THE UNIVERSITY of ADELAIDE

Machine Learning Approaches to Automated Mechanism Design for Public Project Problem

Guanhua Wang

A thesis submitted for the degree of

DOCTOR OF PHILOSOPHY

The University of Adelaide

June 26, 2022

Contents

List of Figures

List of Tables

University of Adelaide

Abstract

Machine Learning Approaches to Automated Mechanism Design for Public Project Problem

by Guanhua Wang

Mechanism design is a central research branch in microeconomics. An effective mechanism can significantly improve performance and efficiency of social decisions under desired objectives, such as to maximize social welfare or to maximize revenue for agents.

However, mechanism design is challenging for many common models including the public project problem model which we study in this thesis. A typical public project problem is a group of agents crowdfunding a public project (e.g., building a bridge). The mechanism will decide the payment and allocation for each agent (e.g., how much the agent pays, and whether the agent can use it) according to their valuations. The mechanism can be applied to various economic scenarios, including those related to cyber security. There are different constraints and optimized objectives for different public project scenarios (sub-problems), making it unrealistic to design a universal mechanism that fits all scenarios, and designing mechanisms for different settings manually is a taxing job. Therefore, we explore automated mechanism design (AMD) (Sandholm, 2003) of public project problems under different constraints.

In this thesis, we focus on the public project problem, which includes many sub-problems (excludable/non-excludable, divisible/indivisible, binary/non-binary). We study the classical public project model and extend this model to other related areas such as the zero-day exploit markets. For different sub-problems of the public project problem, we adopt different novel machine learning techniques to design optimal or near-optimal mechanisms via automated mechanism design.

We evaluate our mechanisms by theoretical analysis or experimentally comparing our mechanisms against existing mechanisms. The experiments and theoretical results show that our mechanisms are better than state-of-the-art automated or manual mechanisms.

List of Abbreviations

AMD	Automated mechanism design
GAN	Generative adversarial network
MLP	Multi-layer perceptions
AMA	Affine maximizer auctions (Likhodedov and Sandholm, 2005)
LP	Linear programming
IR	Individual rationality
SP	Strategy-proofness
SF	Straight-forwardness (Assumption 6.3)
DP	Dynamic programming
PORF	Price-oriented rationing-free (Yokoo, Sakurai, and Terada, 2002)
DSIC	Dominant-strategy incentive compatible
CEC	Conservative equal cost mechanism
CDF	Cumulative distribution function
PDF	Probability density function
SCS	Serial cost sharing mechanism
NN	Neural network
TGA	Truthful genetic algorithm (In Chapter 4)
ATGA	Approximately truthful genetic algorithm (In Chapter 4)
VCG	Vickrey-Clarke-Groves mechanism
UB	Upper bound

Declaration of Authorship

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Guanhua Wang

December 2021

Acknowledgements

Firstly, I want to thank my supervisor Dr. Mingyu Guo. I am honorable working with him in these three years. I very much appreciate his help. When I stock in no research inspiration, he can always figure it out and find a suitable way for me to solve the problems. Mingyu is a humble scholar. His research foresight and research inspiration is lifelong treasure for me.

I would thank Dr. Wei Zhang, and professor Muhammad Ali Babar. They provide some suggestions and comments that helped me to significantly improve the content of this thesis and papers.

And from my Ph.D. role, I also thank my collaborators, lab meta, and my friends for the help with the papers. I would like to mention Ba-Dung Le, Runqi Guo, Yuko Sakurai, Wuli Zuo, Xie Yue, Zhigang Lu for many useful discussions. My study area is not a hot area in this university. Thanks, you give me many useful discussions. Good luck to you in the future. And I will thank Wuli Zuo for checking and correcting my grammar mistakes.

I also want to thank my parents and my grandparents for financially supporting me during my Ph.D. study. And because of Covid-19, it is a hard time to connect you only on the internet. Hopefully, we can reunion soon after Covid-19.

Finally, I will thank the university and its staff. They provide a good environment for my research.

Publications

This thesis is based on the following research papers that have been published in peer-reviewed conferences or journals:

- **Guanhua Wang**, Runqi Guo, Yuko Sakurai, Muhammad Ali Babar and Mingyu Guo. Mechanism Design for Public Projects via Neural Networks. In Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021, Conference). (CORE Rank: A*)

- **Guanhua Wang**, Mingyu Guo. Public Project with Minimum Expected Release Delay. In the 18th Pacific Rim International Conference on Artificial Intelligence (PRICAI, Conference) 2021. (CORE Rank: B)

- **Guanhua Wang**, Wuli Zuo, Mingyu Guo. Redistribution in Public Project Problems via Neural Networks. In the 20th IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT, Conference), Melbourne, Australia, 2021. (CORE Rank: B)

- Mingyu Guo, **Guanhua Wang**, Hideaki Hata, M. Ali Babar. Revenue-Maximizing Markets for Zero-Day Exploits. In Autonomous Agents and Multi-Agent Systems (AAMAS, Journal) 35, 36, 2021. (CORE Rank: A)

Other research paper that has been published in peer-reviewed conferences but not used in this thesis:

- Ba-Dung Le; **Guanhua Wang**; Mehwish Nasim; Muhammad Ali Babar, "Gathering Cyber Threat Intelligence from Twitter Using Novelty Classification," 2019 International Conference on Cyberworlds (CW, Conference), 2019, pp. 316-323, (CORE Rank: B)

Chapter 1

Introduction

Mechanism design is a fundamental and important research area in economics. Since 2000, the Nobel Memorial Prize in Economic Sciences has been twice awarded for mechanism design. Nobel Memorial Prize 2007 in Economics was awarded to Leonid Hurwicz, Eric S. Maskin, and Roger B. Myerson for laying the foundations of mechanism design theory (Drobietz, Loerbroks, and Hansson, 2021). Then the Prize 2020 was awarded to Paul Milgrom and Robert Wilson for improving auction theory (Janssen, 2020). Mechanism design takes an objective-first approach to design economic mechanisms or incentives, toward desired objectives, in strategic settings, where players are individual rational (Lee, 2016).

Mechanism design theory has applications in a number of computer science sub-areas. Mechanism design studies how to make social decisions when we need to take individual preferences into consideration. Many problems in the computer science domain are related to mechanism design. For example, the popularity of e-commerce leads to a long list of new Internet-based markets, such as Advertisement auctions and blockchains. Besides new markets, many sub-areas of computer science involve mechanism design problems, such as multi-agent coordination in multiagent systems, distributed computing, and almost all domains involving competition (i.e., cyber security) and collaboration (i.e., federated machine learning and multiagent reinforcement learning). Furthermore, the latest development in computational techniques also leads to new development of economic results. The main topic of this thesis is along this line (i.e., using machine learning to develop new mechanisms). In this thesis, there is also discussion on applying mechanism design theory to cyber security.

This thesis focuses on the *public project* problem. To illustrate the public project model, we present the following example. Suppose there are n households (often referred to as n agents) living in a community. They plan to build a public project (a project shared by all households, e.g., a swimming pool). We assume that the project costs 1 dollar (without loss of generality). These agents plan to crowdfund this swimming pool. Only those who pay will have access. Every agent has a different valuation for the swimming pool, which is private information. We assume that agent i's valuation for the swimming pool is v_i. v_i is the maximum amount agent i is willing to pay to contribute to the pool/access the pool. A mechanism (social decision rule) maps the agents' preferences (i.e., the v_i) to the social decision (i.e., who can access the pool and how much each agent pays). One example design goal is to design a crowdfunding rule that maximizes the number of agents that can access the pool (i.e., in expectation with respect to the prior distributions of the v_i).

One example mechanism is simply to ask every agent to pay an equal share ($\frac{1}{n}$). If any agent disagrees, then we simply cancel the project. This way, no agents can benefit by lying (disagree with the proposed payment when the agent can afford it, or agree with the proposed payment when the agent cannot afford it) and the total payment collected is exactly 1, which meets the project cost (or 0 when the project is cancelled). Of course, this rule is not ideal because as long as there is one agent whose value is lower than $\frac{1}{n}$, then the project is not built. Given the goal of maximizing the number of agents who can consume the project, we want a rule that charges different agents different amounts (i.e., only agents who have high values for the pool need to pay more – we need to identify agents who can afford to pay more and force them to pay more).

The above task isn't actually easy, because we are designing a function (the crowdfunding social decision rule) that maps an input vector of dimension n (the v_i) to an output vector that describes every agent's allocation and payment (output dimension is $2n$). Furthermore, not all functions are feasible. It is a difficult task in itself to describe what feasible functions look like. We may add a constraint that is, for example, "no manipulation" (i.e., if an agent can afford to pay a high payment, then

we don't want this agent to pretend that her value is low by reporting a low value – recall that the agents' preferences are private information). It is difficult to mathematically derive the set of all functions that prevent manipulation and this is just for one mechanism design constraint. Later on in this thesis, we will formalize commonly used constraints in mechanism design.

In what follows, we present a timeline of mechanism design research.

Mechanism design theory originated from Hurwicz's pioneering work in 1960 (Hurwicz, 1960). For the general problem, it discusses whether and how to design an economic mechanism (such as an auction) for any given economic or social goal under decision-making conditions (such as incentive compatibility). Major mechanisms are under the incentive compatibility (IC) constraint to align the personal interests of economic participants with the mechanism design goals.

In 1967, Harsanyi (Harsanyi, 1967) defined the mechanism as a game of information involving the agents. The agents receive secret "messages" containing information relevant to payoffs. For example, a message may contain information about their preferences or the quality of a good for sale. This information is called the agent's "type". Agents then report a type to buy the goods. The reported types can be strategic false valuations or true valuations. After reporting, the agents are allocated according to the mechanism and are asked to pay accordingly (Figure 1.1). Mechanisms align the personal interests of agents with the mechanism design goal by encouraging agents to reveal their true valuations.

FIGURE 1.1. A typical mechanism payoff structure.

In 1981, Myerson et al. (Myerson, 1981) obtained the optimal auction mechanism

for single-item auctions. Since then, intense research has been carried out to solve optimal auctions in different settings. However, after four decades, for multi-agent or multi-item revenue-maximizing auctions, the optimal auctions remain unknown (Curry, Sandholm, and Dickerson, 2022). When it comes to the specific context of public project problems, designing a multi-agent mechanism meets more difficulties due to lack of characterisations of feasible mechanisms, as well as the variety of design objectives and the problem's high-dimension (when the number of agents is large) (Maghsoudlou, Afshar-Nadjafi, and Niaki, 2017; Carroll, 2019). For example, when we crowdfund a public project (i.e., a piece of software), the project could be non-excludable (the software is open source, therefore can be used by everyone, including the non-paying agents) or excludable (the software can only be used by agents who pay), indivisible (the software is a one-time sale for all agents) or divisible (the software is a subscription-based and an agent may enjoy "a fraction of the software" – i.e., subscribe half of the time).

It is difficult to analytically design public project mechanisms considering the aforementioned difficulties. We try to automatically design optimal or near-optimal mechanisms for the public project problem via machine learning methods. We propose several neural network training techniques for designing public project mechanisms, which also have the potential to be generalized to other mechanism design models. Besides neural networks, we also develop automated mechanism design techniques based on evolutionary computation. For instance, for an example application setting on zero-day exploit markets, we focus on the *Affine Maximizer Auctions* (AMA) (Likhodedov and Sandholm, 2005; Curry, Sandholm, and Dickerson, 2022; Lavi, Ahuva Mu'alem, and Nisan, 2003) and use the Fourier series to characterize the AMA mechanisms. We then adjust the AMA mechanism parameters via evolutionary computation.

1.1 Public Project Mechanism Design Motivation and Research Challenges

In a public project model, a group of agents need to decide whether or not to crowdfund a public project (that is often accessible by every agent once the project is funded), and the mechanism design task is to design a function that maps the agents' private preferences to social decisions (i.e., whether the project is built and how much each agent pays), where the goal is often to maximize agents' welfare. The social decision rule is often also subject to a list of application-specific constraints. The public project model is a well-studied economic model and it also finds application in the computer science domain (i.e., crowdfunded bug bounty systems). In what follows, we summarize the research challenges of mechanism design for the public project problem.

1.1.1 Unknown Feasible/Optimal Mechanism Characterizations

The first difficulty lies in mechanism design itself. Mechanisms collect the valuation from each agent, and design payment and allocation schemes (Figure 1.1). It is obvious that, in the design of the mechanism, the input side is the valuations from the agents, which the mechanism designer can never assign or design. To achieve the desired objectives, the mechanism designer needs to get the true valuations from the agents. The only way we encourage the agents to tell the truth is by controlling the two output functions (allocation and payment functions). The designed allocation and payment functions must ensure that each agent can get the highest utility whenever she reports the true valuation.

Since Myerson solved the single-item optimal auction design problem in his seminal work (Myerson, 1981), intense research has been taken to solve problems under different settings. However, the problem is not completely resolved (Sandholm, 2003). For public project problems, for some variants of the model, we do not know the exact characterizations of the truthful mechanisms. Even when we know the exact characterization of the truthful mechanism, the characterization is difficult to work

with. For example, the characterization may contain an exponential number of parameters, and there are no known techniques for efficient optimisation that works with the characterization. This is different from Myerson's result for single-item auctions, where the proposed optimisation technique allows us to focus on very restricted forms of mechanisms.

1.1.2 Neural Network Mechanism Design Challenges for Public Project Problems

One of the characteristics of the public project problem is that the mechanism needs to decide whether or not to build the project (Sen, 2007). When we apply neural networks to design public project mechanisms, binary decision (build vs not build) often lead to entirely different gradients for the loss function. In stochastic gradient descent, we can only use a small batch of type profiles, which means that often the number of "build" cases plays a more important role than the quality of the mechanism when it comes to calculating the gradient. This causes poor training results and slow training speed.

1.1.3 High Variety of Public Project Problems

In many mechanism design problems, the typical objectives are to maximize social welfare or to maximize revenue (Zou, 2009; Manelli and Vincent, 2007; Amanatidis, Kleer, and Schäfer, 2019; Amanatidis, Birmpas, and Markakis, 2017). In this thesis, we deal with public project problems with a few more objectives besides maximizing for welfare and revenue. Our objectives studied also include minimizing the total and maximum release delay (for public projects where the mechanism designer can set a release time) and maximizing the number of consumers for the public project. Existing manual mechanisms designed by the economists perform poorly toward our proposed objectives.

The performance of mechanisms is often impacted by the number of agents (Zhao, 2014). For example, if there are a large number of agents joining the public project

problem, then the project has a high probability to be built. We could even entirely ignore the "not build" possibility when designing mechanisms. This observation allows us to design asymptotically optimal mechanisms for a few variants of the model. The challenging design cases are sometimes only about small numbers of agents. This plays well with our approach of using neural networks and evolutionary computation to automatically design mechanisms.

1.2 Research Summary

In this thesis, we focus on automatically designing mechanisms for the public project problem (Mezzetti, 2004). In what follows we describe all the variants of the public project models that are studied in this thesis. We also summarize our mechanism design constraints, design objectives, and also our high-level mechanism optimisation techniques. We conclude this section with a detailed list of contribution outlines of the future chapters. While summarizing our research in this section, we briefly define the relevant terminologies when necessary. To avoid breaking the presentation flow, we leave a more detailed discussion of background information and related research to Chapter 2.

1.2.1 Summary of Public Project Problem Variants

A public project problem is a social decision model where a group of agents have to decide whether or not to build a public project (e.g., a swimming pool, public library, open source software, etc., Kaiser, 2007).

Public project problems have many variants. In this thesis we consider all the following:

1. The project may be *non-excludable* or *excludable* (Samuelson, 1954; Ott and Turnovsky, 2006)
2. The project may be *indivisible* or *divisible* (Lipton et al., 2004)
3. We assume that the project is *binary* and *non-rivalrous* (Samuelson, 1954; Wadhwa and Dong, 2020)

Terminology explanations:

- Non-excludable: it is impossible to exclude any individuals from consuming the good. (Samuelson, 1954; Ott and Turnovsky, 2006; Ohseto, 2000)

- Excludable: the mechanism designer can specify which agents can consume the project, for example, it could be that only those who pay for the project have access to its benefits. (Ott and Turnovsky, 2006)

- Non-rivalrous: when one agent consumes the project, others are not prevented from using it. (Samuelson, 1954; Wadhwa and Dong, 2020)

- Binary: a public project is built or not built. A non-binary public project refers to a project with different provisional levels. For example, the agents may face building a better swimming pool, a poor-quality swimming pool, or not build at all. (Lipton et al., 2004)

- Indivisible: a public project can not be divided. For example, the allocation for an agent is either 1 (this agent uses the entire project) or 0 (this agent can not use the project at all). To be more specific, an agent only has two options, consumes the entire project or does not consume the project at all. (Lipton et al., 2004)

- Divisible: a public project can be divided. For example, one agent may consume or use part of the project. The allocation to an agent is in the range from [0,1].

1.2.2 Summary of Mechanism Constraints and Objectives

The mechanism designer expects the agents to report their valuations truthfully so that the overall system objective can be achieved. However, an individual agent wants to maximize her own utility (i.e., receive better allocation and pay less). Therefore, an agent may have the incentive to lie about her true valuation if doing so increases her utility (Matsushima, 2007). An agent also may refuse to join a mechanism if she deems that joining the mechanism leads to a worse utility compared to her utility when staying out. When designing mechanisms, we enforce two constraints: Strategy-Proofness (SP) (Ma, 1994) and Individual Rationality (IR).

- *Strategy-proofness (SP)*: For any agent i, her utility is maximized by reporting her valuation truthfully.

- *Individual rationality (IR)*: For any agent i, her utility is nonnegative when she reports truthfully.

As mentioned earlier, for many mechanism design settings, coming up with a characterization of all strategy-proof and individual rational mechanisms is often a difficult task, which then makes mechanism design under these two constraints difficult.

In the table below, we summarize the mechanism design constraints and objectives used in each chapter.

	SP	IR	Objectives (goals)
Chapter 3	✓	✓	1. Maximize Expected Number of Consumers 2. Maximize Expected Agents' Welfare
Chapter 4	Study Both SP and Almost SP	✓	1. Minimize Expected Total Delay 2. Minimize Expected Max Delay
Chapter 5	✓	IR in Expectation	1. Worst-Case Efficiency Ratio 2. Expected Welfare
Chapter 6	✓	✓	1. Revenue-Maximizing

TABLE 1.1. Mechanism Constraints and Objectives for Each Chapter

1.2.3 Contributions and Outlines of Individual Chapters

In this thesis, we discuss various different methods for automatically designing mechanisms. Our aim is to improve the performance of public project mechanisms, and provide optimal or near-optimal public project mechanisms for real-world applications. We adopt different neural network and evolutionary computation frameworks to automatically design mechanisms for public project problems, while considering different objectives, constraints and model variety (Table 3.1 and Table 1.1).

A commonly used approach in this thesis is that for a small number of agents, we use automated mechanism design to derive mechanisms for our objectives. For a large number of agents, We try to theoretically analyse the automated designed mechanisms and calculate the theoretical performance upper bound. We compare our

mechanisms to the theoretical upper bound and/or existing state-of-the-art mechanisms to demonstrate the effectiveness of our proposed mechanisms.

In Table 1.2, we highlight the mechanism families considered (i.e., we would focus on these families of mechanisms, and then optimize within them) and also our machine learning techniques used.

	Mechanism Families	Machine Learning Approach
Chapter 3	Largest Unanimous Mechanisms	Neural Networks
Chapter 4	Sequential Unanimous Mechanisms	Evolutionary Computation
Chapter 5	VGC Redistribution Mechanisms	Neural Networks
Chapter 6	AMA mechanisms	Evolutionary Computation + Neural Network

TABLE 1.2. Mechanism Families and Machine Learning Techniques used in Each Chapter

Below please find a detailed summary of our contributions in future chapters.

- In Chapter 3, we study both the **non-excludable** and the **excludable** versions of the *binary non-rivalrous indivisible* public project problem. The existing methods (e.g., serial cost-sharing mechanism, Moulin, 1994) are not optimal in general. We identify a sufficient condition on the prior distribution for the conservative equal costs mechanism to be the optimal strategy-proof and individually rational mechanism. We prove that, in some scenarios, the serial cost-sharing mechanism is optimal. For other scenarios, we find that the serial cost-sharing mechanism is far from optimality. For these scenarios, we design better-performing mechanisms via neural networks.

  For non-excludable public project problems, we improve existing mechanisms by using dynamic programming (DP). For excludable problems, we design a better-performing price-oriented rationing-free (PORF) mechanisms via neural networks. We propose three neural network training techniques. The experiments show that our mechanisms are better than previous results and are closer to the theoretical upper bound.

Contributions:

1. We are the first to use the price-oriented rationing-free (PORF) mechanisms to assist the designing of iterative mechanisms via neural networks. PORF mechanisms move the mechanism's complex (*e.g.*, iterative) decision-making process off the neural network to a separate simulation process.

2. We feed prior distribution into the loss function. By feeding the prior distribution's *analytical form* into the cost function, we can provide high-quality gradients.

3. We learn toward existing manual mechanisms as initialization. We use supervision to state-of-the-art manual mechanisms as a systematic way for initialization. It is also effective in fixing constraint violations for heuristic-based mechanisms.

- In Chapter 4, we study a divisible public project model, where an agent can be allocated only part of the project. Specifically, we study settings where the mechanism can set different project release times for different agents. For an agent, the higher she pays, the earlier she can use the project. There are two objectives for this problem: to minimize the expected maximum release delay and to minimize the expected total release delay. The existing mechanisms (without delayed release) can be trivially applied to our model by interpreting them as mechanisms whose release times are either 0s or 1s. Nevertheless, as expected, existing mechanisms do not perform well as they were not designed for our objectives.

  Under our automated mechanism design approach, we first find regularities by analysing cases involving a small number of agents and then generalise the rule to a larger number of agents. For small numbers of agents, we propose the sequential unanimous mechanisms by extending the existing largest unanimous

mechanisms. We then use evolutionary computation to optimize within the sequential unanimous mechanism family. The experiments show that our mechanisms are better than the existing mechanisms. Then we summarize the patterns of these sequential unanimous mechanisms and apply them to a larger number of scale. We end up with the single deadline mechanisms. We theoretically prove that the single deadline mechanisms are asymptotically optimal, regardless of the prior distributions.

Contributions:

1. For a small number of agents, we propose the sequential unanimous mechanism family and apply automated mechanism design via evolutionary computation.

2. For a large number of agents, we propose a novel single deadline mechanism, which is asymptotically optimal.

- In Chapter 5, we focus on the VCG and the VCG redistribution (which is based on VCG) mechanisms for the public project problem. We design mechanisms via neural networks with two welfare-maximizing objectives: optimal in the worst case and optimal in expectation. We design generative adversarial networks and multi-layer perceptions (GAN + MLP) to find the optimal worst-case performance of VCG redistribution mechanisms for public project problems. GAN is used to generate type profiles with poor worst-case performances, which are used for training. The experiments show that our mechanism is better than existing approaches. We use multi-layer perceptions (MLP) to find the optimal-in-expectation VCG redistribution mechanism. Our innovation is a new way to construct the cost function for training, by including the prior distribution of the agent valuations as weights for a training batch. The experiments show that our mechanisms' performances are very close to the theoretical upper bound.

  Contributions:

1. We are the first to utilise a GAN network to generate worst-case type profiles for training toward worst-case optimal mechanisms.

2. We feed prior distribution into loss function to provide quality gradients for the optimal-in-expectation objective.

3. We discuss dimension reduction for handling large agent count. By reducing the input dimension, the neural network converges faster and still retains good performance.

- In Chapter 6, we focus on a specific scenario called zero-day exploits market. In such a market, one zero-day exploit (i.e., an exploit that allows cyber attackers to hack into iOS systems) is sold to multiple offender and defenders. In this model, for the defensive side, as long as any defender gains access to the exploit, the exploit is assumed to be immediately fixed, which benefits all defenders. The defensive side of the our model is very similar to a non-excludable and divisible public project problem. Nevertheless, the overall model is not a public project problem.

  In order to maximize revenue in zero-day exploit markets, we adopt computational feasible technical automated mechanism design approaches (Guo and Conitzer, 2010; Guo, Hata, and Babar, 2017). One commonly used mechanism family for revenue maximization is the Affine Maximizer Auctions (AMA). For this particular model, we observe that an AMA mechanism can be characterized by a single variable function that can be visualized as a curve. We propose two numerical solution techniques, one is based on neural networks and the other one is based on evolutionary computation. We use neural networks to automatically design the optimal curve for the Affine Maximizer Auctions (AMA) mechanisms. We also use evolutionary computation (based on Fourier series) to optimize for a good AMA curve. The experiments show that our mechanisms based on neural networks and evolutionary computation are near-optimal and

get better results compared to the state-of-the-art method, which was based on iterative linear programming (LP).

Contributions:

1. We are the first to use a series of novel techniques to train the AMA mechanisms, including Fourier-series-based evolutionary computation and neural networks.

- In Chapter 7, we summarize this thesis and discuss future directions.

Chapter 2

Literature Review and Definitions

2.1 Mechanism Design

Mechanism design is the art of designing social decision rules so that the agents are motivated to report their true valuations and a suitable (according to a given objective) outcome is chosen. The objectives are varied. It could be social welfare, fairness, or revenue maximization (Cole and Roughgarden, 2014; Morgenstern and Roughgarden, 2015).

FIGURE 2.1. A typical mechanism structure.

Generally speaking, mechanisms specify payments and allocations (as outlined in Figure 2.1). For a certain agent, the higher she pays, the better allocation she gets. For different agents, they may gain different allocations even if they pay the same.

For example, for the public project problem, a mechanism would specify whether the project is built or not, who can consume the project (if the project is excludable), and how much each agent needs to pay.

2.1.1 Mechanism Properties

Self-interested agents may lie about their valuations if doing so increases their own utilities. Therefore, it is necessary to design mechanisms with desired properties. In

this thesis, we focus on designing mechanisms that are strategy-proof and individually rational (Yokoo, Sakurai, and Matsubara, 2004; Dash, Ramchurn, and Jennings, 2004; Azevedo and Budish, 2019).

Individual Rationality

Individual rationality (IR) means that it is unacceptable for an agent to receive less utility than that she would have received if not joining in the mechanism. Under individually rational mechanisms, each individual weakly prefers to join in the mechanism rather than not participating (Dash, Ramchurn, and Jennings, 2004; Anand et al., 1995).

The u_i is agent i's utility. v_i is agent i's type profile. A mechanism is called individual rationality, if for every player i and for every type profile of the other players v_{-i}:

$$u_i(v_i, v_{-i}) \geq 0, \forall i$$

The utility for each agent i is greater or equal to 0.
In this thesis, we don't consider the external utility.

Strategy-proofness

Strategy-proofness (SP) is also called truthfulness or *dominant-strategy incentive-compatibility* (DSIC). It means that no agents can benefit by misreporting their preferences (Azevedo and Budish, 2019; Mookherjee and Reichelstein, 1992). Roughgarden (Roughgarden, 2010) summarized the commonly used definitions of incentive compatibility. The weaker notion is Bayesian-Nash incentive compatibility and the stronger notion is *dominant-strategy incentive compatibility* (also often called strategy-proofness). Typical examples of SP mechanisms are the VCG mechanisms.

A mechanism is called strategyproof, if for every player i and for every type profile of the other players v_{-i}:

$$u_i(v_i, v_{-i}) \geq u_i(v_i', v_{-i}), \forall i'$$

The agent i get best or at least not worse by being truthful (report truthful value v_i), regardless v_{-i}.

In 1981, Myerson (Myerson, 1981) solved the optimal auction design problem when there is a single item for sale. However, after four decades, for multi-item revenue-maximizing auctions, the optimal solutions remain unknown (Curry, Sandholm, and Dickerson, 2022). Researchers have developed some elegant partial characterization results (Manelli and Vincent, 2007; Pavlov, 2011; Yao, 2017) or have developed impressive algorithmic optimal or near-optimal mechanisms (Cai, Daskalakis, and Weinberg, 2012; Hart and Nisan, 2017) for some specific settings. For the public project problem, there are limited characterization results and also limited existing manual mechanisms.

2.1.2 Analytical Mechanism Design

Mechanism design has traditionally been based on manual effort and human-expert experience (Sandholm, 2003). The designer manually designs a certain rule set and then proves that the proposed mechanism is optimal or near-optimal. Often, the designer studies the mechanism design problem from a mathematical structure perspective (Krishna, 2009). Indeed, many impressive and significant mechanisms were designed this way. Armstrong obtained the revenue optimal mechanisms for selling two items to one buyer (Armstrong, 1996). Pavlov et al. (Pavlov, 2011) derived optimal results for two items with symmetric uniform distributions. In this thesis, we also conduct some level of manual mechanism design for certain public projects, and then theoretically prove by mathematical analysis that the derived mechanism is optimal in certain situations or under certain technical assumptions.

2.1.3 Automated Mechanism Design

In 2002, Conitzer et al. (Conitzer and Sandholm, 2002) first proposed the idea of *automated mechanism design* (AMD). The designer first needs to design a general

framework and a given objective for the problem. Then the designer leaves the optimisation process to the computer, and the computer will automatically calculate the suitable parameters and decide on the details of the mechanisms.

Since the proposal of AMD, many researchers used AMD for mechanism design. In 2006, Jurca and Faltings (Jurca and Faltings, 2006) applied Automated Mechanism Design to compute the minimum payments for a reputation system. In 2007, Constantin and Parkes (Constantin and Parkes, 2007) studied dynamic single-item auctions using automated mechanism design for interdependent value agents. The authors used mixed-integer programming to handle a small number of agents. In 2010, Bhattacharya et al. (Bhattacharya et al., 2010) used AMD to study revenue-maximizing multi-item auctions for budget-constrained agents. Likhodedov et al. (Likhodedov and Sandholm, 2005, 2004) applied automated mechanism design to maximize revenue for combinatorial auctions. The authors focused on the family of Affine Maximizer Auctions (AMA), and AMD was used to automatically adjust the mechanism parameters.

2.1.4 Classic Mechanisms

In this section, we will introduce a few well-known/state-of-the-art mechanisms that are relevant to this thesis.

Cost Sharing Mechanisms

Cost sharing mechanisms are mechanisms that can be applied to binary indivisible public project models. Cost sharing mechanisms are **strategy-proof** and **individual rational** (Moulin, 2005). We summarize a few existing cost sharing mechanisms as follows:

1. ***Largest Unanimous Mechanism*** **(Ohseto, 2000)**

   For every nonempty coalition of agents $S = \{S_1, S_2, \ldots, S_k\}$, there is a constant cost share vector $C_S = (c_{S_1}, c_{S_2}, \ldots, c_{S_k})$ with $c_{S_i} \geq 0$ and $\sum_{1 \leq i \leq k} c_{S_i} = 1$. c_{S_i} is agent S_i's cost share under coalition S. If agent i belongs to two coalitions S

and T with $S \subsetneq T$, then i's cost share under S must be greater than or equal to her cost share under T. Agents in S unanimously approve the cost share vector C_S if and only if $v_{S_i} \geq c_{S_i}$ for all $i \in S$. The mechanism picks the largest coalition S^* satisfying that C_{S^*} is unanimously approved. If S^* does not exist, then the decision is not to build. If S^* exists, then it is always unique, in which case the decision is to build. Only agents in S^* are consumers and they pay according to C_{S^*}.

2. ***Serial Cost Sharing Mechanism (SCS)*** **(Moulin, 1994; Guo, Yang, and Ali Babar, 2018)**

The serial cost sharing mechanism is a specific largest unanimous mechanism. For every nonempty subset of agents S with $|S| = k$, the cost share vector is $(\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k})$. The mechanism picks the largest coalition S^* where the agents are willing to pay equal shares.

- If S^* is empty, then the project will not be built
- If S^* is not empty, then every agent in S^* each pays an equal share and only those who pay can consume the project.

The serial cost sharing mechanism is elegant and we also prove that in many situations, they are in fact optimal/near-optimal.

VCG and VCG Redistribution Mechanisms

1. ***VCG Mechanism***

The Vickrey-Clarke-Groves (VCG) mechanism is efficient and strategy-proof but not budget-balanced (Vickrey, 1961; Clarke, 1971; Groves, 1973).

Under the VCG mechanism, each agent i reports her private type θ_i. The outcome that maximizes the agents' total valuations is chosen. Every agent is required to make a VCG payment, which is determined by the other agents'

types θ_{-i}. An agent's VCG payment is usually described as the extent to which the agent's existence harms the other agents, in terms of the total valuation of other agents, which is called the externality in economics. The total VCG payment may be quite large, leading to decreased welfare for the agents. In particular, in the context of the public project problem, where the goal is often to maximize the social welfare (the agents' total utility considering payments), having large VCG payments are undesirable.

2. ***VCG Redistribution Mechanisms***

VCG redistribution mechanisms are also efficient, strategy-proof and not budget-balanced (Guo and Conitzer, 2008; Guo, 2011). They are proposed to address the social welfare loss due to large VCG payments.

Under a VCG redistribution mechanism, we first run VCG, and then redistribute the VCG payments back to the agents as much as possible. The amount that every agent receives (or pays additionally) is called the redistribution payment, and is characterized by a redistribution payment function h. For agent i, its redistribution payment depends on other agents' types θ_{-i}. In this way, the VCG redistribution mechanism remains to be strategy-proof (Naroditskiy et al., 2012).

The VCG redistribution mechanism may bring the agents more benefit due to the redistribution payment, so the total social welfare could be increased compared to the VCG mechanism.

Affine Maximizer Auctions (AMA)

Myerson's (Myerson, 1981) solved for the optimal single-item auction. For combinatorial auctions, Myerson's technique does not generalize beyond single-parameter settings. Revenue maximizing mechanism design remains an open problem. Many revenue-boosting techniques were proposed by researchers. The Affine Maximizer Auctions (AMA) mechanisms are a family of strategy-proof mechanisms that are characterized by a set of parameters (Likhodedov and Sandholm, 2005). By focusing

on the AMA mechanisms, the mechanism designer can focus on tuning the mechanism parameters in order to achieve better revenue. This way, mechanism design is no longer a functional optimisation process, but rather a value optimisation process, which is often much easier.

The family of AMA mechanisms for the public project model was formally defined by Guo et al. (Guo, Hata, and Babar, 2016) as follows:

AMA Mechanisms

- Given a type profile θ, the outcome picked is the following:

$$o^* = \arg\max_{o \in O} \left(\sum_{i=1}^{n} u_i v_i(\theta_i, o) + a_o \right)$$

- Agent i's payment equals:

$$\frac{\max_{o \in O} \left(\sum_{j \neq i} u_j v_j(\theta_j, o) + a_o \right) - \sum_{j \neq i} u_j v_j(\theta_j, o^*) - a_{o^*}}{u_i}$$

Here, O represents the outcome space, Θ_i represents agent i's type space, and $v_i(\theta_i, o)$ represents agent i's valuation for outcome $o \in O$ when her type is $\theta_i \in \Theta_i$.

2.2 Machine Learning Methods for Automated Mechanism Design

Machine learning is a useful tool for finding an acceptable or near-optimal results for optimization problems. Typical machine learning methods include evolutionary computation algorithm, particle swarm optimization algorithm, artificial neural network algorithm, etc. (Mohri, Rostamizadeh, and Talwalkar, 2018)

In 2003, Sandholm (Sandholm, 2003) reported that search algorithms is a new approach to solve the automated mechanism design problem. In 2005, Balcan et al. (Balcan et al., 2005) obtained a unified approach for a variety of revenue-maximizing mechanism design problems. They used sample-complexity techniques in machine

learning theory to simplify the design of revenue-maximizing incentive-compatible mechanisms to standard algorithmic questions. Various novel machine learning methods have been proposed for AMD (Narasimhan, Agarwal, and Parkes, 2016). In the current "Era of Neural Networks" (Sharma and Singh, 2017), many researchers used machine learning methods to find optimal mechanisms (Cai, Daskalakis, and Weinberg, 2012; Alaei et al., 2012) or near-optimal mechanisms (Hart and Nisan, 2017; Hartline and Roughgarden, 2009; Yao, 2014).

In this thesis, we mainly use evolutionary computation and artificial neural networks to find optimal/near-optimal mechanisms for public project problems.

2.2.1 Evolutionary Computation

Evolutionary computation techniques can produce highly optimized solutions by mutation and crossover, which generates new genotypes to find good solutions for a given problem. Evolutionary computation is widely used for a long range of computational tasks (Neumann et al., 2019; Long et al., 2020; Do et al., 2021). Researchers can apply evolutionary computation to mechanism design by treating mechanism design as an engineering problem and bring in engineering design principles (Phelps, McBurney, and Parsons, 2010). We categorize these approaches under the banner of evolutionary mechanism design. Andrews (Andrews, 1994) is the first to apply evolutionary computation to the double-auction design problem with a view to automating the mechanism design process. Cliff (Cliff, 1998) used evolutionary search to explore the parameter space of the zip strategy. After that, more researchers used evolutionary computations to study mechanism design problems (Conitzer and Sandholm, 2007; Cliff, 2002; Phelps et al., 2002, 2003).

Evolutionary computation (in the context of mechanism design) usually involves the following steps:

1. Initialization: An initial batch of mechanisms is created when evolutionary computation starts.

2. Genetic operators (such as crossover and mutation): Small random changes are introduced to the existing mechanisms in the population (Fox and McMahon, 1991).

3. Selection: As weaker mechanisms are stochastically removed, the mechanism population becomes refined.

2.2.2 Artificial Neural Network

A neural network is often formed by thousands of neurons, which are grouped into different layers to pass and process data from the input layer to the output layer. The internal layers are called hidden layers and usually, they are fully connected with their neighbour hidden layers. This pattern forms the structure of the network. Activation functions are often applied to every layer (Sharma and Singh, 2017).

Recent development of automated mechanism design capitalizes on deep learning. Dütting et al. (Dütting et al., 2019) in 2019 first used the neural network (called **RegretNet**) to solve automatic mechanism design problems for learning approximately strategy-proof auctions for multi-bidder multi-item auctions. They also proposed another neural network (called **RochetNet**) for a single bidder, which is perfectly strategy-proof ensured by network construction. Feng et al. (Feng, Narasimhan, and Parkes, 2018) and Golowich et al. (Golowich, Narasimhan, and Parkes, 2018a) extended **RegretNet** to deal with different constraints and objectives. Sakurai et al. (Sakurai et al., 2019) adopted the idea of **RegretNet** to design false-name-proof mechanisms, where false-name-proof violations are added to the cost function, which is then minimized to remove false-name manipulation. Curry et al. (Curry et al., 2020) modified **RegretNet** to be able to verify strategy-proofness of the auction mechanism learned by neural network. Peri et al. (Peri et al., 2021) developed **PreferenceNet** to encode human preference (e.g. fairness) constraints into **RegretNet**.

Later on, many other researchers explored the use of neural networks in automating mechanism design (Manisha, Jawahar, and Gujar, 2018; Shen, Tang, and Zuo, 2018; Rahme, Jelassi, and Weinberg, 2020; Duan et al., 2022; Rahme et al., 2020;

Zhan and Zhang, 2020; Bichler et al., 2021; Brero et al., 2020).

For example, Shen et al. (Shen, Tang, and Zuo, 2018) used neural networks (called **MenuNet**) to automatically design revenue optimal mechanisms for multi-item revenue optimization settings (selling two items to one buyer). They theoretically proved that the mechanism framework is indeed optimal. Rahme et al. (Rahme, Jelassi, and Weinberg, 2020) proposed **ALGNet** to solve two-player game through parameterizing the misreporter. Then Rahme et al. (Rahme et al., 2020) proposed a permutation-equivariant architecture called EquivariantNet for symmetric auctions. Duan et al. (Duan et al., 2022) proposed **CITransNet** which focused on permutation equivariant auctions without symmetric constraints. Zhan et al. (Zhan and Zhang, 2020) used a reinforcement learning-based (DRL-based) solution that can automatically learn the best pricing strategy. Bichler et al. (Bichler et al., 2021) used neural network to study Bayesian Nash equilibrium strategies. They developed a neural pseudogradient ascent (NPGA) methods to learn the equilibrium bid functions.

All evidences suggest that, for many settings, it is possible for the neural network to find optimal or near-optimal truthful mechanisms with proper loss functions and network structures.

For example, **RegretNet** (Dütting et al., 2019) used loss function to ensure approximate strategy-proofness. Whenever the training samples cause truthfulness violations, RegretNet reports a large loss so loss function can force the neural network based mechanism to learn toward strategy-proofness.

Another way to achieve truthfulness is via adopting specific "truthful" neural network structures. Both **RochetNet** (Dütting et al., 2019) and **MenuNet** (Shen, Tang, and Zuo, 2018) were restricted to a single agent, but enforced strategy-proofness at the architectural level (Curry, Sandholm, and Dickerson, 2022). **ALGNet** and **CITransNet** focused on a small number of agents (such as two bidders) for permutation or symmetric equivariant settings. In this thesis, we propose a **PORF network** which is a strategy-proof neural network enforced by network structure.

As an example truthful neural network, Figure 2.2 shows how Manisha et al. used the network structure to ensure the strategy-proof constraint (Manisha, Jawahar, and Gujar, 2018). The input for network is θ_{-i} (n-1 dimensions), and the output (redistribution payment) for agent i is $h(\theta_{-i})$ which is independent from agent i's type θ_i. Since an agent cannot impact her redistribution function under this network, we know for sure that no matter how the parameters are trained, the overall mechanism is always truthful.

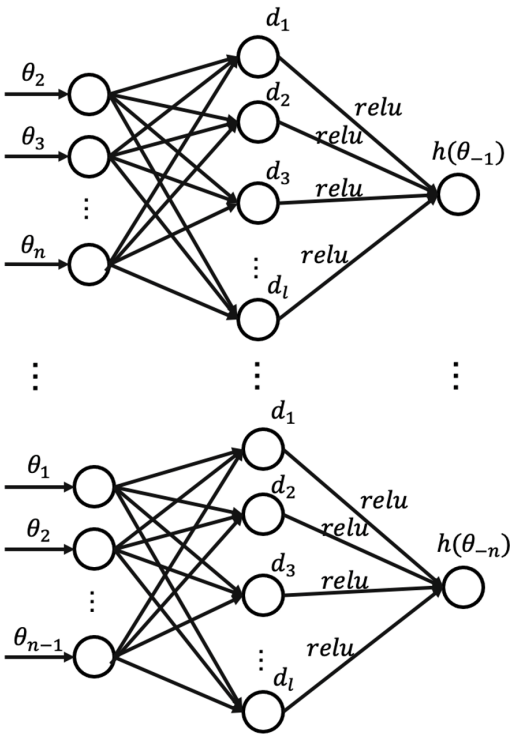


FIGURE 2.2. Manisha's nonlinear network model (Manisha, Jawahar, and Gujar, 2018)

In this thesis, we proposed a few novel neural networks for automated mechanism design, such as networks based on price-oriented rationing-free (PORF) interpretation of strategy-proof iterative (largest unanimous) mechanisms (Yokoo, Sakurai, and Terada, 2002; Yokoo, 2003), and GAN-assisted networks for worst-case mechanism design, where generative adversarial network (GAN) is used to generate worst-case type profiles (Goodfellow et al., 2014, 2020).

Unsupervised Neural Network

Traditionally, neural networks are supervised with labeled training data for recognition tasks. For example, image/face recognition started off as pure supervised but has become hybrid by adopting unsupervised pre-training (Martins, Pires, and Pires, 2007). Typical unsupervised neural network methods include autoencoders, deep belief networks, generative adversarial networks (GAN), and self-organizing maps, etc. (Mohri, Rostamizadeh, and Talwalkar, 2018)

Generative Adversarial Network (GAN)

In 2014, Goodfellow et al. (Goodfellow et al., 2014) first proposed a creative neural network structure to find the "worst cases" for a neural network (discriminator network). Then it immediately became one of the most important neural network frameworks (Yu et al., 2017; Yi, Walia, and Babyn, 2019). Generative adversarial networks (GAN) usually include two different neural networks: a generator network and a discriminator network.

The generator network generates candidates' data while the discriminator network evaluates them. Specific labeled training data are not necessary for GAN. The generator is trained based on whether it succeeds in fooling the discriminator. The discriminator network is trained with samples from the generator and sometimes from the initial training data set. The generator and the discriminator are trained alternatively in each epoch.

Chapter 3

Mechanism Design for Public Projects via Neural Networks

3.1 Introduction

Many multiagent system applications (*e.g.*, crowdfunding) are related to the public project problem. The public project problem is a classic economic model that has been studied extensively in both economics and computer science (Mas-Colell, Whinston, and Green, 1995; Moore, 2006; Moulin, 1988). Under this model, a group of agents decide whether or not to fund a *nonrivalrous* public project — when one agent consumes the project, it does not prevent others from using it.

Table 3.1 highlights the differences in terms of model settings among the different chapters. In each chapter, we study a different public project model variant.

	Binary	Non-rivalrous	Divisible\Indivisible	Excludable\Non-excludable
Chapter 3	✓	✓	Indivisible	Study Both
Chapter 4	✓	✓	Study Both	Excludable
Chapter 5	✓	✓	Indivisible	Non-excludable
Chapter 6 [*]	/	/	Divisible	/

TABLE 3.1. Public Project Model Variants for Each Chapter

[*] In Chapter 6, the general problem is not a public project problem. However, from the perspective of the defenders, they face a model that **highly resembles** a divisible public project model.

In this chapter, We study both the **non-excludable** and the **excludable** versions of the *binary non-rivalrous indivisible* public project problem. The binary decision is either to build or not. If the decision is not to build, then no agents can consume the project. For the *nonexcludable* version, once a project is built, all agents can consume it, including those who do not pay. For example, if the public project is an open source software project, then once the project is built, everyone can consume it. For the *excludable* version, the mechanism has the capability to exclude agents from the built project. For example, if the public project is a swimming pool, then we could impose the restriction that only the paying agents have access to it.

Our aim is to design mechanisms that maximize *expected* performances. We consider two design objectives. One is to maximize the **expected number of consumers** (expected number of agents who are allowed to consume the project).[1] The other objective is to maximize the agents' **expected agents' welfare**. We argue that maximizing the expected number of consumers is *more fair* in some applications. When maximizing the number of consumers, agents with lower valuations are treated as important as high-value agents.

With slight technical adjustments, we adopt the existing characterization results from Ohseto (Ohseto, 2000) for *strategy-proof* and *individually rational* mechanisms for both the nonexcludable and the excludable public project problems. Under various conditions, we show that existing mechanisms or mechanisms derived via classic mechanism design approaches are optimal or near optimal. When existing mechanism design approaches do not suffice, we propose a neural network based approach, which successfully identifies better performing mechanisms. Mechanism design via deep learning/neural networks has been an emerging topic (Manisha, Jawahar, and Gujar, 2018; Golowich, Narasimhan, and Parkes, 2018b; Duetting et al., 2019; Shen, Tang, and Zuo, 2019). Duetting *et.al.* (Duetting et al., 2019) proposed a general approach for revenue maximization via deep learning. The high-level idea is to manually construct often complex network structures for representing mechanisms for different auction types. The cost function is the negate of the revenue. By minimizing

[1] For the nonexcludable version, this is simply to maximize the probability of building.

the cost function via gradient descent, the network parameters are adjusted, which leads to better performing mechanisms. The mechanism design constraints (such as strategy-proofness) are enforced by adding a penalty term to the cost function or using architecture. The penalty is calculated by sampling the type profiles and adding together the constraint violations. Due to this setup, the final mechanism is only approximately strategy-proof. The authors demonstrated that this technique scales better than the classic mixed integer programming based automated mechanism design approach (Conitzer and Sandholm, 2002). Shen *et.al.* (Shen, Tang, and Zuo, 2019) proposed another neural network based mechanism design technique, involving a seller's network and a buyer's network. The seller's network provides a menu of options to the buyers. The buyer's network picks the utility-maximizing menu option. An exponential-sized hard-coded buyer's network is used (*e.g.*, for every discretized type profile, the utility-maximizing option is pre-calculated and stored in the network). The authors mostly focused on settings with only one buyer.

Our approach is different from previous approaches, and it involves three technical innovations, which can be applied to neural network-based mechanism design in general.

1. Calculating mechanism decisions off the network by interpreting mechanisms as price-oriented rationing-free (PORF) mechanisms (Yokoo, 2003)
2. Feeding prior distribution into the cost function
3. Supervisions to manual mechanisms as initialization

Calculating mechanism decisions off the network by interpreting mechanisms as price-oriented rationing-free (PORF) mechanisms (Yokoo, 2003) : A mechanism often involves binary decisions. A common way to model binary decisions on neural networks is by using the *sigmoid* function (or similar activation functions). A mechanism may involve a complex decision process, which makes it impractical to model via *static*

neural networks. For example, for our setting, a mechanism involves *iterative* decision making where the number of "rounds" depends on the agents' types. We could stack multiple sigmoid functions to model this. However, stacking sigmoid functions leads to vanishing gradients and significant numerical errors. Instead, we rely on the PORF interpretation: every agent faces a set of options (outcomes with prices) determined by the other agents. We single out a randomly chosen agent i, and draw a sample of *the other agents' types* v_{-i}. We use a separate program (off the network) to calculate the options i would face. For example, the separate program can be any Python function, so it is trivial to handle complex and iterative decision making. We no longer need to construct complex network structures like the approach in (Duetting et al., 2019) or resort to exponential-sized hard-coded buyer networks like the approach in (Shen, Tang, and Zuo, 2019). After calculating i's options, we link the options together using terms that contain network parameters, which enables backpropagation. One effective way to do this is by making use of the prior distribution as discussed below.

Feeding prior distribution into the cost function: In conventional machine learning, we have access to a finite set of samples, and the process of machine learning is essentially to infer the true probability distribution of the samples. For existing neural network mechanism design approaches (Duetting et al., 2019; Shen, Tang, and Zuo, 2019) (as well as this chapter), it is assumed that the prior distribution is known. After calculating agent i's options, we make use of i's distribution to figure out the probabilities of all the options, and then derive the expected objective value from i's perspective. We assume that the prior distribution is continuous. If we have the *analytical form* of the prior distribution, then the probabilities can provide quality gradients for our training process. This is due to the fact that probabilities are calculated based on neural network outputs. In summary, we combine both samples and distribution in our cost function. (In classic machine learning, the cost function only involves the samples.)

Supervision to manual mechanisms as initialization: We start our training by first conducting supervised learning. We teach the network to mimic an existing manual mechanism, and then leave it to gradient descent. This is essentially a systematic way to improve manual mechanisms. In our experiments, besides the *serial cost sharing mechanism*, we also considered two heuristic-based manual mechanisms as starting points. One heuristic is feasible but not optimal, and the gradient descent process is able to improve its performance. The second heuristic is not always feasible, and the gradient descent process is able to fix the constraint violations. Supervision to manual mechanisms is often better than random initializations. For one thing, the supervision step often pushes the performance to a state that is already somewhat close to optimality. It may take a long time for random initializations to catch up. In computational expensive scenarios, it may never catch up. Secondly, supervision to a manual mechanism is a systematic way to set good initialization point, instead of trials and errors. It should be noted that for many conventional deep learning application domains, such as computer vision, well-performing manual algorithms do not exist. Fortunately, for mechanism design, we often have simple and well-performing mechanisms to be used as starting points.

3.2 Model Description

n agents need to decide whether or not to build a public project. The project is *binary* (build or not build) and *nonrivalrous* (the cost of the project does not depend on how many agents are consuming it). We normalize the project cost to 1. Agent i's type $v_i \in [0, 1]$ represents her private valuation for the public project. We assume that the v_i are drawn *i.i.d.* from a known prior distribution. Let F and f be the CDF and PDF, respectively. We assume that the distribution is continuous and f is differentiable.

- For the nonexcludable public project model, agent i's valuation is v_i if the project is built, and 0 otherwise.
- For the excludable public project model, the outcome space is $\{0, 1\}^n$. Under outcome $(a_1, a_2, \ldots, a_n)$, agent i consumes the public project if and only if $a_i = 1$. If for all i, $a_i = 0$, then the project is not built. As long as $a_i = 1$ for some i, the project is built.

We use $p_i \geq 0$ to denote agent i's payment. We require that $p_i = 0$ for all i if the project is not built and $\sum p_i = 1$ if the project is built. An agent's payment is also referred to as her *cost share* of the project. An agent's utility is $v_i - p_i$ if she gets to consume the project, and 0 otherwise.

We focus on *strategy-proof* and *individually rational* mechanisms. We study two objectives. One is to maximize the expected number of consumers. The other is to maximize the agents' welfare.

3.3 Characterizations and Bounds

We adopt a list of existing characterization results from (Ohseto, 2000), which characterizes strategy-proof and individual rational mechanisms for both nonexcludable and excludable public project problems. A few technical adjustments are needed for the existing characterizations to be valid for our problem. The characterizations in (Ohseto, 2000) were not proved for quasi-linear settings. However, we verify that the assumptions needed by the proofs are valid for our model setting. One exception is that the characterizations in (Ohseto, 2000) assume that every agent's valuation is strictly positive. This does not cause issues for our objectives as we are maximizing for expected performances and we are dealing with continuous distributions.[2] We are also safe to drop the *citizen sovereign* assumption mentioned in one of the characterizations[3], but not the other two minor technical assumptions called *demand monotonicity* and *access independence*.

3.3.1 Nonexcludable Mechanism Characterization

Definition 3.1 (Unanimous mechanism (Ohseto, 2000))**.** *There is a constant cost share vector* $(c_1, c_2, \ldots, c_n)$ *with* $c_i \geq 0$ *and* $\sum c_i = 1$*. The mechanism builds if and only if* $v_i \geq c_i$ *for all* i*. Agent* i *pays exactly* c_i *if the decision is to build. The unanimous mechanism is strategy-proof and individually rational.*

[2] Let M be the optimal mechanism. If we restrict the valuation space to $[\epsilon, 1]$, then M is Pareto dominated by an unanimous/largest unanimous mechanism M' for the nonexcludable/excludable setting. The expected performance difference between M and M' vanishes as ϵ approaches 0. Unanimous/largest unanimous mechanisms are still strategy-proof and individually rational when ϵ is set to exactly 0.

[3] If a mechanism always builds, then it is not individually rational in our setting. If a mechanism always does not build, then it is not optimal.

Theorem 3.1 (Nonexcludable mech. characterization (Ohseto, 2000)). *For the nonexcludable public project model, if a mechanism is strategy-proof, individually rational, and citizen sovereign, then it is weakly Pareto dominated by an unanimous mechanism.*
Citizen sovereign: Build and not build are both possible outcomes.

Mechanism 1 weakly Pareto dominates Mechanism 2 if every agent weakly prefers Mechanism 1 under every type profile.

Example 3.1 (Conservative equal costs mechanism (Moulin, 1994)). *An example unanimous mechanism works as follows: we build the project if and only if every agent agrees to pay* $\frac{1}{n}$.

3.3.2 Excludable Mechanism Characterization

Definition 3.2 (Largest unanimous mechanism (Ohseto, 2000)). *For every nonempty coalition of agents* $S = \{S_1, S_2, \ldots, S_k\}$, *there is a constant cost share vector* $C_S = (c_{S_1}, c_{S_2}, \ldots, c_{S_k})$ *with* $c_{S_i} \geq 0$ *and* $\sum_{1 \leq i \leq k} c_{S_i} = 1$. c_{S_i} *is agent* S_i*'s cost share under coalition* S. *If agent* i *belongs to two coalitions* S *and* T *with* $S \subsetneq T$, *then* i*'s cost share under* S *must be greater than or equal to her cost share under* T. *Agents in* S *unanimously approve the cost share vector* C_S *if and only if* $v_{S_i} \geq c_{S_i}$ *for all* $i \in S$. *The mechanism picks the largest coalition* S^* *satisfying that* C_{S^*} *is unanimously approved. If* S^* *does not exist, then the decision is not to build. If* S^* *exists, then it is always unique, in which case the decision is to build. Only agents in* S^* *are consumers and they pay according to* C_{S^*}. *The largest unanimous mechanism is strategy-proof and individually rational.*

The mechanism essentially specifies a constant cost share vector for every nonempty coalition. The cost share vectors must satisfy that if we remove some agents from a coalition, then under the remaining coalition, every remaining agent's cost share must be equal or higher. The largest unanimously approved coalition become the consumers/winners and they pay according to this coalition's cost share vector. The project is not built if there are no unanimously approved coalitions.

Another way to interpret the mechanism is that the agents start with the grand coalition of all agents. Given the current coalition, if some agents do not approve their cost shares, then they are forever removed. The remaining agents form a smaller coalition, and they face increased cost shares. We repeat the process until all remaining agents approve their shares, or when all agents are removed.

Theorem 3.2 (Excludable mech. characterization (Ohseto, 2000)). *For the excludable public project model, if a mechanism is strategy-proof, individually rational, and satisfies the following assumptions, then it is weakly Pareto dominated by a largest unanimous mechanism.*

Demand monotonicity: Let S be the set of consumers. If for every agent i in S, v_i stays the same or increases, then all agents in S are still consumers. If for every agent i in S, v_i stays the same or increases, and for every agent i not in S, v_i stays the same or decreases, then the set of consumers should still be S.

Access independence: For all v_{-i}, there exist v_i and v_i' so that agent i is a consumer under type profile (v_i, v_{-i}) and is not a consumer under type profile (v_i', v_{-i}).

Example 3.2 (Serial cost sharing mechanism (Moulin, 1994)). *Here is an example largest unanimous mechanism. For every nonempty subset of agents S with $|S| = k$, the cost share vector is $(\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k})$. The mechanism picks the largest coalition where the agents are willing to pay equal shares.*

Deb and Razzolini (Deb and Razzolini, 1999) proved that if we further require an *equal treatment of equals* property (if two agents have the same type, then they should be treated the same), then the only strategy-proof and individually rational mechanism left is the serial cost sharing mechanism. For many distributions, we are able to outperform the serial cost sharing mechanism. That is, equal treatment of equals (or requiring anonymity) does hurt performances.

3.3.3 Nonexcludable Public Project Analysis

We start with an analysis on the nonexcludable public project. The results presented in this section will lay the foundation for the more complex excludable public project model coming up next.

Due to the characterization results, we focus on the family of unanimous mechanisms. That is, we are solving for the optimal cost share vector $(c_1, c_2, \ldots, c_n)$, satisfying that $c_i \geq 0$ and $\sum c_i = 1$.

Recall that f and F are the PDF and CDF of the prior distribution. The *reliability function* $\overline{F}$ is defined as $\overline{F}(x) = 1 - F(x)$. We define $w(c)$ to be the expected utility of an agent when her cost share is c, conditional on that she accepts this cost share.

$$w(c) = \frac{\int_c^1 (x - c) f(x) dx}{\int_c^1 f(x) dx}$$

One condition we will use is *log-concavity*: if $\log(f(x))$ is concave in x, then f is log-concave. We also introduce another condition called *welfare-concavity*, which requires w to be concave.

Theorem 3.3. *If f is log-concave, then the conservative equal costs mechanism maximizes the expected number of consumers. If f is log-concave and welfare-concave, then*

the conservative equal costs mechanism maximizes the expected agents' welfare.

Proof. Let $C = (c_1, c_2, \ldots, c_n)$ be the cost share vector. Maximizing the expected number of consumers is equivalent to maximizing the probability of C getting unanimously accepted, which equals $\overline{F}(c_1)\overline{F}(c_2)\ldots\overline{F}(c_n)$. Its log equals $\sum_{1\leq i\leq n}\log(\overline{F}(c_i))$. When f is log-concave, so is $\overline{F}$ according to (Bagnoli and Bergstrom, 2005). This means that when cost shares are equal, the above probability is maximized.

The expected agents' welfare under the cost share vector C equals $\sum w(c_i)$, conditional on all agents accepting their shares. This is maximized when shares are equal. Furthermore, when all shares are equal, the probability of unanimous approval is also maximized.

□

f being log-concave is also called the *decreasing reversed failure rate* condition (Shao and Zhou, 2016). Bagnoli and Bergstrom (Bagnoli and Bergstrom, 2005) proved log-concavity for many common distributions, including the distributions in Table 3.2 (for all distribution parameters). All distributions are restricted to $[0, 1]$. We also list some limited results for welfare-concavity. We prove that the uniform distribution is welfare-concave, but for the other distributions, the results are based on simulations. Finally, we include the conditions for f being nonincreasing, which will be used in the excludable public project model.

TABLE 3.2. Example Log-Concave Distributions

	Welfare-Concavity	Nonincreasing
Uniform $U(0,1)$	Yes	Yes
Normal	No ($\mu = 0.5, \sigma = 0.1$)	$\mu \leq 0$
Exponential	Yes ($\lambda = 1$)	Yes
Logistic	No ($\mu = 0.5, \sigma = 0.1$)	$\mu \leq 0$

Even when optimal, the conservative equal costs mechanism performs poorly. We take the uniform $U(0, 1)$ distribution as an example. Every agent's cost share is $\frac{1}{n}$.

The probability of acceptance for one agent is $\frac{n-1}{n}$, which approaches 1 asymptotically. However, we need unanimous acceptance, which happens with much lower probability. For the uniform distribution, asymptotically, the probability of unanimous acceptance is only $\frac{1}{e} \approx 0.368$. In general, we have the following bound:

Theorem 3.4. *If f is Lipschitz continuous, then when n goes to infinity, the probability of unanimous acceptance under the conservative equal costs mechanism is $e^{-f(0)}$.*

Without log-concavity, the conservative equal costs mechanism is not necessarily optimal. We present the following dynamic program (DP) for calculating the optimal unanimous mechanism. We only present the formation for welfare maximization.[4] We assume that there is an ordering of the agents based on their identities. We define $B(k, u, m)$ as the maximum expected agents' welfare under the following conditions:

- The first $n - k$ agents have already approved their cost shares, and their total cost share is $1 - m$. That is, the remaining k agents need to come up with m.

- The first $n - k$ agents' total expected utility is u.

The optimal agents' welfare is then $B(n, 0, 1)$. We recall that $\overline{F}(c)$ is the probability that an agent accepts a cost share of c, we have

$$B(k, u, m) = \max_{0 \le c \le m} \overline{F}(c) B(k - 1, u + w(c), m - c)$$

The base case is $B(1, u, m) = \overline{F}(m)(u + w(m))$. In terms of implementation of this DP, we have $0 \le u \le n$ and $0 \le m \le 1$. We need to discretize these two intervals. If we pick a discretization size of $\frac{1}{H}$, then the total number of DP subproblems is about $H^2 n^2$.

[4] Maximizing the expected number of consumers can be viewed as a special case where every agent's utility is 1 if the project is built

To compare the performance of the conservative equal costs mechanism and our DP solution, we focus on distributions that are not log-concave (hence, uniform and normal are not eligible). We introduce the following non-log-concave distribution family:

Definition 3.3 (Two-Peak Distribution $(\mu_1, \sigma_1, \mu_2, \sigma_2, p)$)**.** *With probability p, the agent's valuation is drawn from the normal distribution $N(\mu_1, \sigma_1)$ (restricted to $[0, 1]$). With probability $1 - p$, the agent's valuation is drawn from $N(\mu_2, \sigma_2)$ (restricted to $[0, 1]$).*

The motivation behind the two-peak distribution is that there may be two categories of agents. One category is directly benefiting from the public project, and the other is indirectly benefiting. For example, if the public project is to build bike lanes, then cyclists are directly benefiting, and the other road users are indirectly benefiting (*e.g.*, less congestion for them). As another example, if the public project is to crowdfund a piece of security information on a specific software product (*e.g.*, PostgreSQL), then agents who use PostgreSQL in production are directly benefiting and the other agents are indirectly benefiting (*e.g.*, every web user is pretty much using some websites backed by PostgreSQL). Therefore, it is natural to assume the agents' valuations are drawn from two different distributions. For simplicity, we do not consider three-peak, *etc.*

For the two-peak distribution $(0.1, 0.1, 0.9, 0.1, 0.5)$, DP significantly outperforms the conservative equal costs (CEC) mechanism.

	E(no. of consumers)	E(welfare)
$n = 3$ CEC	0.376	0.200
$n = 3$ DP	0.766	0.306
$n = 5$ CEC	0.373	0.199
$n = 5$ DP	1.426	0.591

3.3.4 Excludable Public Project

Due to the characterization results, we focus on the family of largest unanimous mechanisms. We start by showing that the serial cost sharing mechanism is optimal in some scenarios.

Theorem 3.5. *2 agents case: If f is log-concave, then the serial cost sharing mechanism maximizes the expected number of consumers. If f is log-concave and welfare-concave, then the serial cost sharing mechanism maximizes the expected agents' welfare.*

3 agents case: If f is log-concave and nonincreasing, then the serial cost sharing mechanism maximizes the expected number of consumers. If f is log-concave, nonincreasing, and welfare-concave, then the serial cost sharing mechanism maximizes the agents' welfare.

For 2 agents, the conditions are identical to the nonexcludable case. For 3 agents, we also need f to be nonincreasing. Example distributions satisfying these conditions were listed in Table 3.2.

Proof. We only present the proof for welfare maximization when $n = 3$, which is the most complex case. (For maximizing the number of consumers, all references to the w function should be replaced by the constant 1.) The largest unanimous mechanism specifies constant cost shares for every coalition of agents. We use $c_{1\underline{2}3}$ to denote agent 2's cost share when the coalition is $\{1, 2, 3\}$. Similarly, $c_{\underline{2}3}$ denotes agent 2's cost share when the coalition is $\{2, 3\}$. If the largest unanimous coalition has size 3, then the expected agents' welfare gained due to this case is:

$$\overline{F}(c_{\underline{1}23})\overline{F}(c_{1\underline{2}3})\overline{F}(c_{12\underline{3}})(w(c_{\underline{1}23}) + w(c_{1\underline{2}3}) + w(c_{12\underline{3}}))$$

Given log-concavity of $\overline{F}$ (implied by the log-concavity of f) and welfare-concavity, and given that $c_{\underline{1}23} + c_{1\underline{2}3} + c_{12\underline{3}} = 1$. We have that the above is maximized when all

agents have equal shares.

If the largest unanimous coalition has size 2 and is $\{1, 2\}$, then the expected agents' welfare gained due to this case is:

$$\overline{F}(c_{\underline{1}2})\overline{F}(c_{1\underline{2}})F(c_{12\underline{3}})(w(c_{\underline{1}2}) + w(c_{1\underline{2}}))$$

$F(c_{12\underline{3}})$ is the probability that agent 3 does not join in the coalition. The above is maximized when $c_{\underline{1}2} = c_{1\underline{2}}$, so it simplifies to $2\overline{F}(\frac{1}{2})^2 w(\frac{1}{2})F(c_{12\underline{3}})$. The welfare gain from all size 2 coalitions is:

$$2\overline{F}(\frac{1}{2})^2 w(\frac{1}{2})(F(c_{\underline{1}23}) + F(c_{1\underline{2}3}) + F(c_{12\underline{3}}))$$

Since f is nonincreasing, we have that F is concave, the above is again maximized when all cost shares are equal.

Finally, the probability of coalition size 1 is 0, which can be ignored in our analysis. Therefore, throughout the proof, all terms referenced are maximized when the cost shares are equal. □

For 4 agents and uniform distribution, we have a similar result.

Theorem 3.6. *Under the uniform distribution $U(0, 1)$, when $n = 4$, the serial cost sharing mechanism maximizes the expected number of consumers and the expected agents' welfare.*

For $n \geq 4$ and for general distributions, we propose a numerical method for calculating the performance upper bound. A largest unanimous mechanism can be carried

out by the following process: we make cost share offers to the agents one by one based on an ordering of the agents. Whenever an agent disagrees, we remove this agent and move on to a coalition with one less agent. We repeat until all agents are removed or all agents have agreed. We introduce the following mechanism based on a Markov process. The initial state is $\{(\underbrace{0, 0, \ldots, 0}_{n}), n\}$, which represents that initially, we only know that the agents' valuations are at least 0, and we have not made any cost share offers to any agents yet (there are n agents yet to be offered). We make a cost share offer c_1 to agent 1. If agent 1 accepts, then we move on to state $\{(c_1, \underbrace{0, \ldots, 0}_{n-1}), n-1\}$. If agent 1 rejects, then we remove agent 1 and move on to reduced-sized state $\{(\underbrace{0, \ldots, 0}_{n-1}), n-1\}$. In general, let us consider a state with t users $\{(l_1, l_2, \ldots, l_t), t\}$. The i-th agent's valuation lower bound is l_i. Suppose we make offers $c_1, c_2, \ldots, c_{t-k}$ to the first $t-k$ agents and they all accept, then we are in a state $\{(\underbrace{c_1, \ldots, c_{t-k}}_{t-k}, \underbrace{l_{t-k+1}, \ldots, l_t}_{k}), k\}$. The next offer is c_{t-k+1}. If the next agent accepts, then we move on to $\{(\underbrace{c_1, \ldots, c_{t-k+1}}_{t-k+1}, \underbrace{l_{t-k+2}, \ldots, l_t}_{k-1}), k-1\}$. If she disagrees (she is then the first agent to disagree), then we move on to a reduced-sized state $\{(\underbrace{c_1, \ldots, c_{t-k}}_{t-k}, \underbrace{l_{t-k+2}, \ldots, l_t}_{k-1}), t-1\}$. Notice that whenever we move to a reduced-sized state, the number of agents yet to be offered should be reset to the total number of agents in this state. Whenever we are in a state with all agents offered $\{(c_1, \ldots, c_t), 0\}$, we have gained an objective value of t if the goal is to maximize the number of consumers. If the goal is to maximize welfare, then we have gained an objective value of $\sum_{1 \le i \le t} w(c_i)$. Any largest unanimous mechanism can be represented via the above Markov process. So for deriving performance upper bounds, it suffices to focus on this Markov process.

Starting from a state, we may end up with different objective values. A state has an expected objective value, based on all the transition probabilities. We define $U(t, k, m, l)$ as the maximum expected objective value starting from a state that satisfies:

- There are t agents in the state.

- There are k agents yet to be offered. The first $t-k$ agents (those who accepted the offers) have a total cost share of $1-m$. That is, the remaining k agents are responsible for a total cost share of m.
- The k agents yet to be offered have a total lower bound of l.

The upper bound we are looking for is then $U(n,n,1,0)$, which can be calculated via the following DP process:

$$U(t,k,m,l) = \max_{\substack{0\le l^*\le l \\ l^*\le c^*\le m}} \Bigg(\frac{\overline{F}(c^*)}{\overline{F}(l^*)} U(t,k-1,m-c^*,l-l^*) + (1-\frac{\overline{F}(c^*)}{\overline{F}(l^*)}) U(t-1,t-1,1,1-m+l-l^*) \Bigg)$$

In the above, there are k agents yet to be offered. We maximize over the next agent's possible lower bound l^* and the cost share c^*. That is, we look for the best possible lower bound situation and the corresponding optimal offer. $\frac{\overline{F}(c^*)}{\overline{F}(l^*)}$ is the probability that the next agent accepts the cost share, in which case, we have $k-1$ agents left. The remaining agents need to come up with $m-c^*$, and their lower bounds sum up to $l-l^*$. When the next agent does not accept the cost share, we transition to a new state with $t-1$ agents in total. All agents are yet to be offered, so $t-1$ agents need to come up with 1. The lower bounds sum up to $1-m+l-l^*$.

There are two base conditions. When there is only one agent, she has 0 probability for accepting an offer of 1, so $U(1,k,m,l)=0$. When there is only 1 agent yet to be offered, the only valid lower bound is l and the only sensible offer is m. Therefore,

$$U(t,1,m,l) = \frac{\overline{F}(m)}{\overline{F}(l)} G(t) + (1-\frac{\overline{F}(m)}{\overline{F}(l)}) U(t-1,t-1,1,1-m)$$

Here, $G(t)$ is the maximum objective value when the largest unanimous set has size t. For maximizing the number of consumers, $G(t) = t$. For maximizing welfare,

$$G(t) = \max_{\substack{c_1,c_2,\ldots,c_t \\ c_i \geq 0 \\ \sum c_i = 1}} \sum_i w(c_i)$$

The above $G(t)$ can be calculated via a trivial DP.

Now we compare the performances of the serial cost sharing mechanism against the upper bounds. All distributions used here are log-concave. In every cell, the first number is the objective value under serial cost sharing, and the second is the upper bound. We see that the serial cost sharing mechanism is close to optimality in all these experiments. We include both welfare-concave and non-welfare-concave distributions (uniform and exponential with $\lambda = 1$ are welfare-concave). For the two distributions not satisfying welfare-concavity, the welfare performance is relatively worse.

	E(no. of consumers)	E(welfare)
$n = 5$ $U(0, 1)$	3.559, 3.753	1.350, 1.417
$n = 10$ $U(0, 1)$	8.915, 8.994	3.938, 4.037
$n = 5$ $N(0.5, 0.1)$	4.988, 4.993	1.492, 2.017
$n = 10$ $N(0.5, 0.1)$	10.00, 10.00	3.983, 4.545
$n = 5$ Exponential $\lambda = 1$	2.799, 3.038	0.889, 0.928
$n = 10$ Exponential $\lambda = 1$	8.184, 8.476	3.081, 3.163
$n = 5$ Logistic$(0.5, 0.1)$	4.744, 4.781	1.451, 1.910
$n = 10$ Logistic$(0.5, 0.1)$	9.873, 9.886	3.957, 4.487

Example 3.3. *Here we provide an example to show that the serial cost sharing mechanism can be far away from optimality. We pick a simple Bernoulli distribution, where an agent's valuation is* 0 *with* 0.5 *probability and* 1 *with* 0.5 *probability.*[5] *Under the serial cost sharing mechanism, when there are n agents, only half of the agents are consumers (those who report* 1*s). So in expectation, the number of consumers is $\frac{n}{2}$. Let us consider another simple mechanism. We assume that there is an ordering of the agents based on their identities (not based on their types). The mechanism asks the first agent to accept a cost share of* 1*. If this agent disagrees, she is removed from the system. The mechanism then moves on to the next agent and asks the same, until an agent agrees. If an agent agrees, then all future agents can consume the project for free. The number of removed agents follows a geometric distribution with* 0.5 *success probability. So in expectation,* 2 *agents are removed. That is, the expected number of consumers is $n - 2$.*

3.4 Mechanism Design vs Neural Networks

For the rest of this chapter, we focus on the excludable public project model and distributions that are not log-concave. Due to the characterization results, we only need to consider the largest unanimous mechanisms. We use neural networks and deep learning to solve for well-performing largest unanimous mechanisms. Our approach involves several technical innovations as discussed in Section 3.1.

3.4.1 Mechanism Design via Neural Networks

We start with an overview of automated mechanism design (AMD) via neural networks. Previous papers on mechanism design via neural networks (Manisha, Jawahar, and Gujar, 2018; Golowich, Narasimhan, and Parkes, 2018b; Duetting et al., 2019; Shen, Tang, and Zuo, 2019) all fall into this general category.

[5] This chapter assumes that the distribution is continuous, so technically we should be considering a smoothed version of the Bernoulli distribution. For the purpose of demonstrating an elegant example, we ignore this technicality.

- Use neural networks to represent the full (or a part of the) mechanism. Like mechanisms, neural networks are essentially functions with multi-dimensional inputs and outputs.

- Training is essentially to adjust the network parameters in order to move toward a better performing network/mechanism. Training is just parameter optimization.

- Training samples are not real data. Instead, the training type profiles are generated based on the known prior distribution. We can generate infinitely many fresh samples. We use these generated samples to build the cost function, which is often a combination of mechanism design objective and constraint penalties. The cost function must be differentiable with respect to the network parameters.

- The testing data are also type profiles generated based on the known prior distribution. Again, we can generate infinitely many fresh samples, so testing is based on completely fresh samples. We average over enough samples to calculate the mechanism's expected performance.

3.4.2 Network Structure

A largest unanimous mechanism specifies constant cost shares for every coalition of agents. The mechanism can be characterized by a neural network with n binary inputs and n outputs. The n binary inputs present the coalition, and the n outputs represent the constant cost shares. We use $\vec{b}$ to denote the input vector (tensor) and $\vec{c}$ to denote the output vector. We use NN to denote the neural network, so $NN(\vec{b}) = \vec{c}$. There are several network constraints:

- All cost shares are nonnegative: $\vec{c} \geq 0$.

- For input coordinates that are 1s, the output coordinates should sum up to 1. For example, if $n = 3$ and $\vec{b} = (1, 0, 1)$ (the coalition is $\{1, 3\}$), then $\vec{c}_1 + \vec{c}_3 = 1$ (agent 1 and 3 are to share the total cost).

- For input coordinates that are 0s, the output coordinates are irrelevant. We set these output coordinates to 1s, which makes it more convenient for the next constraint.

- Every output coordinate is nondecreasing in every input coordinate. This is to ensure that the agents' cost shares are nondecreasing when some other agents are removed. If an agent is removed, then her cost share offer is kept at 1, which makes it trivially nondecreasing.

All constraints except for the last is easy to achieve. We will simply use $OUT(\vec{b})$ as output instead of directly using $NN(\vec{b})$[6]:

$$OUT(\vec{b}) = \text{softmax}(NN(\vec{b}) - 1000(1 - \vec{b})) + (1 - \vec{b})$$

Here, 1000 is an arbitrary large constant. For example, let $\vec{b} = (1, 0, 1)$ and $\vec{c} = NN(\vec{b}) = (x, y, z)$. We have

$$OUT(\vec{b}) = \text{softmax}((x, y, z) - 1000(0, 1, 0)) + (0, 1, 0)$$

$$= \text{softmax}((x, y - 1000, z)) + (0, 1, 0)$$

$$= (x', 0, z') + (0, 1, 0) = (x', 1, y')$$

In the above, $\text{softmax}((x, y - 1000, z))$ becomes $(x', 0, y')$ with $x', y' \geq 0$ and $x' + y' = 1$ because the second coordinate is very small so it (essentially) vanishes after softmax. Softmax always produces nonnegtive outputs that sum up to 1. Finally, the 0s in the output are flipped to 1s per our third constraint.

The last constraint is enforced using a penalty function. For $\vec{b}$ and $\vec{b}'$, where $\vec{b}'$ is obtained from $\vec{b}$ by changing one 1 to 0, we should have that $OUT(\vec{b}) \leq OUT(\vec{b}')$,

[6]This is done by appending additional calculation structures to the output layer.

which leads to this penalty:

$$\text{ReLU}(OUT(\vec{b}) - OUT(\vec{b}'))$$

Another way to enforce the last constraint is to use the *monotonic networks* structure (Sill, 1998). This idea has been used in (Golowich, Narasimhan, and Parkes, 2018b) , where the authors also dealt with networks that take binary inputs and must be monotone. However, we do not use this approach because it is incompatible with our design for achieving the other constraints. There are two other reasons for not using the monotonic network structure. One is that it has only two layers. Some argue that having a *deep* model is important for performance in deep learning (Zhou and Feng, 2017). The other is that under our approach, we only need a fully connected network with ReLU penalty, which is highly optimized in state-of-the-art deep learning toolsets (while the monotonic network structure is not efficiently supported by existing toolsets). In our experiments, we use a fully connected network with four layers (100 nodes each layer) to represent our mechanism.

3.4.3 Cost Function

For presentation purposes, we focus on maximizing the expected number of consumers. Only slight adjustments are needed for welfare maximization.

Previous approaches of mechanism design via neural networks used *static* networks (Manisha, Jawahar, and Gujar, 2018; Golowich, Narasimhan, and Parkes, 2018b; Duetting et al., 2019; Shen, Tang, and Zuo, 2019). Given a sample, the mechanism simulation is done on the network. Our largest unanimous mechanism involves iterative decision making, and the number of rounds is not fixed, as it depends on the users' inputs.

To model iterative decision making via a static network, we could adopt the following process. The initial offers are $OUT((1,1,\ldots,1))$. The remaining agents after the first round are then $S = \text{sigmoid}(v - OUT((1,1,\ldots,1)))$. Here, v is the type profile sample. The sigmoid function turns positive values to (approximately) 1s and negative values to (approximately) 0s. The next round of offers are then $OUT(S)$. The remaining agents afterwards are then $\text{sigmoid}(v - OUT(S))$. We repeat this n times because the largest unanimous mechanism have at most n rounds. The final coalition is a converged state, so even if the mechanism terminates before the n-th round, having it repeat n times does not change the result (except for additional numerical errors). Once we have the final coalition S^f, we include $\sum_{x \in S^f} x$ (number of consumers) in the cost function. However, this approach performs *abysmally*, due to the *vanishing gradient problem* and numerical errors caused by stacking n sigmoid functions.

We would like to avoid stacking sigmoid to model iterative decision making or get rid of sigmoid altogether. Sigmoid is heavily used in existing works on neural network mechanism design, but it is the culprit of significant numerical errors. We propose an alternative approach, where decisions are simulated off the network using a separate program (*e.g.*, any Python function). The advantage of this approach is that it is now trivial to handle complex decision making. However, experienced neural network practitioners may immediately notice a pitfall. Given a type profile sample v and the current network NN, if we simulate the mechanism off the network to obtain the number of consumers x, and include x in the cost function, then training will fail completely. This is because x is not a differentiable function of network parameters and cannot support backpropagation at all.[7]

One way to resolve this is to interpret the mechanisms as price-oriented rationing-free (PORF) mechanisms (Yokoo, 2003). That is, if we single out one agent, then her options (outcomes combined with payments) are completely determined by the other agents and she simply has to choose the utility-maximizing option. Under a

[7] We use PyTorch in our experiments. An overview of Automated Differentiation in PyTorch is available here (Paszke et al., 2017).

largest unanimous mechanism, an agent faces only two results: either she belongs to the largest unanimous coalition or not. If an agent is a consumer, then her payment is a constant due to strategy-proofness, and the constant payment is determined by the other agents. Instead of sampling over complete type profiles, we sample over v_{-i} with a random i. To better convey our idea, we consider a specific example. Let $i = 1$ and $v_{-1} = (\cdot, \frac{1}{2}, \frac{1}{2}, \frac{1}{4}, 0)$. We assume that the current state of the neural network is exactly the serial cost sharing mechanism. Given a sample, we use a separate program to calculate the following entries. In our experiments, we simply used Python simulation to obtain these entries.

- The objective value if i is a consumer (O_s). Under the example, if 1 is a consumer, then the decision must be 4 agents each pays $\frac{1}{4}$. So the objective value is $O_s = 4$.
- The objective value if i is not a consumer (O_f). Under the example, if 1 is not a consumer, then the decision must be 2 agents each pay $\frac{1}{2}$. So the objective value is $O_f = 2$.
- The binary vector that characterizes the coalition that decides i's offer ($\vec{O_b}$). Under the example, $\vec{O_b} = (1, 1, 1, 1, 0)$.

O_s, O_f, and $\vec{O_b}$ are constants without network parameters. We link them together using terms with network parameters, which is then included in the cost function:

$$(1 - F(OUT(\vec{O_b})_i))O_s + F(OUT(\vec{O_b})_i)O_f \tag{3.1}$$

$1 - F(OUT(\vec{O_b})_i)$ is the probability that agent i accepts her offer. $F(OUT(\vec{O_b})_i)$ is then the probability that agent i rejects her offer. $OUT(\vec{O_b})_i$ carries gradients as it is generated by the network. We use the analytical form of F, so the above term carries gradients.[8]

[8] PyTorch has built-in analytical CDFs of many common distributions.

The above approach essentially feeds the prior distribution into the cost function. We also experimented with two other approaches. One does not use the prior distribution. It uses a full profile sample and uses one layer of sigmoid to select between O_s or O_f:

$$\text{sigmoid}(v_i - OUT(\vec{O_b})_i)O_s + \text{sigmoid}(OUT(\vec{O_b})_i - v_i))O_f \tag{3.2}$$

The other approach is to feed "even more" distribution information into the cost function. We single out two agents i and j. Now there are 4 options: they both win or both lose, only i wins, and only j wins. We still use F to connect these options together.

In Section 3.5, in one experiment, we show that singling out one agent works the best. In another experiment, we show that even if we do not have the analytical form of F, using an analytical approximation also enables successful training.

There is a summary of the loss function:

1. O_s, O_f, and $\vec{O_b}$ are constants without gradients. We link them together using terms with gradients, which is then included in the cost function:

$$loss = (1 - F(OUT(\vec{O_b})_i))O_s + F(OUT(\vec{O_b})_i)O_f$$

2. $OUT(\vec{O_b})$ is the network output (*i.e.*, cost share offers given coalition O_b). $1 - F(OUT(\vec{O_b})_i)$ is the probability that agent i accepts her offer. $F(OUT(\vec{O_b})_i)$ is then the probability that agent i rejects her offer.

3. $OUT(\vec{O_b})_i$ carries gradients as it is generated by the network. We use the analytical form of F, so the above term carries gradients.

Our network uses the 4 layers fully connected network. The network has 100 nodes in each layer with ReLU active function. The input size is n (the agent number), and the output size is also n. And the batch size is 1.

3.4.4 Supervision as Initialization

We introduce an additional supervision step in the beginning of the training process as a systematic way of initialization. We first train the neural network to mimic an existing manual mechanism, and then leave it to gradient descent. We considered three different manual mechanisms. One is the serial cost sharing mechanism. The other two are based on two different heuristics:

Definition 3.4 (One Directional Dynamic Programming)**.** *We make offers to the agents one by one. Every agent faces only one offer. The offer is based on how many agents are left, the objective value cumulated so far by the previous agents, and how much money still needs to be raised. If an agent rejects an offer, then she is removed from the system. At the end of the algorithm, we check whether we have collected* 1*. If so, the project is built and all agents not removed are consumers. This mechanism belongs to the largest unanimous mechanism family. This mechanism is not optimal because we cannot go back and increase an agent's offer.*

Definition 3.5 (Myopic Mechanism)**.** *For coalition size k, we treat it as a nonexcludable public project problem with k agents. The offers are calculated based on the dynamic program proposed at the end of Subsection 3.3.3, which computes the optimal offers for the nonexcludable model. This is called the myopic mechanism, because it does not care about the payoffs generated in future rounds. This mechanism is not necessarily feasible, because the agents' offers are not necessarily nondecreasing when some agents are removed.*

3.5 Experiments

The experiments are conducted on a machine with Intel i5-8300H CPU.[9] The largest experiment with 10 agents takes about 3 hours. Smaller scale experiments take only about 15 minutes.

In our experiments, unless otherwise specified, the distribution considered is two-peak $(0.15, 0.1, 0.85, 0.1, 0.5)$. The x-axis shows the number of training rounds. Each round involves 5 batches of 128 samples (640 samples each round). Unless otherwise specified, the y-axis shows the expected number of **non**consumers (so lower values represent better performances). Random initializations are based on Xavier normal with bias 0.1.

Figure 3.1 (Left) shows the performance comparison of three different ways for constructing the cost function: using one layer of sigmoid (without using distribution) based on (3.2), singling out one agent based on (3.1), and singling out two agents. All trials start from random initializations. In this experiment, singling out one agent works the best. The sigmoid-based approach is capable of moving the parameters, but its result is noticeably worse. Singling out two agents has almost identical performance to singling out one agent, but it is slower in terms of time per training step.

[9] We experimented with both PyTorch and Tensorflow (eager mode). The PyTorch version runs significantly faster, because we are dealing with dynamic graphs.

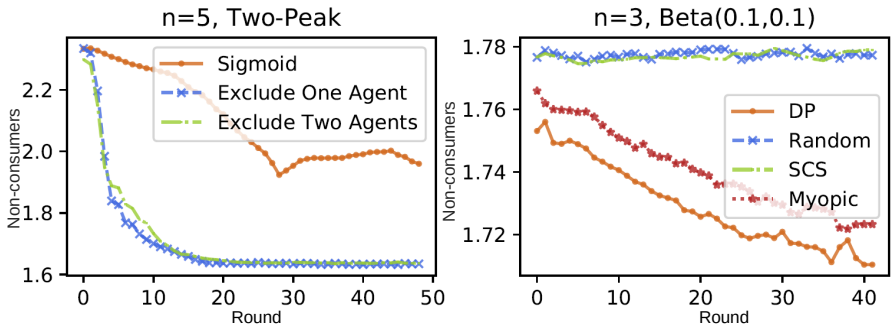


FIGURE 3.1. Effect of Distribution Info on Training (test set result during every training step)

Figure 3.1 (Right) considers the Beta $(0.1, 0.1)$ distribution. We use Kumaraswamy $(0.1, 0.354)$'s analytical CDF to approximate the CDF of Beta $(0.1, 0.1)$. The experiments show that if we start from random initializations (Random) or start by supervision to serial cost sharing (SCS), then the cost function gets stuck. Supervision to one directional dynamic programming (DP) and Myoptic mechanism (Myopic) leads to better mechanisms. So in this example scenario, approximating CDF is useful when analytical CDF is not available. It also shows that supervision to manual mechanisms works better than random initializations in this case.

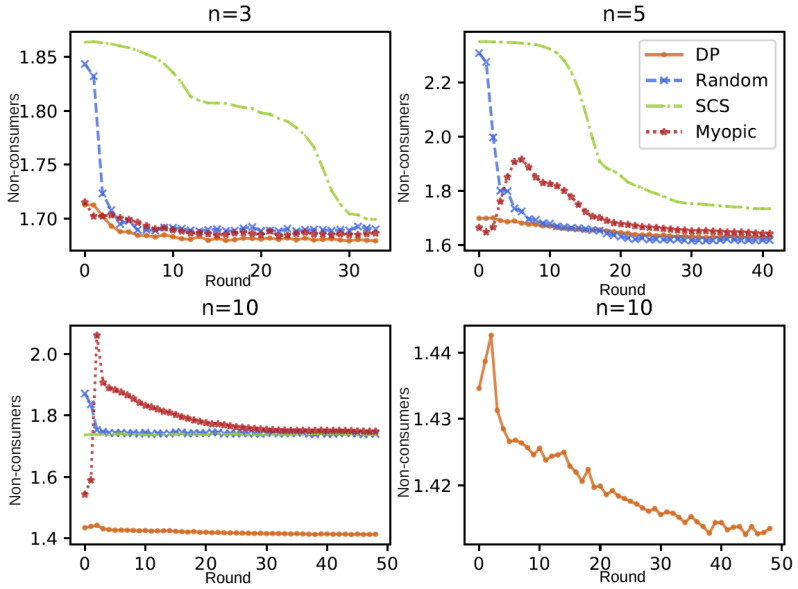


FIGURE 3.2. Experiment result: Supervision to Different Manual Mechanisms (test set result during every training step)

Figure 3.2 (Top-Left $n = 3$, Top-Right $n = 5$, Bottom-Left $n = 10$) shows the performance comparison of supervision to different manual mechanisms. For $n = 3$, supervision to DP performs the best. Random initializations is able to catch up but not completely close the gap. For $n = 5$, random initializations caught up and actually became the best performing one. The Myopic curve first increases and then decreases because it needs to first fix the constraint violations. For $n = 10$, supervision to DP significantly outperforms the others. Random initializations closes the gap with regard to serial cost sharing, but it then gets stuck. Even though it looks like the DP curve is flat, it is actually improving, albeit very slowly. A magnified version is shown in Figure 3.2 (Bottom-Right).

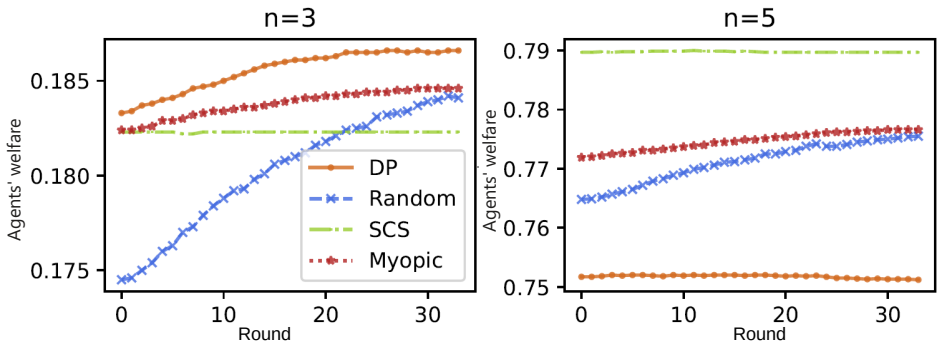


FIGURE 3.3. Experiment result: Maximizing Agents' Welfare (test set result during every training step)

Figure 3.3 shows two experiments on maximizing expected agents' welfare (y-axis) under two-peak $(0.2, 0.1, 0.6, 0.1, 0.5)$. The start of each table is the baseline performance after the supervision training for every baseline (DP, Random, SCS, Myopic). Then we leave it to gradient descent with the loss function. For $n = 3$, supervision to DP leads to the best result. For $n = 5$, SCS is actually the best mechanism we can find (the cost function barely moves).

It should be noted that all manual mechanisms *before training* have very similar welfares: 0.7517 (DP), 0.7897 (SCS), 0.7719 (Myopic). Even random initialization before training has a welfare of 0.7648. In this case, there is just little room for improvement.

3.6 Chapter Summary

In this chapter, we studied optimal-in-expectation mechanism design for the public project model. We are the first to use neural networks to design iterative mechanisms. To avoid modeling iterative decision making via the sigmoid function, we simulate the different options an agent faces under an iterative mechanism and combine the options using distribution information to build the cost function for our optimal-in-expectation objective. We showed that under various conditions, existing mechanisms or mechanisms derived via classic mechanism design approaches are optimal. When classic mechanism design approaches do not suffice, we derived better mechanisms via neural networks by first training the neural networks to mimic manual mechanisms and then improving by gradient descent.

Our experiments show that for different the number of agents, the prior probability distribution will significantly influence the effect of the agent's social welfare or consumers. For example, for uniform or normal distribution, the serial cost-sharing mechanism is best. However, when the prior probability distribution is two-peak, the serial cost-sharing mechanism is no longer best.

Chapter 4

Public Project with Minimum Expected Release Delay

In this chapter, we study a public project model where the project can be released to different agents at different times. The mechanism designer uses "delay in release" to incentivize the agents to pay for the public project. The goal of the mechanism design is to minimize the maximum delay or the total delay.

4.1 Introduction

The public project problem is a fundamental mechanism design model with many applications in multiagent systems. The public project problem involves multiple agents, who need to decide whether or not to build a public project. The project can be **nonexcludable** (*i.e.*, if the project is built, then every agent gets to consume the project, including the non-paying agents/free riders) or **excludable** (*i.e.*, the setting makes it possible to exclude some agents from consuming the project) (Ohseto, 2000).[1] A public project can be **indivisible/binary** or **divisible** (Moulin, 1994). A binary public project is either built or not built (*i.e.*, there is only one level of provision). In a divisible public project, there are multiple levels of provision (*i.e.*, build a project with adjustable quality).

[1] An example nonexcludable public project is a public airport, and an example excludable public project is a gated swimming pool.

In this chapter, we study an excludable public project model that is "divisible" in a different sense. In the model, the level of provision is binary (*i.e.*, the project is either built or not built), but an agent's consumption is divisible. The mechanism specifies when an agent can start consuming the project. High-paying agents can consume the project earlier, and the free riders need to wait. The waiting time is also called an agent's **delay**. The delay is there to incentivize payments. The model was proposed by Guo *et al.* (Guo, Yang, and Ali Babar, 2018). The authors studied the following mechanism design scenario. A group of agents come together to crowd-fund a piece of security information. No agent is able to afford the information by herself.[2] Based on the agents' valuations on the information, the mechanism decides whether or not to crowd-fund this piece of information (*i.e.*, purchase it from the security consulting firm that is selling this piece of information). If we are able to raise enough payments to cover the cost of the security information, then *ideally* we would like to share it to all agents, including the free riders, in order to maximizes the *overall protection of the community*. However, if all agents receive the information regardless of their payments, then no agents are incentivized to pay. To address this, the mechanism releases the information only to high-paying agents in the beginning and the non-paying/low-paying agents need to wait for a delayed release. The mechanism design goal is to minimize the delay as long as the delay is long enough to incentivize enough payments to cover the cost of the information. Guo *et al.* (Guo, Yang, and Ali Babar, 2018) proposed two design objectives. One is to minimize the *max-delay* (*i.e.*, the maximum waiting time of the agents) and the other is to minimize the *sum-delay* (*i.e.*, the total waiting time of the agents). The authors focused on *worst-case mechanism design* and proposed a mechanism that has a constant approximation ratio compared to the optimal mechanism. The authors also briefly touched upon *expected* delay. The authors used simulation to show that compared to their worst-case competitive mechanism, the *serial cost sharing mechanism* proposed by Moulin (Moulin, 1994) has much lower expected *max-delay* and *sum-delay* under various distributions.

In this chapter, we focus on minimizing the expected *max-delay* and the expected *sum-delay*. We propose a mechanism family called the **single deadline mechanisms**.

[2] Zero-day exploits are very expensive (Greenberg, 2012; Fisher, 2015).

For both objectives, under minor technical assumptions, we prove that there exists a single deadline mechanism that is *near optimal* when the number of agents is large, *regardless of the prior distribution*. Furthermore, when the number of agents approaches infinity, the optimal single deadline mechanism approaches optimality asymptotically. For small number of agents, the single deadline mechanism is not optimal. We extend the single deadline mechanisms to multiple deadline mechanisms. We also propose a genetic algorithm based automated mechanism design approach. We use a sequence of offers to represent a mechanism and we evolve the sequences. By simulating mechanisms using multiple distributions, we show that our genetic algorithm successfully identifies better performing mechanisms for small number of agents.

Ohseto (Ohseto, 2000) characterized all strategy-proof and individually rational mechanisms for the binary public project model (both excludable and nonexcludable), under minor technical assumptions. Deb and Razzolini (Deb and Razzolini, 1999) further showed that on top of Ohseto's characterization, if we require *equal treatment of equals* (*i.e.*, if two agents have the same type, then they should be treated the same), then the only strategy-proof and individually rational mechanisms are the *conservative equal cost mechanism* (nonexcludable) and the *serial cost sharing mechanism* (excludable), which were both proposed by Moulin (Moulin, 1994). It should be noted that Ohseto's characterization involves *exponential* number of parameters, so knowing the characterization does not mean it is easy to locate good mechanisms. Wang *et al.* (Wang et al., May 2021) proposed a neural network based approach for optimizing within Ohseto's characterization family.

The authors studied two objectives: maximizing the number of consumers and maximizing the social welfare. It should be noted that Ohseto's characterization does not apply to the model in this chapter, as our model has an additional spin that is the release delay. In this chapter, we propose a family of mechanisms called the sequential unanimous mechanisms, which is motivated by Ohseto's characterization. We apply a genetic algorithm for tuning the sequential unanimous mechanisms. Mechanism design via evolutionary computation (Phelps, McBurney, and Parsons, 2010) and mechanism design via other computational means (such as linear programming (Conitzer

and Sandholm, 2002) and neural networks (Duetting et al., 2019; Shen, Tang, and Zuo, 2019; Wang et al., May 2021)) have long been shown to be effective for many design settings.

4.2 Model Description

There are n agents who decide whether or not to build a public project. The project is binary (build or not build) and nonrivalrous (the cost of the project does not depend on how many agents are consuming it). We normalize the project cost to 1. Agent i's type $v_i \in [0,1]$ represents her private valuation for the project. We use $\vec{v} = (v_1, v_2, \ldots, v_n)$ to denote the type profile. We assume that the v_i are drawn *i.i.d.* from a known prior distribution, where f is the probability density function. For technical reasons, we assume f is *positive* and *Lipschitz continuous over* $[0,1]$.

We assume that the public project has value over a time period $[0,1]$. For example, the project could be a piece of security information that is discovered at time 0 and the corresponding exploit expires at time 1. We assume the setting allows the mechanism to specify each agent's release time for the project, so that some agents can consume the project earlier than the others. Given a type profile, a mechanism outcome consists of two vectors: $(t_1, t_2, \ldots, t_n)$ and $(p_1, p_2, \ldots, p_n)$. *I.e.*, agent i starts consuming the project at time $t_i \in [0,1]$ and pays $p_i \geq 0$. $t_i = 0$ means agent i gets to consume the public project right from the beginning and $t_i = 1$ means agent i does not get to consume the public project. We call t_i agent i's *release time*. We assume the agents' valuations over the time period is uniform. That is, agent i's valuation equals $v_i(1 - t_i)$, as she enjoys the time interval $[t_i, 1]$, which has length $1 - t_i$. Agent i's utility is then $v_i(1 - t_i) - p_i$. We impose the following mechanism design constraints:

- Strategy-proofness: We use t_i and p_i to denote agent i's release time and payment when she reports her true value v_i. We use t'_i and p'_i to denote agent i's release time and payment when she reports a false value v'_i. We should have

$$v_i(1 - t_i) - p_i \geq v_i(1 - t'_i) - p'_i$$

- Individual rationality:

$$v_i(1 - t_i) - p_i \geq 0$$

- Ex post budget balance:

  If the project is not built, then no agent can consume the project and no agent pays. That is, we must have $t_i = 1$ and $p_i = 0$ for all i.

  If the project is built, then the agents' total payment must cover exactly the project cost. That is, $\sum_i p_i = 1$.

Our aim is to design mechanisms that minimize the following design objectives:

1. Expected *Max-Delay*: $E_{v_i \sim f}\left(\max\{t_1, t_2, \ldots, t_n\}\right)$
2. Expected *Sum-Delay*: $E_{v_i \sim f}\left(\sum_i t_i\right)$

4.3 Single Deadline Mechanism

We first describe the *serial cost sharing mechanism (SCS)* proposed by Moulin (Moulin, 1994). Under SCS, an agent's release time is either 0 or 1.[3]

Let $\vec{v}$ be the type profile. We first define the following functions:

$$I(\vec{v}) = \begin{cases} 1 & \exists k \in \{1, 2, \ldots, n\}, k \leq |\{v_i | v_i \geq \frac{1}{k}\}| \\ 0 & \text{otherwise} \end{cases}$$

$I(\vec{v})$ equals 1 if and only if there exist at least k values among $\vec{v}$ that are at least $\frac{1}{k}$, where k is an integer from 1 to n.

$$K(\vec{v}) = \begin{cases} \max\{k | k \leq |\{v_i | v_i \geq \frac{1}{k}\}|, k \in \{1, 2, \ldots, n\}\} & I(\vec{v}) = 1 \\ 0 & I(\vec{v}) = 0 \end{cases}$$

Given $\vec{v}$, there could be multiple values for k, where there exist at least k values among $\vec{v}$ that are at least $\frac{1}{k}$. $K(\vec{v})$ is the largest value for k. If such a k value does not exist, then $K(\vec{v})$ is set to 0.

Definition 4.1 (Serial Cost Sharing Mechanism (Moulin, 1994))**.** *Let $\vec{v}$ be the type profile. Let $k = K(\vec{v})$.*

- *If $k > 0$, then agents with the highest k values are the consumers. The consumers pay $\frac{1}{k}$. The non-consumers do not pay.*
- *If $k = 0$, then there are no consumers and no agents pay.*

[3] Because the concept of release time does not exist in the classic binary excludable public project model.

Essentially, the serial cost sharing mechanism finds the largest k where k agents are willing to equally split the cost. If such a k exists, then we say *the cost share is successful* and these k agents are *joining the cost share*. If such a k does not exist, then we say *the cost share failed*.

Next we introduce a new mechanism family called the single deadline mechanisms.

Definition 4.2 (Single Deadline Mechanisms)**.** *A single deadline mechanism is characterized by one parameter $d \in [0, 1]$. d is called the mechanism's* **deadline**. *We use $M(d)$ to denote the single deadline mechanism with deadline d. The time interval before the deadline $[0, d]$ is called the* **non-free** *part. The time interval after the deadline $[d, 1]$ is called the* **free** *part.*

We run the serial cost sharing mechanism on the non-free part as follows. For the non-free part, the agents' valuations are $d\vec{v} = (dv_1, dv_2, \ldots, dv_n)$. Let $k = K(d\vec{v})$. Agents with the highest k values get to consume the non-free part, and they each needs to pay $\frac{1}{k}$.

The free part is allocated to the agents for free. However, we cannot give out the free part if the public project is not built.

If we give out the free part if and only if $I(d\vec{v}) = 1$, then the mechanism is not strategy-proof, because the free parts change the agents' strategies.[4] Instead, we give agent i her free part if and only if $I(dv_{-i}) = 1$. That is, agent i gets her free part if and only if the other agents can successfully cost share the non-free part without i.

If an agent receives both the non-free part and the free part, then her release time is 0. If an agent only receives the free part, then her release time is d. If an agent does not receive either part, then her release time is 1. Lastly, if an agent only receives the non-free part, then her release time is $1 - d$, because such an agent's consumption interval should have length d (*i.e.*, $[1 - d, 1]$).

Proposition 4.1. *The single deadline mechanisms are strategy-proof, individually rational, and ex post budget balanced.*

[4] For example, an agent may over-report to turn an unsuccessful cost share into a successful cost share, in order to claim the free part.

Proof. Whether an agent receives her free part or not does not depend on her report, so the agents are essentially just facing a serial cost sharing mechanism, where the item being cost shared is d portion of the public project. ■ □

4.4 Max-Delay: Asymptotic Optimality

In this section, we show that when the number of agents is large enough, the optimal single deadline mechanism is asymptotically optimal in terms of expected max-delay, regardless of the prior distribution.

Theorem 4.1. *The optimal single deadline mechanism's expected max-delay approaches* 0 *when the number of agents approaches infinity.*

Proof. We consider a single deadline mechanism $M(d)$. Every agent's valuation is drawn *i.i.d.* from a distribution with PDF f. Let V_i be the random variable representing agent i's valuation. Since f is positive and Lipschitz continuous, we have the following:

$$\forall d, \exists k, P(dV_i \geq \frac{1}{k}) > 0$$

That is, for any deadline d, there always exists an integer k, where the probability that an agent is willing to pay $\frac{1}{k}$ for the non-free part is positive. Let $p = P(dV_i \geq \frac{1}{k})$. We define the following Bernoulli random variable:

$$B_i = \begin{cases} 1 & dV_i \geq \frac{1}{k} \\ 0 & \text{otherwise} \end{cases}$$

B_i equals 1 with probability p. It equals 1 if and only if agent i can afford $\frac{1}{k}$ for the non-free part. The total number of agents in $\vec{v}$ who can afford $\frac{1}{k}$ for the non-free part then follows a Binomial distribution $B(n,p)$. We use B to denote this Binomial variable. If $B \geq k+1$, then every agent receives the free part, because agent i receives the free part if excluding herself, there are at least k agents who are willing to pay $\frac{1}{k}$ for the non-free part. The probability that the max-delay is higher than d is therefore bounded above by $P(B \leq k)$. According to Hoeffding's inequality, when $k < np$,

$$P(B \leq k) \leq e^{-2n\left(p-\frac{k}{n}\right)^2}$$

We immediately have that when n approaches infinity, the probability that the max-delay is higher than d is approaching 0. Since d is arbitrary, we have that asymptotically, the single deadline mechanism's expected max-delay is approaching 0. ■

□

Next, we use an example to show that when $n = 500$, the optimal single deadline mechanism's expected max-delay is close to 0.01. We reuse all notation defined in the proof of Theorem 4.1. We make use of the Chernoff bound. When $k < np$, we have

$$P(B \leq k) \leq e^{-nD\left(\frac{k}{n}||p\right)}, \quad \text{where } D\left(a||p\right) = a\ln\frac{a}{p} + (1-a)\ln\frac{1-a}{1-p}$$

When all agents receive the free part, the max-delay is at most d. Otherwise, the max-delay is at most 1. The expected max-delay is at most

$$P(B \leq k) + d(1 - P(B \leq k)) \leq P(B \leq k) + d$$

Example 4.1. *Let us consider a case where* $n = 500$*. We set* $d = 0.01$ *and* $k = 250$*.*

- f *is the uniform distribution* $U(0,1)$*: We have* $p = 0.6$ *and* $P(B \leq 250) \leq 3.69\text{e}{-5}$*.* $M(0.01)$*'s expected max-delay is then bounded above by* $0.01{+}3.69\text{e}{-5}$*.*
- f *is the normal distribution* $N(0.5, 0.1)$ *restricted to* $[0,1]$*: We have* $p = 0.84$ *and* $P(B \leq 250) \leq 7.45\text{e} - 69$*.* $M(0.01)$*'s expected max-delay is then bounded above by* $0.01 + 7.45\text{e} - 69$*.*

On the contrary, the expected max-delay of the serial cost sharing mechanism is not approaching 0 asymptotically. For example, when $n = 500$, under the uniform distribution $U(0,1)$, the expected max-delay of the serial cost sharing mechanism equals 0.632 which is very close to $1 - \frac{1}{e}$.

Proposition 4.2. *The expected max-delay of the serial cost sharing mechanism equals*

$$1 - (\int_{\frac{1}{n}}^{1} f(x)dx)^n$$

The above expression approaches $1 - e^{-f(0)}$ *asymptotically.*

4.5 Sum-Delay: Asymptotic Optimality

In this section, we show that when the number of agents is large enough, the optimal single deadline mechanism's expected sum-delay approaches optimality, regardless of the prior distribution.

Theorem 4.2. *When the number of agents approaches infinity, the optimal single deadline mechanism is optimal among all mechanisms in terms of expected sum-delay.*

Theorem 4.2 can be proved by combining Proposition 4.5 and Proposition 4.6.

Proposition 4.3. *The optimal expected sum-delay is finite regardless of the distribution.*

Proof. We consider the following mechanism: Pick an arbitrary integer $k > 1$. We offer $\frac{1}{k}$ to the agents one by one. An agent gets the whole interval $[0, 1]$ if she agrees to pay $\frac{1}{k}$ and if the project is built. Otherwise, she gets nothing. We build the project only when k agents agree. Since we approach the agents one by one, after k agents agree to pay $\frac{1}{k}$, all future agents receive the whole interval for free. This mechanism's expected sum-delay is bounded above by a constant. The constant only depends on the distribution. ■

□

The following proposition follows from Proposition 4.3.

Proposition 4.4. *Given a mechanism M and the number of agents n, let $Fail(n)$ be the probability of not building under M. We only need to consider M that satisfies $Fail(n) = O(1/n)$.*

We then propose a relaxed version of the ex post budget balance constraint, and use it to calculate the delay lower bound.

Definition 4.3 (Ex ante budget balance)**.** *Mechanism M is ex ante budget balanced if and only if the expected total payment from the agents equals the probability of building (times project cost 1).*

Proposition 4.5. *Let $Fail(n)$ be the probability of not building the project under the optimal mechanism when there are n agents.*

We consider what happens when we offer o for the whole interval $[0,1]$ to an individual agent. If the agent accepts o then she pays o and gets the whole interval. Otherwise, the agent pays 0 and receives nothing.

We define the delay versus payment ratio $r(o)$ *as follows:*

$$r(o) = \frac{\int_0^o f(x)dx}{o\int_o^1 f(x)dx}$$

r is continuous on $(0,1)$. Due to f being Lipschitz continuous, we have $\lim_{o\to 0} r(o) = f(0)$ and $\lim_{o\to 1} r(o) = \infty$. We could simply set $r(0) = f(0)$, then r is continuous on $[0,1)$.

We define the optimal delay versus payment ratio r^* *as follows:*

$$r^* = \min_{o\in[0,1)} r(o)$$

The expected sum-delay is bounded below by $r^*(1 - Fail(n))$, which approaches r^* asymptotically according to Proposition 4.4.

Outline. If we switch to ex ante budget balance, then it is without loss of generality to focus on anonymous mechanisms. We then face a single agent mechanism design problem where an agent pays $\frac{1-Fail(n)}{n}$ in expectation and we want to minimize her expected delay. Based on Myerson's characterization for single-parameter settings, here every strategy-proof mechanism works as follows: for each infinitesimal time interval there is a price and the price increases as an agent's allocated interval increases in length. There is an optimal price that minimizes the ratio between the delay caused by the price and the payment. The total payment is $1 - Fail(n)$, which means the total delay is at least $r^*(1 - Fail(n))$. ■ □

Proposition 4.6. *Let o^* be the optimal offer that leads to the optimal delay versus payment ratio r^*.*[5]

$$o^* = \arg\min_{o\in[0,1)} r(o)$$

Let $\epsilon > 0$ be an arbitrarily small constant. The following single deadline mechanism's expected sum delay approaches $r^*(1+\epsilon)$ asymptotically.

$$M(\frac{1+\epsilon}{no^* \int_{o^*}^1 f(x)dx})$$

Proof. Let $p = P(V_i(1+\epsilon) \geq o^*)$. Let $k = n\int_{o^*}^1 f(x)dx$. p is the probability that an agent is willing to pay $\frac{1}{k}$ for the non-free part whose length is $\frac{1+\epsilon}{no^* \int_{o^*}^1 f(x)dx}$. We use B to denote the Binomial distribution $B(n,p)$. If $B > k$, then every agent receives the free part, because cost sharing is successful even if we remove one agent. The probability that an agent does not receive the free part is then bounded above by $P(B \leq k)$. According to Hoeffding's inequality, we have that when $k < np$, we have

$$P(B \leq k) \leq e^{-2n(p-\frac{k}{n})^2} = e^{-2n(\int_{\frac{o^*}{1+\epsilon}}^1 f(x)dx - \int_{o^*}^1 f(x)dx)^2} = e^{-2n(\int_{\frac{o^*}{1+\epsilon}}^{o^*} f(x)dx)^2}$$

Let $\beta = \int_{\frac{o^*}{1+\epsilon}}^{o^*} f(x)dx$. The expected total delay when some agents do not receive the free part is then at most $ne^{-2n\beta^2}$, which approaches 0 as n goes to infinity. Therefore, we only need to consider situations where all agents receive the free part and at least k agents receive the non-free part. The expected sum delay on the remaining $n-k$ agents is then at most

$$(n-k)\frac{1+\epsilon}{no^* \int_{o^*}^1 f(x)dx} = (1+\epsilon)\frac{\int_0^{o^*} f(x)dx}{o^* \int_{o^*}^1 f(x)dx} = (1+\epsilon)r^*$$

□

[5] If $o^* = 0$, then we replace it with an infinitesimally small $\gamma > 0$. The achieved sum-delay is then approaching $r(\gamma)(1+\epsilon)$ asymptotically. When γ approaches 0, $r(\gamma)$ approaches r^*.

We then use an example to show that when $n = 500$, under different distributions, the optimal single deadline mechanism's expected sum-delay is close to the optimal value.

Example 4.2. *We consider* $n = 500$ *which is the same as Example 4.1. Simulations are based on* $100,000$ *random draws.*

- f *is the uniform distribution* $U(0,1)$*: The single deadline mechanism* $M(1)$ *(essentially the serial cost sharing mechanism) has an expected sum-delay of* 1.006*, which is calculated via numerical simulation.* $Fail(500)$ *is then at most* 0.002*.* $r^* = 1$*. The lower bound is* 0.998*, which is close to our achieved sum-delay* 1.006*.*

- f *is the normal distribution* $N(0.5, 0.1)$ *restricted to* $[0,1]$*: The single deadline mechanism* $M(1)$*'s expected sum-delay equals* $2.3\text{e}-4$ *in simulation, which is obviously close to optimality.*

- f *is the beta distribution* $Beta(0.5, 0.5)$*: The single deadline mechanism* $M(0.01)$*'s expected sum-delay equals* 1.935 *in simulation.* $Fail(500)$ *is then at most* 0.00387*.* $r^* = 1.927$*. The lower bound equals* $(1 - 0.00387) * r^* = 1.920$*, which is very close to the achieved sum-delay of* 1.935*. The serial cost sharing mechanism* $M(1)$ *is far away from optimality in this example. The expected sum-delay of the serial cost sharing mechanism is much larger at* 14.48*.*

4.6 Automated Mechanism Design for Smaller Number of Agents

For smaller number of agents, the single deadline mechanism family no longer contains a near optimal mechanism. We propose two numerical methods for identifying better mechanisms for smaller number of agents. One is by extending the single deadline mechanism family and the other is via evolutionary computation.

4.6.1 Multiple Deadline Mechanisms

The first method is fairly straightforward. We could extend the single deadline mechanism family as follows:

Definition 4.4 (Multiple Deadline Mechanisms)**.** *A multiple deadline mechanism* $M(d_1, d_2, \ldots, d_n)$ *is characterized by* n *different deadlines. Agent* i*'s non-free part is* $[0, d_i]$ *and her free part is* $[d_i, 1]$*. The mechanism's rules are otherwise identical to the single deadline mechanisms.*

We simply use exhaustive search to find the best set of deadlines. Obviously, this approach only works when the number of agents is tiny.

4.6.2 Automated Mechanism Design via Evolutionary Computation

Ohseto (Ohseto, 2000) characterized all strategy-proof and individually rational mechanisms for the binary public project model (under several minor technical assumptions). We summarize the author's characterization as follows:

- *Unanimous mechanisms* (characterization for the nonexcludable model): Under an unanimous mechanism, there is a cost share vector $(c_1, c_2, \ldots, c_n)$ with $c_i \geq 0$ and $\sum_i c_i = 1$. The project is built if and only if all agents accept this cost share vector.

- *Largest unanimous mechanisms* (characterization for the excludable model): Under a largest unanimous mechanism, for every subset/coalition of the agents, there is a constant cost share vector. The agents initially face the cost share vector corresponding to the grand coalition. If some agents do not accept the current cost share vector, then they are forever excluded. The remaining agents face a different cost share vector based on who are left. If at some point, all remaining agents accept, then we build the project. Otherwise, the project is not built.

We extend the largest unanimous mechanisms by adding the *release time* element.

Definition 4.5 (Sequential unanimous mechanisms)**.** *A cost share vector under a sequential unanimous mechanism includes both the payments and the release time:*

$$T_1, B_1, \quad T_2, B_2, \quad \ldots, \quad T_n, B_n$$

Agent i accepts the above cost share vector if and only if her utility based on her reported valuation *is nonnegative when paying B_i for the time interval $[T_i, 1]$. That is, agent i accepts the above cost share vector if and only if* her reported valuation

is at least $\frac{B_i}{1-T_i}$. $\frac{B_i}{1-T_i}$ *is called the* unit price *agent* i *faces. We require* $B_i \geq 0$ *and* $\sum_i B_i = 1$.

A sequential unanimous mechanism contains m *cost share vectors in a sequence. The mechanism goes through the sequence and stops at the first vector that is accepted by all agents. The project is built and the agents' release time and payments are determined by the unanimously accepted cost share vector. If all cost share vectors in the sequence are rejected, then the decision is not to build.*

The largest unanimous mechanisms (can be interpreted as special cases with binary T_i) form a subset of the sequential unanimous mechanisms. The sequential unanimous mechanisms' structure makes it suitable for genetic algorithms — we treat the cost share vectors as the *genes* and treat the sequences of cost share vectors as the *gene sequences*. The sequential unanimous mechanisms are generally not strategy-proof. However, they can be easily proved to be strategy-proof in two scenarios:

- A sequential unanimous mechanism is strategy-proof when *the sequence contains only one cost share vector* (an agent faces a take-it-or-leave-it offer). This observation makes it easy to generate an initial population of strategy-proof mechanisms.

- If for every agent, as we go through the cost share vector sequence, the unit price an agent faces is *nondecreasing* and her release time is also *nondecreasing*, then the mechanism is strategy-proof. Essentially, when the above is satisfied, all agents prefer earlier cost share vectors. All agents are incentivized to report truthfully, as doing so enables them to secure the earliest possible cost share vector.

The sequential unanimous mechanism family *seems* to be quite expressive.[6] Our experiments show that by optimizing within the sequential unanimous mechanisms, we are able to identify mechanisms that perform better than existing mechanisms. Our approach is as follows:

- Initial population contains 200 strategy-proof mechanisms. Every initial mechanism is a sequential unanimous mechanism with only one cost share vector. The B_i and the T_i are randomly generated by sampling $U(0, 1)$.

- We perform evolution for 200 rounds. Before each round, we filter out mechanisms that are not truthful. We have two different filters:

  - Strict filter: we enforce that every agent's unit price faced and release time must be nondecreasing. With this filter, the final mechanism produced must be strategy-proof. We call this variant the *Truthful Genetic Algorithm (TGA)*.

  - Loose filter: we use simulation to check for strategy-proofness violations. In every evolution round, we generate 200 random type profiles. For each type profile and each agent, we randomly draw one false report and we filter out a mechanism if any beneficial manipulation occurs. After finishing evolution, we use $10,000$ type profiles to filter out the untruthful mechanisms from the final population. It should be noted that, we can only claim that the remaining mechanisms are *probably* truthful. We call this variant the *Approximately Truthful Genetic Algorithm (ATGA)*.

- We perform crossover and mutations as follows:

  - Crossover (Figure 4.1): We call the top 50% of the population (in terms of fitness, *i.e.*, expected max-delay or sum-delay) the *elite population*. For every elite mechanism, we randomly pick another mechanism from the whole

[6] Let M be a strategy-proof mechanism. There exists a sequential unanimous mechanism M' (with exponential sequence length). M' has an approximate equilibrium where the equilibrium outcome is arbitrarily close to M's outcome.

population, and perform a crossover by randomly swapping one gene segment.

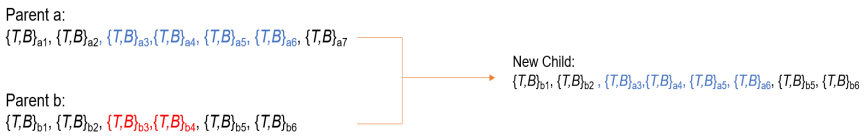


FIGURE 4.1. Genetic algorithm: Crossover

– Mutation (Figure 4.2): For every elite mechanism, with 20% chance, we randomly select one gene, modify the offer of one agent. We insert that new cost share vector into a random position after the original position.

FIGURE 4.2. Genetic algorithm: Mutation

– Neighbourhood Search (Figure 4.3): For every elite mechanism, with 20% chance, we randomly perturb one gene uniformly (from -10% to $+10\%$).

FIGURE 4.3. Neighborhood Search

- Abandon duplication and unused genes: In every evolution round, if a cost share vector is never unanimously accepted or if two cost share vectors are within 0.0001 in terms of L1 distance. then we remove the duplication/unused genes.

4.6.3 Experiments

We present the expected max-delay and sum-delay for $n = 3, 5$ and for different distributions. ATGA is only approximately truthful. We recall that in our evolutionary process, in each round, we only use a very loose filter to filter out the untruthful mechanisms. After evolution finishes, we run a more rigorous filter on the final population (based on $10,000$ randomly generated type profiles). The percentage in the parenthesis is the percentage of mechanisms surviving the more rigorous test. The other mechanisms (TGA and Multiple deadlines) are strategy-proof. SCS is the serial cost sharing mechanism from Moulin (Moulin, 1994). According to Guo *et al.* (Guo, Yang, and Ali Babar, 2018)'s experiments, SCS has the best known expected delays, so we use it as a benchmark.

n=3,sum-delay	ATGA	TGA	Single deadline	Multiple deadline	SCS
Uniform(0,1)	**1.605(95%)**	**1.605**	**1.605**	**1.605**	**1.605**
Beta(0.5,0.5)	**1.756(89%)**	**1.757**	**1.757**	**1.757**	**1.757**
Bernoulli(0.5)	**0.869(100%)**	**0.868**	1.499	1.253	1.498
50% 0, 50% 0.8	**1.699(98%)**	1.873	1.873	1.873	1.873

n=3,max-delay	ATGA	TGA	Single deadline	Multiple deadline	SCS
Uniform(0,1)	**0.705(97%)**	**0.705**	**0.705**	**0.705**	**0.705**
Beta(0.5,0.5)	**0.754(87%)**	0.757	0.782	0.757	0.782
Bernoulli(0.5)	**0.5(100%)**	**0.498**	0.687	**0.50**	0.877
50% 0, 50% 0.8	**0.676(94%)**	0.753	0.749	0.749	0.877

n=5,sum-delay	ATGA	TGA	Single deadline	Multiple deadline	SCS
Uniform(0,1)	1.462(95%)	1.503	**1.415**	**1.415**	**1.415**
Beta(0.5,0.5)	2.279(92%)	2.12	**1.955**	**1.955**	**1.955**
Bernoulli(0.5)	**1.146(100%)**	1.867	2.106	1.711	2.523
50% 0, 50% 0.8	2.432(94%)	2.845	2.323	**2.248**	2.667

n=5,max-delay	ATGA	TGA	Single deadline	Multiple deadline	SCS
Uniform(0,1)	0.677(91%)	0.677	**0.662**	**0.662**	0.678
Beta(0.5,0.5)	0.754(79%)	0.75	**0.73**	**0.73**	0.827
Bernoulli(0.5)	0.506(100%)	**0.50**	0.577	**0.50**	0.971
50% 0, 50% 0.8	**0.666(80%)**	0.751	0.736	0.679	0.968

TABLE 4.1. Experiment result: Our methods' sum-delay and max-delay vs state of the art.
We see that ATGA performs well in many settings. If we focus on *provable* strategy-proof mechanisms, then TGA and the optimal multiple deadline mechanism also often perform better than the serial cost sharing mechanism.

4.7 Chapter Summary

In this chapter, we study the excludable public project model where the decision is binary (build or not build). In a classic excludable and binary public project model, an agent either consumes the project in its whole or is completely excluded. We study a setting where the mechanism can set different project release times for different agents, in the sense that high-paying agents can consume the project earlier than the low-paying agents. The mechanism design objective is to minimize the expected maximum release delay and the expected total release delay. We propose the single deadline mechanisms. We show that the optimal single deadline mechanism is asymptotically optimal for both objectives, regardless of the prior distributions. For a small number of agents, we propose the sequential unanimous mechanisms by extending the largest unanimous mechanisms from Ohseto (Ohseto, 2000). We propose an automated mechanism design approach via evolutionary computation to optimize within the sequential unanimous mechanisms.

Chapter 5

Redistribution in Public Project Problems via Neural Networks

In this chapter, we discuss VCG redistribution mechanisms (variants of the VCG mechanism) for the public project problems. We design mechanisms via neural networks with two welfare-maximizing objectives: optimal in the worst case and optimal in expectation.

We combine generative adversarial networks and multi-layer perceptions (GAN + MLP) to find the optimal worst-case VCG redistribution mechanisms for the public project problem. We use multi-layer perceptions (MLP) combined with a cost function that takes into consideration the agents' prior distributions to find the optimal-in-expectation VCG redistribution mechanisms for the public project problem.

5.1 Introduction

5.1.1 VCG Redistribution Mechanisms

Many important problems in multiagent systems are related to resource allocations. The problem of allocating one or more resources among a group of competing agents can be solved through economic allocation mechanisms that take the agents' reported valuations for the resources as input, and produce an allocation of the resources to the agents, as well as payments to be made by the agents. As a central research branch in

economics and game theory, mechanism design concerns designing collective decision-making rules for multiple agents, to achieve desirable objectives, such as maximizing the social welfare, while each agent pursues her own utility. A mechanism is efficient if the agents who value the resource the most will get it. A mechanism is strategy-proof if the agents have the incentives to report their valuations truthfully, which is to say, an agent's utility is maximized when reporting her true valuation, no matter how the other agents report.

The Vickrey-Clarke-Groves (VCG) mechanism is a celebrated efficient and strategy-proof mechanism. Under the VCG mechanism, each agent i reports her private type θ_i. The outcome that maximizes the agents' total valuations is chosen. Every agent is required to make a VCG payment $t(\theta_{-i})$, which is determined by the other agents' types. An agent's VCG payment is often described as how much this agent's presence hurts the other agents, in terms of the other agents' total valuations. The total VCG payment may be quite large, leading to decreased welfare for the agents. In particular, in the context of the public project problem, where the goal is often to maximize the social welfare (the agents' total utility considering payments), having large VCG payments are undesirable.

To address the welfare loss due to the VCG payments, Cavallo (Cavallo, 2006) suggested that we first execute the VCG mechanism and then redistribute as much of the payments back to the agents, without violating the efficiency and strategy-proofness of the VCG mechanism, and in a weakly budget-balanced way. This is referred to as the VCG redistribution mechanism. The amount that every agent receives (or pays additionally) is called the redistribution payment. To maintain efficiency and strategy-proofness of VCG, the redistribution payment of an agent is required to be independent of her own valuation. To maintain weakly budget-balance, the total amount redistributed should never exceed the total VCG payment. The redistribution payment is characterized by a redistribution function h, where $h(\theta_{-i})$ represents agent i's redistribution payment.

There have been many successes on designing redistribution mechanisms for various multi-unit/combinatorial auction settings (Guo, 2011; Cavallo, 2006; Clippel et al., 2014; Faltings, 2005; Guo and Conitzer, 2009; Moulin, 2009; Gujar and Narahari, 2011; Guo, 2012; Guo and Conitzer, 2014; Tsuruta et al., 2014), including a long list of optimal/near-optimal mechanisms. On the other hand, there hasn't been comparable success in solving for optimal redistribution mechanisms for the public project problem, despite multiple attempts (Naroditskiy et al., 2012; Guo, 2016; Guo and Shen, 2017; Guo, 2019; Guo et al., 2011). In terms of optimal results, Naroditskiy *et al.* (Naroditskiy et al., 2012) solved for the worst-case optimal mechanism for three agents. Unfortunately, the authors' technique does not generalize to more than three agents. Guo (Guo, 2019) proposed a mechanism that is worst-case optimal when the number of agents approaches infinity, but for small number of agents, the mechanism is not optimal. For maximizing expected welfare, there are no existing results, because it is difficult for traditional mathematical analysis(eg: mixed integer programming) to maximize the expectation of welfare.

5.1.2 Designing VCG Redistribution Mechanisms via Neural Networks

A recent emerging topic in mechanism design is to bring tools such as neural networks from machine learning to design mechanisms (Manisha, Jawahar, and Gujar, 2018; Golowich, Narasimhan, and Parkes, 2018b; Duetting et al., 2019; Shen, Tang, and Zuo, 2019; Wang et al., May 2021). Duetting et al. (Duetting et al., 2019) proposed a neural network approach for the automated design of optimal auctions. They model an auction as a multi-layer neural network and frame optimal auction design as a constrained learning problem which can be solved using standard machine learning pipelines. *The training and testing type profiles are generated based on the prior distribution. The cost function involves the mechanism objective and the penalty for property violation.*

Essentially, neural networks were used as tools for functional optimisation. Shen et al. (Shen, Tang, and Zuo, 2019) proposed a neural network based framework to automatically design revenue optimal mechanisms. This framework consists of

a seller's network, which provides a menu of options to the buyers, and a buyer's network, which outputs an action that maximizes her utility. Wang et al. (Wang et al., May 2021) studied mechanism design for the public project problem and proposed several technical innovations that can be applied to mechanism design in general to improve the performance of mechanism design via neural networks.

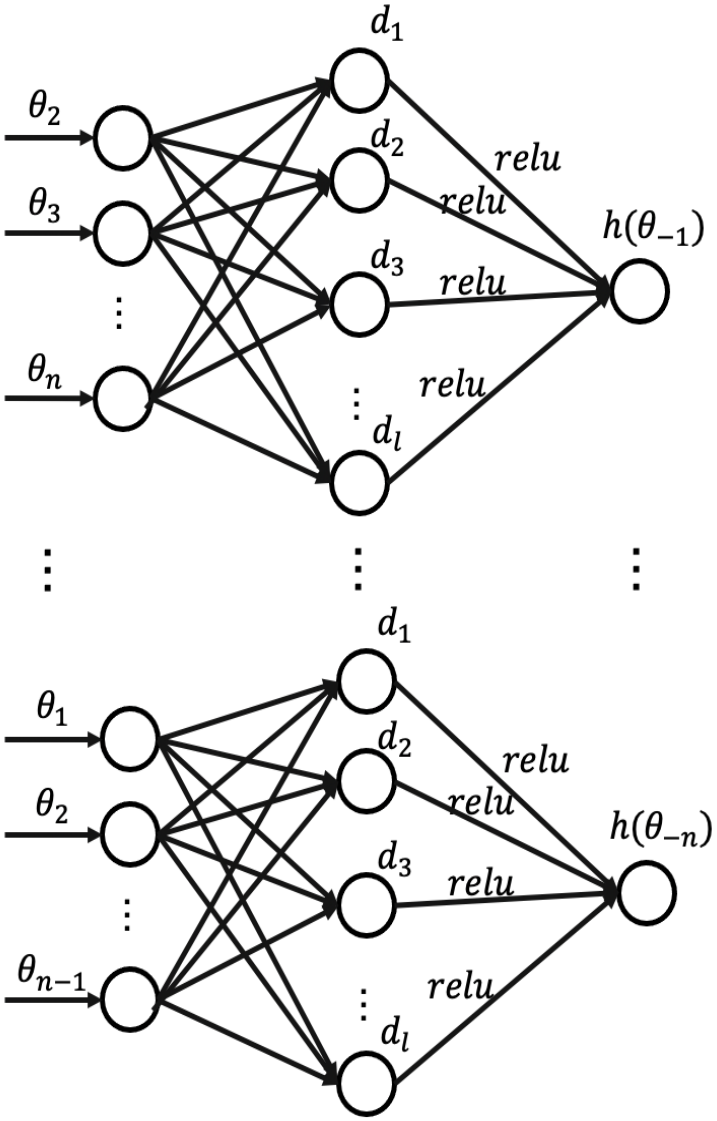


FIGURE 5.1. Neural network structure reported by Manisha et al. (Manisha, Jawahar, and Gujar, 2018)

The work by Manisha et al. (Manisha, Jawahar, and Gujar, 2018) is the first and attempt to design VCG redistribution mechanisms using neural networks, and other researchers (Tacchetti et al., 2019) study the VCG redistribution by following them. They focused on multi-unit auctions with unit demand and studied both worst-case and optimal-in-expectation objectives. By randomly generating a large number of

bid profiles, they train a neural network to maximize the total redistribution payment, while enforcing that the total redistribution should not exceed the total VCG payment. They modelled the redistribution function as a simple network outlined in Figure 5.1. It is a fully connected network with one hidden layer using ReLU activation. It takes the valuations of all the agents other than agent i herself as input, and outputs the predicted redistribution payment for agent i. Their data is sampled randomly from uniform distribution ($\theta_i \in Uniform(0, 1)$).

5.1.3 Improved Neural Networks for Designing VCG Redistribution Mechanisms for the Public Project Problem

In this chapter, we train neural networks to design VCG redistribution functions for the public project problem, which turns out to be a more challenging setting compared to multi-unit auctions studied by Manisha et al. (Manisha, Jawahar, and Gujar, 2018). The public project problem is a classic mechanism design problem that has been studied extensively in economics and computer science (Mas-Colell, Whinston, and Green, 1995; Moore, 2006; Moulin, 1988). In this problem, n agents decide whether or not to build a non-excludable public project, for example, a public bridge that can be accessed by everyone once built. Without loss of generality, we assume that the cost of the project is 1, and $\theta_i (0 \leq \theta_i \leq 1)$ is agent i's valuation for the project if it is built. If the decision is not to build, every agent retains her share of the cost, which is $1/n$.

We first evaluate the simple multilayer perceptron (MLP) model proposed by Manisha et al. (Manisha, Jawahar, and Gujar, 2018). That is, for each agent i, we train a neural network that maps θ_{-i} to agent i's redistribution. The training and testing samples are randomly generated based on the prior distribution. The cost function maximizes the mechanism design objective, as well as enforces mechanism design constraints via penalty. We find that such a simple MLP is not effective enough for the public project problem for the following reasons:

1. From our experiments, by randomly generating the type profiles, we are not getting the true worst-case type profiles for the public project problem (it is a coincidence that for multi-unit auctions with unit demand, it is a lot easier to hit a worst case).

2. Another challenge is the high input dimension when the number of agents is large. For 100 agents, the neural network has to take a 99-dimensional input, which is computationally unrealistic.

3. In the public project problem, the agents' collective payments differ significantly between cases where the decision is to build the public project and cases where the decision is not to build. The number of "build" samples in a training batch significantly impacts the parameter gradient, which results in a wild loss fluctuation during the training process.

To solve the aforementioned problems, we propose a novel neural network approach to design redistribution mechanisms for the public project problems. Our approach involves the following technical innovations.

GAN Network

For the worst-case objective, we introduce a generative adversarial network (GAN) to generate worst-case type profiles, and then use these type profiles to train the redistribution function. Our experiment shows that a mechanism trained only using randomly generated data fails when facing type profiles generated by GAN, so GAN is necessary and effective to derive the worst case.

Dimension Reduction

Instead of feeding θ_{-i} as input to train the mechanism, which has $n-1$ dimensions, we extract a few expressive features from θ_{-i} (e.g. the maximum of types θ_{-i}, the sum of θ_{-i} excluding the maximum, etc.). This reduces the input dimension to 3.

This helps the neural network loss converge faster and still retain good performance.

Supervised Learning

Wang et al. (Wang et al., May 2021) suggested that supervision to manual mechanisms often outperforms random initialization in terms of training speed by pushing the performance to a state that is already somewhat close to optimality. In addition, unlike many other deep learning problems, for mechanism design, there often exist simple and well-performing mechanisms that can be used as starting points. In this particular problem, we first conduct supervised learning to let the network mimic the state of art manual mechanism (Guo, 2019), and then leave it to gradient descent. This approach saves time for larger n in our experiments.

Feeding Prior Distribution into Loss Function

We use probability density function (PDF) of the prior distribution to provide quality gradients. In specific, for each valuation profile $\theta = \{\theta_i\}(i = 1..n)$ generated from a distribution D, we randomly choose θ_i to be replaced by a randomly generated θ_i' from $Uniform(0, 1)$ and update θ to be θ'. This sample is then assigned a weight based on the PDF. In experiments, we see that this approach significantly reduces the loss fluctuation during training. One potential explanation (or observation) is that this approach reduces the fluctuation in the proportion of "build" cases among a batch.

With a more sophisticated network architecture due to the above technical adjustments, we get better results for the worst-case than the state of the art (Guo, 2019). For the optimal-in-expectation objective, our results are close to the theoretical optimal values.

5.2 Model Description

For the public project problem, VCG redistribution mechanisms have the following form (Naroditskiy et al., 2012):

- Build the public project if and only if $\sum_i \theta_i \geq 1$.
- If the decision is to build, agent i receives $\sum_{j \neq i} \theta_j - h(\theta_{-i})$.
- If the decision is not to build, then agent i receives $(n-1)/n - h(\theta_{-i})$.
- h is an arbitrary function and θ_{-i} refers to the types from the agents other than i herself.

A VCG redistribution mechanism is characterized by the function h.
Due to Guo (Guo, 2019),

- $S(\theta) = max\{\sum_i \theta_i, 1\}$ is exactly the first-best total utility. (I.e., if the sum of types is higher than 1, then the efficient decision is to build. Otherwise, the efficient decision is not to build.)
- The agents' welfare (total utility considering payments) under type profile θ is $nS(\theta) - \sum_i h(\theta_{-i})$, which is obtained via simple algebraic simplification based on the definition of VCG redistribution mechanisms.

We consider two objectives. One is to find a mechanism that maximizes the worst-case efficiency ratio, and the other is to maximize the expected efficiency ratio.

5.2.1 Worst-case Optimal Mechanism

The efficiency ratio r is defined as the ratio between the achieved total utility and the first best total utility:

$$r = \frac{nS(\theta) - \sum_i h(\theta_{-i})}{S(\theta)} = n - \frac{\sum_i h(\theta_{-i})}{S(\theta)}$$

The worst-case efficiency ratio is the worst case ratio between the achieved total utility and the first best total utility, namely, the minimum of r over all type profiles. Due to Guo (Guo, 2019), the mechanism has a worst-case efficiency ratio α if and only if:

$$\forall\theta, (n-1) \leq \sum_i h(\theta_{-i})/S(\theta) \leq (n-\alpha) \tag{5.1}$$

In Inequality 5.1, the left side is the constraint for weakly budget-balance, and the right side corresponds to the definition of α.

Therefore, taking the worst-case ratio as the objective, we need to design an h function that:

$$\begin{aligned} &\text{maximize} \quad \alpha \\ &\text{subject to} \\ &\forall\theta, (n-1) \leq \sum_i h(\theta_{-i})/S(\theta) \leq (n-\alpha) \end{aligned} \tag{5.2}$$

5.2.2 Optimal-in-Expectation Mechanism

For this objective, we maximize the expected efficiency ratio r to 1, which is equivalent to minimize $\sum_i h(\theta_{-i})/S(\theta)$ from above to $n-1$, with the consideration of the weakly budget-balance constraint.

We are designing an h function that:

$$\begin{aligned} &\text{minimize} \quad \overline{\sum_i h(\theta_{-i})/S(\theta)} \\ &\text{subject to} \\ &\forall\theta, (n-1) \leq \sum_i h(\theta_{-i})/S(\theta) \end{aligned} \tag{5.3}$$

5.3 Worst-case Optimal Mechanism

In this section, we focus on the worst-case optimal mechanism. We first describe our neural network approach by explaining the network architecture and details of the relating techniques. Then we define the loss function.

5.3.1 Network Architecture

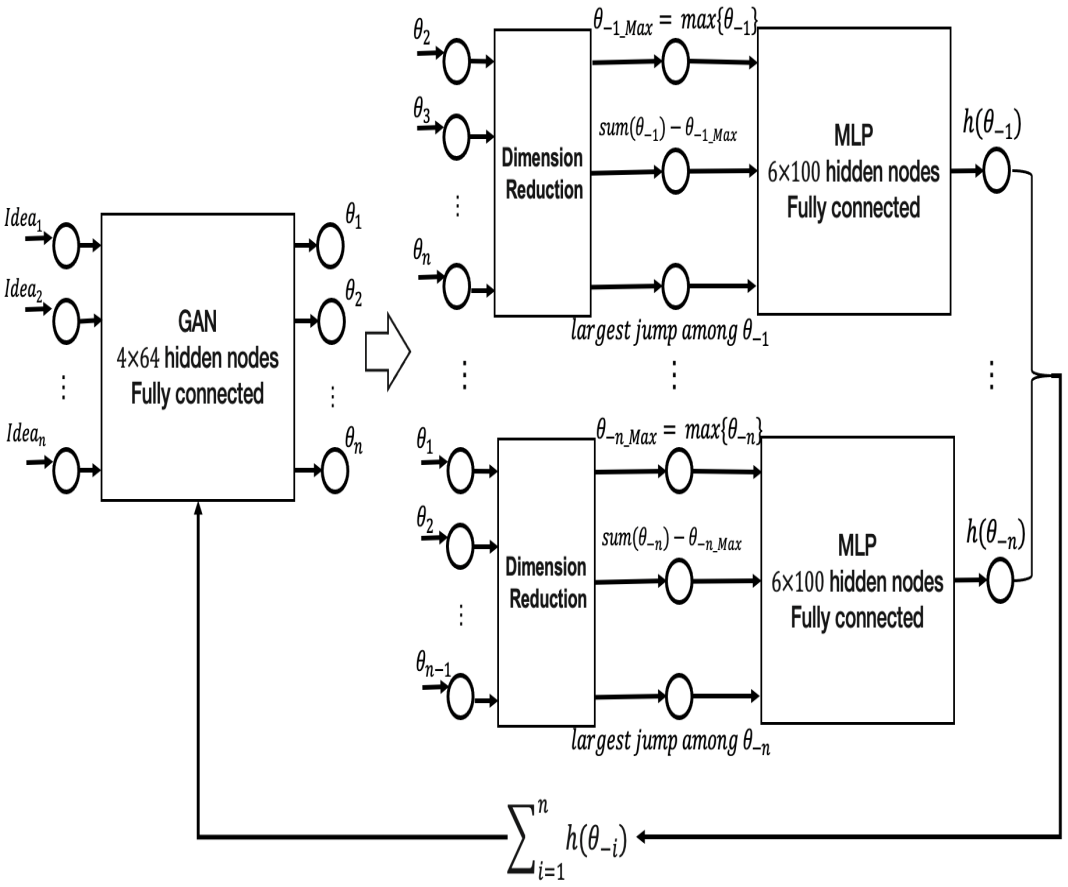


FIGURE 5.2. Our neural network architecture for worst-case optimal scenarios

We construct a neural network to determine the h function. As illustrated in Figure 5.2, it is a network system in which a GAN and an MLP interacting with each other, we call it GAN+MLP.

The GAN works as a Generative Model and is used to generate special samples (type profiles). It takes n randomly generated values as input ideas, and the output

is $\theta = \{\theta_i\}(i = 1...n)$. It is a fully connected network with 4 hidden layers, and each hidden layer contains 64 nodes. For a given batch ($batch\_size = b$), the GAN generates b type profiles: $\theta^{(j)} \in batch = \{\theta^{(1)}, \theta^{(2)}, ..., \theta^{(b)}\}$, with the aim to maximize the difference between the maximum and the minimum of $\sum_i h(\theta_{-i}^{(j)})/S(\theta^{(j)})$. It means to:

$$maximize\ (\sum_i h(\theta_{-i}^{(j_1)})/S(\theta^{(j_1)}) - \sum_i h(\theta_{-i}^{(j_2)})/S(\theta^{(j_2)}))$$

where $\theta^{(j1)}, \theta^{(j2)} \in batch$ is the sample that gives the maximum and minimum of $\sum_i h(\theta_{-i}^{(j)})/S(\theta^{(j)})$, respectively.

The MLP works as a Discriminative Model that learns the samples generated by the GAN. The MLP is a fully connected neural network. For each agent i, the network takes θ_{-i} as the input, and outputs the value of $h(\theta_{-i})$. In the MLP, there are 6 hidden layers, each of which contains 100 nodes and with ReLU as the activation function. We first train the MLP under supervision to the best-performing manual mechanism and then leave it to unsupervised learning. For unsupervised learning, our cost function is the combination of design objective and also penalty due to constraint violation.

5.3.2 Details of the Networks and Evaluations

To improve the result for the neural network, we adopt some technical adjustments. We use a GAN instead of uniform to generate special cases in order to find out the worst case. For cases with a greater number of agents ($n \geq 5$), we adopt two technical tricks: Dimension Reduction and Supervised Learning.

GAN Network

In previous studies, the authors used random generation or fixed data to find the worst case (Manisha, Jawahar, and Gujar, 2018). As mentioned in Section 3.1, We propose a new GAN approach to find out the worst case. We conduct a contrast experiment to verify the validation of the GAN. We use only data generated from uniform distribution to train a network, and test this network with two sets of data.

- Test Set A: 20000 data drawn from $Uniform(0, 1)$
- Test Set B: 10000 data generated from a trained GAN network and 10000 data drawn from $Uniform(0, 1)$

Figure 5.3 outlines the experimental results for $n = 10$ showing the difference of the the network performance on Test Set A and B. In the left figure, the network is tested by randomly generated data set A. It gives $\alpha = 0.896$, and $\sum_i h(\theta_{-i})/S(\theta)$ is between 9 and 9.104. However, in the right figure, the network performs poorly with significant violation of the weakly budget-balance constraint ($\sum_i h(\theta_{-i})/S(\theta)$ is from 3 to 9.2).

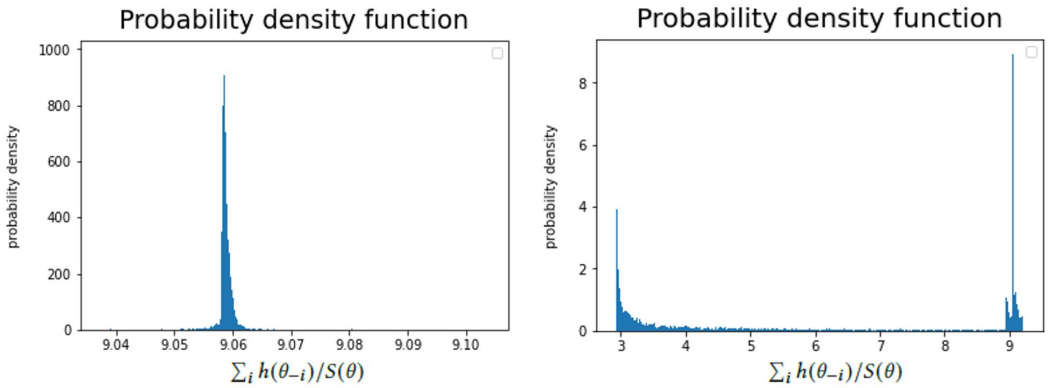


FIGURE 5.3. Experiment result: Use vs not use GAN.
Spread of $\sum_i h(\theta_{-i})/S(\theta)$ evaluated using random type profiles and GAN generated type profiles. Random generation fails to generate true worst cases.

Therefore, random type profile generation as used in Manisha et al. (Manisha, Jawahar, and Gujar, 2018) does not work for this problem. To get the worst-case performance, we need a GAN network to generate higher quality worst-case profiles, and then let the MLP learn these profiles.

Dimension Reduction

The MLP takes a $(n-1)$-dimensional input for n agents, resulting in expensive computation when n is large. This motivates us to look for an effective dimension-reducing technique.

We first manually design a list of features that describe θ_{-i}, and then experimentally search for a good combination of three features to be used for dimension reduction purposes. (We essentially reduce θ_{-i} to three dimensions this way.)

The features we consider include:

- The highest type(s) from θ_{-i}
- The the lowest type(s) from θ_{-i}
- The sum of some types from θ_{-i}
- The standard deviation of some types from θ_{-i}
- The largest jump of adjacent types from θ_{-i}

  Here *jump of adjacent types* is defined as: for a sorted list θ_{-i}, there is $jump_j$ between θ_j and θ_{j+1}:

$$jump_j = \theta_{j+1} - \theta_j$$

We first experimentally derive that the following features are more important than the rest (i.e., removing them results in significant performance loss):

- The highest type from θ_{-i}
- The the lowest type from θ_{-i}
- The sum of some type from θ_{-i}

We then experimentally evaluate different combinations of the above features:

1. Combination 1: the highest type(s) from θ_{-i} & the sum of all the other types
2. Combination 2: the highest type(s) from θ_{-i} & the difference between the highest and the lowest type from θ_{-i}
3. Combination 3: the highest type(s) from θ_{-i} & the standard deviation of all types from θ_{-i}
4. Combination 4: the highest type(s) from θ_{-i} & the standard deviation of all the other types
5. Combination 5: the highest type(s) from θ_{-i} & the largest jump of adjacent types
6. Combination 6: the highest type(s) from θ_{-i}, the lowest type from θ_{-i}, & the largest jump of adjacent types
7. Combination 7: the highest type(s) from θ_{-i}, the sum of all the other types & the largest jump of adjacent types
8. Combination 8: the highest type(s) from θ_{-i}, the lowest type from θ_{-i}, & the sum of all the other types

Figure 5.4 shows α of the networks trained with the different input combinations against the number of agents n. It is found that Combination 1, 7 and 8 performs better than the other combinations.

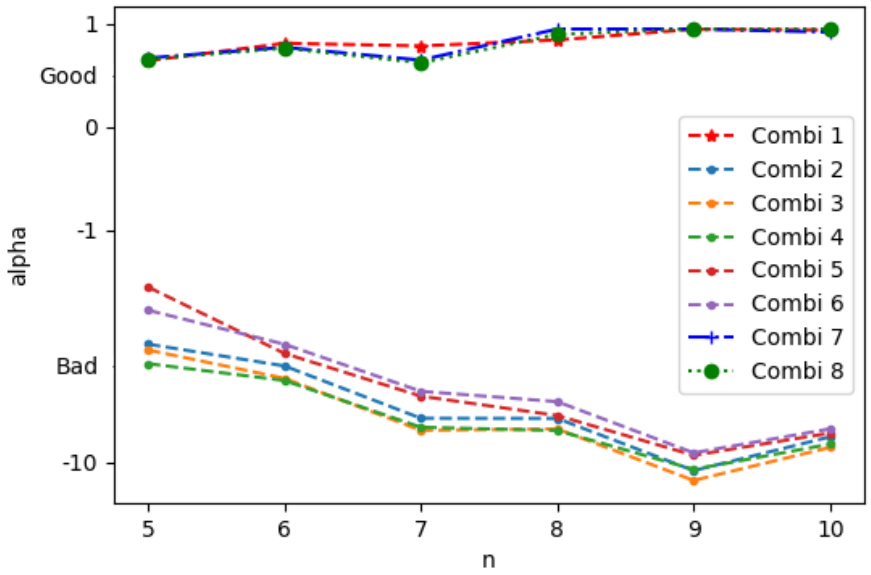


FIGURE 5.4. Experiment result: Effects of different dimension reduction methods.

The worst-case ratio α of the models trained with the input generated by using different feature-combinations against the number of agents $n = 5, ..., 10$.

The above dimension-reducing mechanism improves both the training speed and sometimes the performance. We can infer that by using this dimension-reducing mechanism, the training speed would have a more significant improvement with the increase of the agent number n.

Figure 5.5 shows the difference of the loss of the network between using and not using dimension-reducing mechanism for $n = 10$. In the left figure, the training and test loss is stabilized within 1000 iterations with the application of the dimension-reducing mechanism, while in the right figure, the losses still vibrate in a wider range till 2000 iterations.

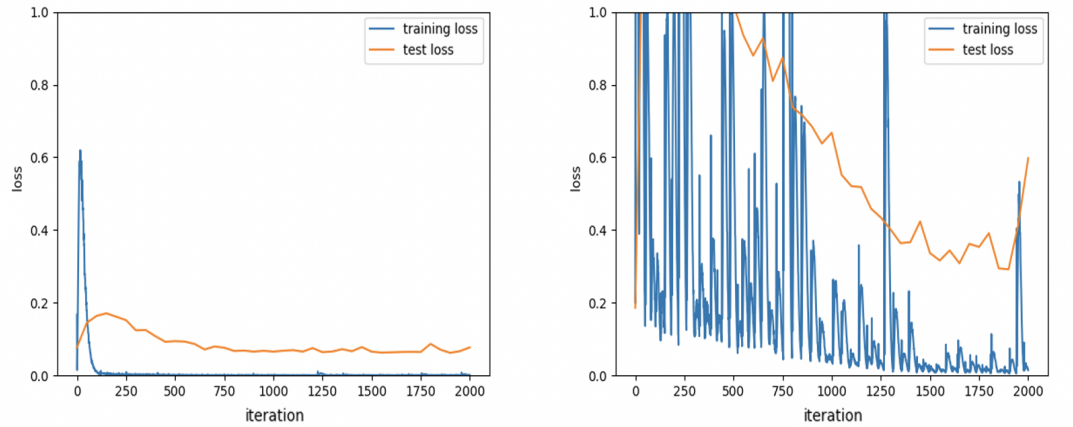


FIGURE 5.5. Experiment result: Speed of dimension reduction.

The loss during the training when using (left) and not using (right) the dimension-reducing mechanism for $n = 10$ ($\theta_i \in Uniform(0, 1)$). Dimension reduction leads to faster convergence.

Figure 5.6 shows the difference on the spreading of $\sum_i h(\theta_{-i})/S(\theta)$ between using and not using dimension-reducing mechanism for $n = 10$. In the left figure, when using the dimension-reducing mechanism, we get an expected performance of $\sum_i h(\theta_{-i})/S(\theta) = 9.003$, which is very close to $n - 1 = 9$. This indicates the corresponding mechanism is close to being optimal. (We recall that for the expected performance, we want the expected ratio to be as close to $n - 1$ as possible.) When not using the dimension-reducing mechanism, our performance is 9.377, which is further away from $n - 1 = 9$.

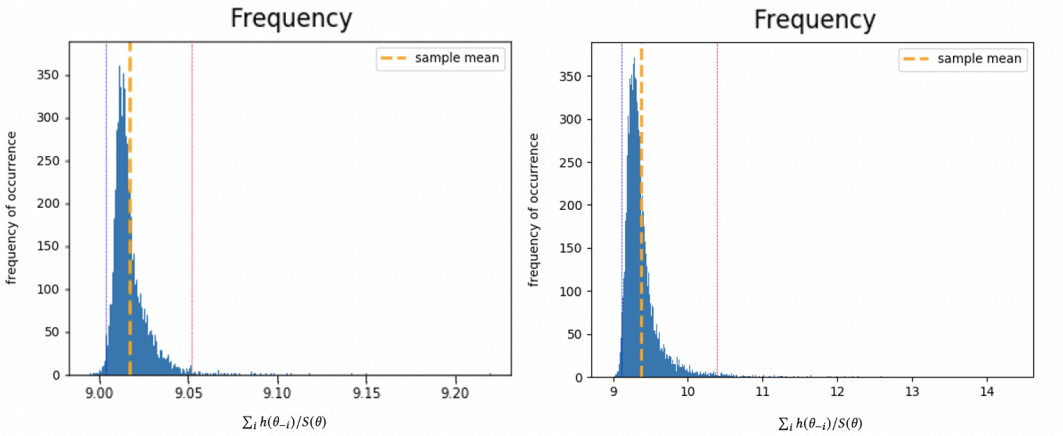


FIGURE 5.6. Experiment result: Use vs not use dimension reduction.

Spreading of $\sum_i h(\theta_{-i})/S(\theta)$ when using (left) and not using (right) the dimension-reducing mechanism for $n = 10$ ($\theta_i \in Uniform(0, 1)$), data size = 10000, bins = 500. Dimension reduction leads to better-performing mechanism.

Supervised Learning

For a greater number of agents, it takes gradient descent a long time to fix the constraint violations. Wang et al. (Wang et al., May 2021) suggested that we first supervise the neural network into the existing best manual mechanism, and then leave it to unsupervised learning. With the best manual mechanism as the starting point, better mechanisms can usually be found. The existing best manual mechanism for worst-case optimal objective is reported by Guo (Guo, 2019).

5.3.3 Loss Function

GAN Network Loss

The GAN network has the following loss:

$$\begin{aligned} loss_{GAN} =& min\{\sum_i h(\theta_{-i}^{(j_1)})/S(\theta^{(j_1)})\} \\ & - max\{\sum_i h(\theta_{-i}^{(j_2)})/S(\theta^{(j_2)})\}, \qquad \theta^{(j_1)}, \theta^{(j_2)} \in batch \end{aligned}$$

Supervised Loss

In supervised learning, we want the predicted h to be as close as possible to the best manual value h_manual (Guo, 2019), so the loss is:

$$loss_{supervised} = (h - h\_manual)^2$$

Unsupervised Loss

In the unsupervised learning stage, we want all the $\sum_i h(\theta_{-i})/S(\theta)$ to maintain the weakly budget-balance constraint, and let α to be close to 1, which means to make the upper bound of $\sum_i h(\theta_{-i})/S(\theta)$ as small as possible.

The loss function includes two parts:

- $objective\_loss = (relu(\sum_i h(\theta_{-i}) - (n - up\_bound)S(\theta)))^2$
- $constraint\_loss = (relu((n-1)S(\theta) - \sum_i h(\theta_{-i})))^2$

Since the weakly budget-balance is a strict constraint, while the objective is soft, we add a multiplier ϵ to weaken the effect of $objective\_loss$. For the worst-case optimal network, we get the best $\epsilon = 0.01$ through experiments.

$$\begin{aligned} loss_{unsupervised} &= \epsilon \cdot objective\_loss + constraint\_loss \\ &= objective\_loss/100 + constraint\_loss \end{aligned}$$

5.4 Optimal-in-Expectation Mechanism

We design the optimal-in-expectation mechanism with slight modifications based on the worst-case mechanism.

The architecture of the MLP stays the same. We do not need a GAN for generating the worst-case since the worst-case does not matter. For a large agent number n, we also adopt dimension reduction and supervised learning as we do for the worst-case mechanism. In addition, we feed the prior distribution into the loss function to achieve a high-quality gradient. This network is called MLP+FEED.

5.4.1 Feed Prior Distribution into Loss Function

In this problem, the decision to build or not to build significantly affects the expectation of a training batch. For batches with different amounts of "build" cases, the gradients fluctuate significantly, causing worse training results and speed.

Wang et al. (Wang et al., May 2021) discovered a way to insert the cumulative distribution function (CDF) of the prior distribution into the cost function to get more accurate loss function. The approach was shown to be effective for optimal-in-expecation mechanism design. We adopt a similar idea, but use probability density function (PDF) from the prior distribution to provide quality gradients for our training process. For each valuation profile $\theta = \{\theta_i\}(i = 1..n)$ generated from a distribution D, we randomly choose θ_i to be replaced by θ_i' which is regenerated from $Uniform(0, 1)$ and update θ to be θ'.

The probability of the profile θ' is proportional to $PDF_D(\theta'_i)$, so the loss should also be multiplied by $PDF_D(\theta'_i)$.

$$loss_{unsupervised\_feeding} = loss_{unsupervised} \cdot PDF_D(\theta'_i)$$

Here,

$$PDF_D(\theta'_i) = 10^{log\_prob(\theta'_i)}$$

log_prob is provided by PyTorch to calculate PDF. PyTorch distributions package is based on Schulman (Schulman et al., 2015). Our experiments show the difference between feeding and not feeding the distribution into loss function. Figure 5.7 shows that for normal distribution ($Normal(0.5, 0.1),\ n = 3$), feeding distribution into the loss function helps get a better gradient and thus dramatically improves the test result.

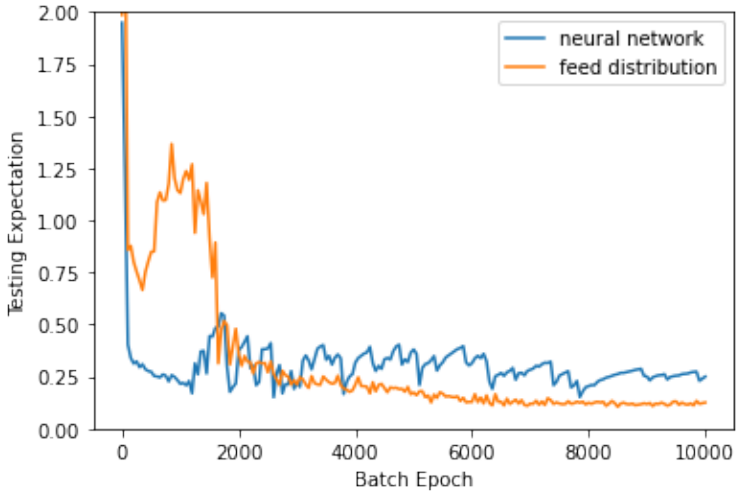


FIGURE 5.7. Experiment result: Feed vs not feed distribution.

Test loss (the distance from $\sum_i h(\theta_{-i})/S(\theta)$ to $(n-1)$) during the training for distribution: $Normal(0.5, 0.1),\ n = 3$). Feeding prior distribution leads to faster convergence and better mechanism.

5.4.2 Loss Function

The loss for the optimal-in-expectation network is similar to that of the worst-case MLP.

Supervised Loss

$$loss_{supervised} = (h - h\_manual)^2$$

Unsupervised Loss

We use the square loss to approximate the loss for the expectation, since the derivative of square loss ($f(x) = x^2$) is a linear function, and can represent the strength of the gradient decedent, which is also linear ($f'(x) = ax + b$).

Loss function consists of $objective\_loss$ and $constraint\_loss$, where

- $objective\_loss = (relu(\sum_i h(\theta_{-i}) - (n-1)S(\theta)))^2$
- $constraint\_loss = (relu((n-1)S(\theta) - \sum_i h(\theta_{-i})))^2$

$objective\_loss$ aims to push $\sum_i h(\theta_{-i})/S(\theta)$ close to $(n-1)$, so that the mechanism redistributes as much of the collected VCG payment back to the agents. $constraint\_loss$ is for weakly budget-balance, to make the total redistributed amount less than the total VCG payment.

Similar with the worst-case optimal network, we use a multiplier ϵ to soften $objective\_loss$, and the experimentally best $\epsilon = 10^{-4}$. So,

$$\begin{aligned} loss_{unsupervised} &= \epsilon \cdot objective\_loss + constraint\_loss \\ &= objective\_loss/10000 + constraint\_loss \end{aligned}$$

With the feeding of prior distribution to loss function as described in Section 4.1, we have:

$$\begin{aligned} &loss_{unsupervised\_feeding} \\ &= (objective\_loss/10000 + constraint\_loss) * PDF_D(\theta_i') \end{aligned}$$

5.5 Experiments and Results

We program with python by using the third-party library *PyTorch*. The experiments are conducted on a computer with an i5-8300H CPU and an Nvidia 1060 GPU. The experiment running time varies from a few minutes up to about 1 hour, depending on different agent numbers and data sizes.

5.5.1 Experiment settings

Generate Data

We randomly generate training and test data from prior distributions, and also use the GAN network. Half of the test data is generated from the GAN and the other half is randomly generated data. The test data size is from 10000 to 100000, and increases with n, as shown in Table 5.1. We generate new test data for each test.

n	4	5	6	7	8	9	10
Data size	10000	20000	20000	20000	50000	50000	100000

TABLE 5.1. Experiment setting: Test data size for different n.

Batch Size

Keskar et al. (Keskar et al., 2016) show that a larger batch leads to a dramatic degradation in the quality of the model. They investigate the cause for this generalization drop in the large-batch regime and present numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions. It shows that the large batch size converges to the sharp minimum, while the small batch size converges to the flat.

Our experiments support Keskar's view and we find that the loss does not reduce with a big batch size ($\geq$ 1024). We set the batch size to 64 through comparative experiments.

Order input

Arora et al. (Arora et al., 2016) shows that it is hard for ReLU to simulate the function $max\{a, b\}$. A single $max\{a, b\}$ needs two layers and 5 nodes with exact weights and biases. So inputting sorted θ_{-i} values to the MLP is a necessary and important step to get good results. We order the valuations such that $\theta_1 \geq \theta_2 \geq ... \geq \theta_n$. For redistribution problems, sorted input values will not influence strategy-proofness.

Initialization and Optimizer

We use Xavier normal initialization for the weights and $Normal(0, 0.01)$ for the bias, and use Adam optimizer with the $learning\_rate = 0.001$ initially, and decays by 0.98 every 100 steps by Pytorch Scheduler.

5.5.2 Results

Worst-case results

We compare our result for the worst-case optimal mechanism(GAN+MLP) with the previously proposed mechanisms:

- SBR: heuristic-based SBR mechanism (Naroditskiy et al., 2012)
- ABR: heuristic-based ABR mechanism (Guo, 2016)
- AMD: mechanisms derived via Automated Mechanism Design (AMD) (Guo and Shen, 2017)
- AO: asymptotically optimal (AO) VCG Redistribution (Guo, 2019)
- UB: the **conjectured** upper bounds (UB) on the efficiency ratios (Naroditskiy et al., 2012)

The result in Table 5.2 shows that our mechanism achieves better worst-case efficiency ratios than all previous results.

n	SBR	ABR	AMD	AO	GAN + MLP	UB
4	0.354	0.459	0.600	0.625	**0.634**	0.666
5	0.360	0.402	0.545	0.600	**0.622**	0.714
6	0.394	0.386	0.497	0.583	**0.592**	0.868
7	n too large	0.360	0.465	0.571	**0.626**	0.748
8	n too large	0.352	0.444	0.563	**0.654**	0.755
9	n too large	0.339	0.422	0.556	**0.682**	0.772
10	n too large	0.336	0.405	0.550	**0.623**	0.882

TABLE 5.2. Experiment result: GAN+MLP Compare with state of the art for worst case.

The main downside of our results is that our worst-case is calculated numerically by trying a large number of type profiles. This is a limitation due to our neural network based approach. (Manisha et al. (Manisha, Jawahar, and Gujar, 2018)'s neural network approach also evaluated the worst-case by randomly generating a large number of type profiles. We showed that our GAN approach is a lot more rigorous compared to simply random profile generation.) It should be noted that SBR, ABR and AMD's worst-cases were also calculated numerically. (AO's worst-case was derived analytically.) To test the stability of the worst-case ratios, we experimented with different test sizes. Table 5.3 shows that for $n = 10$ with different test sizes of 10000, 20000 and 100000, α is stable at 0.623.

Data size	10000	20000	100000
α	0.623	0.623	0.623

TABLE 5.3. Experiment setting: α for Different test size ($n = 10$).

Optimal-in-expectation results

We evaluate the expectation of $\sum_i h(\theta_{-i})/S(\theta)$ under our mechanism obtained via MLP+FEED. As defined in our model, for a case with n agents, the theoretical optimal value for $\sum_i h(\theta_{-i})/S(\theta)$ is $n-1$, so we want the expectation to be as close to $n-1$ as possible from above (to maintain weakly budget-balance). Our results show that the obtained mechanisms are near optimal. For example, we get $E = 4.061$ vs $n-1 = 4$ for $n = 5$, $E = 5.034$ vs $n-1 = 5$ for $n = 6$.

MLP+FEED	Uniform(0,1)	Normal (0.5,0.1)	n-1
n=3	2.079	2.101	2
n=4	3.071	3.111	3
n=5	4.061	4.142	4
n=6	5.027	5.034	5
n=7	6.009	6.067	6
n=8	7.008	7.023	7
n=9	8.002	8.008	8
n=10	9.003	9.023	9

TABLE 5.4. Experiment result: Our result for optimal-in-expectation scenario.

$\sum_i h(\theta_{-i})/S(\theta)$ in Expectation for Different Distributions

Table 5.4 shows that for both data generated from uniform distribution and normal distribution, the average $\sum_i h(\theta_{-i})/S(\theta)$ of our MLP+FEED network is very close to the theoretical optimal value $n-1$, which means that the redistribution function will return the vast majority of the total VCG payment to the agents (particularly for a great number of agents).

5.6 Chapter Summary

In this chapter, we consider designing optimal redistribution mechanisms for the public project problem under two objectives: worst-case optimal and optimal-in-expectation. With effective technical improvements on existing networks, we train a neural network to design good redistribution functions. We use a GAN network to generate valuation profiles to find the worst case, and feed prior distribution into loss function to get quality gradients for the optimal-in-expectation objective. To deal with large numbers of agents, we study different dimension-reducing methods and supervise the network into the existing manual mechanism as initialization. Our experiments show that for the worst case, we could find better worst-case mechanisms compared to existing mechanisms, and for expectation, the neural networks can derive near-optimal redistribution mechanisms.

Chapter 6

Revenue-Maximizing Markets for Zero-Day Exploits

In this chapter, we study a mechanism design model called zero-day exploit market. In such a market, one zero-day exploit (i.e., an exploit that allows cyber attackers to hack into iOS systems) is sold to multiple offender and defenders. In our model, for the defensive side, as long as any defender gains access to the exploit, the exploit is assumed to be immediately fixed, which benefits all defenders. The defensive side of the our model corresponds to a non-excludable public project problem. Otherwise, the model studied in this paper is only very loosely related to the public project model. The main goal of this chapter is to maximize revenue. We propose two numerical solution techniques for tuning the well-studied *Affine Maximizer Auctions* (AMA) mechanisms for revenue maximization, one is based on neural networks and the other one is based on evolutionary computation.

6.1 Introduction

Revenue-maximizing markets for *zero-day exploits* were defined in many research papers (Guo, Hata, and Babar, 2016; Egelman, Herley, and Oorschot, 2013; Hata, Guo, and Babar, 2017; Kanda et al., 2017). Guo et al. (Guo, Hata, and Babar, 2016) gave clear definitions for zero-day exploit markets. The authors proposed a Linear Programming (LP) based approach for tuning the parameters of AMA mechanisms for the purpose of revenue maximization. However, the authors' approaches had limitations. For example, it cannot handle too many constraints, and it failed to reach

optimality in some cases.

Our contributions consist of two machine learning methods (neural networks and evolutionary computation) for optimizing within the AMA mechanism family for revenue maximization. The experiments show that our mechanisms based on these two methods are better than the existing LP mechanisms.

6.1.1 Zero-day Exploit Markets

A zero-day exploit refers to a software bug which has not been disclosed to the public, and is also unknown to the software vendor. The zero-day exploit market has a long history and has been accepted by the security community (Egelman, Herley, and Oorschot, 2013). The market for zero-day exploits is not necessarily a black market. The buyers like software vendors, police or national agencies typically purchase bugs through internal or community-run bug bounty reward programs. It has been widely reported that government agencies use zero-day vulnerabilities to track criminals or for other national security reasons. Some organisations buy exploits to ensure safety for themselves.

6.1.2 Problem Description

Guo, Hata, and Babar, 2016 formally described the zero-day exploit market model. In this model, one exploit can be sold to one or more buyers. The seller is the mechanism designer, who wants to sell the exploit to maximize revenue. For example, the seller can be a cyber security company that sells bugs for profit.

Assumption 6.1. *One exploit is sold over a time frame from 0 to 1 (*$[0, 1]$*). The exploit is available to be traded from time* 0 *(zero-day), and* 1 *is the moment that the exploit's life ends (*e.g.*, due to the end of life of the affected software, or the update of a major service pack). (Guo, Hata, and Babar, 2016)*

According to Assumption 6.1, for each buyer i, we use $t_i \in [0,1]$ to denote the time agent i receives the information regarding the exploit.

Assumption 6.2. *"There are two types of agents (buyers):* defenders *and* offenders.*" (Guo, Hata, and Babar, 2016)*

- *A defender is a buyer who would like to fix the exploit.*
- *An offender is a buyer who buys the exploit in order to to utilize it (or attack it).*

For a given exploit, we assume that it can be fixed by any defender. More specifically, once an exploit is received by any defender, it is immediately fixed, rendering it worthless to all offenders. All defenders can enjoy a shared "protected" time interval from the moment the exploit is fixed (t_{end}) to 1. t_{end} is the earliest time any defender obtains the exploit. We say t_{end} is the ending time of the exploit.

Buyer i's type is a non-negative value. Function $v_i(t)$ is buyer i's instantaneous valuation at time t.

- If i is an offender, and he/she receives the exploit at t_i ($t_i \in [0,1]$). Recall that the exploit gets fixed at t_{end}. The valuation of the buyer i (offender) is a integral, which equals
$$\int_{t_i}^{t_{end}} v_i(t)dt \tag{6.1}$$

- If i is a defender, then her/his value is determined by the interval between t_{end} and the end of the exploit's life cycle, which equals
$$\int_{t_{end}}^{1} v_i(t)dt \tag{6.2}$$

The mechanism should satisfy a few desired properties: strategy-proofness, individual rationality and straight-forwardness. These properties are defined below for zero-day exploit markets:

Definition 6.1. Strategy-proofness (SP)*: For any buyer i, his/her utility is maximized when revealing $v_i(t)$ truthfully.*

Definition 6.2. Individual rationality (IR)*: For any buyer i, his/her utility is non-negative when revealing $v_i(t)$ truthfully.*

Definition 6.3. Straight-forwardness (SF)*: A straight-forward mechanism is defined as follow: before asking for offenders' valuation functions, the mechanism reveals the full details of the exploit to all offenders.*

Here, SF is a property that is specifically introduced for zero-day exploit markets and only for this chapter. An exploit can be regarded as a piece of one-time information. As a result, if the auctioneer discloses the details of the exploit to the buyers before the auction, the buyers may immediately walk away with the information for free. If the auctioneer does not describe what is being sold, it is hard for the buyers to come up with their valuation functions.

Assumption 6.3. *We assume that there are two ways for the seller to describe an exploit: either describe the full details, or describe what can be achieved with the exploit* (e.g.*, with this exploit, anyone can seize full control of a Windows 10 system remotely). (Guo, Hata, and Babar, 2016)*

- *We assume that it is safe for the seller to disclose what can be achieved with the exploit. That is, the buyers will not be able to derive "how it is done" based on "what can be achieved". (Guo, Hata, and Babar, 2016)*
- *If the seller only discloses what can be achieved, then it is difficult for an offender to determine whether the exploit is new, or something she already knows, and thus difficult to come up with their valuation. (Guo, Hata, and Babar, 2016)*

- *We assume that the defenders are able to come up with valuation functions just based on what can be achieved. This is because all zero-day exploits are by definition unknown to the defenders. (Guo, Hata, and Babar, 2016)*

The above assumption leads to the SF property. It is important to note that SF does not require the disclosure of exploitation details to the defenders prior to their bidding. If the seller does so, then the defenders can simply fix the exploit and bid $v_i(t) \equiv 0$. Due to IR, the defenders can go away without paying. Offenders are given the details before they bid, but they cannot simply bid $v_i(t) \equiv 0$ to go away without paying, which is due to the following reasoning.

Guo et al. (Guo, Hata, and Babar, 2016) used the defenders as a "THREAT". That is, if offenders bid low, the auctioneer will disclose the exploit to the defenders early (t_{end} would be smaller). The exploit then becomes less valuable for the offenders according to Equation 6.1. Essentially, the offenders are encouraged to bid/pay more to keep the exploit alive. The higher they bid, the longer the exploit remains alive.

6.1.3 Affine Maximizer Auctions Model Description

For revenue maximization, many researchers developed well performing mechanisms. Myerson (Myerson, 1981)'s optimal auction is optimal for selling a single item. For combinatorial auctions, Myerson's technique does not generalize beyond single-parameter settings. Revenue maximizing mechanism design remains an open problem for general combinatorial auctions. Many revenue-boosting techniques were proposed by researchers (Guo, Deligkas, and Savani, 2014; Guo and Deligkas, 2013; Guo et al., 2015). One particular revenue-boosting technique is the Affine Maximizer Auctions (AMA) mechanisms (Likhodedov and Sandholm, 2005). The AMA mechanism family is a rich family of mechanisms. AMA mechanisms are all strategy-proof and they are characterized by a set of parameters. By focusing on the AMA mechanisms, the original zero-day market design problem is transformed into a value optimization problem where the mechanism designer only needs to adjust the AMA parameters.

The family of AMA mechanisms is formally defined by Guo et al. (Guo, Hata, and Babar, 2016) as follows:

AMA Mechanisms

- Given a type profile θ, the outcome picked is the following:

$$o^* = \arg\max_{o \in O} \left(\sum_{i=1}^{n} u_i v_i(\theta_i, o) + a_o \right)$$

- Agent i's payment equals:

$$\frac{\max_{o \in O} \left(\sum_{j \neq i} u_j v_j(\theta_j, o) + a_o \right) - \sum_{j \neq i} u_j v_j(\theta_j, o^*) - a_{o^*}}{u_i}$$

Here, O represents the outcome space, Θ_i represents agent i's type space, and $v_i(\theta_i, o)$ represents agent i's valuation for outcome $o \in O$ when her type is $\theta_i \in \Theta_i$. Under the model described in 6.1.2, the outcome space is $[0, 1]$. To be more specific, an outcome $o \in [0, 1]$ represents when the exploit ends (revealed to the defenders). Our techniques require that the outcome space be finite, so the outcome space is discretized into a set $(\{0, \frac{1}{k}, \frac{2}{k}, \ldots, 1\})$. The outcome space size is $|O| = k + 1$.

By focusing on AMA mechanisms defined like the above, we only need to adjust the u_i and the a_o, which are the AMA mechanism parameters.

6.2 Optimizing Affine Maximizer Auctions via Neural Networks

Mechanism design via neural networks has recently drawn significant attention in the algorithmic game theory community (Manisha, Jawahar, and Gujar, 2018; Duetting et al., 2019; Shen, Tang, and Zuo, 2019; Wang et al., May 2021). In the context of our model, the high-level approach of tuning AMA parameters using neural networks is as follows:

- Treat the u_i and the a_i as model parameters.
- Initialize the model parameters randomly or start from a known mechanism such as the VCG mechanism.
- In each learning step, we generate a batch of type profiles based on the prior distribution. We evaluate the current AMA mechanism's average revenue on this batch. Parameter gradient is calculated based on this average. Model parameters are adjusted via gradient descent.

The *training data* are randomly sampled based on the prior distribution. In each learning step, we generate a fresh batch of type profiles. After we finish training, we generate another (much larger) fresh batch of type profiles to be the *testing data*. It should be noted that we never need to reuse any type profile, so there is a separation between training and testing data.

One limitation of the neural network approach is that the batch size in each learning step needs to small (*e.g.*, we set the batch size to be 16). If the batch size is very large, the time consumption of each learning step gets too long, and it actually hurts the learning performance (Keskar et al., 2016). Obviously, it is insufficient to use only 16 type profiles to *accurately* estimate a mechanism's expected revenue, but generally speaking, we do not need every learning step to move toward the correct direction. We only need that in *most* learning steps we are moving toward the correct direction. In addition, we divide the timeframe into p small pieces(e.g., $p = 1000$ pieces). We calculate the loss and argmax/min/sum these pieces' gradients according to the loss function. Then if the batch size is large, we need GPU resources out of my computer limitations (we need $p \times batchsize$ rather than $batchsize$). Moreover, our next job is to study the different neural network structures for the mechanism design problem.

Due to the batch size limitation, for our neural network based approach, we focus on the case with only two agents (one defender and one offender). A batch of 16 type profiles can be generated by drawing 4 sample types for each agent.

With two agents, an AMA mechanism can be expressed as

$$M(u_{offender}, u_{defender}, a_0, a_1, \ldots, a_k)$$

It is without loss of generality to set $u_{offender} = 1$. Besides the model parameter $u_{defender}$, the only other mechanism parameters are the a_i. We recall that a_i represents the constant term in the AMA mechanism for outcome $\frac{i}{k}$. That is, the a_i can naturally be represented using a curve $a(t)$ with $t \in [0, 1]$, where $a(t)$ is the constant

term for outcome t. The function $a(t)$ can be expressed using the following neural network:

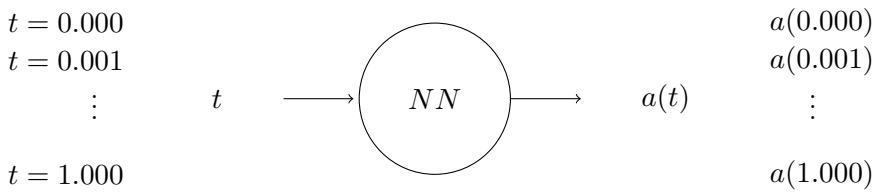


FIGURE 6.1. Neural network representation of the a_i when $k = 1000$

In the context of the above representation, the AMA mechanism's allocation (the ending time) equals

$$t^* = \arg\max_{t\in[0,1]}(\sum_{i\in N} u_i v_i(\theta_i, t) + a(t))$$

$$N = \{offender, defender\}$$

Agent i's payment equals ($i \in \{offender, defender\}$):

$$p_i = \frac{\max\limits_{t\in[0,1]}\left\{\sum_{j\neq i} u_j v_j(\theta_j, t) + a(t)\right\} - \left\{\sum_{j\neq i} u_j v_j(\theta_j, t^*) + a(t^*)\right\}}{u_i}$$

To maximise the total revenue, the loss function is set to be:

$$minimise : loss = -(p_{offender} + p_{defender})$$

In training, we set $u_{defender}$ as an *autograd* [1] parameter, and we use a fully connected network to represent the function $a(t)$. We assume that we have the analytical form of the agents' valuation function $v_i(\theta_i, t)$, which is to facilitate automatic differentiation needed by gradient descent. We will present the experimental results in Section 6.4.

[1]For an overview of automatic differentiation in PyTorch, please refer to (Paszke et al., 2017).

6.3 Optimizing Affine Maximizer Auctions via Evolutionary Computation

As discussed in the above section, an AMA mechanism is characterized by a curve $a(t)$ with $t \in [0, 1]$.[2] In the previous section on neural networks, we used neural networks to model the curve $a(t)$. A natural idea is to consider other methods for expressing curves, such as:

- Piece-wise linear segments: We may join k straight-line segments to form a curve. The coordinates for the end points are $(\frac{j}{k}, c_j)$ for $j = 0, 1, \ldots, k$. To optimize for the best piece-wise linear segments, we just need to adjust the c_j.

- Polynomial:

$$a(t) = c_k t^k + c_{k-1} t^{k-1} + \ldots + c_1 t + c_0$$

  To optimize for the best polynomial representation of $a(t)$, again, we just need to adjust the c_j.

- Fourier series:

$$a(t) = \frac{c_0}{2} + \sum_{j=1}^{N} (c_j cos(\frac{2\pi}{p} jt) + c'_j sin(\frac{2\pi}{p} jt))$$

  To optimize for the best Fourier series representation of $a(t)$, we need to adjust the c_j and the c'_j.

For all the above representation models, we use the following genetic algorithm to adjust the parameters:

[2] For two agents, besides the curve $a(t)$, we also have another model parameter $u_{defender}$, which is a single parameter that can be dealt with separately (using a naive for loop).

Algorithm 1: Genetic Algorithm

Step 1: Create an initial population of 60 curves (PopSize = 60). All initial curves are randomly generated. For piece-wise linear segments, the initial random curves are straight lines $a(t) = st + b$ where s, b are drawn randomly from $U(-100, 100)$. We choose a random slope from $U(-100, 100)$ so that the resulting straight lines have similar *vertical swings* to the curves obtained via linear programming and neural networks. For polynomials and Fourier series, all parameters are drawn randomly from $U(-10, 10)$.

Step 2: Evolution:

while *Have not reached the* 100*-th round* **do**

  for *Each Individual* **do**

    Randomly choose 200 type profiles to test fitness (revenue);

  Sort individuals according to fitness;

  Selection:

  Keep the top 20 individuals in terms of fitness (EliteSize = 20);

  Add new individuals through **Crossover** and **Mutation** until we reach Popsize;

  1. Crossover: Generate 20 new curves via standard two-point crossover.

  2. Mutation: Generate 20 new curves via perturbing existing curves (with 0.1 probability, a parameter is increased or decreased by $\delta = 0.5$).

Step 3: Testing:

Average over 10000 random type profiles to choose the best individual.

We will present the experimental results in Section 6.4.

6.4 Experiments

According to (Greenberg, 2012), an exploit that attacks the Chrome browser sells for at most 200k for offensive clients (USD). According to Google's official bug bounty reward program for the Chrome browser (Projects, 2015), a serious exploit is priced for at most 15k. We adopt an experimental setting based on the numbers above.

There are two agents. Obviously, the offender's valuation function is

$$v(\theta_O, t) = \int_0^t \theta_O(1-x)dx$$

θ_O is drawn uniformly at random from $U(0, 400)$. That is, the offender's valuation for the whole time interval $[0, 1]$ is at most 200. The offender gets less and less interested in the exploit as time goes on (the instantaneous valuation gets to 0 as $1-x$ approaches 0 when x approaches 1).

The defender's valuation function is

$$v(\theta_D, t) = \int_t^1 \theta_D x dx$$

θ_D is drawn uniformly at random from $U(0, 15)$. That is, the defender's valuation for the whole time interval $[0, 1]$ is at most 15. The defender's instantaneous valuation in the exploit does not change over time.

Optimal revenue: 50.55

Since the valuation functions satisfy all the conditions needed for the single-parameter model introduced in Guo, Hata, and Babar, 2016, we are able to derive the revenue-maximizing mechanism. The ending time is based on the following rule:

$$t^* = \arg\max_{t\in[0,1]} \{(2 \times \theta_O - 400)(t - t * t/2) + (2 \times \theta_D - 15)(1-t)\}$$

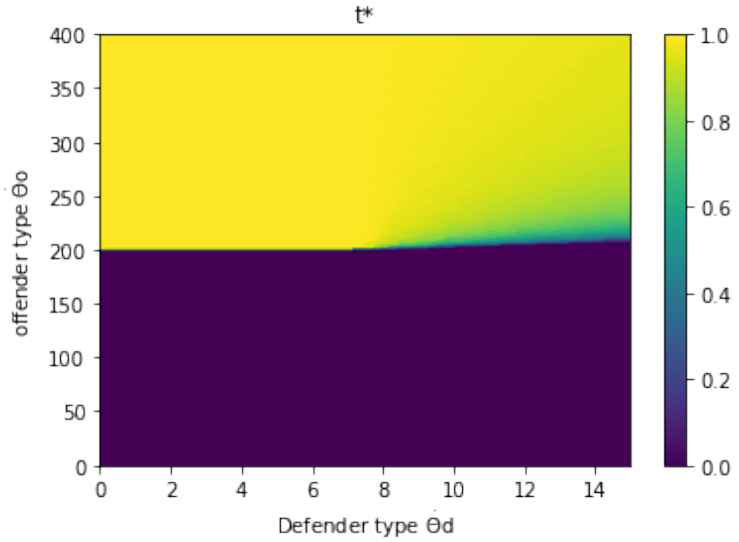


FIGURE 6.2. Experiment result: Ending time t^* function.

Ending time t^* as a function of type profiles for the optimal mechanism

The optimal revenue equals 50.55.

Revenue via neural networks: 50.31

We use the following setup:

- We use a fully connected network with three hidden layers (200 nodes per layer) to represent $a(t)$.
- We use the Adam optimizer with a learning rate of 0.0001.
- The batch size is set to 16.
- The training set consists of 80000 randomly generated type profiles.
- The testing set consists of 20000 randomly generated type profiles.

The achieved expected revenue equals 50.31.

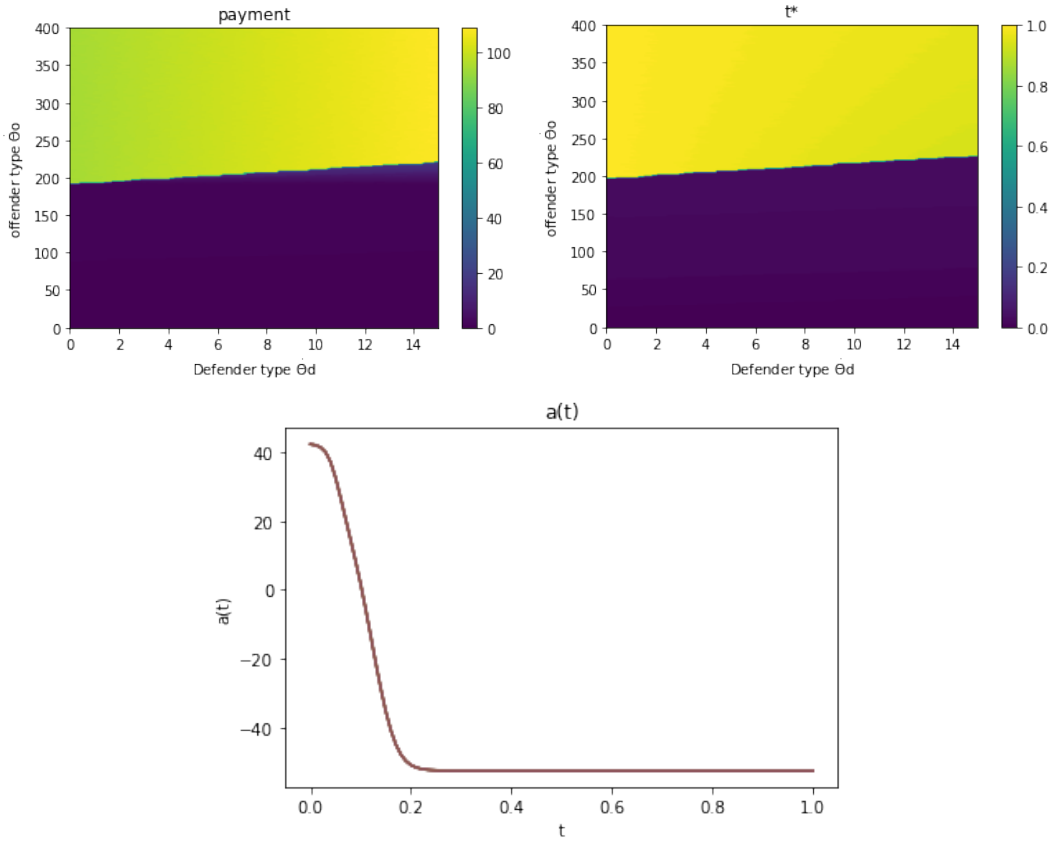


FIGURE 6.3. Experiment result: Total payment, ending time t^*, and $a(t)$ via neural network.

Total payment, ending time t^*, and $a(t)$ as functions of type profiles for the AMA mechanism derived via neural network

Main computational challenges for the neural networks

Setting aside $u_{defender}$, every AMA mechanism is characterized by a **curve** $a(t)$, where $t \in [0, 1]$. We note that the main computational bottleneck is due to the learning batch size. Our fully-connected network structure with 3 layers and 200 nodes per layer is more than *expressive* enough to represent curves. In our experiments, we generate $20,000$ type profiles to evaluate an AMA mechanism's expected revenue. We cannot afford to do this in every learning iteration. Instead, we use a batch size of 16 during learning. That is, we use the average revenue of 16 randomly generated type profiles to estimate the parameter gradient. Certainly, the gradient direction obtained this way is not very accurate. Fortunately, as long as the learning rate is small, and as long as most of the time, the gradient direction is generally correct, then we still are able to successfully train the neural network. Figure 6.4 is an illustration of the neural network training process. As shown in Figure 6.4, the average revenue of 16 randomly generated type profiles perturbs wildly[3], but the mechanism is still improving steadily during the training, which shows that a small batch is good enough for training.

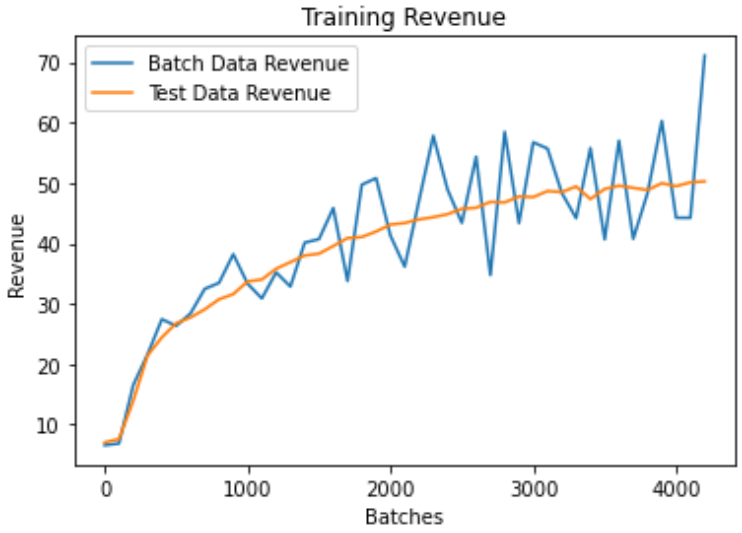


FIGURE 6.4. Experiment result: Neural network training process.

[3]In Figure 6.4, the presented data points for the batch data revenue are the average revenue for every 100 batches.

Revenue via evolutionary computation: 47.67

Our last numerical technique approximates $a(t)$ using segmented straight lines, polynomials, and Fourier series. The parameters (*i.e.*, polynomial coefficients) are adjusted according to an evolutionary algorithm (Algorithm 1).

TABLE 6.1. Experiment result: Evolutionary computation's results.

Segmented straight line (50 segments):	47.67
Quartic polynomial:	38.80
Sextic polynomial:	37.20
Fourier series (N=5, p=2) :	44.54
Fourier series (N=30, p=2) :	46.88

Revenue via evolutionary computation: different representations of $a(t)$

We present all the $a(t)$ functions in the figure that follows.

For Fourier series (N=30), the payment and ending time are as follows:

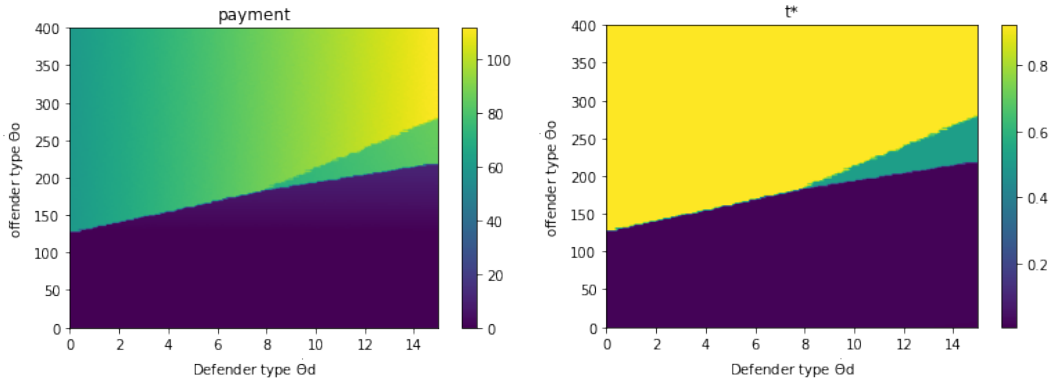
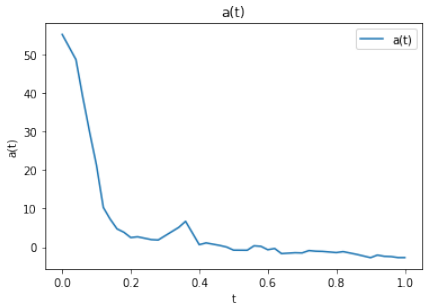


FIGURE 6.5. Experiment result: Total payment and ending time t^* via evolutionary computation.

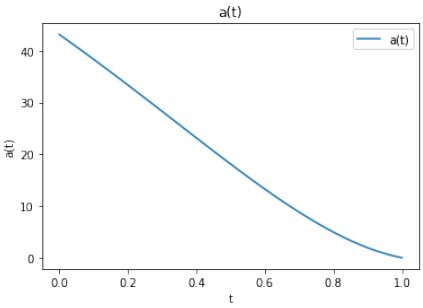
Total payment and ending time t^* as functions of type profiles for the AMA mechanism derived via evolutionary computation (Fourier series N=30)

6.4.1 Comparison of different AMA solution techniques

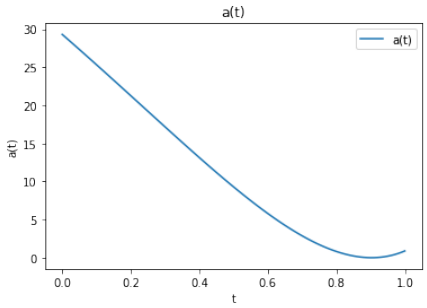
The neural network based approach produces slightly better result than the linear programming based approach. On the other hand, the neural network based approach realistically only works for two agents and cannot deal with black-box valuation functions, as it requires the valuation functions' analytical forms for auto differentiation.



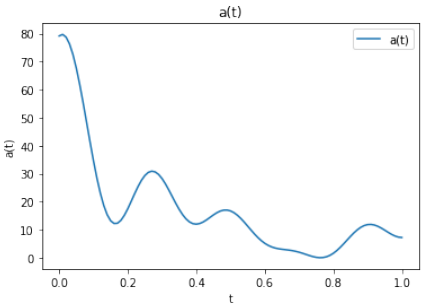
(a) Segmented straight lines (50 segments)



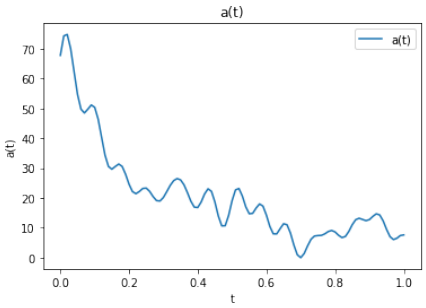
(b) Quartic



(c) Sextic



(d) Fourier series N=5



(e) Fourier series N=30

The evolutionary computation technique scales the best if we adopt a representation with small number of parameters (*i.e.*, if we approximate $a(t)$ using a quadratic polynomial such as $a(t) = c_2t^2 + c_1t + c_0$, then we only need to adjust three parameters). The down-side of the evolutionary computation technique is that all the representations we have tried (polynomials or Fourier series) are not as expressive as neural networks, so the achieved revenue using the evolutionary computation technique is slightly worse.

6.5 Chapter Summary

In this chapter, we study markets for zero-day exploits from a revenue-maximizing mechanism design perspective. For the defenders, as long as any defender receives the exploit, all defenders are protected. So from the perspective of the defenders, the model studied in this chapter can be viewed as a public project model that is not excludable. Nevertheless, otherwise the model studied in this chapter is not related to a typical public project model.

We adopted the computationally feasible automated mechanism design approach. We focused on the AMA mechanisms. To identify an AMA mechanism with high revenue, we need to design an allocation "curve". We propose two numerical solution techniques, one is based on neural networks and the other one is based on evolutionary computation. All techniques are able to produce near-optimal mechanisms.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we used different machine learning methods to design optimal or near-optimal strategy-proof and individual rational mechanisms for public project problems. Specifically, we addressed the following problems:

- In Chapter 3, we study the public project that is indivisible and binary (e.g., a bridge). Indivisible means that an agent either consumes the project in its whole or is completely excluded. Binary means that this project is built or not built. We identified a sufficient condition on the prior distribution for the conservative equal costs mechanism to be the optimal strategy-proof and individually rational mechanism. For the non-excludable model, we designed novel mechanisms, such as dynamic programming, to get optimal results. For the excludable indivisible public project models, we involved several technical innovations that can be applied to mechanism design in general. We interpreted the mechanisms as price-oriented rationing-free (PORF) mechanisms. The experiments showed that our mechanisms are better than previous results and more close to the theoretical upper bound.

- In Chapter 4, we focused on the divisible public projects. In a classic excludable and binary public project model, we study a setting where the mechanism can set different project release times for different agents, which means that for a certain agent, the higher he/she pays, the earlier he/she can use the project. For a small number of agents, we proposed the sequential unanimous mechanisms

by extending the existing mechanisms and used evolutionary computation to optimize them. We proposed the single deadline mechanisms which are shown to be asymptotically optimal. The experiments showed that our mechanisms are better than existing mechanisms.

- In Chapter 5, we studied the VCG redistribution mechanisms for the classic public project problem. Multi-layer perceptrons (MLP) were used in combination with a carefully designed cost function that takes into consideration of the agents' prior distributions for the optimal-in-expectation objective. We designed generative adversarial networks (GAN + MLP) to find the optimal worst-case VCG redistribution mechanisms. The experiments showed that our mechanisms are very close to the theoretical upper bounds and are better than existing mechanisms.

- In Chapter 6, we studied markets for zero-day exploits from a revenue-maximizing mechanism design perspective. We used a neural network to get the optimal curve that characterizes the optimal Affine Maximizer Auctions (AMA) mechanism. A second technique used evolutionary computation to evolve mathematical expressions for representing the optimal AMA curve. The experiments showed that our neural networks and evolutionary computation based techniques both produce near-optimal revenue.

7.2 Future Work

In addition to the public project problems addressed in this thesis, we point out the following open problems that we expect to explore in the future:

- Apply our methods to other models: The techniques proposed in this thesis can not only be used in public project problems, but also have the potential

to be applied to other economic models. One future direction is to extend our methods to other models such as cake cutting, facility location, and auctions.

- Design better neural network structures for mechanism design: In this thesis, we carefully designed our neural networks to ensure strategy-proofness and individual rationality. However, most of our design focused on the mechanism design aspect, not on the neural networks themselves. That is, neural network is used as an assisting tool and it is heavily guided by manual human inputs. Most of our networks are straight-forward feed-forward fully-connected networks. One future direction is to consider more sophisticated neural network structures such as pointer networks, LSTM, permutation invariant networks, etc.

- Combinatorial public project model: Through out this thesis, we assumed that there is only one public project. However, in practise, it is common that we face many public projects and the agents face combinatorial decision making. One future research direction is to see to what extent our techniques and results generalise beyond a single public project.

Bibliography

[1] Tuomas Sandholm. "Automated mechanism design: A new application area for search algorithms". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2003, pp. 19–36.

[2] Anton Likhodedov and Tuomas Sandholm. "Approximating Revenue-Maximizing Combinatorial Auctions". In: *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. Ed. by Manuela M. Veloso and Subbarao Kambhampati. AAAI Press / The MIT Press, 2005, pp. 267–274. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-043.php.

[3] Makoto Yokoo, Yuko Sakurai, and Kenji Terada. "Price-oriented, Rationing-free Protocol: Guideline for Designing Strategy/False-name Proof Auction Protocols". In: (2002).

[4] Marie Drobietz, Adrian Loerbroks, and Nils Hansson. "Who is who in cardiovascular research? What a review of Nobel Prize nominations reveals about scientific trends". In: *Clinical Research in Cardiology* (2021), pp. 1–10.

[5] Maarten CW Janssen. "Reflections on the 2020 Nobel Memorial Prize Awarded to Paul Milgrom and Robert Wilson". In: *Erasmus Journal for Philosophy and Economics* 13.2 (2020), pp. 177–184.

[6] SangMok Lee. "Incentive compatibility of large centralized matching markets". In: *The Review of Economic Studies* 84.1 (2016), pp. 444–463.

[7] Leonid Hurwicz. "Optimality and informational efficiency in resource allocation processes". In: *Mathematical methods in the social sciences* (1960).

[8] John C Harsanyi. "Games with incomplete information played by "Bayesian" players, I–III Part I. The basic model". In: *Management science* 14.3 (1967), pp. 159–182.

[9] Roger B Myerson. "Optimal auction design". In: *Mathematics of operations research* 6.1 (1981), pp. 58–73.

[10] Michael Curry, Tuomas Sandholm, and John Dickerson. "Differentiable Economics for Randomized Affine Maximizer Auctions". In: *arXiv preprint arXiv:2202.02872* (2022).

[11] Hamidreza Maghsoudlou, Behrouz Afshar-Nadjafi, and Seyed Taghi Akhavan Niaki. "Multi-skilled project scheduling with level-dependent rework risk; three multi-objective mechanisms based on cuckoo search". In: *Applied Soft Computing* 54 (2017), pp. 46–61.

[12] Gabriel Carroll. "Robustness in mechanism design and contracting". In: *Annual Review of Economics* 11 (2019), pp. 139–166.

[13] R. Lavi, Ahuva Mu'alem, and N. Nisan. "Towards a characterization of truthful combinatorial auctions". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 2003, pp. 574–583.

[14] Arunava Sen. "The theory of mechanism design: An overview". In: *Economic and Political Weekly* (2007), pp. 8–13.

[15] Xiaoyan Zou. "Double-sided auction mechanism design in electricity based on maximizing social welfare". In: *Energy Policy* 37.11 (2009), pp. 4231–4239.

[16] Alejandro M Manelli and Daniel R Vincent. "Multidimensional mechanism design: Revenue maximization and the multiple-good monopoly". In: *Journal of Economic theory* 137.1 (2007), pp. 153–185.

[17] Georgios Amanatidis, Pieter Kleer, and Guido Schäfer. "Budget-feasible mechanism design for non-monotone submodular objectives: Offline and online". In: *Proceedings of the 2019 ACM Conference on Economics and Computation.* 2019, pp. 901–919.

[18] Georgios Amanatidis, Georgios Birmpas, and Evangelos Markakis. "On budget-feasible mechanism design for symmetric submodular objectives". In: *International Conference on Web and Internet Economics*. Springer. 2017, pp. 1–15.

[19] Xuanhe Zhao. "Multi-scale multi-mechanism design of tough hydrogels: building dissipation into stretchy networks". In: *Soft matter* 10.5 (2014), pp. 672–687.

[20] Claudio Mezzetti. "Mechanism design with interdependent valuations: Efficiency". In: *Econometrica* 72.5 (2004), pp. 1617–1626.

[21] Brooks A Kaiser. "The Athenian trierarchy: Mechanism design for the private provision of public goods". In: *The Journal of economic history* 67.2 (2007), pp. 445–480.

[22] Paul A Samuelson. "The pure theory of public expenditure". In: *The review of economics and statistics* (1954), pp. 387–389.

[23] Ingrid Ott and Stephen J Turnovsky. "Excludable and non-excludable public inputs: Consequences for economic growth". In: *Economica* 73.292 (2006), pp. 725–748.

[24] Richard J Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. "On approximately fair allocations of indivisible goods". In: *Proceedings of the 5th ACM Conference on Electronic Commerce*. 2004, pp. 125–131.

[25] Samir Wadhwa and Roy Dong. "Failure of Equilibrium Selection Methods for Multiple-Principal, Multiple-Agent Problems with Non-Rivalrous Goods: An Analysis of Data Markets". In: *arXiv preprint arXiv:2004.00196* (2020).

[26] Shinji Ohseto. "Characterizations of Strategy-Proof Mechanisms for Excludable versus Nonexcludable Public Projects". In: *Games and Economic Behavior* 32.1 (2000), pp. 51 –66. ISSN: 0899-8256.

[27] Hitoshi Matsushima. "Mechanism design with side payments: Individual rationality and iterative dominance". In: *Journal of Economic Theory* 133.1 (2007), pp. 1–30.

[28] Jinpeng Ma. "Strategy-proofness and the strict core in a market with indivisibilities". In: *International Journal of Game Theory* 23.1 (1994), pp. 75–83.

[29] Hervé Moulin. "Serial Cost-Sharing of Excludable Public Goods". In: *The Review of Economic Studies* 61.2 (1994), pp. 305–325. ISSN: 00346527, 1467937X.

[30] Mingyu Guo and Vincent Conitzer. "Computationally Feasible Automated Mechanism Design: General Approach and Case Studies". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1868.

[31] Mingyu Guo, Hideaki Hata, and M. Ali Babar. "Optimizing Affine Maximizer Auctions via Linear Programming: An Application to Revenue Maximizing Mechanism Design for Zero-Day Exploits Markets". In: *PRIMA 2017: Principles and Practice of Multi-Agent Systems - 20th International Conference, Nice, France, October 30 - November 3, 2017, Proceedings*. 2017, pp. 280–292.

[32] Richard Cole and Tim Roughgarden. "The sample complexity of revenue maximization". In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 2014, pp. 243–252.

[33] Jamie H Morgenstern and Tim Roughgarden. "On the pseudo-dimension of nearly optimal auctions". In: *Advances in Neural Information Processing Systems* 28 (2015).

[34] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. "The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions". In: *Games and Economic Behavior* 46.1 (2004), pp. 174–188.

[35] Rajdeep K Dash, Sarvapali D Ramchurn, and Nicholas R Jennings. "Trust-based mechanism design". In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.* IEEE. 2004, pp. 748–755.

[36] Eduardo M Azevedo and Eric Budish. "Strategy-proofness in the large". In: *The Review of Economic Studies* 86.1 (2019), pp. 81–116.

[37] Paul Anand et al. "Foundations of rational choice under risk". In: *OUP Catalogue* (1995).

[38] Dilip Mookherjee and Stefan Reichelstein. "Dominant strategy implementation of Bayesian incentive compatible allocation rules". In: *Journal of Economic Theory* 56.2 (1992), pp. 378–399.

[39] Tim Roughgarden. "Algorithmic game theory". In: *Communications of the ACM* 53.7 (2010), pp. 78–86.

[40] Gregory Pavlov. "Optimal mechanism for selling two goods". In: *The BE Journal of Theoretical Economics* 11.1 (2011).

[41] Andrew Chi-Chih Yao. "Dominant-strategy versus bayesian multi-item auctions: Maximum revenue determination and comparison". In: *Proceedings of the 2017 ACM Conference on Economics and Computation*. 2017, pp. 3–20.

[42] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. "An algorithmic characterization of multi-dimensional mechanisms". In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 2012, pp. 459–478.

[43] Sergiu Hart and Noam Nisan. "Approximate revenue maximization with multiple items". In: *Journal of Economic Theory* 172 (2017), pp. 313–347.

[44] Vijay Krishna. *Auction theory*. Academic press, 2009.

[45] Mark Armstrong. "Multiproduct nonlinear pricing". In: *Econometrica: Journal of the Econometric Society* (1996), pp. 51–75.

[46] Vincent Conitzer and Tuomas Sandholm. "Complexity of Mechanism Design". In: *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*. Ed. by Adnan Darwiche and Nir Friedman. Morgan Kaufmann, 2002, pp. 103–110.

[47] Radu Jurca and Boi Faltings. "Minimum payments that reward honest reputation feedback". In: *Proceedings of the 7th ACM Conference on Electronic Commerce*. 2006, pp. 190–199.

[48] Florin Constantin and David C Parkes. "On revenue-optimal dynamic auctions for bidders with interdependent values". In: *Agent-Mediated Electronic Commerce and Trading Agent Design and Analysis*. Springer, 2007, pp. 1–15.

[49] Sayan Bhattacharya, Gagan Goel, Sreenivas Gollapudi, and Kamesh Munagala. "Budget constrained auctions with heterogeneous items". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 379–388.

[50] Anton Likhodedov and Tuomas Sandholm. "Methods for Boosting Revenue in Combinatorial Auctions". In: *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*. Ed. by Deborah L. McGuinness and George Ferguson. AAAI Press / The MIT Press, 2004, pp. 232–237. URL: http://www.aaai.org/Library/AAAI/2004/aaai04-037.php.

[51] Hervé Moulin. *The price of anarchy of serial cost sharing and other methods*. Tech. rep. Citeseer, 2005.

[52] Mingyu Guo, Yong Yang, and Muhammad Ali Babar. "Cost Sharing Security Information with Minimal Release Delay". In: *PRIMA 2018: Principles and Practice of Multi-Agent Systems*. Cham: Springer International Publishing, 2018, pp. 177–193. ISBN: 978-3-030-03098-8.

[53] William Vickrey. "Counterspeculation, Auctions, and Competitive Sealed Tenders". In: *Journal of Finance* 16 (1961), pp. 8–37.

[54] Edward H Clarke. "Multipart pricing of public goods". In: *Public choice* (1971), pp. 17–33.

[55] Theodore Groves. "Incentives in teams". In: *Econometrica: Journal of the Econometric Society* (1973), pp. 617–631.

[56] Mingyu Guo and Vincent Conitzer. "Undominated VCG redistribution mechanisms". In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. Citeseer. 2008, pp. 1039–1046.

[57] Mingyu Guo. "VCG Redistribution with Gross Substitutes". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3733.

[58] Victor Naroditskiy, Mingyu Guo, Lachlan Dufton, Maria Polukarov, and Nicholas R. Jennings. "Redistribution of VCG Payments in Public Project Problems". In: *Internet and Network Economics - 8th International Workshop, WINE 2012, Liverpool, UK, December 10-12, 2012. Proceedings*. Ed. by Paul W. Goldberg. Vol. 7695. Lecture Notes in Computer Science. Springer, 2012, pp. 323–336.

[59] Mingyu Guo, Hideaki Hata, and M. Ali Babar. "Revenue Maximizing Markets for Zero-Day Exploits". In: *PRIMA 2016: Princiles and Practice of Multi-Agent Systems - 19th International Conference, Phuket, Thailand, August 22-26, 2016, Proceedings*. 2016, pp. 247–260.

[60] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[61] M-F Balcan, Avrim Blum, Jason D Hartline, and Yishay Mansour. "Mechanism design via machine learning". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. IEEE. 2005, pp. 605–614.

[62] Harikrishna Narasimhan, Shivani Brinda Agarwal, and David C Parkes. "Automated mechanism design without money via machine learning". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. 2016.

[63] Poonam Sharma and Akansha Singh. "Era of deep neural networks: A review". In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE. 2017, pp. 1–5.

[64] Saeed Alaei, Hu Fu, Nima Haghpanah, Jason Hartline, and Azarakhsh Malekian. "Bayesian optimal auctions via multi-to single-agent reduction". In: *arXiv preprint arXiv:1203.5099* (2012).

[65] Jason D Hartline and Tim Roughgarden. "Simple versus optimal mechanisms". In: *Proceedings of the 10th ACM conference on Electronic commerce*. 2009, pp. 225–234.

[66] Andrew Chi-Chih Yao. "An n-to-1 bidder reduction for multi-item auctions and its applications". In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 92–109.

[67] Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. "Evolutionary diversity optimization using multi-objective indicators". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Anne Auger and Thomas Stützle. ACM, 2019, pp. 837–845. DOI: 10.1145/3321707.3321796. URL: https://doi.org/10.1145/3321707.3321796.

[68] Qian Long, Zihan Zhou, Abhinav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. "Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020. URL: https://openreview.net/forum?id=SJxbHkrKDH.

[69] Viet Anh Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. "Analysis of Evolutionary Diversity Optimisation for Permutation Problems". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2021*. 2021.

[70] Steve Phelps, Peter McBurney, and Simon Parsons. "Evolutionary mechanism design: a review". In: *Autonomous agents and multi-agent systems* 21.2 (2010), pp. 237–264.

[71] Martin Andrews. "Genetic programming for the acquisition of double auction market strategies". In: *Advances in genetic programming* (1994).

[72] Dave Cliff. "Evolving parameter sets for adaptive trading agents in continuous double-auction markets". In: *Agents-98 workshop on artificial societies and computational markets, Minneapolis, MN*. 1998, pp. 38–47.

[73] Vincent Conitzer and Tuomas Sandholm. "Incremental Mechanism Design." In: *IJCAI*. 2007, pp. 1251–1256.

[74] Dave Cliff. "Evolution of market mechanism through a continuous space of auction-types". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*. Vol. 2. IEEE. 2002, pp. 2029–2034.

[75] Steve Phelps, Peter McBurney, Simon Parsons, and Elizabeth Sklar. "Co-evolutionary auction mechanism design: A preliminary report". In: *International Workshop on Agent-Mediated Electronic Commerce*. Springer. 2002, pp. 123–142.

[76] Steve Phelps, Peter McBurney, Simon Parsons, and Elizabeth Sklar. "Applying genetic programming to economic mechanism design: evolving a pricing rule for a continuous double auction". In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. 2003, pp. 1096–1097.

[77] BR Fox and MB McMahon. "Genetic operators for sequencing problems". In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 284–300.

[78] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David Parkes, and Sai Srivatsa Ravindranath. "Optimal auctions through deep learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1706–1715.

[79] Zhe Feng, Harikrishna Narasimhan, and David C Parkes. "Deep learning for revenue-optimal auctions with budgets". In: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. 2018, pp. 354–362.

[80] Noah Golowich, Harikrishna Narasimhan, and David C Parkes. "Deep Learning for Multi-Facility Location Mechanism Design." In: *IJCAI*. 2018, pp. 261–267.

[81] Yuko Sakurai, Satoshi Oyama, Mingyu Guo, and Makoto Yokoo. "Deep False-Name-Proof Auction Mechanisms". In: *PRIMA 2019: Principles and Practice of Multi-Agent Systems - 22nd International Conference, Turin, Italy, October 28-31, 2019, Proceedings*. Vol. 11873. Lecture Notes in Computer Science. Springer, 2019, pp. 594–601. DOI: 10.1007/978-3-030-33792-6_45. URL: https://doi.org/10.1007/978-3-030-33792-6_45.

[82] Michael Curry, Ping-Yeh Chiang, Tom Goldstein, and John Dickerson. "Certifying strategyproof auction networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4987–4998.

[83] Neehar Peri, Michael Curry, Samuel Dooley, and John Dickerson. "PreferenceNet: Encoding Human Preferences in Auction Design with Deep Learning". In: *Advances in Neural Information Processing Systems* 34 (2021).

[84] Padala Manisha, C. V. Jawahar, and Sujit Gujar. "Learning Optimal Redistribution Mechanisms Through Neural Networks". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. Ed. by Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018, pp. 345–353.

[85] Weiran Shen, Pingzhong Tang, and Song Zuo. "Automated mechanism design via neural networks". In: *arXiv preprint arXiv:1805.03382* (2018).

[86] Jad Rahme, Samy Jelassi, and S Matthew Weinberg. "Auction learning as a two-player game". In: *arXiv preprint arXiv:2006.05684* (2020).

[87] Zhijian Duan, Jingwu Tang, Yutong Yin, Zhe Feng, Xiang Yan, Manzil Zaheer, and Xiaotie Deng. "A Context-Integrated Transformer-Based Neural Network for Auction Design". In: *arXiv preprint arXiv:2201.12489* (2022).

[88] Jad Rahme, Samy Jelassi, Joan Bruna, and S Matthew Weinberg. "A permutation-equivariant neural network architecture for auction design". In: *arXiv preprint arXiv:2003.01497* (2020), p. 4.

[89] Yufeng Zhan and Jiang Zhang. "An incentive mechanism design for efficient edge learning by deep reinforcement learning approach". In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE. 2020, pp. 2489–2498.

[90] Martin Bichler, Maximilian Fichtl, Stefan Heidekrüger, Nils Kohring, and Paul Sutterer. "Learning equilibria in symmetric auction games using artificial neural networks". In: *Nature Machine Intelligence* 3.8 (2021), pp. 687–695.

[91] Gianluca Brero, Alon Eden, Matthias Gerstgrasser, David C Parkes, and Duncan Rheingans-Yoo. "Reinforcement learning of simple indirect mechanisms". In: *arXiv preprint arXiv:2010.01180* (2020).

[92] Makoto Yokoo. "Characterization of Strategy/False-name Proof Combinatorial Auction Protocols: Price-oriented, Rationing-free Protocol". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 733–739.

[93] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[94] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[95] Joao F Martins, V Ferno Pires, and Armando J Pires. "Unsupervised neural-network-based algorithm for an on-line diagnosis of three-phase induction motor stator fault". In: *IEEE Transactions on Industrial Electronics* 54.1 (2007), pp. 259–264.

[96] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[97] Xin Yi, Ekta Walia, and Paul Babyn. "Generative adversarial network in medical imaging: A review". In: *Medical image analysis* 58 (2019), p. 101552.

[98] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[99] J. Moore. *General Equilibrium and Welfare Economics: An Introduction*. Springer, 2006.

[100] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.

[101] Noah Golowich, Harikrishna Narasimhan, and David C. Parkes. "Deep Learning for Multi-Facility Location Mechanism Design". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*.

International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 261–267.

[102] Paul Duetting, Zhe Feng, Harikrishna Narasimhan, David Parkes, and Sai Srivatsa Ravindranath. "Optimal Auctions through Deep Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, 2019, pp. 1706–1715.

[103] Weiran Shen, Pingzhong Tang, and Song Zuo. "Automated Mechanism Design via Neural Networks". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. Montreal QC, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 215–223. ISBN: 978-1-4503-6309-9.

[104] Rajat Deb and Laura Razzolini. "Voluntary cost sharing for an excludable public project". In: *Mathematical Social Sciences* 37.2 (1999), pp. 123 –138. ISSN: 0165-4896.

[105] Mark Bagnoli and Ted Bergstrom. "Log-concave probability and its applications". In: *Economic Theory* 26.2 (2005), pp. 445–469. ISSN: 1432-0479. DOI: 10.1007/s00199-004-0514-4.

[106] Ran Shao and Lin Zhou. "Optimal allocation of an indivisible good". In: *Games and Economic Behavior* 100 (2016), pp. 95 –112. ISSN: 0899-8256.

[107] Joseph Sill. "Monotonic Networks". In: *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*. NIPS '97. Denver, Colorado, USA: MIT Press, 1998, pp. 661–667. ISBN: 0-262-10076-2.

[108] Zhi-Hua Zhou and Ji Feng. "Deep Forest: Towards an Alternative to Deep Neural Networks". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. Melbourne, Australia: AAAI Press, 2017, pp. 3553–3559. ISBN: 978-0-9992411-0-3.

[109] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*. 2017.

[110] Andy Greenberg. *Shopping For Zero-Days: A Price List For Hackers' Secret Software Exploits*. 2012.

[111] Dennis Fisher. *VUPEN Founder Launches New Zero-Day Acquisition Firm Zerodium*. July 24, 2015 online: https://threatpost.com/vupen-launches-new-zero-day-acquisition-firm-zerodium/113933/. 2015.

[112] Guanhua Wang, Runqi Guo, Yuko Sakurai, Ali Babar, and Mingyu Guo. "Mechanism Design for Public Projects via Neural Networks". In: *20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021, online)*. May 2021.

[113] Ruggiero Cavallo. "Optimal Decision-making with Minimal Waste: Strategyproof Redistribution of VCG Payments". In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '06. Hakodate, Japan: ACM, 2006, pp. 882–889. ISBN: 1-59593-303-4. DOI: 10.1145/1160633.1160790. URL: http://doi.acm.org/10.1145/1160633.1160790.

[114] Geoffroy de Clippel, Victor Naroditskiy, Maria Polukarov, Amy Greenwald, and Nicholas R. Jennings. "Destroy to save". In: *Games and Economic Behavior* 86 (2014), pp. 392–404.

[115] Boi Faltings. "A Budget-Balanced, Incentive-Compatible Scheme for Social Choice". eng. In: *Lecture notes in computer science*. Berlin: Springer, 2005, pp. 30–43. ISBN: 9783540297376.

[116] Mingyu Guo and Vincent Conitzer. "Worst-case optimal redistribution of VCG payments in multi-unit auctions". In: *Games and Economic Behavior* 67.1 (2009), pp. 69–98.

[117] Hervé Moulin. "Almost budget-balanced VCG mechanisms to assign multiple objects". In: *JET* 144.1 (2009), pp. 96–119.

[118] Sujit Gujar and Y. Narahari. "Redistribution Mechanisms for Assignment of Heterogeneous Objects". In: *J. Artif. Intell. Res.* 41 (2011), pp. 131–154.

[119] Mingyu Guo. "Worst-case optimal redistribution of VCG payments in heterogeneous-item auctions with unit demand". In: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012*

(3 Volumes). Ed. by Wiebe van der Hoek, Lin Padgham, Vincent Conitzer, and Michael Winikoff. IFAAMAS, 2012, pp. 745–752. URL: http://dl.acm.org/citation.cfm?id=2343803.

[120] Mingyu Guo and Vincent Conitzer. "Better redistribution with inefficient allocation in multi-unit auctions". In: *Artificial Intelligence* 216 (2014), pp. 287–308.

[121] Shunsuke Tsuruta, Masaaki Oka, Taiki Todo, Yujiro Kawasaki, Mingyu Guo, Yuko Sakurai, and Makoto Yokoo. "Optimal false-name-proof single-item redistribution mechanisms." In: *AAMAS*. Citeseer. 2014, pp. 221–228.

[122] Mingyu Guo. "Competitive VCG Redistribution Mechanism for Public Project Problem". In: *PRIMA 2016: Princiles and Practice of Multi-Agent Systems - 19th International Conference, Phuket, Thailand, August 22-26, 2016, Proceedings*. Vol. 9862. Lecture Notes in Computer Science. Springer, 2016, pp. 279–294.

[123] Mingyu Guo and Hong Shen. "Speed up Automated Mechanism Design by Sampling Worst-Case Profiles: An Application to Competitive VCG Redistribution Mechanism for Public Project Problem". In: *PRIMA 2017: Principles and Practice of Multi-Agent Systems - 20th International Conference, Nice, France, October 30 - November 3, 2017, Proceedings*. Vol. 10621. Lecture Notes in Computer Science. Springer, 2017, pp. 127–142.

[124] Mingyu Guo. "An Asymptotically Optimal VCG Redistribution Mechanism for the Public Project Problem". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 315–321.

[125] Mingyu Guo, Victor Naroditskiy, Vincent Conitzer, Amy Greenwald, and Nicholas R. Jennings. "Budget-Balanced and Nearly Efficient Randomized Mechanisms: Public Goods and beyond". In: *Internet and Network Economics - 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011. Proceedings*. Ed. by Ning Chen, Edith Elkind, and Elias Koutsoupias. Vol. 7090. Lecture Notes in Computer Science. Springer, 2011, pp. 158–169.

[126] Andrea Tacchetti, DJ Strouse, Marta Garnelo, Thore Graepel, and Yoram Bachrach. "A neural architecture for designing truthful and efficient auctions". In: *arXiv preprint arXiv:1907.05181* (2019).

[127] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. "Gradient estimation using stochastic computation graphs". In: *arXiv preprint arXiv:1506.05254* (2015).

[128] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *ICLR 2017* (2016).

[129] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. "Understanding deep neural networks with rectified linear units". In: *arXiv preprint arXiv:1611.01491* (2016).

[130] Serge Egelman, Cormac Herley, and Paul C. van Oorschot. "Markets for Zero-Day Exploits: Ethics and Implications". In: *Proceedings of the 2013 New Security Paradigms Workshop*. NSPW '13. Banff, Alberta, Canada: Association for Computing Machinery, 2013, 41–46. ISBN: 9781450325820. DOI: 10.1145/2535813.2535818. URL: https://doi.org/10.1145/2535813.2535818.

[131] Hideaki Hata, Mingyu Guo, and M. Ali Babar. "Understanding the Heterogeneity of Contributors in Bug Bounty Programs". In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*. Ed. by Ayse Bener, Burak Turhan, and Stefan Biffl. IEEE, 2017, pp. 223–228.

[132] Tetsuya Kanda, Mingyu Guo, Hideaki Hata, and Kenichi Matsumoto. "Towards understanding an open-source bounty: Analysis of bountysource". In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2017, pp. 577–578.

[133] Mingyu Guo, Argyrios Deligkas, and Rahul Savani. "Increasing VCG Revenue by Decreasing the Quality of Items". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014,

pp. 705–711. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8186.

[134] Mingyu Guo and Argyrios Deligkas. "Revenue Maximization via Hiding Item Attributes". In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 157–163. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6909.

[135] Mingyu Guo, Hong Shen, Taiki Todo, Yuko Sakurai, and Makoto Yokoo. "Social Decision with Minimal Efficiency Loss: An Automated Mechanism Design Approach". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*. Ed. by Gerhard Weiss, Pinar Yolum, Rafael H. Bordini, and Edith Elkind. ACM, 2015, pp. 347–355. URL: http://dl.acm.org/citation.cfm?id=2772925.

[136] The Chromium Projects. *Severity Guidelines for Security Issues*. Accessed September 15, 2015 online: https://www.chromium.org/developers/severity-guidelines. 2015.