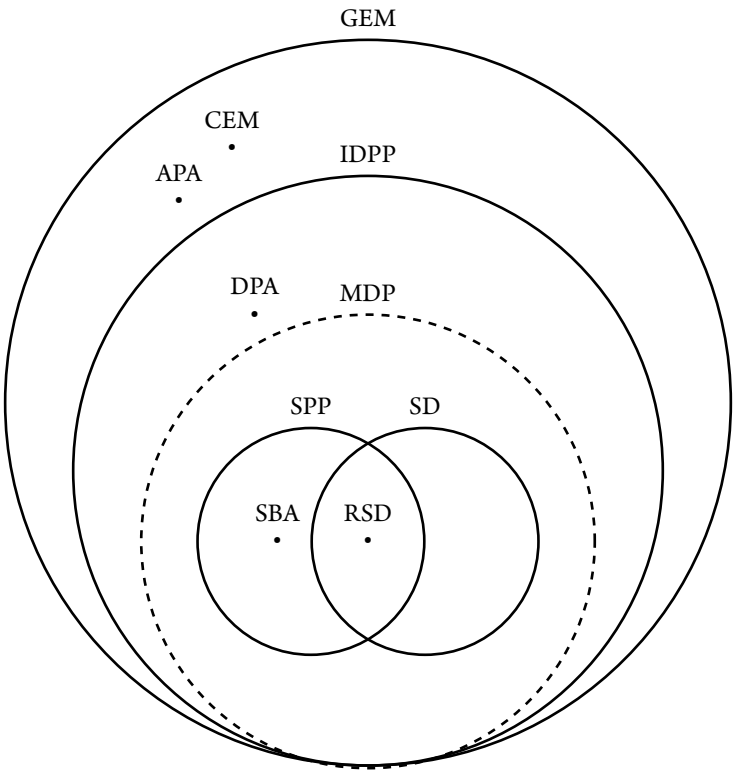


Figure 3.3.1: Subclasses of the Generalized Eating Mechanism

Eating Mechanism is recovered by ignoring the initial reports, and allowing an agent to eat only from the item they point to in a particular round.

Example 3.3.1 (Classical Eating Mechanism as GEM)**.** The Classical Eating Mechanism is a GEM s.t.:

- For $j = c_i^{(t)}$ we have $v_{ij}^{(t)} = 1$ and $v_{ij'}^{(t)} = 0$ for $j' \neq j$.
- $a^{(t+1)} = \{1\}^n$
- $p^{(t)} = \{0\}^{n \times m}$
- $b^{(t)} = \{0\}^{n \times m}$

We can also recover the Ascending Auction as follows. If in any period all agents are pointing at different items, let agents have eating speed 1 for the item they are pointing to and speed 0 for other items. Otherwise, increment the price for items which multiple agents are pointing to.

Example 3.3.2 (Ascending Auction as GEM)**.** The Ascending Auction is a GEM s.t.:

- If there exists item j and agents k, l s.t. $c_k^{(t)} = c_l^{(t)} = j$, then:
  - $v^{(t)} = \{0\}^{n \times m}$

- $a^{(t+1)} = \{1\}^n$
- $p_{ij}^{(t)} = p_{ij}^{(t-1)} + \varepsilon \; \forall i$ for each item j s.t. $\exists k, l \; s.t. \; c_k^{(t)} = c_l^{(t)} = j$
- $p_{ij}^{(t)} = p_{ij}^{(t-1)} \; \forall i$ for all other items j

- Otherwise:
  - $v_{ij}^{(t)} = 1$ if $c_i^{(t)} = j$ and 0 otherwise.
  - $a^{(t)} = \{0\}^n$
  - $p^{(t)} = p^{(t-1)}$
- $b^{(t)} = \{\infty\}^n$

The first case corresponds to two agents pointing to the same item. In that case, no agents get to eat ($v^{(t)} = \{0\}^{n \times m}$), and all agents remain ($a^{(t+1)} = \{1\}^n$). We increment the price ($p_{ij}^{(t)} = p_{ij}^{(t-1)} + \varepsilon$) for all items j with multiple people pointing to them. This price goes up for all agents, not just the ones pointing. We keep the other prices constant.

The second case corresponds to all agents pointing to different items or no item. In that case, each agent gets eating speed 1 for the item they're pointing at ($v_{ij}^{(t)} = 1$ if $c_i^{(t)} = j$ and 0 otherwise), all agents are removed ($a^{(t)} = \{0\}^n$), and prices are the same as the previous round ($p^{(t)} = p^{(t+1)}$). Agents have eating speeds 1 or 0 for items, and each item has at most one agent who can eat from it. This means an agent consumes all or none of an item, and the items are assigned deterministically at the end of the round. Budgets are unconstrained.

Ascending Auctions in practice often have "activity rules" which constrain what actions an agent can take. For example, if in round t an agent says they prefer item 1 to item 2, then in round $t+1$ the price of item 2 increases relative to item 1, the agent is not allowed to say they prefer item 2 to item 1. GEM does not currently learn any rules for constraining the agent pointing action $c^{(t)}$. On the other hand, GEM's *speed* function can implement some state-based activity rules that deal with consumption: for example, if an agent has consumed more than half of an item, they cannot consume any more of it. However, GEM does not track agents' past actions except as reflected in the partial allocation, so parameterizing history-intensive activity rules would require adjusting the class somewhat.

3.3.1 Interactive Dynamic Posted Price Mechanisms

The *speed* function of the GEM can implement a turn-based mechanism by allowing only one agent at a time to have nonzero eating speed. This defines a natural subclass of GEM where only one agent can consume in a round and any consumption of items will result in purchasing those items with certainty. We call this the Interactive Dynamic Posted Price (IDPP) class:

Definition 3.3.1 (IDPP as GEM)**.** Let M be a GEM. We say that M is an *Interactive Dynamic Posted Price Mechanism* if:

- At most one agent at a time has eating speed one for all items, and the rest have eating speed zero: $v_i^{(t)} = \{1\}^m$ for some i and $v_j^{(t)} = \{0\}^m$ for all $j \neq i$
- An agent assigned eating speed one is removed from the remaining agents at the end of the round.

By having at most one agent with eating speed $\{1\}^m$ and the rest with $\{0\}^m$ we implement "turns" by ensuring that one agent can consume any number of items with certainty in any round. Removing that agent at the end of the round simplifies their consumption strategy space; it is a dominant strategy to consume their favorite set given the prices they face. If agents were allowed to remain, they might strategically not consume some item in order to get it at a better price in the future.

Example 3.3.3 (Descending Price Auction as IDPP)**.** The Descending Price Auction is an IDPP s.t.:

- $p^{(0)}$ is set high enough that no agents want to buy any items
- If some agent i takes action $c_i^{(t)} \neq m + 1$, i.e. not the no-item action:
  - $v_i^{(t)} = \{1\}^m$ and $v_{i'} = \{0\}^m$ for all agents $i' \neq i$
  - $p^{(t)} = p^{(t-1)}$
  - If multiple such agents, ties broken randomly
- Otherwise:
  - $v^{(t)} = \{0\}^{n \times m}$
  - $a^{(t+1)} = a^{(t)}$
  - $p^{(t)} = p^{(t-1)} - \{\varepsilon\}^{n \times m}$
- $b^{(t)} = \{\infty\}^n$

The first case corresponds to some agent i choosing to buy, i.e. not choosing the no-item action. In that case, it is this agent's "turn" ($v_i^{(t)} = \{1\}^m$), and we freeze prices ($p^{(t)} = p^{(t-1)}$). We then remove that agent (all IDPP do this). The second case corresponds to all agents choosing the no-item action. In that case, we set all speeds to zero ($v = \{0\}^{n \times m}$), all agents remain ($a^{(t+1)} = a^{(t)}$), and prices decrement for all items and agents $p^{(t)} = p^{(t-1)} - \{\varepsilon\}^{n \times m}$. Budgets are unconstrained.

3.3.2 Static Posted Price Mechanisms

Restricting *speed*, *price*, and *budget* further results in subclasses of IDPP (and thus of GEM). One natural such restriction is to make speed, prices, and budgets only dependent on initial agent reports, ignoring history and partial allocation. This yields the Static Posted Price (SPP) class:

Definition 3.3.2 (SPP as IDPP)**.** Let M be an IDPP mechanism. Denote $(s,h)_{-r}$ as the parts of a (state, history) tuple (s,h) that are not the initial agent reports r. We say M is a *Static Posted Price Mechanism* if there exists some function $f : \mathbb{R}^{n\times m} \to \mathbb{R}^n$ from agent reports to agent scores s.t.:

- $v_i^{(t)} = \{1\}^m$ iff. $i \in \arg\max_{i'} f(r)_{i'} a_{i'}^{(t)}$ (if multiple, ties broken randomly)
- $price(r,(s,h)_{-r}) = price(r,(s,h)'_{-r})$ for any $(s,h)_{-r}, (s,h)'_{-r}$
- $budget(r,(s,h)_{-r}) = budget(r,(s,h)'_{-r})$ for any $(s,h)_{-r}, (s,h)'_{-r}$

The first bullet means that an agent is visited ($v_i^{(t)} = \{1\}^m$) iff. they have the highest score of remaining agents ($i \in \arg\max_{i'} f(r)_{i'} a_{i'}^{(t)}$). Ties are broken randomly. The second and third bullets mean that the functions *price* and *budget* ignore everything about (s,h) except for the agent reports r. So, the scores that f outputs induce an order over agents, which the mechanism will visit sequentially regardless of the purchasing decisions of agents. For the Single-Bid Auction, the scores are just the reports, so the mechanism visits agents in decreasing order of this "bid".

Example 3.3.4 (Single-Bid Auction as SPP)**.** The Single-Bid Auction is an SPP s.t.:

- $f(r)_i = r_i$
- $p_{ij}^{(t)} = r_i$
- $b^{(t)} = \{\infty\}^n$

RSD is another example of an SPP, where all agents have equal scores and thus the same probability of going next at all times.

Example 3.3.5 (RSD as SPP)**.** Random Serial Dictatorship is an SPP s.t.:

- $f(r) = \{1\}^n$
- $p_{ij}^{(t)} = 1$
- $b = \{1\}^n$

In RSD, the prices and budgets are simply to ensure agents take maximum one item each. The payments can be made with fake money.

World domination is such an ugly phrase. I prefer to call it world optimisation.

Harry–*Harry Potter and the Methods of Rationality*

4

Reinforcement Learning Methods

In this chapter, we lay out the methods we use for learning good mechanisms within the Interactive Dynamic Posted Price (IDPP) class. In particular, we model the mechanism design problem as a Markov Decision Process (MDP) and employ state of the art reinforcement learning algorithms to learn good policies (mechanisms).

In Section 4.1, we provide background on MDPs and outline how mechanism design problems can be modeled as MDPs. In Section 4.2, we formalize and parameterize an MDP that captures a subclass of IDPP. In Section 4.3, we present our methods for optimizing policy parameters, including our optimization algorithm Adam and our gradient approximation method.

Our MDP does not capture the full generality of IDPP; in particular, it does not incorporate the mechanism history $h^{(t)}$ (the previous round's prices and current round's agent actions) into the policy decision in round t. This excludes mechanisms such as the Descending Price Auction, which require the previous round's prices to set the current round's prices. Nonetheless, the MDP is sufficiently general to describe the settings we will be studying in detail in Chapter 5.

4.1 MDP Background

An MDP is a problem where an agent must repeatedly decide what action to take based upon the current state. When the agent takes an action, they receive some reward and move to a new state.

Definition 4.1.1 (MDP)**.** A Markov Decision Process is defined by:

- A set of states S, including a starting state and a terminal state.
- A set of possible actions A
- A reward function $R : S \times A \to \mathbb{R}$
- A transition function $T : S \times A \to S$

Previous work by Bello et al. [2] and Dai et al. [10] has modeled algorithmic problems as MDPs and used reinforcement learning to learn good solution algorithms. The Traveling Salesperson Problem is a well-known NP-hard problem where the input is a set of two-dimensional real points (graph), and the output is a tour (ordering of the vertices). This can be modeled as an MDP where:

- The state is the graph and a partial tour.
- An action is the choice of which vertex to add to the tour next.
- The reward is negated distance traveled.
- Choosing vertex v adds it to the partial tour to create the new state.

The learning problem is to train an expected-reward-maximizing *policy* for choosing which vertex to add (action) given the graph and current partial tour (state). Here the expectation is over some distribution of input graphs. An expected-reward-maximizing policy for this MDP corresponds to an expected-distance-minimizing algorithm for solving the TSP. Bello et al. [2] and Dai et al. [10] parameterize policies with neural nets and use reinforcement learning to train these policies for TSP and other problems on graphs.

The success of this previous work on imperative algorithms inspired us to make use of these methods for indirect mechanism design. Instead of a policy representing a solution algorithm, it represents a mechanism. To build intuition, we will now sketch an MDP for the Serial Dictatorship setting. The state is the remaining items and agents, as well as the private agent valuations for the items. The action is an (agent, item) pairing. The policy proceeds in two steps:

1. The mechanism: the policy chooses some agent a to visit next. For this step, the private agent values are *hidden* from the policy.
2. The buyer behavior: the policy queries some fixed function *buyer* that determines what item i, if any, the chosen agent will select (purchase).

Then the policy takes action (a, i). The environment removes agent a and item i from the state, and the policy receives reward equal to agent a's value for item i. When there are either no more agents or no more items, the policy terminates.

In this formulation, the MDP policy encodes two things: the mechanism itself and the buyer behavior, which for Serial Dictatorship is all there is to agent strategies. This was a stylistic choice; it would be equally valid to encode the buyer behavior as part of the environment, in the transition function between states. Agents' private valuations are included as part of the state but hidden during the mechanism step of the policy; it would be cheating for the mechanism to know the private valuations. Agent valuations are only visible during the buyer step of the policy.

There is some uncertainty in the starting state because of the probabilistic nature of the agent valuations. We optimize the policy for expected reward (expected welfare), where the expectation is over starting states (agent valuations). We only optimize the mechanism part of the policy, i.e. which agent to visit next. The buyer behavior is fixed. Intuitively, as the mechanism designer, we only have control over the mechanism and not the agent strategies, so that is the only aspect of the policy we can train.

For Serial Dictatorship, optimal buyer behavior is simple to encode; an agent just chooses the item they value the most. The settings we study in this thesis all have similarly simple, dominant agent strategies, although the fully general Generalized Eating Mechanism does not. An MDP for the fully general GEM would have agent strategies that are trained alongside the mechanism (though with a different objective), as a way to handle mechanisms without dominant strategy equilibria.

We will soon see that an MDP formulation slightly more general the one above for SD can encode a large subclass of the Interactive Dynamic Posted Price subclass. The reinforcement learning (RL) mechanism starts in the starting state. One time step of the MDP proceeds as follows. The RL mechanism observes its current state $s \in S$, takes some action $a \in A$, receives reward $R(s, a)$, and moves to a new state $T(s, a)$. If the new state is the terminal state, the MDP terminates; otherwise, the next time step follows. The optimization problem is for the RL mechanism to learn a *policy*, a map from states to actions, that maximizes it's total reward.[1] The MDPs we consider have discrete state and action spaces, deterministic rewards and transitions, and finite time steps.

4.2 IDPP as a Markov Decision Process

To make this concrete, we make the simplifying assumption that agents have unit demand, which means that each agent has a value for each item, and an agent's value for a set of items is just the maximum of the item values. This means that on an agent's turn, they will purchase a maximum of one item.

For now, we restrict attention to mechanisms that only use the mechanism state (reports, agents remaining, partial allocation) and ignore the mechanism history (the previous round's prices and current round's agent actions). In particular, this leaves out mechanisms such as the Descending Price Auction. In terms of generality, this class is somewhere between SPP and IDPP.

[1] We consider only settings with bounded time steps. With unbounded time steps, the agent needs to optimize time-discounted lifetime reward.

The MDP policy encodes two things:

1. The parameterized mechanism.
2. The agent purchasing decisions.

However, only the mechanism parameters are learnable. The reinforcement learner cannot control the agent strategies. The state of the MDP in round t is just a state tuple of GEM. This includes:

- The initial agent reports $r \in \mathbb{R}^n$
- The remaining agents $a^{(t)} \in \{0,1\}^n$
- The partial allocation $x^{(t)} \in \{0,1\}^{n \times m}$

We also include the private agent valuations $u \in \mathbb{R}^{n \times m}$ as a part of the state. However, this is hidden from the mechanism's decisions and is only used to query the agents for purchasing decisions.[2] The policy's action is:

- An (agent, item) pairing, where the item can be the null option: $(a, i) \in [n] \times ([m] \cup \{\emptyset\})$

The mechanism part of the policy chooses which agent a to visit and what prices $p^{(t)}$ to set. The decision of what item an agent will buy if visited is queried from a static function $buyer : [n] \times \mathbb{R}^{n \times m} \times \mathbb{R}^{n \times m} \to [m] \cup \{\emptyset\}$, which takes the agent valuations and prices and returns an item or no item $i \in [m] \cup \{\emptyset\}$. The policy then chooses action (a, i). The state transition from time t to $t+1$ is computed as follows:

- Initial reports r are unchanged
- Remove agent a from the remaining agents $a^{(t+1)}$
- If $i \neq \emptyset$, allocate item i to agent a in $x^{(t+1)}$

The reward for an action (a, i) is agent a's valuation for item i if $i \neq \emptyset$ and 0 otherwise. In the settings we study in Chapter 5, there are no agent reports. Meanwhile, the agent puchasing decisions are simple to model: an agent just takes the item that gives them the best value minus payment.

[2] This is how we decided to operationalize the buyer strategies, but in an alternate MDP formulation, the agent valuations could be pushed into the environment

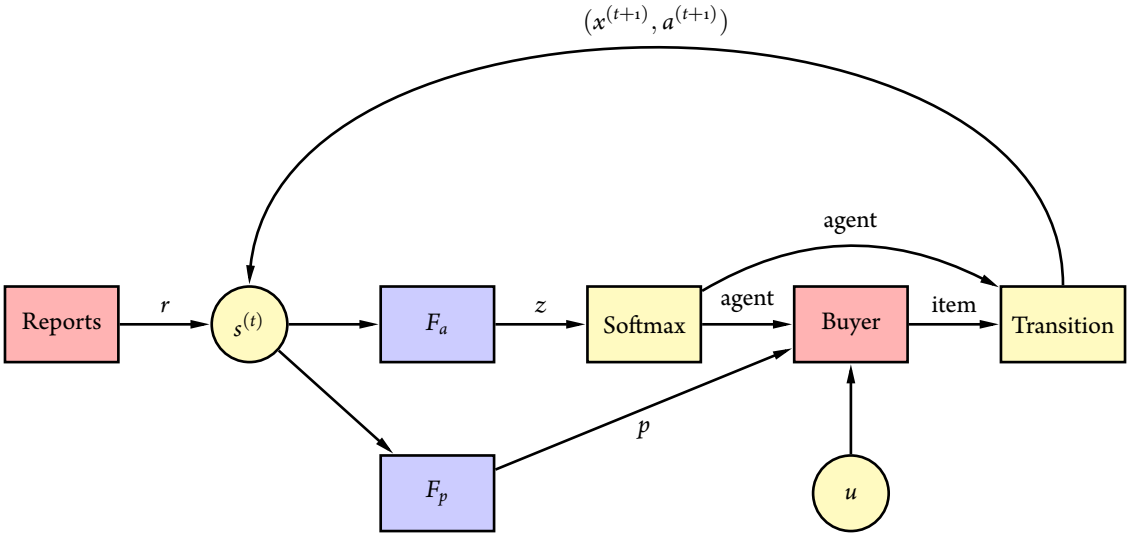


Figure 4.2.1: MDP Architecture

4.2.1 Parameterizing the Policy

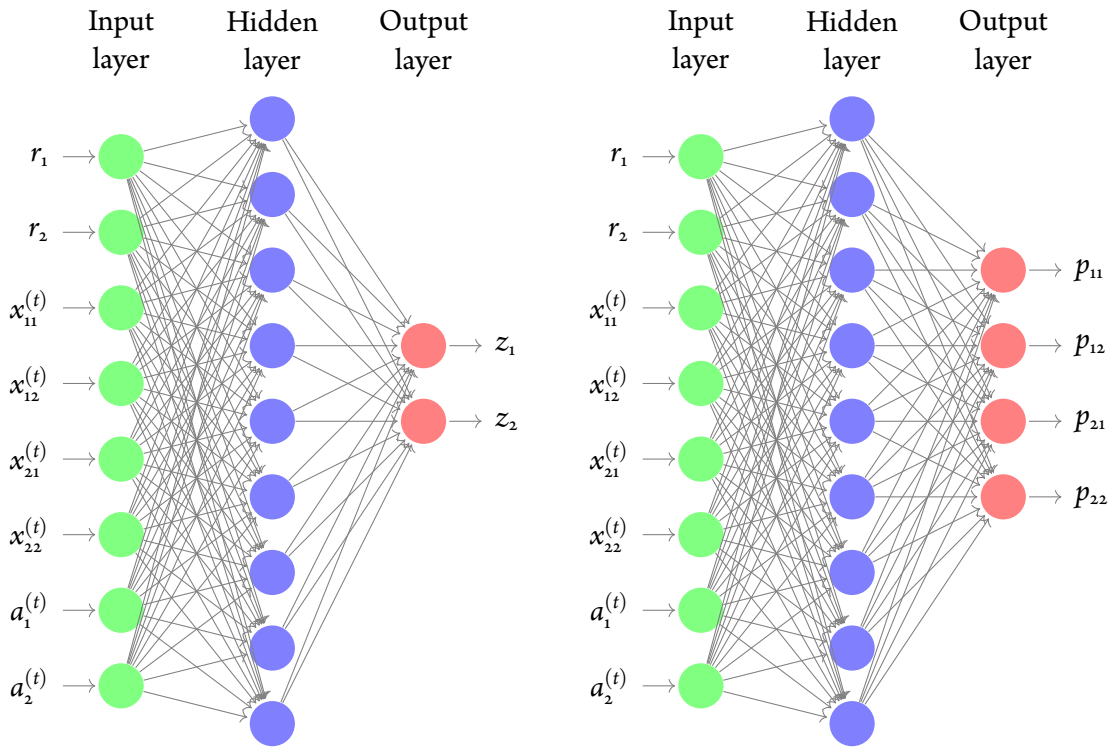
One tool we use in our MDP implementation is the softmax function. Softmax takes a vector as input and outputs a probability distribution as follows:

$$softmax\begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} = \frac{1}{\sum_{i=1}^{n} e^{z_i}} \begin{pmatrix} e^{z_1} \\ \vdots \\ e^{z_n} \end{pmatrix}$$

Our learned policy functions as follows:

1. A neural network F_a maps the state to a vector $z \in \mathbb{R}^n$ of scores for each agent.
2. We take a softmax over the scores for agents who remain, and we choose which agent to visit next by drawing from the resulting distribution.
3. A neural network F_p maps the state to a price matrix $p^{(t)} \in \mathbb{R}^{n \times m}$.
4. We query $item \leftarrow buyer(agent, p^{(t)}, u)$. The valuation u is hidden from F_a and F_p but is necessary to simulate the purchasing decisions.
5. We take action $(agent, item)$.

This process is illustrated in Figure 4.2.1. The blue boxes denote aspects of the MDP controlled by the mechanism, the red boxes denote aspects controlled by the agents, and the yellow boxes denote aspects



(a) F_a (determines order) **(b)** F_p (determines prices)

Figure 4.2.2: Neural Network Architecture

controlled by neither, for example the Transition function, which is just trivial bookkeeping. The distribution over reports, along with the *buyer* function, encode the agent strategies, while the neural networks F_a and F_p encode the mechanism. So the learned parameters θ of the policy are the weights of F_a and F_p. We structure F_a and F_p as single-layer fully-connected neural networks with a hidden layer of size 256. Figure 4.2.2 illustrates the architecture of F_a and F_p in a setting with two agents and two items; the hidden layer is not to scale. Each network takes in $s^{(t)} = (r, x^{(t)}, a^{(t)})$ as input, where r is the initial agent reports, $x^{(t)}$ is the partial allocation at time t, and $a^{(t)}$ is the remaining agents at time t. F_a outputs scores z for each agent to determine which agent goes next, while F_p outputs prices p for each (agent, item) pair. The GEM subclasses we study in Chapter 5 ignore the agent reports r and only use $x^{(t)}$ and $a^{(t)}$ for order and pricing decisions.

4.2.2 Generality of MDP

Theorem 4.2.1. *In the unit demand setting, the MDP formulation exactly captures the subclass of IDPP that ignores the mechanism history, always chooses to visit an agent, and doesn't set budget constraints:*

- $price(s, h) = price(s, h')$ *for any two mechanism histories* $h, h' \in H$

- $speed(s, h) = speed(s, h')$ *for any two mechanism histories* $h, h' \in H$
- $v_i^{(t)} = \{1\}^m$ *for some agent* i *for all times* t
- $budget(s, h) = \{\infty\}^n$ *for any state-history tuple* $(s, h) \in S \times H$

We defer the proof to Appendix A. As described previously, the Descending Price Auction lies within IDPP but outside the MDP because it uses the previous round's prices to set the current round's prices. This level of generality is sufficient for the settings we study in Chapter 5. However, future work will want an MDP that can capture the full generality of GEM, and in richer settings than unit demand. There are no conceptual barriers to such a formulation. The state would be a GEM (state, history) tuple, and the mechanism would have a continuous action of $(speeds, prices, budgets) \in \mathbb{R}^{n \times m} \times \mathbb{R}^{n \times m} \times \mathbb{R}^n$. Agent in-round actions and eating decisions would be a part of the environment and reside in the reward and transition functions rather than the policy. The policy itself would encode only the mechanism. In many ways, this formulation is more intuitive, and we plan to move in that direction for future work. Nonetheless, our current formulation is sufficiently general for the purposes of this thesis, and it benefits from a small, discrete action space, which simplifies our learning problem.

4.3 Optimization

We now present our techniques for optimizing the parameters of our MDP policy. The optimization algorithm we use is a variant of stochastic gradient descent called Adam. Our method for approximating gradients is a variant of Monte Carlo policy gradients called the Advantage Actor Critic (A2C).

4.3.1 Adam

Adam is an extension of stochastic gradient descent that employs parameter-specific adaptive learning rates [20]. These learning rates are determined by the global learning rate, as well as estimations of the first and second moments of the gradients. The name *Adam* is derived from adaptive moment estimation. Adam has a number of desirable properties: it is invariant to scaling the gradients, it performs well with sparse gradients, and the learning rates naturally grow smaller near an optimum. Across a range of settings, Adam converges as or more efficiently than other commonly employed optimization algorithms.

The formal algorithm for Adam is shown in Algorithm 1, replicated from the original paper by Kingma and Ba [20]. We calculate exponential moving averages to estimate the first and second moments of the gradients. The hyperparameters β_1 and β_2 control the exponential weights of these averages. The moment vectors are initialized to 0, which biases the averages toward zero; $\hat{m}_t$ and $\hat{v}_t$ represent bias-corrected estimates of the first and second moments respectively. Letting α be the global learning rate hyperparameter,

Algorithm 1: Adam

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second moment estimate)
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\hat{v} \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$ (Update parameters)
end

classical gradient descent would update parameters as:

$$\theta_t \leftarrow \theta_{t-1} + \alpha g_t$$

Adam instead updates parameters as:

$$\theta_t \leftarrow \theta_{t-1} + \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

The division and square root operations here are performed element-wise. Using $\hat{m}_t$ instead of g_t creates *momentum* in the gradients; the parameter update in time t is partially updating in the direction of gradients from previous time periods. Dividing by $\sqrt{\hat{v}_t}$ scales down gradient coordinates with a low mean-to-variance ratio. One effect of this normalization is that the effective learning rate decays near an optimum. It also makes the parameter update invariant to scaling the gradient. We add ε in the denominator to avoid dividing by zero.

We use Adam to optimize the parameters θ of our MDP policy. For the gradient calculation step, $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$, we use the Advantage Actor Critic method.

4.3.2 Advantage Actor Critic

The policy objective is expected reward, where the expectation is over trajectories τ.

$$J(\theta) = \mathbb{E}_\tau[r(\tau)]$$

Below is the gradient of J with respect to θ [34]. We substitute current and future rewards G_t for the reward of the full trajectory (no need to incorporate past rewards into the current decision).

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t))r(\tau)\Big] \\ =& \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t))G_t\Big]\end{aligned}$$

Below is the Monte Carlo gradient approximation. We approximate the expectation by sampling trajectories. We also incorporate a baseline b to reduce the variance of the gradients.

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_{i,t}|s_{i,t}))(G_{i,t} - b(s_{i,t}))$$

We use the Advantage Actor Critic (A2C), which defines the baseline as follows [29]:

$$G_t - b(s_t) := A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Here $V(s_t)$ is the critic's estimation of the baseline value of being in state s_t, so the *advantage* $A(s_t, a_t)$ captures how much better taking action a_t is than the other actions available. The critic's estimates V are the output of a neural net which is trained alongside the policy (actor). The structure of the neural net is the same as the neural nets depicted in Figure 4.2.2, just with a single output node $V(s^{(t)})$. Below is the critic loss (for a critic parameterized by θ):

$$\mathcal{L}(\theta_v) = \sum_{t=0}^{T-1} |A_\theta(s_t, a_t)|^2$$

As the policy learns better actions, the critic learns better estimates of the values of states, which enables the policy to learn yet better actions. Intuitively, the critic is useful because it reduces variance in gradients, which leads to more stable learning.

4.3.3 Softmax

For A2C to work properly, the distribution over policy actions ((agent, item) pairs) must be differentiable in the policy parameters. Otherwise, gradients would be zero and the RL mechanism would be unable to

learn. We accomplish this differentiability by using the softmax function:

$$softmax \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} = \frac{1}{\sum_{i=1}^{n} e^{z_i}} \begin{pmatrix} e^{z_1} \\ \vdots \\ e^{z_n} \end{pmatrix}$$

Instead of deterministically visiting the agent with the highest score (F_a output), the mechanism takes a softmax over the scores and chooses the agent to visit by drawing from that distribution. This makes the distribution over policy actions differentiable in the weights of the order network F_a. Then instead of buyers deterministically choosing the item that gives them the best value minus price, a buyer takes a softmax over items, where the scores are the value minus price for each item, and draws the item from the resulting distribution. This makes the distribution over policy actions differentiable in the weights of the price network F_p.

A computer once beat me at chess, but it was no match for me at kick boxing.

Emo Philips

5 Results

In this chapter, we present three economic settings and show that our reinforcement learning mechanism is able to learn the optimal policy in each. As described in Section 4.3.3, we use the softmax function to enable the RL mechanism to learn properly. However, this trick for helping the mechanism learn is not necessary for evaluating a mechanism. In fact, it adds noise that is unreflective of the true performance of the mechanism. So to evaluate our RL mechanism, after every training episode we run an evaluation episode where the mechanism just visits the agent with the highest score at each time step. It is this performance that we discuss, rather than the reward from training episodes. In these evaluation episodes, agents still technically made their purchasing decisions through a softmax, but for the most part their value minus payment was far enough apart that the decisions were basically deterministic.

5.1 Static Serial Dictatorship

We first test a Serial Dictatorship setting where the optimal welfare can be achieved by a static order. For the Serial Dictatorship subclass, the policy only chooses which agent to visit next; prices are set to zero. In this setting, there is persistent asymmetry between agent valuations, so the learning task is to learn to visit agents with generally high valuations earlier in the ordering. Our reinforcement learning mechanism successfully is able to successfully learn this behavior.

There are 100 agents, numbered 0 to 99. There are nine items, numbered 1 to 9. Agent i's value for item j is

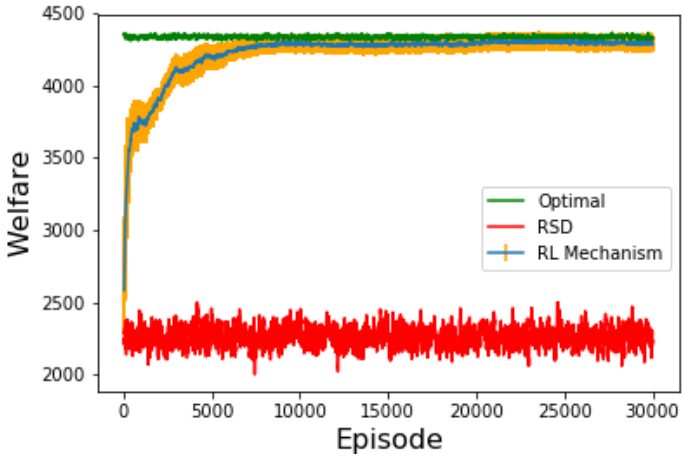


Figure 5.1.1: Performance of RL Static SD by training episode.

distributed as $u_i(j) \sim \max(N(ij, ij), 0)$. In words, it is normally distributed with a mean of ij and a standard deviation of $\sqrt{ij}$, truncated to be non-negative. Values are drawn independently. The optimal policy will visit agent 99 first, then agent 98, then agent 97, always choosing the highest agent remaining because in expectation they have the highest value for each item. This policy achieves expected welfare $\approx$ 4336.7. Meanwhile, RSD achieves expected welfare $\approx$ 2265.6, which is $\approx 52\%$ of the optimal.

Figure 5.1.1 plots the social welfare achieved by our RL Mechanism against the number of training episodes. The blue is the average performance of ten learning trajectories, and the orange is the error bars (plus or minus a standard deviation). The policy starts with randomly initialized weights and so is similar to RSD, but it quickly learns to put agents with high valuations earlier in the order. While the figure only plots performance up to 30,000 training episodes, the mechanism slowly improves even beyond that. By episode 200,000, our mechanism achieves $\approx 99.5\%$ of the optimal expected welfare. One of the RL mechanisms' executed order in episode 200,000 was $(97, 99, 96, 92, 94, 98, 95, 93, 90)$. This is not precisely optimal, but it's close enough for all practical purposes.

5.2 Dynamic Serial Dictatorship

We next test a Serial Dictatorship setting where the optimal welfare can only be achieved by a dynamic policy. In this setting, each agent only gets nontrivial utility from one type of item, so the policy must learn to only visit an agent if their preferred item type remains. Our reinforcement learning mechanism is able to successfully learn the optimal policy.

There are $3n$ agents: n red, n yellow, and n blue. There are $2n$ items: n red and n yellow. Red agents have value 10 for each red item and some small ε for each yellow item. Yellow agents have value 10 for each yellow item and ε for each red item. Blue agents' valuations are realized as follows:

- Draw $X \sim Unif(0, 1)$
- Each blue agent's valuation is drawn i.i.d.:
  - With probability X, value 20 for all red items and ε for all yellow items.
  - With probability $1 - X$, value 20 for all yellow items and ε for all red items.

The optimal social welfare is achieved by each blue agent receiving their preferred item type, then the remaining red items going to red agents and yellow items going to yellow agents. The optimal dynamic serial dictatorship achieves this by visiting the n blue agents first. Then if there are r red items and y yellow items left, it visits r red agents and y yellow agents in the next $r + y$ time steps.

This optimal policy needs to be dynamic. If the blue agents take an equal number of red and yellow items, the mechanism should visit an equal number of red and yellow agents in the next n time steps. If instead the blue agents take all red items, the mechanism must visit only yellow agents to ensure the remaining yellow items go to yellow agents.

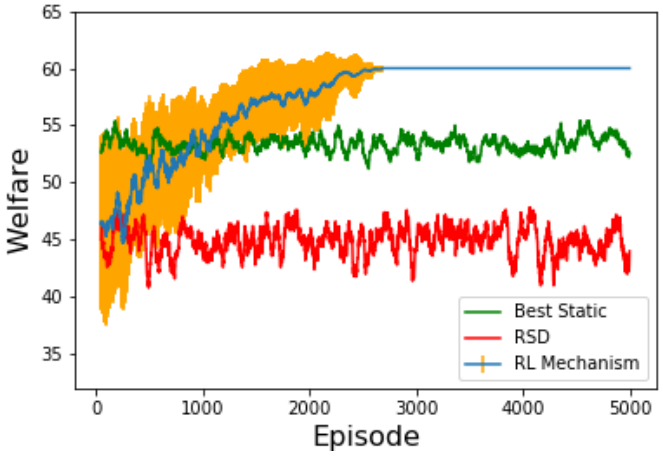
The best Static Serial Dictatorship is to visit the blue agents first, then an even split of red and yellow agents:

Theorem 5.2.1. *The policy that visits the n blue agents, then $\lceil \frac{n}{2} \rceil$ red agents, then $\lfloor \frac{n}{2} \rfloor$ yellow agents achieves weakly the greatest expected welfare among Static Serial Dictatorships.*

We defer the proof to Appendix A. However, this policy cannot achieve the full welfare when the blue agents take an uneven split of red and yellow items. Thus, our RL agent needs to effectively learn a dynamic policy in order to perform optimally.

Figure 5.2.1a plots the social welfare achieved by our RL Mechanism against the number of training episodes for $n = 2$. The blue is the average performance of ten learning trajectories, and the orange is the error bars (plus or minus a standard deviation). The policy starts with randomly initialized weights and so is similar to RSD, but immediately starts to learn, surpassing the best static policy and converging to the optimal policy, which achieves welfare 60 every time.

Figure 5.2.1b illustrates the dynamic behavior of one of the RL mechanisms in evaluation episode 5,000. The "Agent Order" column represents the ordering of the scores for each agent, with the agent of highest score first and lowest score last. In the first two rounds, the mechanisms puts the blue agents first; one takes a red item, then the other takes a yellow item. In round 3, the mechanism puts the two red agents ahead of the two yellow agents; agent R_1 takes the remaining red item. But in round 4, because there is only a yellow item left, the mechanism puts the yellow agents ahead of the remaining red agent, and agent Y_1 ends up with the last yellow item. This ensures that each item ends up with either a blue agent or an agent of the item's color.



(a) Performance by training episode

Time	Items	Agent Order	Action
1	$Y_a, Y_b, R_a, R_b,$	$(B_1, B_2, R_1, R_2, Y_1, Y_2)$	(B_1, Y_a)
2	Y_b, R_a, R_b	$(B_2, R_1, R_2, Y_1, Y_2)$	(B_2, R_a)
3	Y_b, R_b	(R_1, R_2, Y_1, Y_2)	(R_1, R_b)
4	Y_b	(Y_1, Y_2, R_2)	(Y_1, Y_b)

(b) Dynamic Behavior

Figure 5.2.1: Dynamic Serial Dictatorship

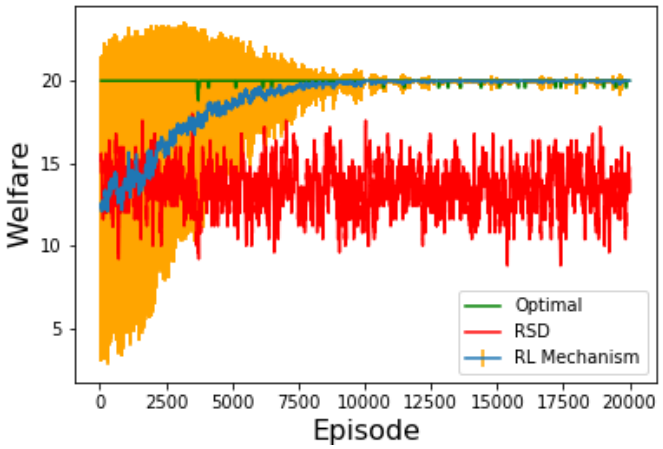
5.3 Dynamic Price

Finally, we test a setting where no Serial Dictatorship can outperform RSD; instead, the mechanism must learn to set appropriate prices to achieve optimal welfare. Our reinforcement learning mechanism is able to learn these optimal prices.

There are ten agents and one item. Each agent's valuation for the item is drawn i.i.d.:

- With probability 0.5, value 20.
- With probability 0.5, value ε.

The optimal allocation is to distribute the item to an agent with value 20, if they exist. However, this cannot be achieved with a serial dictatorship; the mechanism does not know a priori which agents have high valuations and which have low valuations. In fact, because agent valautions are symmetric and independent, there is no advantage whatsoever to optimizing the order. Instead, the optimal mechanism sets a price for the item between 0 and 20 (for example, 10), which ensures that when it visits an agent, that agent will buy the item iff. they have value 20 for it. Our RL mechanism is allowed to learn dynamic prices, although in this setting static prices are sufficient to achieve optimal welfare.



(a) Performance by training episode

Agent	Price
1	10.4
2	10.2
3	9.9
4	10.1
5	10.2
6	10.1
7	10.0
8	10.2
9	10.0
10	10.4

(b) Learned Prices

Figure 5.3.1: Dynamic Price Mechanism

As previously discussed, for our policy to learn properly, the distribution over buyer actions (buy the item or not) needs to be differentiable in the price. We accomplish this by having an agent choose to buy or not by drawing from the distribution $softmax([value - price, 0])$ where the 0 represents not buying the item. One byproduct of this probabilistic implementation is that buyers sometimes make irrational decisions; if the value minus price for the item is -0.4, the agent will sometimes choose to buy it anyway. Conversely, if the value minus price for the item is 0.6, the agent will sometimes choose not to buy it. The practical upshot is that for our RL mechanism, prices close to 0 and 20, while theoretically optimal, do not achieve optimal welfare in the environment. Prices close to 10 ensure that an agent is extremely unlikely to buy the item with value 0 or pass on it with value 20. With price 10, an agent will make an irrational decision $< 0.01\%$ of the time.

Figure 5.3.1a plots the social welfare achieved by our RL mechanism against the number of training episodes. The blue is the average performance of ten learning trajectories, and the orange is the error bars (plus or minus a standard deviation). The policy starts with prices close to zero and so is similar to RSD, which achieves $\approx 66\%$ of the optimal welfare. However, over time the mechanism learns to raise prices, thus excluding agents from buying the item if they have a low valuation for it. Figure 5.3.1b shows the prices for each agent for one of the RL mechanisms in the first time period of evaluation episode 20,000. Notice they are all close to 10; intuitively, this makes sense because as discussed in the previous paragraph, this ensures agents take the item *exactly if* they have value 20 for it. The reason neither our RL mechanism nor the optimal mechanism (which sets all prices at 10) is able to achieve welfare 20 every time is that 0.1% of the time, none of the agents have value 20 for the item, so the only welfare possible is 0. The trajectories are smoothed for clarity, which is why these present in the graph as small dips rather than spikes down to zero.

5.4 Discussion

The optimal design of indirect mechanisms is a relatively unstudied problem, and to our knowledge this is the first use of reinforcement learning for indirect mechanism design. The right parameterization of a mechanism depends on the economic environment, but GEM is a sufficiently general framework to capture an important range of possible settings. In the Static Serial Dictatorship setting, our RL mechanism was able to learn that some agents have persistently higher valuations than others and put those agents earlier. In the Dynamic Serial Dictatorship setting, our RL mechanism learned to put the blue agents first because they have the highest values and also to adjust the order based upon which items were left after the blue agents went. In a setting where prices were necessary to achieve optimal welfare, our RL mechanism was able to learn efficient prices. These results are encouraging and indicate that reinforcement learning can be an effective tool for the automated design of indirect mechanisms. In particular, our learning framework seems promising in its ability to generalize to additional environments. See Chapter 6 for some discussion of potential future directions.

By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.

Elieze Yudkowsky

6 Conclusion

Optimal resource allocation and social choice under computational constraints remains an open problem. While there exists a rich literature of direct mechanisms, little is known about settings where full preference communication is difficult and indirect mechanisms are required. Neural networks have previously been shown to be effective tools for the automated design of direct mechanisms. Separately, other researchers have used reinforcement learning to train parameterized MDP policies for the solution of imperative algorithmic problems. In this thesis, we combined elements of these two agendas and presented the first use of neural networks and reinforcement learning for indirect mechanism design.

We presented the Generalized Eating Mechanism, a large parametric class of indirect mechanisms, and showed that a number of well-known indirect mechanisms exist as special cases of GEM. We formulated the indirect mechanism design problem as a Markov Decision Process and studied the performance of a reinforcement learner on a few different special settings. In a setting where a static serial dictatorship can achieve the optimal welfare, our RL mechanism was able to achieve $> 99\%$ of the optimal welfare, compared to $\approx 52\%$ for RSD. In a setting where a dynamic serial dictatorship is necessary to achieve optimal welfare, our RL mechanism exhibited this dynamic behavior and achieved 100% of the optimal welfare, compared to $\approx 75\%$ for RSD and $\approx 89\%$ for the best static serial dictatorship. In a setting where order is unimportant and prices are necessary to achieve optimal welfare, our RL mechanism was able to learn optimal prices and achieve 100% of the optimal welfare, compared to $\approx 66\%$ for RSD.

There are a number of ways in which future work can build on this thesis. The most immediate is to

train the reinforcement learning mechanism in a setting where the optimal price must be set dynamically. Another natural next step is to extend the MDP to fully capture GEM, as outlined in Chapter 4. This would give the reinforcement learning mechanism more degrees of freedom over which to optimize, as well as let us study its performance in settings beyond unit demand.

One challenge to including agent reports, in-round actions, eating speeds, and so forth, is that optimal agent strategies quickly become unclear. One possible approach is to train the mechanism on some fixed model of agent behavior, then train agent behavior on the learned mechanism, then train the mechanism on the new agent behavior, and back and forth until the mechanism and strategies converge. Perdomo et al. [24] discuss conditions for the convergence of such a back-and-forth training loop in a more general setting.

Another possible approach is to use imitation learning to model human behavior, for example through Behavioral Cloning [25] or Generative Adversarial Imitation Learning [19]. Recall in Section **??** we described a loop of automated mechanism design. We observe human play of video games with mechanisms via crowdsourcing, train imitation learning agents to mimic that behavior, train good mechanisms for those imitation learners, then roll out those mechanisms to collect more human play and continue the loop. This thesis is the result of work on the mechanism training step, and we look forward to "closing the loop" and enabling rapid prototyping and testing of mechanisms across a host of different environments.

The Internet? We are not interested in it.

Bill Gates, 1993

A Proofs

A.1 Proof of Theorem 4.2.1

Theorem 4.2.1. *In the unit demand setting, the MDP formulation exactly captures the subclass of IDPP that ignores the mechanism history, always chooses to visit an agent, and doesn't set budget constraints:*

- $price(s, h) = price(s, h')$ *for any two mechanism histories* $h, h' \in H$
- $speed(s, h) = speed(s, h')$ *for any two mechanism histories* $h, h' \in H$
- $v_i^{(t)} = \{1\}^m$ *for some agent* i *for all times* t
- $budget(s, h) = \{\infty\}^n$ *for any state-history tuple* $(s, h) \in S \times H$

Proof. Recall an IDPP mechanism is described by three functions:

- *speed*, which takes as input the (state, history) and outputs a choice of some agent to visit (receive eating speeds $\{1\}^m$) or no agent. If an agent is visited, they are removed prior to the next round.
- *price*, which takes as input the (state, history) and outputs prices for each (agent, item) pair.
- *budget*, which takes as input the (state, history) and outputs a budget for each agent.

The neural network F_a maps the state to a distribution over agents; this corresponds to a choice of which agent will be visited. Because the subclass ignores the mechanism history, it is without loss of expressiveness to require that the mechanism visits an agent each round. Suppose some history-agnostic IDPP mechanism chose not to visit an agent when in state $s^{(t)}$ at time t. Then, no agent will consume, and the state will be the same in the next round. In round $t+1$, because the $s^{(t+1)} = s^{(t)}$, the mechanism will set prices $p^{(t+1)} = p^{(t)}$ and again choose not to visit an agent. Recall the termination criteria for GEM: no agents remaining, no items remaining, or prices constant across consecutive periods without any agent consumption. This implies that if a history-agnostic IDPP mechanism ever chooses not to visit an agent, it terminates with no

further consumption. This can be equivalently achieved by setting large prices for all (agent, item) pairs so that no more items are consumed. So, allowing the MDP to not visit an agent does not provide additional expressiveness.

The IDPP condition that agents are removed after receiving positive eating speed is modeled through the MDP transition function. The *price* function is exactly the IDPP *price* function except that, as required by this subclass, the neural network F_p does not take the mechanism history as input.

Because we operate in a unit demand setting where agents consume single items deterministically, agents will choose to purchase at most one item. This means it suffices for agent consumption decisions to be modeled by the *buyer* function which simply outputs a choice of item or no item. Additionally, because agents will buy at most one item, having budgets only serves to exclude some items from being purchased by some agents, and this could be equivalently achieved through setting large prices for those (agent, item) pairs. Thus, leaving budgets in the MDP does not provide additional expressiveness.

□

A.2 Proof of Theorem 5.2.1

Theorem 5.2.1. *The policy that visits the n blue agents, then* $\lceil \frac{n}{2} \rceil$ *red agents, then* $\lfloor \frac{n}{2} \rfloor$ *yellow agents achieves weakly the greatest expected welfare among Static Serial Dictatorships.*

Proof. First, we will prove that the best static policy puts the n blue agents first. First note that permuting the first n agents does not affect welfare because there are n of each type of item, so the first n agents will always get their preferred item. So for maximal welfare purposes, it is WLOG to assume that orders lead with some number x blue agents, then $n - x$ non-blue agents, then some mix of agents beyond that. Now consider some order in which some blue agent lies outside the first n positions, and let b be the first such agent, with position y. We will show that swapping b with the agent r in position $x + 1$, who is red or yellow (WLOG red), leads to a weak increase in expected social welfare. We proceed by case analysis:

1. b's valuation realizes such that they prefer red items. Then switching b and r is a weak increase in expected welfare because it is weakly more likely a red item is available for an agent in position $x + 1$ than position y.

2. b's valuation realizes such that they prefer yellow items. Then if we switch, b takes a yellow item in position $x + 1$, so after position $x + 1$, there is one fewer yellow item and one more red item than there would be otherwise.

    - If there are no red items available before position y, then some agent i between positions $x + 1$ and y took a red item when they otherwise (without a switch) would have taken a yellow item. The decisions of the other agents are the same as without a switch. So in effect, by switching we have transferred a yellow item from i to b, transferred a red item from r to i, and transferred whatever b would have gotten in their original position, either nothing or a yellow item, to r.
        - If the b to r transfer is a yellow item, then agent i must be a red agent (by construction, they cannot be a blue agent). Otherwise, they would not choose a red item when yellow items were still available. In this case, b keeps the same type of item, while i and r switch types of items. But agent i values the red item as much as agent r does, so this transfer maintains social welfare.

- If the b to r transfer is nothing, then b would have gotten nothing in the original order. The change in b's value from the transfer is thus $+20$. The change in r's value from the transfer is -10. So even if agent i is yellow, leading to loss of $-(10-\varepsilon)$, the change in welfare caused by the transfer is still $+\varepsilon$

- If there is a red item available before position y, then agent r takes it and maintains the same personal value.
  - If the agents between positions $x+1$ and y take the same items as they would without the switch, this means b would have gotten a yellow item originally, so welfare is maintained.
  - No agent between positions $x+1$ and y will take a yellow item if they prefer a red item because there is at least one red item available.
  - If some agent i between positions $x+1$ and y takes a red item when they would have take a yellow item originally, this means b would have originally gotten a red item. The welfare increase from b going from a red item to a yellow item is weakly larger than the welfare decrease for agent i going from a yellow item to a red item.

So by first permuting the top n positions so that the blue agents are first, then iteratively swapping the highest-ranked (lowest index) blue agent that is outside the top n positions with the highest-ranked non-blue agent within the top n, we make only weakly welfare-improving swaps and end with an order that has the n blue agents first.

Now we will prove that following the n blue agents with $\lceil \frac{n}{2} \rceil$ red agents, then $\lfloor \frac{n}{2} \rfloor$ yellow agents is weakly optimal. Suppose that the next n agents are made up of r red agents and $n-r$ yellow agents. A red agent is able to select a red item iff. the number of red items left after the blue agents go is greater than the number of red agents ahead of them. This follows from the fact that yellow agents will only take red items if there are no yellow items left and all the remaining items are red. Symmetrically, a yellow agent is able to select a yellow item iff. the number of yellow items left after the blue agents go is greater than the number of yellow agents ahead of them. A consequence of this fact is that it is without loss of optimality to permute the second n agents. So without loss of optimality we have n blue agents, then r red agents, then $n-r$ yellow agents. Now all that remains to be shown is that an equal split of red and yellow agents among the second n agents is optimal. Consider the case where n is even. Suppose that $r < \frac{n}{2}$, so there are $n-r > \frac{n}{2}$ yellow agents. It would be welfare-improving for the last yellow agent to be a red agent because the probability that there are $\geq \frac{n}{2}$ red items left after the blue agents go is larger than the probability that there are $\geq \frac{n}{2}+1$ yellow items left after the blue agents go. So $r \leq \frac{n}{2}$. By symmetry, this means $r = \frac{n}{2}$. In the case where n is odd, it does not matter if the extra agent is red or yellow. So it is a weakly optimal order to have n blue agents, then $\lceil \frac{n}{2} \rceil$ red agents, then $\lfloor \frac{n}{2} \rfloor$ yellow agents. □

Never be certain of anything. It's a sign of weakness.

The Doctor–*Doctor Who*

B

The Boston Mechanism as GEM

The Boston Mechanism is so named for its use in assigning students to Boston Public Schools prior to 2005 [23]. A slightly modified version is presented below:

Example B.0.1 (Boston Mechanism)**.** Assign items in the following manner:

- In rounds $t > 0$, each agent points at one item. For each item that has at least one agent pointing to it, assign the item at random to one of those agents. Remove agents who have been allocated an item.

Example B.0.2 (Boston Mechanism as GEM)**.** The Boston Mechanism is a GEM s.t.:

- If $x_{ij} = 0$ for all j:
  - $v_{ij}^{(t)} = \frac{1}{d_j^{(t)}}$ for $j = c_i^{(t)}$, where $d_j^{(t)}$ is the number of agents who point at item j in round t, and $v_{ij'}^{(t)} = 0$ for $j' \neq c_i^{(t)}$
  - $a_i^{(t+1)} = 1$
- Otherwise:
  - $v_i^{(t)} = \{0\}^m$
  - $a_i^{(t+1)} = 0$
- $p^{(t)} = \{0\}^{n \times m}$
- $b^{(t)} = \{0\}^n$

The wrinkle to implementing Boston as a GEM is that an agent's eating speed for the item they point to is inversely proportional to the number of agents pointing at that item. This implements a random assignment

of the item to one of the agents pointing at it, which occurs before the next round. In the next round, any agents who were allocated an item previously are given zero eating speed and removed.

In 2005, the Deferred Acceptance Algorithm [15], replaced the Boston Mechanism for use in the Boston Public School matching. Deferred Acceptance has a number of desirable properties, including strategyproofness on the student (agent) side [26]. Unfortunately, it is only well-defined in a two-sided matching setting where both students and schools have preferences over each other. In the combinatorial assignment setting we operate in, items don't have preferences over agents, so Deferred Acceptance is not well-defined.

References

[1] Atila Abdulkadiroğlu and Tayfun Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998. ISSN 00129682.

[2] Irwan Bello, Hieu Pham, Quoc Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv.org*, 2017. URL http://search.proquest.com/docview/2075163784/.

[3] Anna Bogomolnaia and Hervé Moulin. A new solution to the random assignment problem. 100(2): 295–328, 2001. ISSN 0022-0531.

[4] Mark Braverman, Jieming Mao, and S Weinberg. Interpolating between truthful and non-truthful mechanisms for combinatorial auctions. *arXiv.org*, 2015. URL http://search.proquest.com/docview/2083860133/.

[5] Y. Cai, C. Daskalakis, and S. M. Weinberg. An algorithmic characterization of multi-dimensional mechanisms. In *Proceedings of the 44th ACM Symposium on Theory of Computing*, pages 459–478, 2012.

[6] Y. Cai, C. Daskalakis, and S. M. Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science*, pages 130–139, 2012.

[7] Y. Cai, C. Daskalakis, and S. M. Weinberg. Understanding incentives: Mechanism design becomes algorithm design. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science*, pages 618–627, 2013.

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

[9] V. Conitzer and T. Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 132–141, 2004.

[10] Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv.org*, 2018. URL http://search.proquest.com/docview/2071625514/.

[11] Nikhil Devanur, Jamie Morgenstern, Vasilis Syrgkanis, and S Weinberg. Simple auctions with simple strategies. In *Proceedings of the Sixteenth ACM Conference on economics and computation*, EC '15, pages 305–322. ACM, 2015. ISBN 9781450334105.

[12] P. Dütting, F. Fischer, P. Jirapinyo, J. Lai, B. Lubin, and D. C. Parkes. Payment rules through discriminant-based classifiers. *ACM Transactions on Economics and Computation*, 3(1):5, 2014.

[13] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David C. Parkes, and Sai Srivatsa Ravindranath. Optimal Auctions through Deep Learning. In *Proc. 36th Int. Conf. on Machine Learning*, pages 1706–1715, 2019.

[14] Z. Feng, H. Narasimhan, and D. C. Parkes. Deep learning for revenue-optimal auctions with budgets. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, pages 354–362, 2018.

[15] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. ISSN 0002-9890. URL http://www.tandfonline.com/doi/abs/10.1080/00029890.1962.11989827.

[16] N. Golowich, H. Narasimhan, and D. C. Parkes. Deep learning for multi-facility location mechanism design. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 261–267, 2018.

[17] Jerry Green and Jean-Jacques Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45(2):427, 1977. ISSN 0012-9682. URL http://search.proquest.com/docview/1296437590/.

[18] Terrence Hendershott, Charles M. Jones, and Albert J. Menkveld. Does algorithmic trading improve liquidity? *Journal of Finance*, 66(1):1–33, 2011. ISSN 0022-1082.

[19] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems 29*, pages 4565–4573. 2016. URL http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv.org*, 2017. URL http://search.proquest.com/docview/2075396516/.

[21] Paul Milgrom. *Putting Auction Theory to Work*. 2004. ISBN 9780521551847.

[22] David Parkes and Michael Wellman. Economic reasoning and artificial intelligence. *Science (New York, N.Y.)*, 349:267–72, 07 2015. doi: 10.1126/science.aaa8403.

[23] Parag A. Pathak and Tayfun Sönmez. Leveling the playing field: Sincere and sophisticated players in the boston mechanism. *American Economic Review*, 98(4):1636–1652, 2008. ISSN 0002-8282.

[24] Juan Perdomo, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. Performative prediction. *arXiv.org*, 2020. URL http://search.proquest.com/docview/2357106621/.

[25] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991. ISSN 0899-7667.

[26] Alvin Roth. Deferred acceptance algorithms: history, theory, practice, and open questions. *International Journal of Game Theory*, 36(3):537–569, 2008. ISSN 0020-7276.

[27] Alvin E Roth. Repugnance as a constraint on markets. 21(3):37–58, 2007. ISSN 0895-3309.

[28] W. Shen, P. Tang, and S. Zuo. Automated mechanism design via neural networks. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems*, 2019. Forthcoming.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning : an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts ; London, England, second edition. edition, 2018. ISBN 9780262039246.

[30] A. Tacchetti, D.J. Strouse, M. Garnelo, T. Graepel, and Y. Bachrach. A neural architecture for designing truthful and efficient auctions. *CoRR*, abs/1907.05181, 2019.

[31] Paul Tylkin, David C. Parkes, and Goran Radanovic. Multi-player atari: Designing helper-ais in rich environments. Technical report, Harvard University, 2020.

[32] Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the USENIX Workshop on Electronic Commerce*, 1995.

[33] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961. ISSN 0022-1082.

[34] Ronald Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 0885-6125.

[35] Lin Zhou. On a conjecture by gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990. ISSN 0022-0531.