Autonomous Agents and Multi-Agent Systems (2024) 38:16
https://doi.org/10.1007/s10458-024-09647-8

# Mechanism design for public projects via three machine learning based approaches

Mingyu Guo[1] · Diksha Goel[2] · Guanhua Wang[3] · Runqi Guo[1] · Yuko Sakurai[4] · Muhammad Ali Babar[1]

Accepted: 25 March 2024

## Abstract

We study mechanism design for nonexcludable and excludable binary public project problems. Our aim is to maximize the expected number of consumers and the expected agents' welfare. We first show that for the nonexcludable public project model, there is no need for machine learning based mechanism design. We identify a sufficient condition on the prior distribution for the existing *conservative equal costs mechanism* to be the optimal strategy-proof and individually rational mechanism. For general distributions, we propose a dynamic program that solves for the optimal mechanism. For the excludable public project model, we identify a similar sufficient condition for the existing *serial cost sharing mechanism* to be optimal for 2 and 3 agents. We derive a numerical upper bound and use it to show that for several common distributions, the serial cost sharing mechanism is close to optimality. The serial cost sharing mechanism is not optimal in general. We propose three machine learning based approaches for designing better performing mechanisms. We focus on the family of *largest unanimous mechanisms*, which characterizes all strategy-proof and individually rational mechanisms for the excludable public project model. A largest unanimous mechanism describes an *iterative* mechanism, which is defined by an exponential number of mechanism parameters. Our first approach describes the largest unanimous mechanism family using a neural network and training is carried out by minimizing a cost function that combines the mechanism design objective and the constraint violation penalty. We interpret the largest unanimous mechanisms as price-oriented rationing-free (PORF) mechanisms, which enables us to move the mechanisms' iterative decision making off the neural network, to a separate simulation process, therefore avoiding the *vanishing gradient* problem. We also feed the prior distribution's *analytical form* into the cost function to achieve high-quality gradients for efficient training. Our second approach treats the mechanism design task as a *Markov Decision Process* with an exponential number of states. During the Markov decision process, the non-consumers are gradually removed from the system. We train multiple neural networks, each for a different number of remaining agents, to learn the optimal value function on the states. Training is carried out by supervised learning toward a set of manually prepared base cases and the Bellman equation. Our third approach is based on *reinforcement learning* for a *Partially Observable Markov Decision Process*. Each RL episode randomly draws a type profile, which is hidden from the RL agent (mechanism designer). The RL agent only observes

Extended author information available on the last page of the article

which cost share offers have been accepted under the largest unanimous mechanism under discussion. We use a continuous action space reinforcement learning approach to adjust the offer policy (i.e., adjust mechanism parameters). Lastly, our first two approaches use "supervision to manual mechanisms" as a systematic way for network initialization, which is potentially valuable for machine learning based mechanism design in general.

**Keywords** Public Project · Cost Sharing · Automated Mechanism Design · Mechanism Design via Neural Networks · Mechanism Design via Reinforcement Learning

## 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 [9, 11, 12, 19–21, 23, 32, 33]. 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. We study both the **nonexcludable** and the **excludable** versions of the *binary* 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.[1] 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).[2] The other objective is to maximize the agents' **expected 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.

Guo *et.al.* [12] studied an objective that is very similar to maximizing the expected number of consumers. The authors studied the problem of crowdfunding security information. There is a premium time period. If an agent pays more, then she receives the information earlier. If an agent pays less or does not pay, then she incurs a time penalty—she receives the information slightly delayed. The authors' objective is to minimize the expected delay. If every agent either receives the information at the very beginning of the premium period, or at the very end, then minimizing the expected delay is equivalent to maximizing the expected number of consumers. The public project is essentially the premium period. It should be noted that when crowdfunding security information, it is desirable to have more agents protected, whether their valuations are high or low. Hence, in this application domain, maximizing the number of consumers is more suitable than maximizing social welfare. However, since any delay that falls *strictly* inside the premium period is not valid

[1] The nonexcludable binary public project model is loosely related to binary social choice models, where the agents have the ability to pay to increase their influence on the final election outcome, such as quadratic voting [7, 16, 31]. The main difference is that for our model, all agents prefer the outcome "build". Additionally, there is a project cost that needs to be collectively covered by the agents.

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

for our *binary* public project model, the mechanisms proposed in [12] do not apply to our setting. For a survey on other game-theoretical models on cybersecurity, please refer to [15].

With slight technical adjustments, we adopt the existing characterization results from Ohseto [24] 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. Before summarizing our results, we introduce the following notation. We assume the agents' valuations are drawn independently and identically from a known distribution, with $f$ being the probability density function.

For the nonexcludable public project problem, we propose the following sufficient condition for the *conservative equal costs mechanism* [22][3] to be optimal. For maximizing the expected number of consumers, $f$ being *log-concave* ($\log f$ being concave) is a sufficient condition. For maximizing social welfare, besides log-concavity, we propose a new condition on $f$ called *welfare-concavity*. For distributions not satisfying the above conditions, we propose a dynamic program that solves for the optimal mechanism.

For the excludable public project problem, we also propose a sufficient condition for the *serial cost sharing mechanism* [22][4] to to be optimal. Our condition only applies to cases with 2 and 3 agents. For 2 agents, the condition is identical to the nonexcludable version. For 3 agents, we also need $f$ to be nonincreasing. For more agents, we propose a numerical technique for calculating the objective upper bound. For a few example log-concave distributions, including common distributions like *uniform* and *normal*, our experiments show that the serial cost sharing mechanism is close to optimality.

On the other hand, for general distributions (for example, without log-concavity), the serial cost sharing mechanism can be far away from optimality. We propose three machine learning based approaches for designing better performing mechanisms.

Mechanism design via machine learning/neural networks has been an emerging topic [6, 8, 10, 13, 18, 25, 27]. Duetting *et.al.* [6] 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 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. 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 [4]. Shen *et.al.* [27] 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). There are also studies focusing on mechanism design using reinforcement learning. Shen et al. [28] studied the mechanism design for dynamic pricing problems in sponsored search auctions, where the search engine may dynamically modify the reserve prices while interacting with

[3] The conservative equal costs mechanism is described in Example 1.

[4] The serial cost sharing mechanism is described in Example 2.

the bidders. The authors modelled the dynamic mechanism design problem as a Markov Decision Process and used reinforcement learning methods to solve the problem. Tang et al. [30] studied the design mechanism for industrial environments. The goal is to make the designer use the data generated during the process to improve future designs automatically. The author formulated the mechanism design task as a Markov Decision Process and applied the deep deterministic policy gradient algorithm, to handle the continuous action space. Hu et al. [14] designed an inference-aided reinforcement mechanism that uses reinforcement learning techniques in order to learn the policies to incentivize high-quality labels from crowdsourcing. Another line of research is on using neural network to solve for equilibria for a given game [3].

Our first approach models the largest unanimous mechanism family using a neural network and training is carried out by minimizing a cost function that combines the mechanism design objective and the constraint violation penalty. One complication we face is that the largest unanimous mechanisms involve iterative binary decisions. A common way to model binary decisions on neural networks is by using the *sigmoid* function. Unfortunately, stacking sigmoid functions to model iterative decision making leads to vanishing gradients and significant numerical errors. Instead, we rely on the price-oriented rationing-free (PORF) interpretation [37]: 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. We no longer need to construct complex network structures like the approach in [6] or resort to exponential-sized hard-coded buyer networks like the approach in [27]. After calculating $i$'s options, we link the options together using terms that contain network parameters, which enables backpropagation. Specifically, after calculating agent $i$'s options, we make use of the *analytical form* of $i$'s distribution (assumed to be differentiable) to figure out the probabilities of all the options, and then derive the expected objective value from $i$'s perspective. The training gradient comes from the probabilities, which are calculated based on neural network outputs.

Our second approach treats the mechanism design task as a *Markov Decision Process (MDP)* with an exponential number of states, where states are the accepted cost share vectors (by the agents) under the optimal largest unanimous mechanism. MDP typically can be solved using *Dynamic Program (DP)*; however, considering the exponential-sized state space, it is computationally infeasible to solve our DP exactly. Therefore, we use neural networks to approximate the DP solution [35], where we train the NN, which acts as a value function, to learn toward a set of manually prepared base cases and the recursive relationship between the states (i.e., Bellman equation). During the Markov Decision Process, the non-consumers are gradually removed from the system. We train multiple neural networks, each for a different number of remaining agents. That is, when there are $i$ agents left during the execution of the largest unanimous mechanism, we use a neural network with $i$ inputs as the value function, which determines the next round of offers. The combination of all these neural networks (i.e., a series of networks with 2 to $n$ inputs) define the overall mechanism.

Our third approach is based on *reinforcement learning*. We model the mechanism design problem as a *Partially Observable Markov Decision Process (POMDP)*. The state contains the agents' type profile (true valuations) and also the agents' current accepted cost share vector. The reinforcement learning agent (mechanism designer) can only observe the accepted cost share vector, but not the type profile. In each episode, the RL agent faces a newly generated type profile. The RL agent interacts with the environment repeatedly in order to learn how to optimally set the cost share vectors for the agents. We use the deep deterministic policy gradient algorithm [17] to train the RL agent.

Lastly, our first two approaches both use "supervision to manual mechanisms" as a systematic way for network initialization, which we believe is potentially valuable for machine learning based mechanism design in general. Manual mechanisms refer to manually designed existing mechanisms. In our first approach, we teach the network to mimic an existing manual mechanism, and then leave it to gradient descent. In our experiments, we considered different 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. In our second approach that approximates DP using NN, to speed up training, we first supervise the NN to learn toward the serial cost sharing mechanism. This gives us a "not far off" value function as the starting point (instead of starting from a random function). Our experiments also show that this additional supervision step leads to better mechanism performances. We believe supervision to manual mechanisms is often better than random initializations, because 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.

## 2 Model description

There are $n$ agents who 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. We use $v_i$ to represent agent $i$'s private valuation for the public project. We assume an agent's valuation for the public project is nonnegative, which is a standard assumption in public project mechanism design literature [9, 11, 12, 19–21, 23, 32, 33]. That is, $v_i \in [0, 1]$. 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.

- Agent $i$'s valuation is $v_i$ if she gets to consume the project, and 0 otherwise.
- 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, agent $i$'s valuation is $v_i$ if the project is built and $i$ is selected as a consumer, and 0 otherwise.
- For the excludable model, the outcome space is $\{0, 1\}^n$. We use $a_i$ to represent agent $i$'s allocation outcome. Under outcome $(a_1, a_2, \dots, a_n)$, 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.

Agent $i$'s payment $p_i$ is nonnegative. 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*. 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. Under a strategy-proof mechanism, it is a dominant strategy for an agent to truthfully report her valuation. Under an individually rational mechanism, an agent's utility is at least 0.

We study two objectives. One is to maximize the expected number of consumers. The other is to maximize the agents' welfare.

Let $v = (v_1, v_2, \dots, v_n)$ be a type profile. Let $M$ be a mechanism. We define the *benefit* function $B_M(v)$ as follows:

- If the objective is to maximize the expected number of consumers, then $B_M(v)$ denotes the number of consumers under $M$ for type profile $v$.
- If the objective is to maximize the social welfare, then $B_M(v)$ denotes the agents' total utility under $M$ for type profile $v$.

Our goal is to maximize the expectation $E_{v \sim f}(B_M(v))$. Here, $v \sim f$ denotes that when calculating the expectation of $B_M(v)$, we assume that the type profile $v$ is drawn based on the prior distribution $f$.

## 3 Characterizations and bounds

We adopt a list of existing characterization results from [24], 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 [24] were not proved for quasi-linear settings. However, we verify that the assumptions needed by the proofs remain valid for our model setting. One exception is that the characterizations in [24] 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.[5] We are also safe to drop the *citizen sovereign* assumption mentioned in Theorem 1,[6] but not the other two minor technical assumptions called *demand monotonicity* and *access independence*.

### 3.1 Nonexcludable public project: mechanism characterization

**Definition 1** (*Unanimous mechanism* [24]) There is a constant cost share vector $(c_1, c_2, \dots, c_n)$ 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.

**Theorem 1** (Nonexcludable mech. characterization [24]) *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.*

[5] 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 $\epsilon$ approaches 0. Unanimous/largest unanimous mechanisms are still strategy-proof and individually rational when $\epsilon$ is set to exactly 0.

[6] 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.

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

Below we present an example unanimous mechanism:

***Example 1*** (Conservative equal costs mechanism [22]) We build the project if and only if every agent agrees to pay $\frac{1}{n}$.

## 3.2 Excludable public project: mechanism characterization

**Definition 2** (*Largest unanimous mechanism* [24]) 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 non-empty 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 2** (Excludable mech. characterization [24]) *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 $(v_i', v_{-i})$.*

Below we present an example largest unanimous mechanism:

***Example 2*** (Serial cost sharing mechanism [22]) For every nonempty subset of agents $S$ with $|S| = k$, the cost share vector is $(\frac{1}{k}, \frac{1}{k}, \dots, \frac{1}{k})$. The mechanism picks the largest coalition where the agents are willing to pay equal shares.

Deb and Razzolini [5] 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. The experiments in this paper will show that, via machine learning approaches, we are able to derive mechanisms that outperform the serial cost sharing mechanism for a variety of distributions and for both objectives. That is, equal treatment of equals (or requiring anonymity) does hurt performances.

## 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 $(c_1, c_2, \dots, c_n)$, 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** *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 $C = (c_1, c_2, \dots, c_n)$ be the cost share vector. Maximizing the expected number of consumers is equivalent to maximizing the probability of $C$ getting unanimously accepted, which equals $\overline{F}(c_1)\overline{F}(c_2)\dots\overline{F}(c_n)$. 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 [1]. 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. □

We use the following example to better convey the idea behind Theorem 3:

**Table 1** Example log-concave distributions

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

***Example 3*** Let us consider 2 agents, whose prior distributions are from uniform $U(0, 1)$. That is, $f(x) = 1$, $F(x) = x$ and $\overline{F}(x) = 1 - x$.

Furthermore, $w(c) = \frac{\int_c^1 (x-c)f(x)dx}{\int_c^1 f(x)dx} = \frac{\int_c^1 (x-c)dx}{\int_c^1 dx} = \frac{\frac{1}{2}+\frac{c^2}{2}-c}{1-c} = \frac{1-c}{2}$.

For maximizing the expected number of consumers, we are essentially maximizing $\overline{F}(c_1)\overline{F}(c_2) = (1 - c_1)(1 - c_2)$. Since $c_1 + c_2 = 1$, the expression is maximized when $c_1 = c_2 = \frac{1}{2}$. That is, equal cost shares are optimal.

For maximizing the expected welfare, we are essentially maximizing $\overline{F}(c_1)\overline{F}(c_2)(w(c_1) + w(c_2)) = (1 - c_1)(1 - c_2)(\frac{1-c_1}{2} + \frac{1-c_2}{2}) = (1 - c_1)(1 - c_2)\frac{1}{2}$, which is the same as maximizing the number of consumers, ignoring the difference on the constant factor. Therefore, equal cost shares are also optimal for welfare maximization.

*f* being log-concave is also called the *decreasing reversed failure rate* condition [26]. Bagnoli and Bergstrom [1] proved log-concavity for many common distributions, including the distributions in Table 1 (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.

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 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.[7]

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:

[7] 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.

- 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 $B(n, 0, 1)$. Here, $k = n$, $u = 0$ and $m = 1$. When $k = n$, we have $n - k = 0$. When $m = 1$, we have $1 - m = 0$. For $B(n, 0, 1)$, 0 agents have already approved their cost shares, and the already approved cost share is 0 and the first 0 agents' total expected utility is 0. Therefore, $B(n, 0, 1)$ represents the original problem we aim to solve via DP. 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 $H^2n^2$.

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** (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 |

## 3.4 Excludable public project analysis

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 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 1.

***Proof*** We only present the proof for welfare maximization when $n = 3$.

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 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 $c_{\underline{1}23} + c_{1\underline{2}3} + c_{12\underline{3}} = 1$, we have that the above is maximized when all agents have equal shares. If the largest unanimous coalition 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 then $2\overline{F}(\frac{1}{2})^2 w(\frac{1}{2})(F(c_{\underline{1}23}) + F(c_{1\underline{2}3}) + F(c_{12\underline{3}}))$. 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. 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 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 formulated as a Markov Decision 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 $c_1, c_2, \dots, c_{t-k}$ to the first $t-k$ agents and they all accept, then we are in a state $\{(\underbrace{c_1, \dots, c_{t-k}}_{t-k}, \underbrace{l_{t-k+1}, \dots, l_t}_{k}), k\}$. The next offer is $c_{t-k+1}$. If the next agent accepts, then we move on to $\{(\underbrace{c_1, \dots, c_{t-k+1}}_{t-k+1}, \underbrace{l_{t-k+2}, \dots, l_t}_{k-1}), k-1\}$. If she disagrees (she is then the first agent to disagree), then we move on to a reduced-sized state $\{(\underbrace{c_1, \dots, c_{t-k}}_{t-k}, \underbrace{l_{t-k+2}, \dots, l_t}_{k-1}), t-1\}$. 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 $\{(c_1, \dots, c_t), 0\}$, 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 Decision Process. So for deriving performance upper bounds, it suffices to focus on this MDP.

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:

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

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,

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

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 |

In summary, for the nonexcludable public project model, if the distribution satisfies the condition in Theorem 3, then the conservative cost sharing mechanism is optimal. For other distributions, we proposed a dynamic programming technique, which produces the optimal mechanism. For the excludable public project model, if the distribution and the number of agents satisfy the condition in Theorem 5 or Theorem 6, then the serial cost sharing mechanism is optimal. For a few common distributions, we calculated the performance upper bound, and demonstrated that the serial cost sharing mechanism is close to optimality. However, there does exist distribution under which the serial cost sharing is far away from optimality.

***Example 4*** Here we provide an example to show that the serial cost sharing mechanism can be far away from optimality. Specifically, in the following constructed setting, the serial cost sharing mechanism's expected number of consumers is $\frac{n}{2}$, while a trivial mechanism can achieve $n-2$. That is, the optimality gap approaches infinity as $n$ increases. We pick a simple Bernoulli distribution, where an agent's valuation is 0 with 0.5 probability and 1 with 0.5 probability.[8] Under the serial cost sharing mechanism, when there are $n$ agents, only half of the agents are consumers (those who report 1s). 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 $n-2$.

For the rest of this paper, 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.

## 4 Machine learning based mechanism design approach 1: modelling iterative mechanisms as price-oriented rationing-free mechanisms

We start with an overview of automated mechanism design (AMD) via neural networks. Our approach 1 and previous papers on mechanism design via neural networks [6, 8, 18, 27] all fall into this general category.

- 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.

### 4.1 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

[8] Our paper 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.

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 $NN(\vec{b})$[9]:

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

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

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

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 $OUT(\vec{b}) \leq OUT(\vec{b}')$, which leads to this penalty:

$$\text{ReLU}(OUT(\vec{b}) - OUT(\vec{b}')) \tag{2}$$

During training, the above ReLU penalties easily reach near 0. In the context of our model, a mechanism that slightly violates the ReLU penalties can be easily fixed by slightly perturbing the offers to ensure that the offers are monotone (i.e., during the execution of the mechanism, we can simply refuse to ever lower the offers). Since we assume continuous prior distributions, slight adjustments on the offers do not significantly hurt the objectives, as the objectives are defined in terms of expectations.

Another way to enforce the last constraint is to use the *monotonic networks* structure [29]. This idea has been used in [8], 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. [38]

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

argues that having a *deep* model is important for performance in deep learning. 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.

### 4.2 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 [6, 8, 18, 27]. 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 $OUT((1, 1, \dots, 1))$. The remaining agents after the first round are then $S = \text{sigmoid}(v - OUT((1, 1, \dots, 1)))$. Here, $v$ is the type profile sample. 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. 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, 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.

One way to resolve this is to interpret the mechanisms as price-oriented rationing-free (PORF) mechanisms [37]. 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 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.

- 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:

$$(1 - F(OUT(\vec{O}_b)_i))O_s + F(OUT(\vec{O}_b)_i)O_f \tag{3}$$

$1 - F(OUT(\vec{O}_b)_i)$ 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.[10]

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 $O_f$:

$$\text{sigmoid}(v_i - OUT(\vec{O}_b)_i)O_s + \text{sigmoid}(OUT(\vec{O}_b)_i - v_i))O_f \tag{4}$$

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 Sect. 4.4, 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.

Below we summarise our approach 1:

Summary of Approach 1

**Neural network inputs:** binary vector of size $n$ that represents the coalition, where agent $i$ belongs to the coalition if and only if the $i$-th coordinate is 1.

**Neural network outputs:** non-negative real vector of size $n$ that represents the cost share vector, where the total cost share for agents in the coalition must sum up to 1 (achieved by Equation 1).

**Cost function:** Monotonicity violation penalty (Equation 2) minus the expected objective over the training batch (Equation 3).

### 4.3 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 4** (*One directional dynamic programming*) We make offers to the agents one by one. Every agent faces an offer based on how many agents are left, the objective value

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

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. At the end of the algorithm, if we collected 1, 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 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. This mechanism is not necessarily feasible, because the agents' offers are not necessarily nondecreasing when some agents are removed.

## 4.4 Experiments specific to approach 1

In this subsection, we present a few experimental results specific to our first machine learning approach. The experiments are conducted on a machine with Intel i5-8300H CPU. As mentioned earlier, we use a fully connected network with four layers (100 nodes each layer) to represent our mechanism. Random initializations are based on Xavier normal with bias 0.1.

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).

Figure 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 (4), singling out one agent based on (3), 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.

Figure 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 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 Fig. 2 (Bottom-Right).

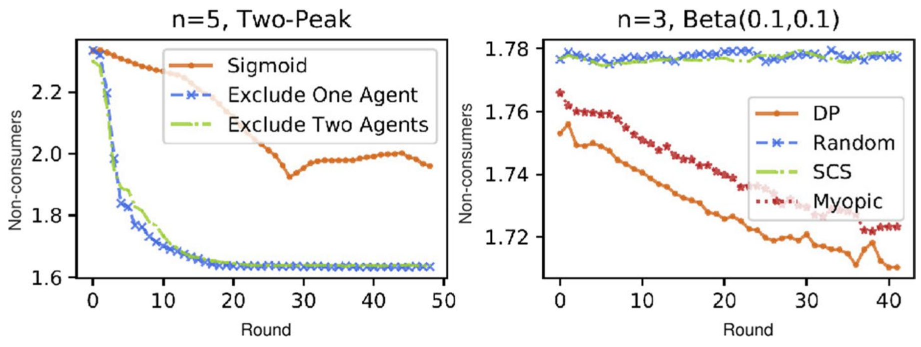

**Fig. 1** Effect of Distribution Info on Training

Figure 3 shows two experiments on maximizing expected agents' welfare (y-axis) under two-peak (0.2, 0.1, 0.6, 0.1, 0.5). 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.

## 5 Machine learning based mechanism design approach 2: neural network approximated dynamic program (NN_DP)

In this approach, we interpret the mechanism design task as a ***Markov Decision Process (MDP)***, which should not be confused with the MDP from Sect. 3.4, as the two MDP formulations are different. We aim to design a mechanism that maximizes the expected performances, i.e., the expected number of consumers and the agent's expected social welfare. We consider the agents' accepted cost share offer as a state that can be described as a vector of size $n$, where $n$ is the number of ***alive agents***.[11] The initial state vector (accepted cost share offer) can be represented as:

$$\underbrace{<0,0,0,0>}_{\text{Length of state vector} =} \quad \text{\# Alive agents} \tag{5}$$

Each coordinate in Eq. (5) represents each agent's accepted cost share offer. Initially, each coordinate has a value of 0, which indicates that no offer has been made to the agents yet.

**State transition process:** Given a state vector as shown in Eq. (5), we make cost share offer to one of the agents. On being offered, the agent either accepts or rejects the offer, and we reach a new state, as illustrated in Fig. 4. If the agent's cost share offer is less than her private valuation (drawn from the prior distribution), then the agent accepts the offer, otherwise rejects it. When an agent rejects the offer, the corresponding agent is removed from the state vector. We repeatedly make cost share offers to the agents until we reach the ***base states***. For our model, we have the following two base states:

[11] Alive agents are the agents who have approved their cost share offers.

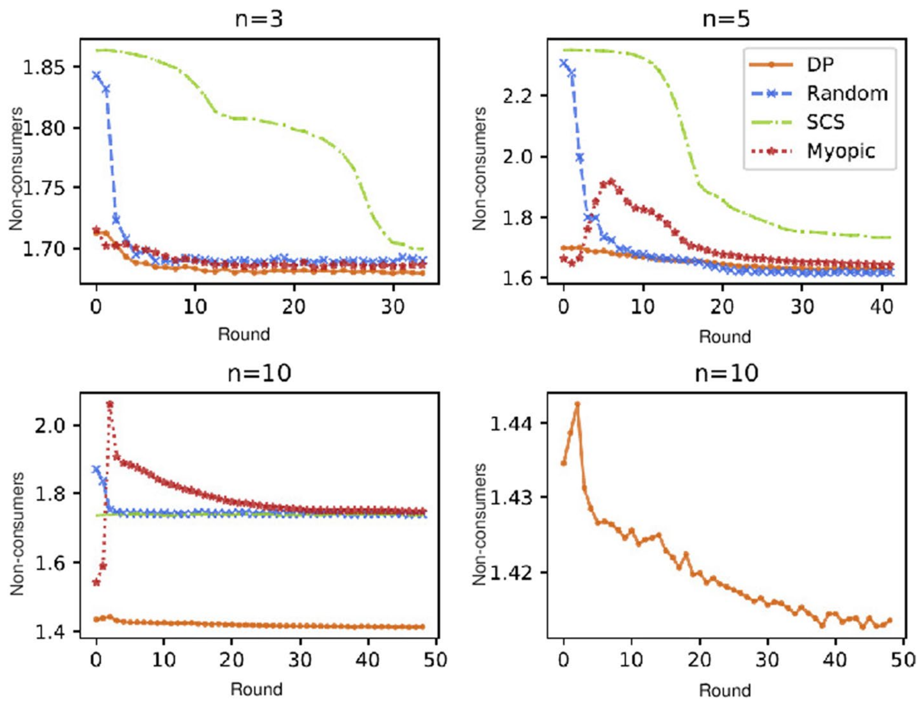

**Fig. 2** Supervision to different manual mechanisms

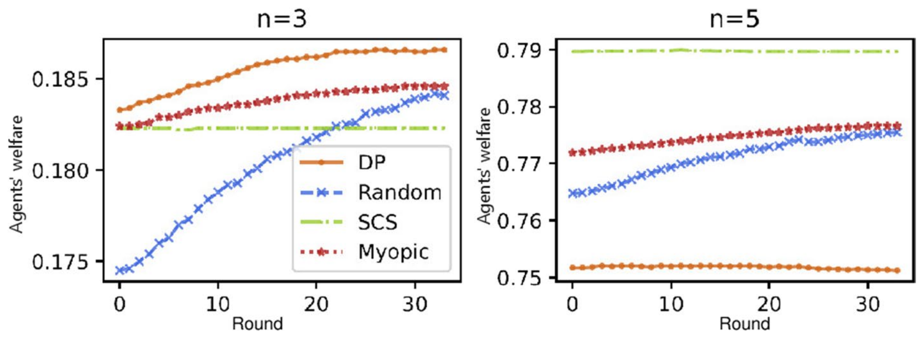

**Fig. 3** Maximizing agents' welfare

1. When all the agents disagree with their cost share offer, the resulting state is an empty state vector. In this case, the final decision is not to build the project.
2. When the sum of accepted cost share offers of all agents is equal to 1. In this case, the final decision is to build the project.

In this manner, we get a sequence of states that decides whether or not to build a public project.

$$\underbrace{<0,0,0,0>}_{\text{Current state vector}} \xrightarrow{\text{offer } c_1 \text{ to agent } 1} \underbrace{<c_1,0,0,0>}_{\text{Agree}} \text{ or } \underbrace{<0,0,0,0>}_{\text{Disagree}}$$

**Fig. 4** Illustration of the state transition process

**Converting the problem to Dynamic Program:** For a state $s$, ***admissible set of actions*** $A(s)$ are the agents who approve their current cost share offer (alive agents), and we can offer increased cost share to them. Notably, we do not consider the agents who have not approved their cost share offer and have been removed from the state vector.

Given a state $s$ and an admissible set of actions $A(s)$, we compute the future states (cost share offer) the agents may face while deciding whether or not to build a public project. The maximum size of the state space for this MDP can be $(\frac{1}{I})^n$, where $I$ represents a small constant offer increment (0.02 for our model). Notably, the size of this state space is very large. Besides, we do not consider all the possible states in the state space; instead, we only consider the states (cost share offers) that the agents may face while trying to reach a decision.

We can solve the above-described MDP for the public project problem using Dynamic Program. The principle of optimality [2] states that we can get the optimal overall solution to the problem by making optimal choices at each decision step. In addition, it provides us with a top-down instance to design a recursive relation that connects the solution of a problem and its subproblems. For a given state $s$ and action $a \in A(s)$, the DP function for maximizing the expected performance can be written as:

$$f(s) = \max_{a \in A(s)} \left( \sum_{s' \in \delta(s,a)} Pr(s' \mid s,a) f(s') \right) \tag{6}$$

Here, $f(s)$ denotes the design objective we are considering at state $s$, i.e., either the number of consumers or social welfare, $\delta(s,a)$ represents the set of future states that the agent encounter on performing action $a$ on state $s$, $Pr(s' \mid s,a)$ represents the transition probability, i.e., action $a$ on state $s$ will result in state $s'$.

We can solve Eq. (6) using backward induction; however, it is computationally infeasible considering the exponential-sized state space. Therefore, we propose to use Neural Network to solve the DP function.

**Neural Network:** The classical NN training is based on supervised learning, which requires a lot of training ($s$, $f(s)$) pair samples. Nevertheless, the supervised learning method is not practical for our problem, as estimating even one exact $f(s)$ may take exponential time. Besides, it is computationally intractable to solve the DP considering the large state space. Therefore, we adopt an alternative learning approach and train the NN to learn the DP function [36], which involves learning; (1) base cases, (2) recursive relationship between states. Notably, during the initial training phases, the NN is inaccurate. Therefore, we supervise the NN to learn toward the serial cost sharing mechanism as the initialization step. Afterwards, we train the NN to learn the recursive relationship. The trained NN approximately solves the DP equation, and therefore, we call this approach as ***Neural Network Approximated Dynamic Program (NN_DP)***. We then follow the policy (action suggested by the trained NN) of the trained NN when implementing the mechanism. However, since it is an approximated solution, the result will not be optimal. The proposed NN_DP is a heuristic approach that we expect to work well.

Given a state vector (cost share offers) as input, the NN outputs agents' social welfare (or the number of consumers). We start with an initial state vector as shown in Eq. (5) and generate learning samples using the state transition process. *Learning samples* are the sequence of states (cost share offers) obtained using the state transition process where we start from the initial state vector and keep transiting to a new state until we reach a base state, i.e., the final decision. We consider the collected states as the cost share offers that the agents may face while trying to reach a decision, and therefore, we use these states as learning samples in order to train the NN. To avoid the NN falling into local optima, especially in the initial training phases when our NN is highly inaccurate, we propose using the *exploration technique* for generating the learning samples. We set the value of exploration parameter $\epsilon$ to 0.5. For every state, we either take an optimal action (the action suggested by the trained NN) with a probability of $\epsilon$ or take an arbitrary action with a probability of $1 - \epsilon$. The action suggested by the NN may not always be optimal; therefore, we consider taking arbitrary action to explore other actions. We keep generating learning samples till we reach a decision (base state). We train the NN on the generated learning samples. We first supervise the NN to learn the DP base states and their corresponding values as follow:

- When all the agents disagree with their cost share offers and no agent is left willing to pay. In this case, the expected social welfare (or expected number of consumers) is always 0.
- When the sum of the agents' accepted cost share offer is equal to 1, then the expected welfare is computed as the sum of the differences between the agent's type (private valuation) and accepted cost share. In addition, the expected number of consumers is the number of agents who accepted their cost share offers.

For the rest of the states, the NN is trained to learn the recursive relation of the DP function (Eq. 6). Let $f(s;\theta)$ represent the value estimated by the NN, where $s$ is the input state, and $\theta$ are model's parameters. In the ideal situation, $f(s;\theta)$ should strictly be a DP function; however, there is a gap between the right and left term in Eq. (7), and our goal is to minimize the gap. We train the NN using the mean squared error loss function, and the loss $\mathcal{L}(\theta)$ is computed as follows:

$$\mathcal{L}(\theta) = \sum_{s \in S} \left( f(s;\theta) - \max_{a \in A(s)} \left( \sum_{s' \in \delta(s,a)} Pr(s' \mid s, a) f(s') \right) \right)^2 \tag{7}$$

where $S$ is a set of all states.

We generate learning samples assuming the current $f$ is the accurate DP function and train the NN on the generated learning samples. The advantage of this approach is that as the DP function $f$ improves with training, it is highly possible that these generated learning samples (states) are more likely to occur, i.e., the agents may face these offers while trying to reach a decision. Besides, instead of training only one NN for computing the expected performances, we train a series of NNs, one for each problem size, in order to enhance the approximation accuracy of NNs. For instance, for a problem with $n$ agents, we train $n$ neural networks.

Below we summarise our approach 2:

Summary of Approach 2

**Neural network inputs:** There are multiple neural networks with different input dimensions. When there are $k$ agents in the coalition, we use a non-negative real vector of size $k$ as the input vector. The $i$-th coordinate of the input vector specifies the cost share offer agent $i$ has already agreed to.

**Neural network outputs:** A single real value that represents the optimal state value. The optimal state value is the expected mechanism design objective under the optimal policy at the current state.

**Cost function:** Deviation from the Bellman equation (Eq. 7).

## 6 Machine learning based mechanism design approach 3: reinforcement learning

In this approach, we treat the mechanism design task as a ***Partially Observable Markov Decision Process (POMDP)***. We propose a Reinforcement Learning based framework to learn a policy (for giving cost share offers) to maximize the expected performance for the public project problem. The overall idea of the proposed framework is to make the RL agent learn the policy by interacting with the designed environment. The designed reward function directs the RL agent to maximize the expected performance.

Our designed environment works as follows: In each *episode*, we randomly draw the agent's type profile (private valuation) from the prior distribution. The initial *observation* considers all the agent's initial accepted cost share offer to be 0 (i.e., all agents agreed to pay 0), and all the agents are alive. In each *step*, we increase one agent's offer; the resulting observation is whether or not this agent has agreed to this offer and her newly accepted cost share offer. We aim to maximize the reward received during an episode, which equals to the mechanism design objective for the given type profile under the current RL policy. This section first describes the environment design and then the training algorithm.

**Environment design:** We describe the mechanism design task as a POMDP $(\Omega, S, A, Tr, R)$, where $\Omega$ is the observation space, $S$ denotes the state space, $A$ denotes the action space, $Tr$ is the state transition function and $R$ denotes the reward. In POMDP, the RL agent can not observe the states directly; instead, the agent only receives the observations and tries to learn the best decision based on the partial information (observations) received from the environment. The details of the environment are discussed below:

- **Observation:** Observation $o$ is a vector of size $2n$, where $n$ is the total number of agents. For each agent, we keep track of two things. First is the agent's already accepted cost share offer. The coordinate in the observation vector representing the agent's accepted cost share offer ranges from 0 to 1. The second is whether or not the agent has approved her current cost share offer. If the agent has approved her current cost share offer, then the corresponding coordinate in the observation vector is 1, and we call such agents as ***alive agents***. On the other hand, if the agent rejects the cost share offer, then the value of the corresponding coordinate is 0. We can represent the observation vector $o$ as:

$$< \overbrace{\underbrace{0,0}_{\text{Agent 1}} \;, \;\; . \;\; . \;\; . \;\; . \;\; , \underbrace{0,0}_{\text{Agent n}}}^{\text{length of observation vector = 2}} > \times \text{ number of agents} \tag{8}$$

- **State:** State $s$ is a vector of size $3n$, consisting of observation and the agents' private valuation. The state is not directly accessible to the RL agent; the agent only gets access to the observation, whereas the private valuation part of the state is hidden. Besides, we have following two base states: (1) when all the agents do not approve their cost share offers; hence, the final decision is not to build the project, (2) when the sum of accepted cost share offers of all the agents is equal to 1; hence, the decision is to build the project. We can represent the state vector $s$ as:

$$< \overbrace{\underbrace{\text{Observation}}_{2n} \underbrace{\text{Private valuation}}_{n}}^{\text{length of state vector = 3}} > \times \text{ number of agents} \tag{9}$$

- **Action:** An action $a$ is a value from 0 to 1 and denotes the offer increment for the agent. For instance, an action $a$ indicates increasing the corresponding agent's offer by $a$.
- **Transition function:** For a given agent $i$ (agent to make offer to) and the offer increment $a$ (action), the new cost share offer $c$ for the agent $i$ is computed as below:

$$new\ c[i] = old\ c[i] + \min\left(a,\ 1 - \sum_{j \in \text{Alive agents}} c[j]\right) \tag{10}$$

  An agent $i$ accepts the new cost share offer only when the new offer is less than the agent's private valuation. If the agent accepts the offer, then the agent's accepted cost share offer in the resulting observation is set to the new offer. On the other hand, if the agent does not accept the new offer, then both the agent's accepted cost share offer and alive status are set to 0 in the resulting observation. In this way, we get a new observation. We keep transiting to a new observation until all the agents disagree or the sum of accepted cost share offer by alive agents is 1.
- **Reward:** We define the reward as below:

  - ***For maximizing the number of consumers:*** We define the reward $r(s, a)$ at state $s$ on selecting action $a$ as *the number of alive agents*, given the sum of cost share offers accepted by the agents is 1. Otherwise, the reward is 0.
  - ***For maximizing the social welfare:*** We define the reward $r(s, a)$ at state $s$ on selecting action $a$ as the *sum of the difference between the agent's private valuation and cost share offer*, given that the sum of cost share offers accepted by the agents is 1. Otherwise, the reward is 0.

**Training algorithm:** The RL agent does not possess any prior expert knowledge about the environment in which it operates. For each episode, we draw agents' private valuations independently and identically from a known distribution. Initial observation considers all the agents' accepted cost share offer to 0 and agents' alive status to 1. Besides, we always make cost share offer to the agents sequentially, starting from the first agent to the last, in a circular manner and skip those agents whose alive status is 0. It should be noted that this offering order is without loss of generality. Whenever any agent disagrees with their cost share offer, both her accepted cost share offer and alive status are set to 0, and the RL agent starts again by offering to the first alive agent. Notably, the agent can not directly access the state but only the partial information from the state, i.e., observation.

At each timestamp $t$, the RL agent receives state $s_t$; however, the RL agent can only access observation $o_t$ from state $s_t$ and then selects an action $a_t$ and obtains a reward $r_{t+1}$ according to the designed reward function, and move to the state $s_{t+1}$ according to the transition function. The process repeats till the base state is reached, i.e., all the agents disagree, or the sum of all accepted cost share offers by the agents is 1. This way, we have a sequence of states, observations, actions and rewards that ends at the base state. The RL agent aims to learn a policy that maximizes the total reward received during an episode. We use *Deep Deterministic Policy Gradient* (DDPG) reinforcement learning algorithm, which is based on the actor-critic concept. The critic is used to learn the action value function, whereas the actor learns the policy. In this way, the critic provides the policy gradient to the actor.

Below we summarise our approach 3:

Summary of Approach 3 Reinforcement Learning

**Neural network inputs (observation):** A non-negative real vector of size $2n$. We use two coordinates for each agent. For each agent, we keep track of whether she belongs to the coalition (binary) and the cost share offer she already agreed to.

**Neural network outputs (action):** In every step, the environment focuses on one agent (round-robin style). The action is a non-negative real value, which represents how much we should increase the focused agent's cost share offer (Equation 10).

**Summary of environment:** In every episode, we randomly draw a type profile, which is hidden from the reinforcement learning agent. Episode ends when all agents in the coalition unanimously agree with their cost share offers or when no agents are left in the coalition. The reinforcement learning agent only collects the reward at the end of the episode, which is simply the mechanism design objective.

## 7 Experiments comparing different approaches

In this section, we discuss the performance of the proposed approaches by performing exhaustive experiments. We performed the experiments on a high-performance cluster server with Intel Gold 6148/6248 CPUs. We implemented all the code in PyTorch.

The training parameters for approach 1 has been described in Sect. 4.4. Below we present the parameters for the other two approaches.

***Training parameters for neural network approximated dynamic program approach*** We use a simple fully connected NN, and a ReLU activation function follows each layer. The last layer of NN is followed by a sigmoid function, which maps the NN's output to a value from 0 to 1; the value indicates the expected performance. We fix the number of layers to 4 and the size of each layer to 64. We use mean square error to calculate the loss. Parameters are trained using Adam optimizer with a learning rate of 0.001 and weight decay of $5e-4$. We train the model for 200 rounds. In each round, we collect 256 states and then train the model in a batch of 16 states (therefore, we have 16 batches, each with 16 states and overall 256 states in each round). We train a series of NNs, one for each problem size. We set the offer increment to a small constant of 0.02. We perform experiments 5 times with a seed from 0 to 4; the average across all seed is the final expected performance.

It is impractical to use DP to evaluate the exact number of consumers or social welfare corresponding to the accepted cost share offers of the agents due to the large state space.

Besides, the trained NN might not output the accurate expected performance for a given accepted cost share offer (state vector). Consequently, we simulate the expected performance (the number of consumers and social welfare) using the Monte Carlo simulations. We simulate the trained NN on the cost share offer (initially, all accepted cost share offers are 0) using Monte Carlo simulations over 20,000 runs to determine the number of consumers and agents' social welfare.

***Training parameters for reinforcement learning approach*** We implement our RL environment in the OpenAI Gym toolkit and utilize Tianshou [34] framework to train the RL model. We use a simple multi-layer perceptron NN and set the size of the hidden layer to 128. We train the model using the deep deterministic policy gradient algorithm, Adam optimizer with an actor learning rate of 0.0001, critic learning rate of 0.001 and batch size of 128 states. The model is trained for a maximum of 1000 epochs with 20,000 steps per epoch. The reward discount factor is set to 0.99, reward normalization to false, the target weight to 0.005 and the exploration noise to 0.1. The experience buffer can store 20,000 samples.

To determine the performance of the trained policy in the designed environment, we collect states, observations, actions and rewards from the environment using the learnt policy. We simulate the environment for 1000 episodes, and the reward at the end of the episode is the required expected performance (the number of consumers or the social welfare).

We use DP, Random, SCS, Myopic to denote the performances of approach 1 when we start from these initializations (i.e., in SCS, we first use supervision to train the network to learn toward the serial cost sharing mechanism (SCS), and then perform gradient descent according to approach 1. NN_DP and RL refer to our approach 2 and 3, respectively. We use UB to denote the objective upper bound, calculated using the DP approach described in Sect. 3. Also, as comparison, we use SCS_Ori to denote the performance of the original serial cost sharing mechanism, which perform significantly worse than the best machine learning based mechanism.

**Results** Table 2 presents the comparison of proposed approaches for *maximizing the number of consumers*. We performed experiments for 3, 5, and 10 agents and have considered the two-peak distribution (0.15, 0.1, 0.85, 0.1, 0.5). Results from the table show that for $n = 3$, the reinforcement learning approach RL performs the best with 1.58 number of consumers. Supervision to DP performs better than the other approaches by achieving 1.33 number of consumers but is still far away from the results of the RL approach. Besides, the RL approach outperforms all other approaches for $n = 5$ and 10. For $n = 5$, the RL approach results in 3.7 number of consumers, whereas the second best performing approach is random initializations. Similarly, for $n = 10$, RL performs the best with 8.88 number of consumers. Our results demonstrate that the RL approach consistently achieves the best performance for maximizing the number of consumers. Approach 1 consistently achieves the second best performance and NN_DP performs poorly.

Table 3 presents the comparison of the proposed approaches for *maximizing the social welfare* under two-peak distribution (0.2, 0.1, 0.6, 0.1, 0.5). For this objective, we experimented with 3 and 5 number of agents. Our results demonstrate that for $n = 3$, our neural network approximated dynamic program technique NN_DP outperforms the RL and other techniques by a significant number. NN_DP results in 0.2821 welfare, whereas the second best performing approach is RL which results in 0.2589 welfare. These two techniques perform considerably better than the other approaches for $n = 3$. For $n = 5$, the RL-based approach performs significantly better than the rest of the approaches and results in social welfare of 0.9396, which is much higher than the second best performing approach NN_DP. Notably, for maximizing social welfare, NN_DP outperforms the RL approach for

**Table 2** Comparison of proposed approaches for maximizing the number of consumers

| Approach ↓ | #Agents → | | |
|---|---|---|---|
| | *n = 3* | *n = 5* | *n = 10* |
| DP | <u>1.33</u> | 3.37 | <u>8.60</u> |
| Random | 1.32 | <u>3.39</u> | 8.26 |
| SCS | 1.30 | 3.32 | 8.26 |
| Myopic | <u>1.33</u> | 3.36 | 8.25 |
| NN_DP | 1.04 | 1.38 | 4.31 |
| RL | **1.58** | **3.70** | **8.88** |
| UB | 1.63 | 3.92 | 9.15 |
| SCS_Ori | 0.52 | 1.40 | 4.24 |

Bold results indicate the best results and the second best results are underlined

**Table 3** Comparison of proposed approaches for maximizing the social welfare

| Approach ↓ | #Agents → | |
|---|---|---|
| | *n = 3* | *n = 5* |
| DP | 0.1870 | 0.7517 |
| Random | 0.1835 | 0.7648 |
| SCS | 0.1820 | 0.7897 |
| Myopic | 0.1840 | 0.7719 |
| NN_DP | **0.2711** | <u>0.8307</u> |
| RL | <u>0.2589</u> | **0.9396** |
| UB | 0.2744 | 1.1112 |
| SCS_Ori | 0.1982 | 0.8205 |

Bold results indicate the best results and the second best results are underlined

3 agents. The reason for the high performance of NN_DP is that given unlimited computation power, NN_DP will provide us with optimal solutions. In addition, for maximizing social welfare, the reward is continuous rather than discrete; therefore, it is comparatively easy to learn the expected performance for this case. Another factor can be the small size of problem (only 3 agents), due to which NN_DP is able to achieve the near-optimal solution.

Based on the results presented above, RL emerges as the most effective approach for maximizing the expected number of consumers. Approach 1 also demonstrates commendable performance, positioning it as a viable alternative to RL. In contrast, the NN_DP approach underperforms, rendering it less suitable for maximizing the expected number of consumers. When it comes to maximizing the expected welfare, both NN_DP and RL exhibit comparable performances. There is no distinct advantage of one approach over the other for the objective of maximizing the expected welfare."

## 8 Conclusion

In this paper, we studied optimal-in-expectation mechanism design for nonexcludable and excludable binary public project problems. Under various conditions, we showed that existing mechanisms or mechanisms derived via classic mechanism design approaches are optimal or near optimal. For excludable public project problem, and especially for non-log-concave prior distributions, we proposed three different machine learning approaches for deriving better performing mechanisms. Our techniques range from unsupervised learning, supervised learning, to reinforcement learning. Some of our training techniques have the potential to be applied to other machine learning based mechanism design models, such as modelling iterative mechanisms as PORF to avoid the vanishing gradient issue, directly feeding the analytical form of the prior distribution into the cost function to achieve higher-quality gradient, and finally using supervision to manual mechanisms as a systematic way of initialization.

**Funding** Open Access funding enabled and organized by CAUL and its Member Institutions.

## References

1. Bagnoli, M., & Bergstrom, T. (2005). Log-concave probability and its applications. *Economic Theory, 26*(2), 445–469. https://doi.org/10.1007/s00199-004-0514-4
2. Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society, 60*(6), 503–515.
3. Bichler, M., Fichtl, M., Heidekrüger, S., Kohring, N., & Sutterer, P. (2021). Learning equilibria in symmetric auction games using artificial neural networks. *Nature Machine Intelligence, 3*, 1–9.
4. Conitzer, V., & Sandholm, T. (2002). Complexity of mechanism design. In Darwiche A, 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, Morgan Kaufmann, (pp. 103–110)
5. Deb, R., & Razzolini, L. (1999). Voluntary cost sharing for an excludable public project. *Mathematical Social Sciences, 37*(2), 123–138.
6. Dütting, P., Feng, Z., Narasimhan, H., Parkes, D., & Ravindranath, S.S. (2019). Optimal auctions through deep learning. In International conference on machine learning, PMLR, (pp. 1706–1715)
7. Goeree, J. K., & Zhang, J. (2017). One man, one bid. *Games and Economic Behavior, 101*, 151–171. special Issue in Honor of John O. Ledyard.
8. Golowich, N., Narasimhan, H., & 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, International joint conferences on artificial intelligence organization, (pp. 261–267)
9. Guo, M. (2021). An asymptotically optimal VCG redistribution mechanism for the public project problem. *Auton Agents Multi Agent Syst, 35*(2), 40.
10. Guo, M. (2024). Worst-case vcg redistribution mechanism design based on the lottery ticket hypothesis. In The 38th AAAI conference on artificial intelligence (AAAI), Vancouver, Canada, arXiv:2305.11011
11. Guo, M., Naroditskiy, V., Conitzer, V., Greenwald, A., & Jennings, N.R. (2011). 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, Springer, Lecture Notes in Computer Science, (vol 7090, pp. 158–169)
12. Guo, M., Yang, Y., & Babar, M.A. (2018). Cost sharing security information with minimal release delay. In PRIMA 2018: Principles and practice of multi-agent systems - 21st international conference, Tokyo, Japan, October 29 - November 2, 2018, Proceedings, Springer, Lecture Notes in Computer Science, (vol. 11224, pp. 177–193)
13. Guo, M., Wang, G., Hata, H., & Babar, M. A. (2021). Revenue maximizing markets for zero-day exploits. *Auton Agents Multi Agent Syst, 35*(2), 36.
14. Hu, Z., Liang, Y., Zhang, J., Li, Z., & Liu, Y. (2018). Inference aided reinforcement learning for incentive mechanism design in crowdsourcing. Advances in Neural Information Processing Systems *31*
15. Iqbal, A., Gunn, L. J., Guo, M., Babar, M. A., & Abbott, D. (2019). Game theoretical modelling of network/cybersecurity. *IEEE Access, 7*, 154167–154179.
16. Lalley, S.P., & Weyl, E.G. (2019). Nash equilbria for quadratic voting. arXiv:1409.0264
17. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. In International conference on learning representations (ICLR)
18. Manisha, P., Jawahar, C.V., & Gujar, S. (2018). Learning optimal redistribution mechanisms through neural networks. In E. André, S. Koenig, M. Dastani, G. Sukthankar (Eds.) Proceedings of the 17th international conference on autonomous agents and multiagent systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018, International foundation for autonomous agents and multiagent systems Richland, SC, USA / ACM, (pp. 345–353)
19. Mas-Colell, A., Whinston, M., & Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press.
20. Moore, J. (2006). *General Equilibrium and Welfare Economics: An Introduction*. Berlin: Springer.
21. Moulin, H. (1988). *Axioms of Cooperative Decision Making*. Cambridge: Cambridge University Press.
22. Moulin, H. (1994). Serial cost-sharing of excludable public goods. *The Review of Economic Studies, 61*(2), 305–325.
23. Naroditskiy, V., Guo, M., Dufton, L., Polukarov, M., & Jennings, N.R. (2012). 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, Springer, Lecture Notes in Computer Science, (vol. 7695, pp. 323–336)
24. Ohseto, S. (2000). Characterizations of strategy-proof mechanisms for excludable versus nonexcludable public projects. *Games and Economic Behavior, 32*(1), 51–66.
25. Sakurai, Y., Oyama, S., Guo, M., & Yokoo, M. (2019). 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, Springer, Lecture Notes in Computer Science, (vol 11873, pp. 594–601)
26. Shao, R., & Zhou, L. (2016). Optimal allocation of an indivisible good. *Games and Economic Behavior, 100*, 95–112.
27. Shen, W., Tang, P., & Zuo, S. (2019). Automated mechanism design via neural networks. In Proceedings of the 18th international conference on autonomous agents and multiagent systems, international foundation for autonomous agents and multiagent systems, Richland, SC, AAMAS '19, (pp. 215–223)
28. Shen, W., Peng, B., Liu, H., Zhang, M., Qian, R., Hong, Y., Guo, Z., Ding, Z., Lu, P., & Tang, P. (2020). Reinforcement mechanism design: With applications to dynamic pricing in sponsored search auctions. *Proceedings of the AAAI Conference on Artificial Intelligence, 34*, 2236–2243.
29. Sill, J. (1998). Monotonic networks. In Proceedings of the 1997 conference on advances in neural information processing systems 10, MIT Press, Cambridge, MA, USA, NIPS '97, (pp. 661–667)
30. Tang, P. (2017). Reinforcement mechanism design. In IJCAI, (pp. 5146–5150)
31. Vragov, R., & Smith, V. (2023). A method for identifying parameterizations of the compensation election and quadratic voting that admit pure-strategy equilibria. *Mathematical Social Sciences, 122*, 7–16.
32. Wang, G., & Guo, M. (2021). Public project with minimum expected release delay. In PRICAI 2021: Trends in artificial intelligence - 18th pacific rim international conference on artificial intelligence, PRICAI 2021, Hanoi, Vietnam, November 8-12, 2021, Proceedings, Part I, Springer, Lecture Notes in Computer Science, (vol 13031, pp. 101–112)
33. Wang, G., Zuo, W., & 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
34. Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, H., & Zhu, J. (2021). Tianshou: A highly modularized deep reinforcement learning library. arXiv preprint arXiv:2107.14171

35. Xu, S., Panwar, S. S., Kodialam, M., & Lakshman, T. (2020). Deep neural network approximated dynamic programming for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence, 34*, 1684–1691.
36. Yang, F., Jin, T., Liu, T.Y., Sun, X., & Zhang, J. (2018). Boosting dynamic programming with neural networks for solving np-hard problems. In: Asian conference on machine learning, PMLR, (pp. 726–739)
37. Yokoo, M. (2003). 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, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'03, (pp. 733–739)
38. Zhou, Z.H., & Feng, J. (2017). Deep forest: Towards an alternative to deep neural networks. In Proceedings of the 26th international joint conference on artificial intelligence, AAAI Press, IJCAI'17, (pp. 3553–3559)

## Authors and Affiliations

**Mingyu Guo[1] · Diksha Goel[2] · Guanhua Wang[3] · Runqi Guo[1] · Yuko Sakurai[4] · Muhammad Ali Babar[1]**

✉ Mingyu Guo
mingyu.guo@adelaide.edu.au

Diksha Goel
diksha.goel@data61.csiro.au

Guanhua Wang
chris.wang@vu.edu.au

Runqi Guo
sharkrg.joker@gmail.com

Yuko Sakurai
yukoskr@gmail.com

Muhammad Ali Babar
ali.babar@adelaide.edu.au

1 University of Adelaide, Adelaide, Australia

2 CSIRO, Melbourne, Australia

3 Victoria University, Melbourne, Australia

4 Nagoya Institute of Technology, Nagoya, Japan