Worst-Case VCG Redistribution Mechanism Design Based on the Lottery Ticket Hypothesis

Mingyu Guo

School of Computer and Mathematical Sciences
University of Adelaide, Australia
mingyu.guo@adelaide.edu.au

Abstract

We study worst-case VCG redistribution mechanism design for the public project problem. The mechanism design task comes down to designing a payment function that maximizes the worst-case allocative efficiency ratio.

We propose a suite of techniques for worst-case mechanism design via neural networks. We use a multilayer perceptron (MLP) with RELU activation to model the payment function and use mixed integer programming (MIP) to solve for the worst-case type profiles that maximally violate the mechanism design constraints. We collect these worst-case type profiles and use them as training samples to train toward better worst-case mechanisms.

In practice, we require a tiny neural network structure for the above approach to scale. The Lottery Ticket Hypothesis states that a large network is likely to contain a "winning ticket" – a much smaller subnetwork that "won the initialization lottery", which makes its training particularly effective. Motivated by this hypothesis, we train a large network and prune it into a tiny subnetwork (i.e., "draw" a ticket). We run MIP-based worst-case training on the drawn subnetwork and evaluate the resulting mechanism's worst-case performance (i.e., "scratch" the ticket). If the subnetwork does not achieve good worst-case performance, then we record the type profiles that cause the current draw to be bad. To draw again, we restore the large network to its initial weights and prune using recorded type profiles from earlier draws (i.e., redraw from the original ticket pot while avoiding drawing the same ticket twice). We expect to eventually encounter a tiny subnetwork that leads to effective training for our worst-case mechanism design task. Lastly, a by-product of multiple ticket draws is an ensemble of mechanisms with different worst cases, which improves the worst-case performance further.

Using our approach, we find previously unknown optimal mechanisms for up to 5 agents. Our results confirm the tightness of conjectured theoretical upper bounds. For up to 20 agents, we derive significantly improved worst-case mechanisms, surpassing a long list of existing manual results.

Introduction

The *public project problem* (Mas-Colell, Whinston, and Green 1995; Moore 2006; Moulin 1988) is a fundamental mechanism design model that has been well studied in both computer science and economics. Under this model, a group of agents decide whether or not to build a public project (i.e., a bridge). In this paper, we focus on *binary* and *non-excludable* public project. That is, the mechanism decision is binary – we either build the project or not. The project is *non-excludable* in the sense that once it is built, it can be consumed by every agent, regardless of their contributions. Our model follows prior works on *VCG redistribution mechanism design for the public project problem* (Naroditskiy et al. 2012; Guo 2016; Guo and Shen 2017; Guo 2019; Wang, Zuo, and Guo 2021). Below we describe the model.

Definition 1 (Public Project Problem)**.** n agents decide whether or not to build a public project that costs 1. Agent i's type is θ_i ($0 \leq \theta_i \leq 1$). If the mechanism decision is NOT BUILD, then agent i retains her share of the project cost $\frac{1}{n}$ and her valuation is $\frac{1}{n}$ for this outcome. If the mechanism decision is to BUILD, then agent i's valuation is θ_i.

VCG redistribution mechanisms all share the same allocation rule. As the name suggests, we allocate *efficiently* according to the VCG mechanism (Vickrey 1961; Clarke 1971; Groves 1973). In the context of the public project problem, the project is built if and only if the agents' total valuation of the project reaches the project cost. That is, the project is built if and only if $\sum_i \theta_i \geq 1$. VCG redistribution mechanisms first charge the VCG payments, which ensures *strategy-proofness*. Then on top of the VCG payments, every agent receives back a redistribution payment. Agent i's redistribution must be independent of her own type θ_i, which ensures that the redistribution term does not affect the original VCG mechanism's efficiency and strategy-proofness. We further require that the total amount redistributed does not exceed the total VCG payment collected (i.e., we require the *non-deficit* constraint). Actually, due to Holmström's characterization (Holmström 1979), for our model, *the VCG redistribution mechanisms are the only mechanisms that are strategy-proof and efficient*. Also due to Holmström's characterization, VCG redistribution mechanisms are essentially *non-deficit Groves mechanisms* (Groves 1973).

Definition 2 (VCG redistribution mechanisms for the public project problem (Naroditskiy et al. 2012))**.** The project is built if and only $\sum_i \theta_i \geq 1$. Agent i receives $\sum_{j \neq i} \theta_j - h(\theta_{-i})$ if the project is built and receives $\frac{n-1}{n} - h(\theta_{-i})$ if the project is not built.

Function h (often called the *Groves term*) characterizes a specific VCG redistribution mechanism. The mechanism design task comes down to designing h.

There is only one mechanism design constraint. We use $\vec{\theta}$ to denote the type profile. We define $s(\vec{\theta}) = \max\{\sum_i \theta_i, 1\}$, which is the agents' total valuation under the efficient allocation. Based on the above definition, the agents' total payment received is $(n-1)s(\vec{\theta}) - \sum_i h(\theta_{-i})$. The non-deficit constraint is therefore $(n-1)s(\vec{\theta}) \leq \sum_i h(\theta_{-i})$.

The mechanism design objective is to maximize the *worst-case allocative efficiency ratio* (Moulin 2009), which is defined as the worst-case ratio between the agents' achieved total utility and the *first-best* total utility (the maximum total utility assuming agents do not lie, which is an upper bound on achievable total utility by non-deficit mechanisms). For our model, the first-best total utility is exactly $s(\vec{\theta})$. The agents' total utility is the total valuation $s(\vec{\theta})$ plus the total payment received, which sum up to $ns(\vec{\theta}) - \sum_i h(\theta_{-i})$. We use α to denote the worst-case allocative efficiency ratio, which implies $ns(\vec{\theta}) - \sum_i h(\theta_{-i}) \geq \alpha s(\vec{\theta})$.

In summary, the mechanism design constraint and objective can be summarized into one inequality:

$$\forall \vec{\theta}, \quad (n-1)s(\vec{\theta}) \leq \sum_i h(\theta_{-i}) \leq (n-\alpha)s(\vec{\theta}) \qquad (1)$$

The whole mechanism design task is to find the largest worst-case allocative efficiency ratio α, by designing a function h that satisfies Inequality 1 for all type profiles.

In this paper, we design h using *worst-case* mechanism design via neural networks.

Related Research

VCG Redistribution Mechanism Design

VCG redistribution mechanisms have been studied extensively for various auction settings, including multi-unit auctions with unit demand (Moulin 2009; de Clippel et al. 2014) or with nonincreasing marginal values (Guo and Conitzer 2009), hetereogenous-item auctions with unit demand (Gujar and Narahari 2011; Guo 2012), combinatorial auctions with gross substitutes (Guo 2011), false-name-proof single-item auction (Tsuruta et al. 2014), combinatorial auctions (Cavallo 2006) and diffusion auctions on networks (Gu et al. 2023).

(Naroditskiy et al. 2012) first studied worst-case VCG redistribution mechanism design for the public project problem. The authors proposed a theoretical upper bound on the worst-case allocative efficiency ratio. The authors *conjectured* that the theoretical upper bound is tight. For 3 agents only, the authors manually derived a mechanism whose worst-case allocative efficiency ratio matches the theoretical upper bound. That is, the theoretical upper bound is indeed tight for 3 agents. For more agents, despite many attempts, no optimal mechanisms were found and we do not know whether the theoretical upper bounds are tight or not.

The existing works on VCG redistribution mechanism design relied mostly on manual efforts or classic automated mechanism design (Conitzer and Sandholm 2002). There have also been works on designing VCG redistribution mechanisms via neural networks. (Manisha, Jawahar, and Gujar 2018) studied a different redistribution model (auctions) and relied on *Monte Carlo simulation* for worst-case performance evaluation. (Wang, Zuo, and Guo 2021) studied the same model as this paper's. The authors also relied on Monte Carlo for worst-case evaluation. Unfortunately, Monte Carlo leads to *significantly inflated* worst-case performances. In this paper, we propose a suite of techniques for worst-case mechanism design via neural networks, which involves *exact worst-case evaluation*. For example, for $n = 10$, we derived one example mechanism whose worst-case allocative efficiency ratio is 0.685 (evaluated exactly via mixed integer programming). This example mechanism's worst-case allocative efficiency ratio inflates to 0.765 if we simulate using $10,000$ random type profiles. Even with inflated numbers, (Wang, Zuo, and Guo 2021)'s achieved ratios are worse than this paper's results.

Neural Network Based Mechanism Design

(Dütting et al. 2019) proposed a general framework for mechanism design via neural networks. The authors' key idea is the "regret" term that models the current network's deviation from strategy-proofness. By minimizing regret, we end up with a mechanism that is *approximately* strategy-proof. (Golowich, Narasimhan, and Parkes 2018) applied regret and max-min monotone network (Sill 1998) to facility location. (Wang et al. 2021) studied public project mechanism design by proposing neural training techniques for iterative mechanisms.

(Curry, Sandholm, and Dickerson 2022) studied randomized affine maximizer auctions (AMAs). The authors named these mechanisms "lottery" AMAs. Here, "lottery" refers to the fact that the allocations are random. The authors also referenced the Lottery Ticket Hypothesis and noted that "some version of the lottery ticket hypothesis is in play here". That is, while allowing many possible allocations, only a tiny number of allocations are ever being used. Retaining initialization values for those allocations being used (by analysing an already designed mechanism) is a better way to initialize. The authors' approach does not involve pruning or subnetwork, so it is overall very loosely related to the hypothesis. (Guo et al. 2021) also applied neural networks for parameter tuning of AMA auctions, specifically in the context of revenue-maximizing markets for zero-day exploits.

(Curry et al. 2020) is the most relevant work to ours. The authors used RELU-based networks to model auctions and applied mixed integer programming to solve for the exact constraint violation (called certified regret). The authors showed that certified regret is higher than empirical regret. The authors also pointed out an example where the average certified regret is small, but the distribution of regret is skewed in the sense that there exist type profiles with much higher regrets, which also suggests that using Monte Carlo for worst-case analysis is a flawed approach. The certification process is a tool for analyzing mechanisms that have already been trained using random samples. In our paper, we do not consider the *average* maximum constraint violation. We consider the *absolute worst-case*. Furthermore, the

worst-case violation *amount* itself is not something we particularly care about. What we actually need are the worst-case *type profiles* that lead to the worst-case constraint violation, as these worst-case type profiles can be fed into the training process to improve worst-case performance.

Summary of Contributions

We propose a suite of novel techniques for worst-case mechanism design via neural networks, many of which are model-independent general techniques. We use multilayer perceptron (MLP) with RELU activation to model mechanisms and apply mixed integer programming (MIP) to derive worst-case type profiles that maximally violate the mechanism constraints. *These worst-case profiles are collected and used as training samples to train toward better worst-case mechanisms.*

MIP-based worst-case mechanism analysis requires *one binary variable for every network node and for every agent* (for modelling RELU), which does not scale beyond tiny networks. Facing this challenge, we propose a systematic way to derive tiny *trainable* networks based on the Lottery Ticket Hypothesis. The Lottery Ticket Hypothesis (Frankle and Carbin 2019) states that a large network is likely to contain a "winning ticket" – a much smaller subnetwork that "won the initialization lottery", which makes its training particularly effective. For example, a large network with 40 nodes contains $\binom{40}{10}$ subnetworks with 10 nodes (considering only node pruning). Among the *exponential number of* subnetworks, there is a good chance that some have lucky initializations that make stochastic gradient descent particularly effective. The hypothesis states that training is essentially to "magnify" the winning subnetwork (i.e., reduce the weights of the rest of the nodes). Motivated by this hypothesis, we train a large network and prune it into a tiny subnetwork (i.e., "draw" a ticket) using random type profiles. That is, we identify a subnetwork that is *estimated* to have good worst-case performance based on random type profiles. We run MIP-based worst-case training on the drawn subnetwork and evaluate the training result's *exact* worst-case performance (i.e., "scratch" the ticket[1]). If the subnetwork can not achieve good worst-case performance, then we record the type profiles that cause the current draw to be bad. To draw again, we restore the large network to its initial weights and prune using recorded type profiles from earlier draws (i.e., redraw from the original ticket pot[2] while avoiding drawing the same ticket twice[3]). With a large enough initial network and a large enough number of draws, we expect to eventually encounter a winning ticket – a tiny trainable subnetwork that leads to effective training for our worst-case mechanism design task. Lastly, a by-product of multiple ticket draws is an ensemble of mechanisms with different worst cases,[4] which often improves the worst-case performance further.

Equally importantly, we managed to design (near) optimal or significantly improved worst-case VCG redistribution mechanisms, outperforming a long list of existing *manual* results, which also confirms the effectiveness of our worst-case mechanism design techniques.

The task of worst-case optimal VCG redistribution mechanism design has been *completely solved for various auction settings* (i.e., proven-optimal mechanisms were identified for any number of agents), including multi-unit auctions with unit demand (Moulin 2009), multi-unit auctions with nonincreasing marginal values (Guo and Conitzer 2009) and hetereogenous-item auctions with unit demand (Gujar and Narahari 2011; Guo 2012). *On the contrary, there has been almost no success for non-auction models.* For our model, the only known optimal mechanisms are restricted to 3 agents (Naroditskiy et al. 2012; Guo and Shen 2017).

(Naroditskiy et al. 2012) proposed a theoretical upper bound on the worst-case allocative efficiency ratio and conjectured that the bound is tight. For 4 agents, the conjectured tight bound is $\frac{2}{3}$. The authors managed to reach a worst-case allocative efficiency ratio of $\frac{1}{3}$ via their SBR mechanism. (Guo 2016)'s ABR mechanism improved the ratio to 0.459. (Guo and Shen 2017) applied automated mechanism design to push up the ratio to 0.600. (Guo 2019) finally improved the ratio to 0.625 by combining automated mechanism design and manual efforts. Still, the achieved ratio was below the theoretical upper bound. Before this paper, it was not known whether the theoretical upper bound $\frac{2}{3}$ is tight or not. For 5 agents, the story is similar. Despite many attempts, no optimal mechanisms for 5 agents were known and it was not clear whether the conjectured bound $\frac{5}{7}$ is attainable.

In this paper, *we finally identified (near) optimal mechanisms for* 4 *and* 5 *agents, whose worst-case allocative efficiency ratios match the conjectured upper bounds, with tiny gaps less than* 0.0001. Unfortunately, our mechanisms were trained via neural networks, so floating point errors are unavoidable. For 4 agents, h takes three input values a_0, a_1, a_2 ($a_0 \leq a_1 \leq a_2$). Our (near) optimal mechanism uses only 5 hidden nodes. Our mechanism is given as $h(a_0, a_1, a_2) =$
$\text{RELU}(-0.7220 \cdot a_0 - 0.5927 \cdot a_1 - 0.5925 \cdot a_2 + 0.5926) +$
$\text{RELU}(-0.4485 \cdot a_0 - 0.5939 \cdot a_1 - 0.3858 \cdot a_2 + 0.3856) +$
$\text{RELU}(0.1925 \cdot a_0 + 0.4570 \cdot a_1 + 0.4436 \cdot a_2 - 0.2218) -$
$\text{RELU}(-0.4820 \cdot a_0 - 0.3097 \cdot a_1 - 0.0915 \cdot a_2 + 0.3667) +$
$0.9197 \cdot a_0 + 0.6558 \cdot a_1 + 0.6646 \cdot a_2 + 0.2218$

For 5 agents, the (near) optimal mechanism uses 20 hidden nodes.

For 3 agents, we also identified a previously unknown optimal mechanism with the smallest possible network structure. It is represented using only 2 hidden nodes![5] When there are 3 agents, h takes two input values x and y. We assume $x \leq y$ without loss of generality. Our mechanism is $h(x, y) = \frac{2}{3}\text{RELU}(x + y - 1) + \frac{1}{6}\text{RELU}(5x + 3y - 2) + \frac{2}{3}$.

We would like to point out that we cannot skip the prun-

[1] Scalable as we only exactly evaluate tiny subnetworks.

[2] Restoring weights is different from complete re-initialization, as we desire to keep the original "ticket pot". If we reinitialize completely, then our experience from past draws (i.e., which subnetworks perform poorly) becomes no longer applicable.

[3] Recorded type profiles from previous draws "invalidate" tickets that we have already seen, by demonstrating that they perform poorly, which causes the pruning process to switch subnetworks.

[4] Our process ensures different worst cases for the next draw.

[5] We identified a long list of optimal mechanisms for 3 agents.

ing process and directly start from tiny networks (i.e., start from 2 nodes for 3 agents), because for most initializations, stochastic gradient descent would send us to bad local optimum. Certainly, we could try reinitializing tiny networks repeatedly as an alternative, but this is practically not scalable. The Lottery Ticket Hypothesis suggests that among an *exponential* number of subnetworks, some have lucky initializations (i.e., lucky initializations may be quite difficult to come by and may take exponential number of trials). The typical timescale of our approach is that it takes 1 hour to train and evaluate a network as we need to run MIP in every training epoch, so we cannot afford many trials. Therefore, a systematic approach for identifying trainable tiny networks is necessary. We show in an ablation study that our pruning-based approach is superior to repeated reinitializations (i.e., drawing 10 tickets via pruning gives us a much better tiny network than direct reinitializing tiny networks 10 times).

For up to 20 agents, we derive *significantly improved worst-case mechanisms*, surpassing existing manual results.

Worst-Case Training Algorithm

In this section, we present a worst-case training algorithm, which will be used as a *building block* in our main algorithm.

As mentioned, when it comes to worst-case VCG redistribution mechanism design for the public project problem, the whole mechanism design task is to maximizes the worst-case allocative efficiency ratio α, by designing a function h that satisfies Inequality 1 for every type profile. h takes θ_{-i} ($n-1$ dimensions) as inputs and the output has a single dimension. We use a fully connected multilayer perceptron (MLP) with only RELU activation functions to model h. We use $[k_1, k_2, \ldots, k_L]$ to denote the hidden layers' sizes, where k_i is the number of hidden nodes in the i-th hidden layer. I.e., $[20, 20]$ has 2 hidden layers with 20 hidden nodes per layer.

(Naroditskiy et al. 2012) manually designed the optimal mechanism for 3 agents, whose h function divides the domain into 4 regions, and each region takes a different form, but the overall function is continuous. A natural question to ask is "what the optimal mechanism looks like for $n \geq 4$?". If it so happens that the optimal h is *discontinuous* for $n \geq 4$ (i.e., it could be that the domain still divides into regions, but it is discontinuous cross regions), then MLP is not a suitable representation. Fortunately, we do not have to worry about discontinuity. We prove below that for any n, there exists a continuous h function that is arbitrarily close to optimality in terms of worst-case allocative efficiency ratio. Then based on Theorem 2 of (Hornik 1991), the *Universal Approximation Theorem* applies to our model due to that our type space is *compact*. That is, MLP with RELU is expressive enough to describe near optimal VCG redistribution mechanisms.

Theorem 1. *For any constant number of agents n, let α^* be the optimal worst-case allocative efficiency ratio. For any $\epsilon > 0$, there exists a VCG redistribution mechanism whose underlying h function is continuous, and has a worst-case efficiency ratio that is at least $\alpha^* - \epsilon$.*

The above theorem merely says that it is fine to focus on continuous h functions for our mechanism design objective, which confirms that MLP with RELU is expressive enough if we allow large networks, directly quoting the Universal Approximation Theorem. Essentially, the *prerequisite* for applying the Lottery Ticket Hypothesis is confirmed – a large enough MLP+RELU network will be near optimal for our model. The practical challenge is to find tiny trainable subnetworks.

The main idea of the algorithm in this section is to collect the worst-case type profiles that *maximally violate* Inequality 1, given h represented by a specific set of weights/biases. We then use these type profiles as training samples. The cost function represents the total constraint violation over a batch of sample type profiles. By minimizing cost, the mechanism's worst-case constraint violation generally decreases, which leads to better worst-case mechanisms.

Given h in the form of a RELU-activated MLP with specific weights/biases, we use mixed integer programming (MIP) to derive the worst-case type profiles that maximally violate the mechanism constraints. The left side of Inequality 1 is $(n-1)s(\vec{\theta}) \leq \sum_i h(\theta_{-i})$, which describes the non-deficit constraint. In our MIP, we maximize the violation $(n-1)s(\vec{\theta}) - \sum_i h(\theta_{-i})$. The MIP variables include θ_i, which are n continuous variables. Without loss of generality, we assume $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 1$. Also without loss of generality, we assume h's inputs are also ascending. That is, $h(\theta_{-i}) = h(\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n)$. Given h, whose weights/biases are treated as constants, we use one auxiliary binary variable *for each hidden node and for each agent* to model RELU. This is considering that for a specific hidden node, its activation status may be different when evaluating $h(\theta_{-i})$ and $h(\theta_{-j})$ for different agent i and j. The s function also involves another auxiliary binary variable. Via the above MIP, we can solve for the worst-case type profile that maximally violate the left side of Inequality 1 as well as the worst-case violation amount. We can construct a similar MIP to handle the right side of Inequality 1. For the right side, α is provided as a constant (i.e., a constant "goal" ratio). In summary, given h, we run two MIPs to derive

ϵ_L and ϵ_R: the maximum violation amounts on the left and the right side of Inequality 1

$\vec{\theta}_L$ and $\vec{\theta}_R$: the worst-case type profiles that maximally violate Inequality 1

Since ϵ_L and ϵ_R are *maximum violation amounts*, we have
$\forall \vec{\theta}, \quad (n-1)s(\vec{\theta}) - \epsilon_L \leq \sum_i h(\theta_{-i}) \leq (n-\alpha)s(\vec{\theta}) + \epsilon_R$
We define $h'(\theta_{-i}) = h(\theta_{-i}) + \frac{\epsilon_L}{n}$. Since $s(\vec{\theta}) \geq 1$, we have
$\forall \vec{\theta}, \quad (n-1)s(\vec{\theta}) \leq \sum_i h'(\theta_{-i}) \leq (n-(\alpha-\epsilon_L-\epsilon_R))s(\vec{\theta})$
That is, h' corresponds to a VCG redistribution mechanism that never violates the non-deficit constraint, and its worst-case allocative efficiency ratio is at least $\alpha - \epsilon_L - \epsilon_R$. In our training, the final mechanism deliverable is h' instead of h. The achieved worst-case performance is $\alpha - \epsilon_L - \epsilon_R$.

The pseudocode of our worst-case training algorithm is presented in Algorithm 1. It involves 3 training tricks:

Run MIPs only when necessary (line 10): We recall that in our MIPs, we need one binary variable for each hidden node and for each agent. The MIPs are the performance bottleneck. The idea of line 10 is that if the average loss is still large, then that means it should be relatively easy to encounter type profiles that violate the constraints via ran-

Algorithm 1: Worst-Case Training Algorithm

Input: Best manually-achieved worst-case allocative efficiency ratio $\alpha^L = \frac{n+1}{2n}$ (Guo 2019)
Theoretical upper bound on the worst-case allocative efficiency ratio α^U (Naroditskiy et al. 2012)

1: Initialize the store of worst-case type profiles WCP to empty
2: Initialize $\alpha^G = \alpha^L$ (α^G is the training goal for worst-case allocative efficiency ratio)
3: **while** TRUE **do**
4: Run Adam SGD on h with learning rate 0.0001 for 500 epochs
5: Training batch consists of:
6: 16 latest calculated worst-case type profiles (i.e., WCP[-16:])
7: 16 randomly sampled worst-case type profiles from earlier (i.e., from WCP[:-16])
8: 16 random type profiles
9: $n+1$ type profiles where the agents either report $\frac{1}{\lfloor n/2 \rfloor}$ or 0 (i.e., type profiles for deriving the conjectured upper bound (Naroditskiy et al. 2012))
10: **if** average loss over 500 epochs is above a loss threshold (threshold increases over time) **then**
11: Go back to line 4 (train again on existing samples)
12: **end if**
13: Run two MIPs with α^G to obtain ϵ_L, ϵ_R, $\vec{\theta}_L$, $\vec{\theta}_R$, also add $\vec{\theta}_L$ and $\vec{\theta}_R$ to WCP
14: **if** $\epsilon_L + \epsilon_R \leq 0.001$ **then**
15: $\alpha^L = \alpha^G$ and $\alpha^G = \frac{\alpha^U + \alpha^G}{2}$
16: **else**
17: $\alpha^G = \frac{\alpha^L + \alpha^G}{2}$
18: **end if**
19: **end while**

dom sampling (i.e., line 8). We train on these "free" samples instead of running MIPs to obtain more worst-case samples.

Revisit old worst-case type profiles (line 7): An intuition to explain this line is that training could enter a phase that is like the "Whac-A-Mole" game. Fixing type profile A's worst-case performance may cause type profile B to deteriorate. Fixing B then reverses the improvement on A. That is, during training, it's possible for a type profile to reappear over and over again as the worst-case type profile.

Adaptively change the "goalpost" by adjusting α^G (line 13-16): α^L is what we have already achieved. We expect to beat the best manual result so we start from the best manual result. α^L is nondecreasing throughout the process. α^U is the theoretical upper bound so it stays constant. α^G is our goal, which is adaptively adjusted according to a binary search style. We do not set the goal to the theoretical upper bound because the theoretical upper bound may not be achievable. Even if the bound is theoretically tight, the current network size and the current network's initialization may be insufficient to close the gap with respect to the theoretical upper bound. We adaptively adjust the goal to avoid learning an *impossible* task. Furthermore, we refer back to the "Whac-A-Mole" analogy, by gradually increasing α^G, we are essentially prioritizing on whacking the "tallest mole".

Ablation Study on the Three Training Tricks

We have already solved for the (near) optimal mechanisms for 4 and 5 agents, so in this ablation study, we focus on 6 to 10 agents, where we have not found the optimal mechanisms. The results are summarized in Table 1. The quality of training is naturally the "gap" between the achieved worst-case allocative efficiency ratio and the theoretical upper bound. We use WCT to denote our worst-case training algorithm. Earlier we mentioned three training tricks. We use WCT-i to denote the algorithm after removing the i-th trick. That is, WCT-1 runs MIPs even when the average loss is still high. WCT-2 does not revisit old worst-case type profiles. WCT-3 does not adaptively adjust the goal. Instead, it fixates the goal to the theoretical upper bound. We use network size $[10, 10]$. For each n and for each random seed from 0 to 4, we allocate 1 hour. The hardware allocated to each job is 1 CPU core from Intel Xeon Platinum 8360Y (for running MIPs) and 1 GPU core from Nvidia A100 (for neural network training). When presenting mechanisms' worse-case allocative efficiency ratios, we present "gaps with respect to theoretical upper bound", so lower numbers are better. For each algorithm, we present the average gaps and the minimum gap, both over 5 random seeds. Results show that all three training tricks are useful. The min gap is the more important metric. WCT is best for $n \in \{6, 9, 10\}$ and is the close second for $n = 8$. For $n = 7$, the achieved min gap is small, so fixating the gap to 0 turned out to be working well for WCT-3 (as a fluke as the avg gap of WCT-3 is worse).

Lottery Ticket Hypothesis Motivated Worst-Case Training Algorithm

Empirical Evidences Toward the Hypothesis

A limitation of WCT is the computational bottleneck caused by MIPs. We recall that the MIPs require one binary variable for every hidden node and for every agent, which makes it not scalable unless the network is tiny. WCT is not capable of training for too many epochs as training is slowed down by MIPs (i.e., if it takes 10 minutes to run one round of MIPs, then not much training can be done in 1 hours). For this reason, we focus on training using tiny (sub)networks with the help of the Lottery Ticket Hypothesis. We first present the empirical evidences leading to the hypothesis.

The first observation is that training results on tiny networks heavily depend on the initialization. As mentioned in the summary of contributions, we have identified an elegant optimal mechanism for 3 agents that uses only 2 hidden nodes. That is, the network structure $[2]$ is enough to represent an optimal mechanism for 3 agents. However, if we directly train using $[2]$ as the network structure, then we never encountered get any useful results. For example, we initialize $[2]$ for 10 trials using random seed 0 to 9, we run WCT for each seed for 1 hour. The best trial ended up with a gap of 0.1179 with respect to the theoretical upper bound (i.e., cannot discover optimal mechanisms). 8 out of 10 trials failed

	Avg Gap				Min Gap			
n	WCT	WCT-1	WCT-2	WCT-3	WCT	WCT-1	WCT-2	WCT-3
6	**0.2079**	0.2250	0.2167	0.2762	**0.1775**	0.1830	0.1906	0.2213
7	**0.1099**	0.1368	0.1258	0.1227	0.0888	0.0933	0.0916	**0.0644**
8	0.1665	**0.1280**	0.1479	0.1960	0.0973	**0.0939**	0.1078	0.1054
9	**0.1735**	0.2460	0.2021	0.2023	**0.1329**	0.1917	0.1575	0.1683
10	**0.3188**	0.3358	0.3334	0.4288	**0.2576**	0.3086	0.2848	0.3514

Table 1: Ablation Study on Worst-Case Training Algorithm

completely (i.e., not getting positive worst-case allocative efficiency ratios). We then experimented with network structures $[5]$, $[10]$ and $[15]$, also for 10 trials via random seed 0 to 9. We managed to discover near optimal mechanisms for 2, 3 and 7 trials out of 10, respectively. That is, having the "optimal" network structure (i.e., $[2]$) is not enough. *The larger the network, the higher chance there is for us to encounter a trainable initialization.*

Furthermore, even when we start from a large network, the final resulting mechanisms actually do not *need* all the nodes. The (near) optimal mechanisms discovered in this paper for 4 and 5 agents were obtained by *node pruning* using our lottery worst-case training algorithm, which will be introduced in the next section. For these (near) optimal mechanisms, we pruned from $[20]$ to $[2]$ for $n = 3$, from $[100]$ to $[5]$ for $n = 4$, and from $[100]$ to $[20]$ for $n = 5$. For $n = 3$, if we start from $[20]$, then we almost always can get the near optimal mechanism. If we directly analyse $[20]$ using MIPs for $n = 3$, then we are dealing with $20 \cdot 3 = 60$ binary variables, which is fine in terms of scalability. So for $n = 3$, we indeed do not require pruning. For $n = 4$ and 5, the story is different. For example, we used $[100]$ as the large network to solve for the optimal mechanisms for $n = 4$ and 5. It is very hard to handle MIPs with 400 or 500 binary variables as we need to run MIP in every epoch. In this paper, we consider up to $n = 20$ agents, which would be 2000 binary variables if $[100]$ is the network size, so pruning is needed for $n > 3$.

Lastly, for 3 agents, we have identified a long list of optimal mechanisms. We suspect that there are many optimal mechanisms for $n \geq 4$ as well, which is also a helpful property for the lottery-drawing approach (i.e., with many possible winning tickets, it is easier to encounter one).

Lottery Worst-Case Training Algorithm

We start with a large network. We use $[20, 20]$ in all experiments. We search for a tiny trainable subnetwork, which is our "winning ticket". To draw the first ticket, we use random type profiles to train the large network. Every time the average loss gets below a threshold, we prune one node until we reach the target network size.[6] For a node with weights $w_1, w_2, \ldots, w_k$ (i.e., in the next layer, the weights multiplied on this node's value), we define its *importance* as $\sum_i |w_i|$. The node's *relative importance* is its importance divided by the total importance from all nodes from the same layer. We prune the node that is the least important globally. After drawing the first ticket, we run the worst-case training algorithm WCT. The computational resources we allocate to each ticket in our experiments is 100 rounds of MIPs. Running WCT is like "scratching the ticket" to see whether it wins. Most likely we do not win from the first ticket. Before every next draw, we restore the large network to its initial weights. That is, the pot of lottery tickets stays the same. We aim to not draw any previous ticket again. The way we prevent drawing the same ticket again is as follows. We save the last 16 worst-case type profiles from the first ticket and use them for drawing the second ticket. The intuition is that, if the first ticket is not good enough, then it means the last 16 worst-case type profiles contain profiles that violate the mechanism design constraints significantly. When we draw the second ticket, we train not only using random type profiles, but also using these saved 16 profiles, which prevents us from drawing the same ticket (as the ticket we get is expected to work well for the type profiles we *already exposed* to it during its training). In summary, every time we draw a ticket, we evaluate it using WCT and save 16 additional worst-case type profiles. We use this cumulated set of type profiles for future draws to prevent old tickets. As empirical evidences point to the hypothesis, we expect that a winning ticket exists and we will eventually encounter it.

Ablation Study, Ensemble, and Scalability Test

We first present an ablation study in Table 2. For 6 to 10 agents, we start from $[20, 20]$ (40 nodes) and prune to 10 nodes[7] (left side of the table) and 20 nodes (right side of the table). For every setting, we draw 10 tickets and record the best result. That is, the computational resources we allocate for each trial is 100×10 rounds of MIPs. (We will also show an ablation study where we allocate the same amount of wall-clock time to each task.) LOTT. is the result from our lottery worst-case training algorithm. The table includes three other algorithms for comparison. Again, all numbers in tables represent gaps from theoretical upper bound.

REINIT1: Same as LOTT. but without the pruning part. That is, we start with $[5, 5]$ for the left side of the table and

[6] We prune nodes instead of edges as the number of nodes is a better indicator of the computational cost of the resulting subnetwork – recall that in MIPs we need one binary variable for each node and for each agent.

[7] Pruning from 40 nodes down to 10 nodes implies reducing the number of binary variables in MIPs from $40n$ to $10n$, which is significant speed up, from "impossible to run once" to "fast enough to run once in each training epoch".

	Min Gap for Ticket Size 10				Min Gap for Ticket Size 20			
n	LOTT.	REINIT0	REINIT1	REINIT2	LOTT.	REINIT0	REINIT1	REINIT2
6	**0.1929**	0.3951	0.2194	0.2289	**0.1662**	0.2020	0.1940	0.1863
7	**0.0768**	0.1186	0.1067	0.1102	**0.0489**	0.0853	0.0732	0.0819
8	**0.0967**	0.1216	0.1159	0.1306	**0.0699**	0.0904	0.0890	0.0939
9	**0.1226**	0.2840	0.1376	0.1396	**0.0984**	0.1122	0.1015	0.1161
10	**0.2118**	0.3117	0.2766	0.2781	**0.2002**	0.2369	0.2194	0.2226

Table 2: Ablation Study on Lottery Worst-Case Training Algorithm

start with $[10, 10]$ for the right side of the table. No pruning is needed as the ticket size is already the target size. For every ticket draw, we apply random initialization instead. Everything else stays the same as LOTT.. LOTT. produces the best result in every trial. REINIT1 is worse, which demonstrates that starting from small network is less likely to encounter good tickets. We need to start from a large network, train it for a while so that a good subnetwork emerges as training magnifies its relative weights, and then we can pick out the winning/trainable subnetwork via pruning.

REINIT2: Same as REINIT1 but we no longer share worst-case type profiles across different draws. REINIT2 is worse than REINIT1 in 9 out of 10 cases.

REINIT0: Same as REINIT2 but we no longer reinitialize the network across different draws. REINIT0 is basically to start with the same network over and over again and expect randomness brought in by training samples would lead to different performances over different trials. It is worse than REINIT2 in 7 out of 10 cases, which confirms again that initialization plays a big role in training quality.

For LOTT. results in Table 2, as we drew 10 tickets for each setting, we also examine whether we are indeed getting *new* tickets (i.e., different subnetworks) and *better* tickets (i.e., tickets that lead to better worst-case performance after 100 MIP rounds). We almost always get new tickets (78% of the tickets are new). We also mostly get better tickets, as 88% of the tickets are better than all previous tickets. It should be noted that the above results are after only 10 draws. If we keep drawing, we eventually *expect* duplicate draws and tickets that are not showing improvement.

In Table 3, we present our best achieved gaps and compare against the best gaps from existing manual results. We try ticket size $10, \ldots, 20$ (6 sizes), and we allocate 24 hours for each ticket size and for each n. LOTT. records the best result (i.e., for each n, we spent 24×6 hours to search for a good ticket). The hardware allocated to each job is 1 CPU (16 cores) from Intel Xeon Platinum 8360Y (for running MIPs) and 1 GPU core from Nvidia A100 (for network training).

ENSEMBLE is by combining the best two tickets each with 0.5 probability. The parenthesis next to the ENSEMBLE column shows the ticket sizes of the top two tickets (selected from the experiments in the previous paragraph). The idea of ENSEMBLE is that in our training, we intentionally aimed for tickets with different worst-case profiles. By creating ensemble, different tickets can "cover for each other" in terms of worst-case performance. Lastly, the ENSEMBLE is a free by-product of multiple draws. An ENSEMBLE is still a MLP, just with one more linear layer, so its worst-case analysis is conducted in the same way. For some n, evaluating the ensemble takes hours, but this evaluation only needs to be done once, not repeatedly as part of the training process. As expected, ensemble gave the best result.

In Table 4, we test our algorithm's scalability. We use a ticket size of 6. Anything larger than that often causes out-of-memory issue when running MIPs (even with 128 GB memory allocated) and is in general too slow. At this point, we think 20 agents is reaching the limit of our approach in terms of scalability – but this is already much larger in scale compared to existing works on neural network mechanism design. Even with tiny tickets with just 6 nodes, our LOTTERY mechanism produces better results than the best manual result. Also, as an ablation study, we compare LOTTERY against the best run from REINIT0, REINIT1 and REINIT2 (recorded in column REINIT). We allocate 24 hours for each algorithm. That is, REINIT in total spent 24×3 hours and performed worse than LOTTERY, which only spent 24 hours. The hardware specifications are the same as the before.

n	LOTT.	ENSEMBLE	MANUAL
6	0.1541	**0.1531** $(18 + 20)$	0.2847
7	0.0364	**0.0354** $(16 + 16)$	0.1766
8	0.0629	**0.0621** $(12 + 18)$	0.1925
9	0.0845	**0.0837** $(16 + 16)$	0.2164
10	0.1929	**0.1924** $(14 + 14)$	0.3320

Table 3: More Draws and Ensemble

n	LOTT.	REINIT	MANUAL
12	**0.1690**	0.1947	0.2537
14	**0.2767**	0.3408	0.3560
16	0.2662	**0.2613**	0.2889
18	**0.3610**	0.4013	0.3708
20	**0.2903**	0.3136	0.3124

Table 4: Scalability Test

Acknowledgments

I extend my sincere gratitude to Max Ward for directing my attention to the Lottery Ticket Hypothesis.

References

Cavallo, R. 2006. 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, 882–889. New York, NY, USA: ACM. ISBN 1-59593-303-4.

Clarke, E. H. 1971. Multipart pricing of public goods. *Public Choice*, 11: 17–33.

Conitzer, V.; and Sandholm, T. 2002. Complexity of Mechanism Design. In Darwiche, A.; and Friedman, N., eds., *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, 103–110. Morgan Kaufmann.

Curry, M.; Chiang, P.-y.; Goldstein, T.; and Dickerson, J. 2020. Certifying Strategyproof Auction Networks. In *Advances in Neural Information Processing Systems*, volume 33, 4987–4998. Curran Associates, Inc.

Curry, M. J.; Sandholm, T.; and Dickerson, J. P. 2022. Differentiable Economics for Randomized Affine Maximizer Auctions. *CoRR*, abs/2202.02872.

de Clippel, G.; Naroditskiy, V.; Polukarov, M.; Greenwald, A.; and Jennings, N. R. 2014. Destroy to Save. *Games and Economic Behavior*, 86: 392–404.

Dütting, P.; Feng, Z.; Narasimhan, H.; Parkes, D.; and Ravindranath, S. S. 2019. Optimal Auctions through Deep Learning. In *International Conference on Machine Learning*, 1706–1715. PMLR.

Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Golowich, N.; Narasimhan, H.; and Parkes, D. C. 2018. Deep Learning for Multi-Facility Location Mechanism Design. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 261–267. International Joint Conferences on Artificial Intelligence Organization.

Groves, T. 1973. Incentives in Teams. *Econometrica*, 41(4): 617–631.

Gu, S.; Zhang, Y.; Zhao, Y.; and Zhao, D. 2023. A Redistribution Framework for Diffusion Auctions.

Gujar, S.; and Narahari, Y. 2011. Redistribution Mechanisms for Assignment of Heterogeneous Objects. *Journal of Artificial Intelligence Research*, 41: 131–154.

Guo, M. 2011. 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*. AAAI Press.

Guo, M. 2012. Worst-Case Optimal Redistribution of VCG Payments in Heterogeneous-Item Auctions with Unit Demand. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, 745–752. IFAAMAS.

Guo, M. 2016. 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*, volume 9862 of *Lecture Notes in Computer Science*, 279–294. Springer.

Guo, M. 2019. An Asymptotically Optimal VCG Redistribution Mechanism for the Public Project Problem. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 315–321. ijcai.org.

Guo, M.; and Conitzer, V. 2009. Worst-Case Optimal Redistribution of VCG Payments in Multi-Unit Auctions. *Games and Economic Behavior*, 67(1): 69–98.

Guo, M.; and Shen, H. 2017. 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*, volume 10621 of *Lecture Notes in Computer Science*, 127–142. Springer.

Guo, M.; Wang, G.; Hata, H.; and Babar, M. A. 2021. Revenue maximizing markets for zero-day exploits. *Auton. Agents Multi Agent Syst.*, 35(2): 36.

Holmström, B. 1979. Groves' Scheme on Restricted Domains. *Econometrica: Journal of the Econometric Society*, 1137–1144.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257.

Manisha, P.; Jawahar, C. V.; and Gujar, S. 2018. Learning Optimal Redistribution Mechanisms through Neural Networks. In André, E.; Koenig, S.; Dastani, M.; and Sukthankar, G., eds., *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, 345–353. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Mas-Colell, A.; Whinston, M.; and Green, J. R. 1995. *Microeconomic Theory*. Oxford University Press.

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

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

Moulin, H. 2009. Almost Budget-Balanced VCG Mechanisms to Assign Multiple Objects. *JET*, 144(1): 96–119.

Naroditskiy, V.; Guo, M.; Dufton, L.; Polukarov, M.; and Jennings, N. R. 2012. Redistribution of VCG Payments in

Public Project Problems. In Goldberg, P. W., ed., *Internet and Network Economics - 8th International Workshop, WINE 2012, Liverpool, UK, December 10-12, 2012. Proceedings*, volume 7695 of *Lecture Notes in Computer Science*, 323–336. Springer.

Qin, T.; He, F.; Shi, D.; Huang, W.; and Tao, D. 2022. Benefits of Permutation-Equivariance in Auction Mechanisms. In *Advances in Neural Information Processing Systems*, volume 35, 18131–18142. Curran Associates, Inc.

Sill, J. 1998. Monotonic Networks. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, 661–667. Cambridge, MA, USA: MIT Press. ISBN 0-262-10076-2.

Tsuruta, S.; Oka, M.; Todo, T.; Kawasaki, Y.; Guo, M.; Sakurai, Y.; and Yokoo, M. 2014. Optimal False-Name-Proof Single-Item Redistribution Mechanisms. In Bazzan, A. L. C.; Huhns, M. N.; Lomuscio, A.; and Scerri, P., eds., *International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, 221–228. IFAAMAS/ACM.

Vickrey, W. 1961. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance*, 16: 8–37.

Wang, G.; Guo, R.; Sakurai, Y.; Babar, A.; and Guo, M. 2021. Mechanism Design for Public Projects via Neural Networks. In *20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021, Online)*.

Wang, G.; Zuo, W.; and Guo, M. 2021. 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), Melbourne, Australia, 2021*.