Strategyproof Reinforcement Learning for Online Resource Allocation

Sebastian Stein*, Mateusz Ochal◇, Ioana-Adriana Moisoiu*
Enrico Gerding*, Raghu Ganti•, Ting He°, Tom La Porta°
* University of Southampton, UK. {S.Stein,iam1g16,E.Gerding}@soton.ac.uk
◇ University of Edinburgh, UK. M.Ochal@sms.ed.ac.uk
• IBM T.J. Watson Research Center, USA. rganti@us.ibm.com
° Penn State University, USA. {tzh58,tfl12}@psu.edu

ABSTRACT

We consider an online resource allocation problem where tasks with specific values, sizes and resource requirements arrive dynamically over time, and have to be either serviced or rejected immediately. Reinforcement learning is a promising approach for this, but existing work on reinforcement learning has neglected that task owners may misreport their task requirements or values strategically when this is to their benefit. To address this, we apply mechanism design and propose a novel mechanism based on reinforcement learning that aims to maximise social welfare, is strategyproof and individually rational (i.e., truthful reporting and participation are incentivised). In experiments, we show that our algorithm achieves results that are typically within 90% of the optimal social welfare, while outperforming approaches that use fixed pricing (by up to 86% in specific cases).

KEYWORDS

Mechanism Design; Reinforcement Learning; Resource Allocation

ACM Reference Format:
Sebastian Stein, Mateusz Ochal, Ioana-Adriana Moisoiu, Enrico Gerding, Raghu Ganti, Ting He, and Tom La Porta. 2020. Strategyproof Reinforcement Learning for Online Resource Allocation. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

We consider settings where a mechanism needs to make dynamic (online) decisions about whether to allocate scarce resources to a stream of incoming tasks that have varying resource requirements and valuations, and that need to be serviced immediately. A prominent example of such a setting is the allocation of processing resources to real-time computational tasks in edge clouds [20]. Here, users (i.e., task owners) wish to offload tasks from mobile devices to nearby edge resources, in order to save limited battery power or to run more complex applications than they could run locally [18]. These tasks are often time-sensitive, and so we assume that tasks need to be either serviced or rejected immediately. This is common in domains such as virtual/augmented reality [13], autonomous vehicles [14] or real-time analysis of data from Internet-of-Things devices [1], including for voice assistants or live video surveillance. Our overall aim here is to maximise the social welfare, i.e., the sum of valuations of all tasks that are completed within a specific time horizon, e.g., over a day or a week.

There are several key challenges that need to be addressed in these settings. First, tasks arrive dynamically and the system does not know the details of future tasks. This means any allocation mechanism needs to balance the immediate reward of allocating resources and servicing a task with the potential to earn higher future rewards by saving its resources. Second, an accurate statistical model of task arrivals may not initially be available. This means the mechanism needs to learn this and adapt over time. Finally, in many real-world applications, task owners may be self-interested and behave strategically. Thus, if the mechanism is not carefully designed, task owners may misreport their task requirements or valuations if this results in a better outcome for them (e.g., reporting a higher valuation or different requirements may lead to a task being serviced that would otherwise be rejected). Thus, allocation mechanisms should be *strategyproof*, i.e., robust to this type of strategic behaviour by incentivising truthful reporting.

Existing work has looked at some of these challenges. The literature on secretary problems and optimal stopping [9, 16] considers the first challenge, but typically focuses on worst-case analysis and assumes tasks arrive in random order. Some work investigates strategyproof mechanisms in this context [3, 15] and in similar online resource allocation settings [6, 23, 24], but does not consider the impact and opportunity of learning and adaptation over time. This adaptation is addressed separately by the field of reinforcement learning (RL) [26], and RL has been applied successfully to online resource allocation [11, 19, 27]. However, there is little work that combines these two approaches, i.e., designing strategyproof online mechanisms that learn adaptively.

As an exception to this, some related work combines machine learning and mechanism design. Balcan et al. [5] apply machine learning to the design of strategyproof offline auctions, while Blum and Hartline [7] consider the online case. There is also work on designing strategyproof multi-armed bandit mechanisms [4]. Within this literature, Babaioff et al. [2] consider a problem closely related to ours, where streams of agents are offered items that must be immediately accepted or rejected. However, none of these machine learning approaches consider complex state spaces, where current allocation decisions affect future states and opportunities (something that is naturally handled by RL). Cai et al. [10] look at the application of RL to mechanism design, in order to allocate limited resources (e-commerce impressions) to strategic agents, and Du et al. [12] apply RL to cloud computing. However, these focus on revenue maximisation rather than strategyproofness.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

To address these shortcomings, we investigate how to design strategyproof RL mechanisms for online resource allocations problems. This is challenging, because RL algorithms can learn very general policies that may be open to manipulation. Thus, we propose a novel way of constraining the learning process, which ensures that the learnt policies are always strategyproof. More specifically, we make the following contributions: (1) we formalise a general online resource allocation problem with strategic task owners and show that this problem is NP-hard (even if tasks are known in advance); (2) we show that existing RL approaches are vulnerable to strategic behaviour and we then propose a set of conditions that are sufficient for an RL-based mechanism to be strategyproof and individually rational in our setting; (3) we propose a new mechanism called SP-RL (strategyproof RL) that meets these conditions; and (4) we show experimentally that SP-RL performs close to optimal (typically within 90%) and is consistently better than price-based approaches that are commonly used in practice (by up to 86%).

2 ONLINE RESOURCE ALLOCATION

Here we formally describe the online resource allocation problem we are concerned with. We start with an overview of the system, followed by a description of how allocation policies are defined. We then give complexity results and finally describe how we model strategic task owners, which is a key focus of this paper.

2.1 System Model

We consider a setting with a finite time horizon $t_{\max}$ and discrete time steps $T = \{1, 2, \dots, t_{\max}\}$. A resource provider has different types of resources that can be used for completing tasks, given by the set $R = \{1, 2, \dots, r_{\max}\}$. For each resource type $r \in R$, the provider has a limited quantity of resource units, given by $a_r \in \mathbb{N}$. In an edge cloud setting, $R = \{1, 2, 3\}$ could, for example, represent CPUs, GPUs and GBs of RAM. Here, $a_1 = 100$ would indicate there are 100 available CPUs.

At every time step, a single task i is submitted to the system and is characterised by a type $\theta_i = (v_i, d_i, \boldsymbol{q_i})$, where $v_i \in \mathbb{R}^+$ is the value of the task, $d_i \in \mathbb{N}$ is the number of time steps the task will need for completion and $\boldsymbol{q_i} = [q_{i,1}, q_{i,2}, \dots, q_{i,r_{\max}}] \in \mathbb{N}^{r_{\max}}$ is a vector denoting the quantity of each resource required while the task is being completed. We assume a task serviced at time $t \in T$ is completed just before the start of time step $t + d_i$. Due to the correspondence between time steps and tasks, we will use their indices interchangeably, i.e., we will index the type of the task arriving at time step $t \in T$ by θ_t. Note that time steps could be arbitrarily granular, e.g., representing seconds or fractions of a second and the model can easily be extended to settings where tasks do not arrive at each time step. Since this is an online problem, at a given time step i, the resource provider will only be aware of the current task θ_i and the tasks that arrived before time step i, which we denote by $\theta_{<i}$. We will use θ to denote the set of all tasks that arrive by (and including) time step $t_{\max}$.

2.2 Allocation Policy

We are interested in deriving an allocation policy $\pi(\theta_i, \theta_{<i}) \in \{0, 1\}$, which decides, for every task θ_i, whether to reject it ($\pi(\theta_i, \theta_{<i}) = 0$) or to service it ($\pi(\theta_i, \theta_{<i}) = 1$). We assume this policy can only make feasible allocations, i.e., allocations that do not exceed the resources available and that complete before the start of $t_{\max} + 1$:

$$\sum_{j=1}^{t} \pi(\theta_j, \theta_{<j}) \mathbf{1}(j + d_j > t) \cdot q_{j,r} \leq a_r \ \ \forall t, r, \theta, \tag{1}$$

$$t + \pi(\theta_t, \theta_{<t}) \cdot d_t \leq t_{\max} + 1 \ \ \forall t, \theta \tag{2}$$

where $\mathbf{1}(\cdot)$ is the indicator function. We focus on deterministic allocation policies for now, but, to allow for exploration, we will extend our results to certain stochastic policies later.

Our objective is to design an allocation policy π^* that maximises the expected sum of all values of the allocated tasks (we refer to this as the *social welfare*), i.e., $\pi^* = \text{argmax}_\pi \, \mathbb{E}_\theta \left[\sum_{i=1}^{t_{\max}} \pi(\theta_i, \theta_{<i}) \cdot v_i \right]$.

2.3 Problem Complexity

Theorem 1. *Finding an optimal policy for the online resource allocation problem is NP-hard.*

Proof. We prove this via a reduction from the 0-1 knapsack problem. In this problem, there is a finite set I of items, each with a weight w_i and value u_i. The goal is to find a subset $I' \subseteq I$ that maximises $\sum_{i \in I'} u_i$, subject to the constraint $\sum_{i \in I'} w_i \leq W$, where W is the capacity of the knapsack.

To reduce an instance of the 0-1 knapsack problem to our problem, we set $r_{\max} = 1$, $t_{\max} = |I|$, $a_1 = W$. We then assume a deterministic distribution of tasks, where each task θ_i corresponds to one item, with $\theta_i = (v_i = u_i, d_i = t_{\max} - i + 1, \boldsymbol{q}_i = [w_i])$. As one task is created for each item, this can be done in linear time. Given an optimal algorithm π^* to our problem, we now include an item i in the original knapsack problem if and only if $\pi^*(\theta_i, \theta_{<i}) = 1$.

To conclude the proof, we note that a solution to the original knapsack problem is feasible if and only if the corresponding solution to our problem is also feasible. Specifically, by our construction, Equation 2 is always satisfied. The indicator function $\mathbf{1}(j + d_j > t)$ in Equation 1 is always 1, so the LHS of that equation increases monotonically with t. At $t_{\max}$, the equation is equivalent to the knapsack constraint ($\sum_{\{j \mid \pi^*(\theta_j, \theta_{<j})=1\}} q_{j,1} \leq a_1 \iff \sum_{\{j \mid \pi^*(\theta_j, \theta_{<j})=1\}} w_j \leq W$). As π^* maximises $\sum_{i=1}^{t_{\max}} \pi(\theta_i, \theta_{<i}) \cdot v_i$ (and $v_i = u_i$), an optimal solution to our constructed resource allocation problem is also optimal for the 0-1 knapsack problem. □

As this proof uses a deterministic version of the online resource allocation problem, we note that the online resource allocation problem remains NP-hard, even if all tasks are known in advance.

2.4 Strategic Behaviour

A particular focus of our work is the potential presence of strategic task owners. Here, we wish to investigate whether self-interested task owners may have an incentive to misreport their task characteristics in order to increase their individual utility. A system accommodating such behaviour is undesirable, as it may make incorrect decisions based on false information, affecting its performance. To mitigate this strategic behaviour, we will assume that the resource allocation mechanism can impose payments on the task owners and we will use techniques from mechanism design to incentivise truthful behaviour. We first introduce some definitions.

Definition 1 (Agent reports). *We assume the true type of a task, $\theta_i = (v_i, d_i, \boldsymbol{q}_i)$, is private information of the task owner (henceforth called agent). The agent reports $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{\boldsymbol{q}}_i)$ to the resource allocation mechanism on arrival. If $\theta_i = \hat{\theta}_i$, we say the agent reports truthfully; if $\theta_i \neq \hat{\theta}_i$, it misreports.*

Due to this assumption, the allocation policy needs to operate on the agents' reports (we write $\pi(\hat{\theta}_i, \hat{\theta}_{<i})$ when we need to emphasise this). Note that due to the real-time aspect of our setting, we assume that agents cannot manipulate the order in which they are considered by the mechanism. This is realistic in settings where agents have no advance knowledge of their own tasks and where tasks have to be serviced immediately. We also assume that agents interact with the mechanism only once, i.e., no agent has multiple tasks. This assumption is common in mechanism design [6, 22, 23] and, concretely, implies that agents do not need to strategise about how one interaction will affect the outcomes of future interactions.[1] Given this, we will also refer to the owner of task i as agent i.

Definition 2 (Payments). *In addition to making an allocation decision, the resource allocation mechanism imposes payments on the agents, as given by payment policy $p(\hat{\theta}_i, \hat{\theta}_{<i}) \geq 0$. We will make the reasonable assumption that unallocated agents pay 0, i.e., $\pi(\hat{\theta}_i, \hat{\theta}_{<i},) = 0 \implies p(\hat{\theta}_i, \hat{\theta}_{<i}) = 0$.*

Hence, a mechanism $\mathcal{M}$ is defined by its allocation policy π and its payment policy p ($\mathcal{M} = (\pi, p)$). Given this, we can define the utility of an agent and then two desirable properties that we wish our mechanisms to satisfy.

Definition 3 (Agent utility). *The utility of agent i under mechanism $\mathcal{M}$ is given by $u_i(\hat{\theta}_i, \theta_i, \hat{\theta}_{<i}) = \pi(\hat{\theta}_i, \hat{\theta}_{<i})\phi(\hat{\theta}_i, \theta_i)v_i - p(\hat{\theta}_i, \hat{\theta}_{<i})$, where $\phi(\hat{\theta}_i, \theta_i)$ is 1 if agent i has requested* at least *its required resources and duration, and 0 otherwise.*[2]

Definition 4 (Strategyproof Mechanism). *A mechanism $\mathcal{M}$ is strategyproof if reporting the true type always maximises an agent's utility. Formally: $u_i(\theta_i, \theta_i, \hat{\theta}_{<i}) \geq u_i(\hat{\theta}_i, \theta_i, \hat{\theta}_{<i},), \forall \theta_i, \hat{\theta}_i, \hat{\theta}_{<i}$. This is also known as dominant strategy incentive compatibility (DSIC).*

Definition 5 (Individual Rationality). *A mechanism $\mathcal{M}$ is individually rational if an agent always receives a non-negative utility when reporting truthfully. Formally: $u_i(\theta_i, \theta_i, \hat{\theta}_{<i}) \geq 0, \forall \theta_i, \hat{\theta}_{<i}$.*

We note that our setting is a specific case of one with *single-minded* agents [8], i.e., where agents request a set of items (here resources) and derive a given value (v_i) if they receive this set or a superset, and 0 otherwise (that is, the agent gains no utility for receiving less than the requested set of items). For this setting, we can adapt an existing result (based closely on Lemma 11.9 by Blumrosen and Nisan [8], but extended for our setting):

Theorem 2. *A mechanism $\mathcal{M}$ is strategyproof if it satisfies:*

- ***Monotonicity**: If an agent is allocated by the mechanism, it would remain allocated for any other report with* at least *the same value and* at most *the same resource requirements (including duration). Formally: $\pi(\theta_i, \theta_{<i}) = 1 \implies \pi(\theta_j, \theta_{<i}) = 1, \forall \theta_i \preceq \theta_j, \theta_{<i}$, where $\theta_i \preceq \theta_j \iff (v_i \leq v_j) \wedge (d_i \geq d_j) \wedge (q_{i,1} \geq q_{j,1}) \wedge \ldots \wedge (q_{i,r_{max}} \geq q_{j,r_{max}})$.*
- ***Critical Payment**: An allocated agent pays the minimum value it could have reported while remaining allocated. Formally: $p(\theta_i, \theta_{<i}) = \inf_{\hat{v}_i} \hat{v}_i$, s.t., $\pi((\hat{v}_i, d_i, \boldsymbol{q}_i), \theta_{<i}) = 1$.*

Proof. We will show that an agent i with type θ_i cannot improve its utility by misreporting any other type $\hat{\theta}_i$. First, if the misreporting agent is unallocated ($\pi(\hat{\theta}_i, \hat{\theta}_{<i}) = 0$) or if it holds that $\phi(\hat{\theta}_i, \theta_i) = 0$, the agent clearly cannot benefit from this misreport (due to Definition 3). Thus, we next consider only cases where the agent is allocated with $\hat{\theta}_i$ and it holds that $\phi(\hat{\theta}_i, \theta_i) = 1$.

Given this, agent i cannot be worse off reporting $\hat{\theta}'_i = (\hat{v}_i, d_i, \boldsymbol{q}_i)$ rather than $\hat{\theta}_i$, i.e., switching to its true d_i and true $\boldsymbol{q}_i$. As $\phi(\hat{\theta}_i, \theta_i) = 1$ holds, it follows that $\hat{\theta}_i \preceq \hat{\theta}'_i$. Thus, by monotonicity, reporting $\hat{\theta}'_i$ also results in i being allocated. Moreover, it must hold that $p(\hat{\theta}'_i, \hat{\theta}_{<i}) \leq p(\hat{\theta}_i, \hat{\theta}_{<i})$ (otherwise monotonicity would not hold for the two types $(v', \hat{d}_i, \hat{\boldsymbol{q}}_i) \preceq (v', d_i, \boldsymbol{q}_i)$, where $v' = p(\hat{\theta}_i, \hat{\theta}_{<i})$).

Finally, we show that the agent cannot be worse off switching from $\hat{\theta}'_i = (\hat{v}_i, d_i, \boldsymbol{q}_i)$ to its true type $\theta_i = (v_i, d_i, \boldsymbol{q}_i)$. Here, there are two cases: (1) if both θ_i and $\hat{\theta}'_i$ are allocated (this only happens when $p(\theta_i, \hat{\theta}_{<i}) \leq v_i$), there is no change in utility, as the payments are the same; (2) if θ_i is not allocated (this only happens when $p(\theta_i, \hat{\theta}_{<i}) > v_i$), then the agent avoids a negative utility due to a payment that is higher than its value, receiving 0 instead.

□

As truthful agents never pay more than their valuation, such a mechanism is also individually rational. Having defined the core problem considered in this paper, we next discuss how to solve it using reinforcement learning (RL).

3 REINFORCEMENT LEARNING

In this section, we first formulate the problem as a Markov Decision Process (MDP), then describe how we apply linear function approximation to estimate the action-value function of this MDP, and finally we outline an RL algorithm based on SARSA [25].

3.1 MDP Formulation

Most RL algorithms assume that the underlying problem can be formulated as an MDP. We now describe how our problem can be mapped directly to an MDP, defined here as a tuple (S, A, p, ϱ):

States S: Each state $s \in S$ consists of a time step (i), the report of the current agent ($\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{q}_{i,1}, \ldots, \hat{q}_{i,r_{\max}})$) and the occupancy of the resources, $o_{r,j}$, where $r \in R$ and $j \in \{1, 2, \ldots, a_r\}$. Here, $o_{r,j}$ indicates the number of time steps until the jth-next unit of resource type r will become available again. E.g., if $o_1 = [0, 0, 1, 1, 5]$, there are two free units of resource type 1, two more will become free in the next time step, and one more will become free in five time steps. Thus:

$$s = \Big[i, \hat{v}_i, \hat{d}_i, \hat{q}_{i,1}, \ldots, \hat{q}_{i,r_{\max}}, o_{1,1}, \ldots, o_{1,a_1}, \ldots, o_{r_{\max},1}, \ldots, o_{r_{\max},a_{r_{\max}}} \Big] \quad (3)$$

[1] In practice, this is also reasonable when the frequency and volume of repeated interactions from one agent is low compared to the overall volume of tasks.

[2] Formally, $\phi(\hat{\theta}_i, \theta_i) = \mathbf{1}((\hat{d}_i \geq d_i) \wedge (\hat{q}_{i,1} \geq q_{i,1}) \wedge \ldots (\hat{q}_{i,r_{\max}} \geq q_{i,r_{\max}}))$. This applies when the agent has strict resource requirements with no utility for partial allocations, and if the mechanism always allocates the exact resources requested.

Note that states with $i = t_{\max} + 1$ are terminal states: no more actions are possible and there are no more rewards.

Actions A: There are two possible actions, $A = \{0, 1\}$, indicating whether a task should be rejected or serviced, respectively (thus corresponding to the outputs of the allocation policy π). In states where allocation is infeasible, as given by Equations 1 and 2, we assume only the 0 action is available.

Transition Function p: The transition function $p(s'|s, a) \in [0, 1]$ defines the probability of transitioning to state s' when taking action a in state s. In our problem, all uncertainty in this transition function concerns the new reported task $\hat{\theta}_{t+1}$, which depends on a (usually unknown) distribution of task arrivals. The other parts of the new state s' are deterministic: time increases ($t' = t + 1$) and most occupancy values decrease by 1 (never below 0). If task i was scheduled, some of the occupancy values will be set to $d_i - 1$.

Reward function ϱ: The reward when taking an action a in a particular state s is $\varrho(s, a) = \hat{v}_i$ when $a = 1$ and 0 otherwise.

In reinforcement learning, we are typically interested in estimating the expected sum of future rewards (i.e., *return*) when taking a given action a in a given state s and then following policy[3] π, as given by the action-value function $Q(s, a)$:

$$Q(s, a) = \varrho(s, a) + \Sigma_{s'} p(s'|s, a) Q(s', \pi(s')) \tag{4}$$

Many widely-used reinforcement learning algorithms estimate this action-value function using temporal-difference learning, i.e., they improve estimates of a given state-action pair with immediate observed rewards and action-value estimates for successor states, based on Equation 4 [26]. They then couple this with a policy that picks actions greedily with respect to this estimate, iteratively improving both the policy and the estimated Q-function.

Given this, we note that an optimal policy can be derived when the distribution of tasks is known and discrete, using backwards induction from the terminal states and dynamic programming. However, the distributions of tasks are often initially unknown in realistic settings, where the mechanism may have limited experience, and valuations may be from continuous intervals. More importantly, such an approach would scale with $O(|S|)$, which is impractical for larger settings (e.g., with hundreds of resources and fine-grained time steps). For the same reason, tabular RL approaches that store an estimated Q-value for each state-action pair are not suitable here. We address this in the following section.

3.2 Linear Function Approximation

Due to the potentially large state space, we use an approximator to estimate the value of taking a particular action in a given state (i.e., to estimate the $Q(s, a)$ function). While it is possible to apply deep reinforcement learning approaches, we choose a linear function approximator, because it will help us achieve monotonicity and allows us to easily derive appropriate payment policies.

Specifically, we use a d-dimensional feature vector $\boldsymbol{f}(s)$ to represent the current state s, and two d-dimensional weight vectors, $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ that we use to estimate the Q-values for each possible action in the current state s, $Q(s, a, \boldsymbol{w}) = \boldsymbol{w}_a \boldsymbol{f}(s)^\intercal = \sum_{i=1}^{d} w_{a,i} f_i(s)$.

[3]To align our notation with the RL literature, we here parameterise the allocation policy $\pi(s)$ with the state s, noting that the state can be derived directly from the current task θ_i and the sequence of preceding tasks $\theta_{<i}$.

While there are many ways of choosing appropriate feature vectors, we outline two possible ways below that work well in practice, and that we will use later in our evaluation.

Definition 6 (Full Features). *A full feature vector is a direct mapping from our state representation (Equation 3) to a feature vector, with a bias term (1) added, i.e.,* $\boldsymbol{f}(s) = [1, t, \hat{v}_i, \hat{d}_i, \hat{q}_{i,1}, \ldots, \hat{q}_{i,r_{max}}, o_{1,1}, \ldots, o_{1,a_1}, \ldots, o_{r_{max},1}, \ldots, o_{r_{max},a_{r_{max}}}]$

Since this representation can become large when there are many resource types and a_r is large, we also use a simpler representation:

Definition 7 (Simple Features). *A simple feature vector summarises the current occupancy of the system using simple aggregate statistics as follows:* $\boldsymbol{f}(s) = [1, t, \hat{v}_i, \hat{d}_i, \hat{q}_{i,1}, \ldots, \hat{q}_{i,r_{max}}, \bar{o}, \bar{o}^2, \tilde{o}]$, *where* $\bar{o}$ *is the average occupancy across all resource units (i.e., all* $o_{r,j}$ *values) and* $\tilde{o}$ *is the standard deviation of those values.*

However, as we show in the next section, a key problem with this approach arises when agents are strategic.

3.3 Strategic Manipulation

As discussed above, it is common for RL algorithms to follow a policy that is greedy with respect to the estimated $Q(s, a)$ function, i.e., $\pi(s) = \arg\max_a Q(s, a, \boldsymbol{w})$.[4] Assuming a state representation as given in Definitions 6 and 7 above, agents can clearly directly manipulate the outcome of the policy. Since the reported type is part of the state representation, an agent could strategically misreport parts of its type to ensure that $\pi(s) = 1$ (i.e., the task is allocated), as highlighted in the following example.

Example 1. *Assume that the linear function approximator has learnt* $w_{1,3} = a > 0$ *(the weight of* $\hat{v}_i$*) and* $w_{1,4} = b > 0$ *(the weight of* $\hat{d}_i$*), as well as* $w_{0,3} = 0$ *and* $w_{0,4} = 0$*. These are not unreasonable weights, especially if larger tasks tend to be significantly more valuable than smaller tasks.*[5] *It is clear that by sufficiently increasing either* $\hat{v}_i$ *or* $\hat{d}_i$*, agent* i *can ensure that* $\pi(s) = 1$*. Specifically, increasing* $\hat{d}_i$ *to ensure allocation directly violates the monotonicity condition in Theorem 2, indicating that naïve linear function approximation cannot be strategyproof.*

In the next section, we discuss how to introduce appropriate constraints for the weights to ensure that RL with linear function approximation is strategyproof.

4 STRATEGYPROOF RL

In this section, we first discuss how to achieve monotonicity (the first condition in Theorem 2) for RL with a linear function approximator and greedy action selection. Then we outline how critical payments can be computed (the second condition). We conclude by detailing a specific strategyproof reinforcement learning algorithm.

4.1 Achieving Monotonicity of the Mechanism

As discussed, we note that a given state depends directly on the report $\hat{\theta}_i$ of the agent i that arrives in that state. To emphasise this dependency, we denote by $s_i(\hat{\theta}_i, s_{-i})$ the state at time i when agent

[4]We assume ties are broken in favour of allocation, $a = 1$.

[5]We have verified experimentally that such examples occur in practice, especially early on during learning.

i reports $\hat{\theta}_i$ (directly affecting $\hat{v}_i, \hat{d}_i, \hat{q}_{i,1}, \ldots, \hat{q}_{i,r_{\max}}$ in Equation 3) and the rest of the state is s_{-i} (i.e., the current time and resource occupancy). We can now define some important characteristics of the function $\boldsymbol{f}(s)$ used to derive a feature vector for state s:

Definition 8 (Monotonic Features). *A feature $f_k(s)$ is* type-dependent *if there exist reports $\hat{\theta}_i \neq \hat{\theta}'_i$ and state s_{-i}, such that $f_k(s_i(\hat{\theta}_i, s_{-i})) \neq f_k(s_i(\hat{\theta}'_i, s_{-i}))$, i.e., it is possible for an agent to manipulate the feature via its report. Given this, we say a feature is* monotonically increasing *in agent reports if it is type-dependent and $f_k(s_i(\hat{\theta}_i, s_{-i})) \leq f_k(s_i(\hat{\theta}'_i, s_{-i})), \forall \hat{\theta}_i \preceq \hat{\theta}'_i, s_{-i}$. It is* monotonically decreasing *in agent reports if it is type-dependent and $f_k(s_i(\hat{\theta}_i, s_{-i})) \geq f_k(s_i(\hat{\theta}'_i, s_{-i})), \forall \hat{\theta}_i \preceq \hat{\theta}'_i, s_{-i}$.*

Definition 9 (Monotonic Feature Vector). *Let $\mathcal{T}(\boldsymbol{f}) = \{k \mid f_k(s)$ is type-dependent$\}$ be the set of type-dependent feature indices in a feature vector $\boldsymbol{f}$, $\mathcal{I}(\boldsymbol{f})$ the set of monotonically increasing feature indices and $\mathcal{D}(\boldsymbol{f})$ the set of monotonically decreasing feature indices. We now say $\boldsymbol{f}$ is a* monotonic feature vector *if all type-dependent features are either monotonically increasing or monotonically decreasing in agent reports, i.e., $\mathcal{T}(\boldsymbol{f}) = \mathcal{I}(\boldsymbol{f}) \cup \mathcal{D}(\boldsymbol{f})$.*

As an example, the feature $f_k(s) = t$ is not type-dependent, because it cannot be influenced by an agent's report. The feature $f_k(s) = \hat{v}_i$ is type-dependent and also monotonically increasing (as reporting a higher value leads to a higher value for this feature). However, the feature $f_k(s) = \hat{v}_i \hat{d}_i$ is type-dependent, but neither monotonically increasing nor decreasing (as for two reports $\hat{\theta}_i \preceq \hat{\theta}'_i$ it could either increase or decrease, depending on the specific values for $\hat{v}_i, \hat{v}'_i, \hat{d}_i$ and $\hat{d}'_i$).

Lemma 1. *Both the full and the simple feature vectors described in Definitions 6 and 7 are monotonic feature vectors.*

Proof. All type-dependent features are direct representations of the agent's reported type (i.e., $f_3(s) = \hat{v}_i$, $f_4(s) = \hat{d}_i$ and $f_{r=5\ldots4+r_{\max}}(s) = \hat{q}_{i,r}$). Clearly, these are either monotonically increasing or decreasing in agent reports. □

Given this, we can now restrict the set of possible weights to achieve monotonicity of the allocation policy π:

Lemma 2. *An allocation policy π that is greedy with respect to a linear function approximator $\boldsymbol{w}_a \boldsymbol{f}(s)^\intercal$ is monotonic if the following two conditions are satisfied:*

- *$\boldsymbol{f}$ is a monotonic feature vector.*
- *$\forall j \in \mathcal{I}(\boldsymbol{f}), w_{1,j} \geq w_{0,j} \wedge \forall j \in \mathcal{D}(\boldsymbol{f}), w_{1,j} \leq w_{0,j}$, i.e., if a feature is monotonically increasing in agent reports, then its weight for the allocation action ($a = 1$) must be at least as high as that for the reject action ($a = 0$); and the converse needs to be true for features that are monotonically decreasing.*

Proof. We will show this by assuming two arbitrary types $\theta_i \preceq \theta'_i$ and that θ_i is allocated. We will then show that θ'_i must also be allocated, thus proving that monotonicity holds.

Let s_{-i} be the remaining state, with $\pi(s_i(\theta_i, s_{-i})) = 1$ (for brevity, we will write $s(\theta_i) = s_i(\theta_i, s_{-i})$ in the following). First, we note that if condition 1 in Lemma 2 holds, we can break the action-value estimate into three parts: $\sum_{j=1}^{d} w_{a,j} f_j(s(\theta_i)) = \sum_{j \notin \mathcal{T}(\boldsymbol{f})} w_{a,j} f_j(s(\theta_i)) + \sum_{j \in \mathcal{I}(\boldsymbol{f})} w_{a,j} f_j(s(\theta_i)) + \sum_{j \in \mathcal{D}(\boldsymbol{f})} w_{a,j} f_j(s(\theta_i))$.

Due to the definition of type-dependence, we also have $f_i(s(\theta_i)) = f_i(s(\theta'_i)), \forall j \notin \mathcal{T}(\boldsymbol{f})$ and therefore: $\sum_{j=1}^{d} w_{a,j} f_j(s(\theta'_i)) = \sum_{j=1}^{d} w_{a,j} f_j(s(\theta_i)) + \sum_{j \in \mathcal{I}(\boldsymbol{f})} w_{a,j}(f_j(s(\theta'_i)) - f_j(s(\theta_i))) + \sum_{j \in \mathcal{D}(\boldsymbol{f})} w_{a,j}(f_j(s(\theta'_i)) - f_j(s(\theta_i)))$.

To prove monotonicity, we need to show that:

$$\sum_{j=1}^{d} w_{1,j} f_j(s(\theta'_i)) \geq \sum_{j=1}^{d} w_{0,j} f_j(s(\theta'_i)) \tag{5}$$

By using the equalities above, we can re-arrange this to:

$$\sum_{j=1}^{d} w_{1,j} f_j(s(\theta_i)) + \sum_{j \in \mathcal{I}(\boldsymbol{f})} (w_{1,j} - w_{0,j})(f_j(s(\theta'_i)) - f_j(s(\theta_i))) \geq \sum_{j=1}^{d} w_{0,j} f_j(s(\theta_i)) + \sum_{j \in \mathcal{D}(\boldsymbol{f})} (w_{0,j} - w_{1,j})(f_j(s(\theta'_i)) - f_j(s(\theta_i))) \tag{6}$$

By definition of the greedy allocation and our assumption that $\pi(s_i(\theta_i, s_{-i})) = 1$, we know $\sum_{j=1}^{d} w_{1,j} f_j(s(\theta_i)) \geq \sum_{j=1}^{d} w_{0,j} f_j(s(\theta_i))$, so we can re-write Equation 6 to $\sum_{j \in \mathcal{I}(\boldsymbol{f})} (w_{1,j} - w_{0,j})(f_j(s(\theta'_i)) - f_j(s(\theta_i))) \geq \sum_{j \in \mathcal{D}(\boldsymbol{f})} (w_{0,j} - w_{1,j})(f_j(s(\theta'_i)) - f_j(s(\theta_i)))$. By condition 2 in Lemma 2, and the definitions of $\mathcal{I}(\boldsymbol{f})$ and $\mathcal{D}(\boldsymbol{f})$, we can see that the LHS of this inequality must be zero or positive and the RHS must be zero or negative, thus proving Equation 5 holds. Therefore, $\pi(s(\theta'_i)) = 1$ must hold, confirming monotonicity.

□

As discussed earlier, condition 1 in Lemma 2 is fulfilled by both feature vectors proposed in Definitions 6 and 7. To ensure condition 2, appropriate constraints have to be imposed on the weights. We do this by projecting the weights into the feasible space that meets the condition after every update. Specifically, if the new unconstrained weight vector after an update is $\boldsymbol{w}'_a$ (we assume that weights for only one action a are updated at a time), we set:

$$w_{a,i} = \begin{cases} w'_{a,i} & \text{if } i \notin \mathcal{T}(\boldsymbol{f}) \\ \min(w'_{a,i}, w'_{1-a,i}) & \text{if } (a = 1 \wedge i \in \mathcal{D}(\boldsymbol{f})) \\ & \vee (a = 0 \wedge i \in \mathcal{I}(\boldsymbol{f})) \\ \max(w'_{a,i}, w'_{1-a,i}) & \text{otherwise} \end{cases} \tag{7}$$

Finally, we note that RL policies often employ some randomisation to encourage exploration. A prominent example are ϵ-greedy policies, which choose a random action with some small probability ϵ (that is independent of the current state, including the current agent's report) and follow the greedy policy otherwise.

Corollary 1. *Lemma 2 continues to hold for ϵ-greedy policies.*

We omit a formal discussion for brevity, but we need to consider two cases: (1) π explores with probability ϵ and chooses a random action — here, the agent's reported type has no impact, so monotonicity holds trivially; (2) π exploits with probability $1 - \epsilon$ and chooses a greedy action, where monotonicity holds due to Lemma 2.

Having shown how to achieve monotonicity of the allocation function (condition 1 in Theorem 2), we next discuss the payments.

4.2 Calculating Critical Payments

To ensure our mechanism is strategyproof, we need to define an appropriate payment function $p(\theta_i, \theta_{<i})$, which charges the agent i the minimum value it could have reported to be allocated. This

corresponds exactly to the value of $\hat{v}_i$ where the action-value of the allocate action is equal to the value of the reject action, i.e., $\sum_{j=1}^{d} w_{1,j} f_j(s(\theta)) = \sum_{j=1}^{d} w_{0,j} f_j(s(\theta))$.

Let us assume that $\hat{v}_i$ appears only once as a simple linear feature in the feature vector, at index l, i.e., $f_l(s) = \hat{v}_i$ (in both features vectors discussed earlier in Definitions 6 and 7, this is the case). Then we can re-arrange the above to obtain the critical payment:

$$\hat{v}_i = \frac{1}{w_{1,l} - w_{0,l}} \sum_{j=1, j\neq l}^{d} (w_{0,j} - w_{1,j}) f_j(s(\theta)) \tag{8}$$

Note in the special case that $w_{1,l} = w_{0,l}$, the reported value does not affect the allocation decision. In this case, we set $\hat{v}_i = 0$ if the agent was allocated and $\hat{v}_i = \infty$ if it was not allocated. If $\hat{v}_i < 0$, we also set the payment to 0 (the mechanism remains strategyproof, as task valuations are non-negative).

Theorem 3. *A mechanism that uses an allocation policy π as described in Lemma 2, whose feature vector contains $\hat{v}_i$ exactly once as a feature at index l (i.e., $f_l(s) = \hat{v}_i$) and that uses Equation 8 as its payment function for allocated agents (and 0 for unallocated agents) is strategyproof and individually rational.*

Proof. Strategyproofness follows from Lemma 2 and Theorem 2. The mechanism is individually rational, because unallocated agents pay 0 (thus their utility is 0) and allocated agents pay the critical value, which, by definition, is less than or equal to the value reported by the agent (thus their utility is non-negative). □

A similar approach can be used for feature vectors that contain functions of $\hat{v}_i$, but we focus on the simple case here for brevity and because the proposed feature vectors already work well in practice. Again, the theorem applies if an ϵ-greedy policy is used, as long as agents that are serviced due to exploration pay 0.

4.3 Strategyproof Mechanism

We now propose a novel RL mechanism for the resource allocation problem called SP-RL (strategyproof RL). This is based on semi-gradient SARSA, using linear function approximation [26], but with several extensions that ensure monotonicity.

Algorithm 1 gives the full pseudocode for the SP-RL mechanism. This algorithm takes as input a monotonic feature vector $\boldsymbol{f}$ (with $\hat{v}$ appearing exactly once as feature $f_l(s) = \hat{v}_i$), a learning rate $\alpha \in [0, 1]$, an exploration probability $\epsilon \in [0, 1]$ and a history size $\zeta \geq 1$, which is the number of recent state transitions that the algorithm keeps in memory for learning (similar to an experience replay buffer in deep reinforcement learning [21], we found that sampling from a history of previous transitions improves performance). The algorithm first initialises the weight vectors and history (Lines 2–4).

The algorithm then starts each episode, which is one run of the algorithm, from time step 1 to $t_{\max}$. For each time step i, the algorithm receives the next task report $\hat{\theta}_i$, constructs a new state based on this, s_i, and generates a random number n for the ϵ-greedy decision (Lines 8–10). If there are insufficient resources or with probability $\frac{\epsilon}{2}$ (for exploration), the task is rejected (Line 12). If not, with a further probability $\frac{\epsilon}{2}$ (also exploration), the task is accepted (Line 14). In both cases, the payment for agent i is zero, as these decisions are unaffected by the report.

Algorithm 1 Strategyproof RL Algorithm (SP-RL)

1: **procedure** SP-RL($\boldsymbol{f}$, α, ϵ, ζ)
Inputs: $\boldsymbol{f}$ (monotonic feature vector), α (learning rate), ϵ (exploration probability), ζ (history size)
2: $\boldsymbol{w}_0 \leftarrow [0, \ldots, 0]$ ▷ Weights for reject action
3: $\boldsymbol{w}_1 \leftarrow [0, \ldots, 0]$ ▷ Weights for allocate action
4: history $\leftarrow [\,]$ ▷ Transition history
5: **for** each episode **do**
6: $o_{j,r} \leftarrow 0 \quad \forall r \in R, i \leq a_r$ ▷ Occupancy
7: **for** $i = 1 \ldots t_{\max}$ **do**
8: $\hat{\theta}_i \leftarrow$ Next task report
9: $s_i \leftarrow s_i(\hat{\theta}_i, s_{-i})$ ▷ Construct state
10: $n \in [0, 1]$ ▷ Random number
11: **if** $\neg$CanSchedule($\hat{\theta}_i$, $\boldsymbol{o}$) $\vee\ n \leq \frac{\epsilon}{2}$ **then**
12: $\pi_i, p_i, \varrho_i \leftarrow 0, 0, 0$ ▷ Reject
13: **else if** $n \leq \epsilon$ **then**
14: $\pi_i, p_i, \varrho_i \leftarrow 1, 0, \hat{v}_i$ ▷ Allocate
15: **else**
16: $q_0, q_1 \leftarrow \boldsymbol{w}_0 \boldsymbol{f}(s_i)^\intercal, \boldsymbol{w}_1 \boldsymbol{f}(s_i)^\intercal$
17: **if** $q_1 \geq q_0$ **then**
18: $\pi_i, \varrho_i \leftarrow 1, \hat{v}_i$ ▷ Allocate
19: **if** $w_{1,l} = w_{0,l}$ **then**
20: $p_i \leftarrow 0$
21: **else**
22: $p_i \leftarrow \max(0, \frac{1}{w_{1,l} - w_{0,l}} \cdot \sum_{j=1, j\neq l}^{d} (w_{0,j} - w_{1,j}) f_j(s_i))$
23: **else**
24: $\pi_i, p_i, \varrho_i \leftarrow 0, 0, 0$ ▷ Reject
25: Take allocation action π_i
26: **if** $i > 1$ **then**
27: Add $(s_{i-1}, \pi_{i-1}, \varrho_{i-1}, s_i, \pi_i)$ to history
28: UpdateWeights(history, $\boldsymbol{w}$, α, ζ)
29: Update all $o_{j,r}$
30: Add $(s_{t_{\max}}, \pi_{t_{\max}}, \varrho_{t_{\max}}, 0, 0)$ to history
31: UpdateWeights(history, $\boldsymbol{w}$, α, ζ)

32: **procedure** UpdateWeights(history, $\boldsymbol{w}$, α, ζ)
33: **while** |history| $> \zeta$ **do**
34: Remove oldest element from history
35: $(s_j, \pi_j, \varrho_j, s_{j+1}, \pi_{j+1}) \in$ history ▷ Random
36: **if** $s_{j+1} = 0$ **then** ▷ Terminal state
37: $\Delta Q \leftarrow \varrho_j - \boldsymbol{w}_{\pi_j} \boldsymbol{f}(s_j)^\intercal$
38: **else**
39: $\Delta Q \leftarrow \varrho_j + \boldsymbol{w}_{\pi_{j+1}} \boldsymbol{f}(s_{j+1})^\intercal - \boldsymbol{w}_{\pi_j} \boldsymbol{f}(s_j)^\intercal$
40: $\boldsymbol{w}_{\pi_j} \leftarrow \boldsymbol{w}_{\pi_j} + \alpha \Delta Q \boldsymbol{f}(s_j)$
41: **for** $i \in \mathcal{T}(\boldsymbol{f})$ **do** ▷ Monotonicity Correction
42: **if** $\pi_j = 1 \wedge i \in \mathcal{D}(\boldsymbol{f}) \vee \pi_j = 0 \wedge i \in \mathcal{I}(\boldsymbol{f})$ **then**
43: $\boldsymbol{w}_{\pi_j, i} \leftarrow \min(\boldsymbol{w}_{\pi_j, i}, \boldsymbol{w}_{1-\pi_j, i})$
44: **else**
45: $\boldsymbol{w}_{\pi_j, i} \leftarrow \max(\boldsymbol{w}_{\pi_j, i}, \boldsymbol{w}_{1-\pi_j, i})$

If no exploration takes place and there are sufficient resources, the algorithm switches to a greedy action with respect to the estimated action values. If allocation is chosen (Line 18), the payment for agent i is calculated using Equation 8. Otherwise, the task is rejected and the payment is 0. At every time step after $i = 1$, the previous state transition and reward are stored in the transition history (Line 27) and the weights are updated using the UpdateWeights

procedure. This also happens again after the final time step of every episode, to capture the last terminal state (Line 27).

The UpdateWeights procedure picks a random transition from the history and performs a single stochastic gradient descent step to update the weights (Lines 35–40). Lines 41–45 implement the monotonicity corrections, as given in Equation 7.

In practice, this algorithm could run and learn continuously during deployment (each run, or episode, from time step 1 to $t_{\max}$ might represent a single day). Alternatively, there could be a separate training phase, where task arrivals are simulated based on historical traces for a certain number of episodes, after which the weights are frozen and learning stops. Or it could be a mixture, where the policy is trained on simulated data initially, but then continues to be improved during execution.

Finally, as we will show in the next section, SP-RL tends to converge to good solutions. However, we leave a thorough investigation of how the monotonicity correction affects the known theoretical convergence properties [26] of semi-gradient SARSA with linear function approximation to future work. To conclude, we note that the SP-RL algorithm meets all conditions in Theorem 3. Thus:

Corollary 2. *The SP-RL algorithm is strategyproof and individually rational.*

5 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate our proposed algorithm in a range of settings.

5.1 Experimental Setup

Unless specified otherwise, we use the following parameters for SP-RL: $\epsilon = 0.1$, $\alpha = 0.1$, $\zeta = t_{\max}$, as these worked well in practice for a range of settings. We train our algorithm for 10,000 training episodes to ensure convergence to a good policy, and we linearly decrease both ϵ and α to 0 by the end of training. We test SP-RL using the simple and full feature vectors, as well as with and without monotonicity correction. We will refer to these variants as Simple (SP-)RL and Full (SP-)RL (omitting SP without the correction). We also compare performance to the following benchmarks:

- Optimal: This is the optimal policy, obtained using backwards induction and dynamic programming.
- Offline Greedy: In large settings where the optimal is infeasible to compute, this is used as a substitute benchmark. The algorithm assumes full knowledge of future tasks (which is clearly unrealistic in practice). It greedily considers each task i in order of decreasing value density ($v_i/(d_i \Sigma_r q_{i,r})$), servicing i if there are still sufficient remaining resources.
- First-Come-First-Served (FCFS): This algorithm services tasks as long as there are still sufficient resources available.
- Optimal Pricing: This algorithm sets a price per resource unit (ψ) and services tasks if $v_i \geq \psi d_i \Sigma_r q_{i,r}$. This mirrors approaches where resources are sold for fixed prices, as is common in existing cloud settings [17]. We choose ψ optimally from a discretised range of prices (in steps of 0.01).

For all algorithms, we average performance over 1,000 episodes and report 95% confidence intervals. For SP-RL we retrain the algorithm from scratch every 20 episodes, by resetting all weights to 0 and re-running it for 10,000 training episodes (without recording performance during training). This ensures that noise due to the learning process is reduced and that the results are representative of the algorithm's average performance, as SP-RL will learn slightly different policies after each training run.

5.2 Results

In the following, we consider three increasingly complex settings.

5.2.1 Small discrete setting. We start by considering a particularly challenging setting for the Optimal Pricing and FCFS benchmarks. We assume $t_{\max} = 20$, there is only one resource type ($r_{\max} = 1$) and the system has ten units available ($a_1 = 10$). There are only two task types (a and b): $\theta_a = (v_a = 1, d_a = 11, q_{a,1} = 1)$ and $\theta_b = (v_b \geq v_a, d_b = 1, q_{b,1} = 1)$ (where we vary v_b in the experiments). Tasks of type a arrive at all time steps, except for $t = 11$, at which a task of type b arrives (note that no more tasks can be started after $t = 11$, as they would not complete by $t_{\max}$). This is challenging for FCFS, because all resources will be occupied by the time θ_b arrives (thus, it will lose out on the potentially higher-valued task). It is challenging for Optimal Pricing, because ψ can only be set to service either all tasks or only θ_b. Optimal schedules all but one of the θ_a tasks, leaving one resource for θ_b.

In Figure 1, we show the average social welfare of SP-RL and other benchmarks as we vary v_b from 1 to 20. FCFS performs worst, as expected. When $v_b > 10$, Optimal Pricing increases ψ to ensure that task b is serviced, but always achieves 9 utility points less than Optimal. In terms of the SP-RL variants, we note that using a full feature vector here leads to an average utility that is always close to optimal (between 97-98% of the optimal), showing the RL can outperform both Optimal Pricing and FCFS by up to 86%. The simple feature vector achieves close to optimal when $v_b \geq 10$ (between 92-99% of the optimal), but does not learn a better policy than the benchmarks when $v_b = 5$ (due to the limited features and the smaller difference between the action values that are estimated by those features). We note that in cases where RL is close to optimal on average, the policy typically converges to the optimal, but in rare cases diverges slightly (due to randomness in exploration).

Importantly, we note that there is no significant difference between the monotonic and non-monotonic versions of SP-RL. Intuitively, this is because monotonicity should hold in the optimal action-value function in any case. Nevertheless, it is important to constrain the weights, especially during early learning, where monotonicity may otherwise be violated. As these trends hold generally, we focus only on the monotonic versions of SP-RL next.

5.2.2 Large discrete setting. Next, to investigate whether similar trends hold in more realistic settings, we now assume $t_{\max} = 100$, $r_{\max} = 1$ and $a_1 = 5$. There are two task types, $\theta_a = (v_a = 20, d_a = 25, q_{a,1} = 1)$ and $\theta_b = (v_b = 100, d_b = 1, q_{b,1} = 1)$, i.e., long low-value tasks and short high-value tasks. At each time, a random type arrives, and we vary the probability of the high-value tasks, p_b.

Figure 2 shows the performance of the algorithms. Here, when the probability of a high value task is non-zero, FCFS performs worst, as it cannot keep resources free for the high-value tasks.

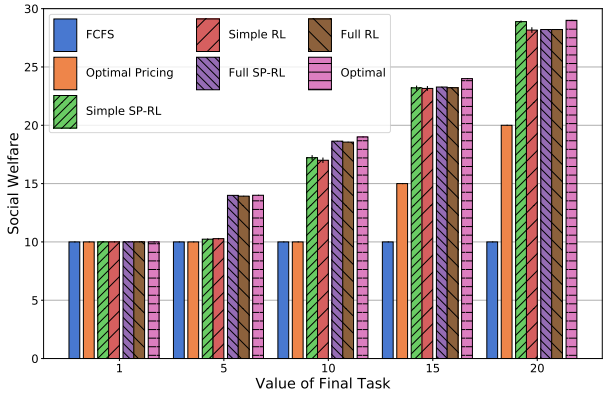

Figure 1: Performance in small discrete setting.

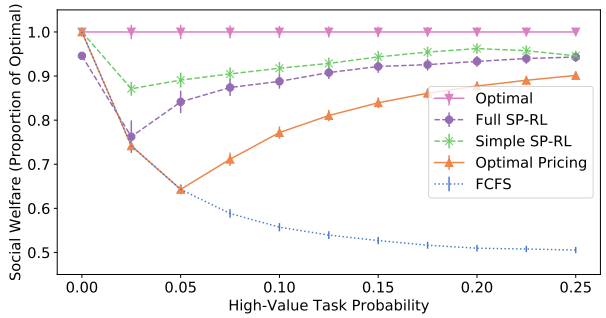

Figure 2: Performance in large discrete setting.

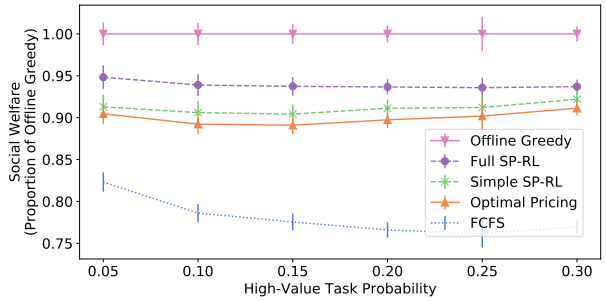

Figure 3: Performance in large continuous setting.

Optimal Pricing initially follows this trend, but for higher probabilities switches completely to the high-value tasks. In contrast, both SP-RL algorithms learn to keep at least one resource free for the high-value tasks. Here, the simple feature vector performs better than the full feature vector, highlighting that the choice of feature vector matters and may be dependent on the specific environment (although both outperform FCFS and Optimal Pricing in most settings). Overall, Simple SP-RL typically achieves 90% or more of the optimal, while Full SP-RL achieves over 80% in most settings. In the best case, Simple SP-RL outperforms Optimal Pricing by 39% and FCFS by 87%.

5.2.3 Large continuous setting. We now consider a larger setting with many resources and continuous values. Here, $t_{\max} = 200$, $r_{\max} = 1$, $a_1 = 400$, $a_2 = 200$, $a_3 = 40$. We again assume lower-value tasks and higher-value tasks. Lower-value tasks (θ_a) have a duration d_a drawn uniformly at random from $\{1, \ldots, 30\}$, and require between $\{1, \ldots, 10\}$ resource units for two of the resources and between $\{1, \ldots, 100\}$ units of the remaining resource (this resource is chosen uniformly at random). The value of the task is $v_a = \mathcal{N}^+(1, 1)d_a\Sigma_r q_{a,r}$, where $\mathcal{N}^+(\mu, \sigma)$ is a random value sampled from a normal distribution with mean μ and standard deviation σ that is truncated below 0. Higher-value tasks (θ_b) have a duration chosen between $\{1, \ldots, 30\}$, and have one of three different resource requirements: $\{1, \ldots, 10\}$ units of either resource 2 or 3 (and none of the others), or $\{1, \ldots, 10\}$ units of all three resource types. Their value is chosen as $v_b = \mathcal{N}^+(100, 10)d_a\Sigma_r q_{a,r}$. This scenario was chosen to represent highly heterogeneous tasks, where resource requirements and value are generally correlated.

Figure 3 shows the results for this setting, as we vary the probability of high value tasks (p_b). Here, Optimal Pricing generally performs better than in previous settings. Due to the large number of resources and the continuous valuation distributions, it can set good prices to admit a mixture of low and high-value tasks (achieving around 90% of Offline Greedy). Nevertheless, both SP-RL approaches achieve a higher utility. Here, the full feature vector performs better than the simple feature vector, as it allows policies that depend more specifically on resource availability. Overall, Full SP-RL consistently achieves 93–95% of Offline Greedy.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have considered how to design strategyproof reinforcement learning mechanisms for online resource allocation. We first showed that the allocation problem we consider is NP-hard and then we proposed a set of sufficient conditions for ensuring that a mechanism based on linear function approximation is both strategyproof and individually rational. We developed a novel algorithm, SP-RL, that fulfils these conditions and showed experimentally that it performs close to the optimal (or an offline benchmark), while outperforming all benchmarks. However, there are clearly limitations of linear function approximation (e.g., it is difficult to encode complex time dependencies without adding new features). Thus, in future work, we will consider other function approximation techniques. We will also investigate settings where agents have multiple tasks and thus interact several times with the mechanism. Finally, for settings where financial payments are impractical, we will look at the use of alternative virtual currencies.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Hany Atlam, Robert Walters, and Gary Wills. 2018. Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing* 2, 2 (4 2018), 10. DOI: https://doi.org/10.3390/bdcc2020010

[2] Moshe Babaioff, Shaddin Dughmi, Robert Kleinberg, and Aleksandrs Slivkins. 2015. Dynamic Pricing with Limited Supply. *ACM Transactions on Economics and Computation (TEAC)* 3, 1 (2015), 4:1–4:26. DOI: https://doi.org/10.1145/2559152

[3] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. 2008. Online Auctions and Generalized Secretary Problems. *ACM SIGecom Exchanges* 7, 2, Article 7 (2008), 11 pages. DOI: https://doi.org/10.1145/1399589.1399596

[4] Moshe Babaioff, Yogeshwer Sharma, and Aleksandrs Slivkins. 2009. Characterizing Truthful Multi-armed Bandit Mechanisms: Extended Abstract. In *Proceedings of the 10th ACM Conference on Electronic Commerce (EC '09)*. ACM, New York, NY, USA, 79–88. DOI: https://doi.org/10.1145/1566374.1566386

[5] Maria-Florina Balcan, Avrim Blum, Jason D Hartline, and Yishay Mansour. 2005. Mechanism design via machine learning. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*. IEEE, 605–614. DOI: https://doi.org/10.1109/SFCS.2005.50

[6] Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. 2019. A truthful online mechanism for resource allocation in fog computing. In *The 16th Pacific Rim International Conference on Artificial Intelligence (PRICAI)*. 363–376. DOI: https://doi.org/10.1007/978-3-030-29894-4_30

[7] Avrim Blum and Jason D Hartline. 2005. Near-optimal online auctions. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*. Society for Industrial and Applied Mathematics, 1156–1163.

[8] Liad Blumrosen and Noam Nisan. 2007. Combinatorial auctions. In *Algorithmic Game Theory*, Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani (Eds.). Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511800481.013

[9] F. Thomas Bruss. 1984. A Unified Approach to a Class of Best Choice Problems with an Unknown Number of Options. *Ann. Probab.* 12, 3 (08 1984), 882–889. DOI: https://doi.org/10.1214/aop/1176993237

[10] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for e-Commerce. In *Proceedings of the 2018 World Wide Web Conference (WWW'18)*. 1339–1348. DOI: https://doi.org/10.1145/3178876.3186039

[11] Mingxi Cheng, Ji Li, and Shahin Nazarian. 2018. DRL-cloud: Deep Reinforcement Learning-based Resource Provisioning and Task Scheduling for Cloud Service Providers. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASPDAC '18)*. IEEE Press, Piscataway, NJ, USA, 129–134. DOI: https://doi.org/10.1109/ASPDAC.2018.8297294

[12] Bingqian Du, Chuan Wu, and Zhiyi Huang. 2019. Learning Resource Allocation and Pricing for Cloud Profit Maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 7570–7577. DOI: https://doi.org/10.1609/aaai.v33i01.33017570

[13] Mohammed S. Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. 2018. Toward Low-Latency and Ultra-Reliable Virtual Reality. *IEEE Network* 32, 2 (3 2018), 78–84. DOI: https://doi.org/10.1109/MNET.2018.1700268

[14] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. 2019. Mobile Edge Computing for the Internet of Vehicles: Offloading Framework and Job Scheduling. *IEEE Vehicular Technology Magazine* 14, 1 (3 2019), 28–36. DOI: https://doi.org/10.1109/MVT.2018.2879647

[15] Mohammad Taghi Hajiaghayi, Robert Kleinberg, and David C. Parkes. 2004. Adaptive Limited-supply Online Auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC '04)*. ACM, New York, NY, USA, 71–80. DOI: https://doi.org/10.1145/988772.988784

[16] Anna Karlin and Eric Lei. 2015. On a Competitive Secretary Problem. In *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. 944–950.

[17] Ian A Kash, Peter Key, and Warut Suksompong. 2019. Simple Pricing Schemes for the Cloud. *ACM Transactions on Economics and Computation (TEAC)* 7, 2 (6 2019), 7:1–7:27. DOI: https://doi.org/10.1145/3327973

[18] Pavel Mach and Z Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1628–1656. DOI: https://doi.org/10.1109/COMST.2017.2682318

[19] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 50–56. DOI: https://doi.org/10.1145/3005745.3005750

[20] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358. DOI: https://doi.org/10.1109/COMST.2017.2745201

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[22] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511800481

[23] David C. Parkes. 2007. Online mechanisms. In *Algorithmic Game Theory*, Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani (Eds.). Cambridge University Press, 411–440. DOI: https://doi.org/10.1017/CBO9780511800481.018

[24] David C Parkes and Satinder P Singh. 2004. An MDP-based approach to online mechanism design. In *Proceedings of the 16th International Conference on Neural Information Processing Systems (NIPS'03)*. 791–798.

[25] Gavin A. Rummery and Mahesan Niranjan. 1994. *On-Line Q-Learning Using Connectionist Systems*. Technical Report. Cambridge University.

[26] Richard Sutton and Andrew Barto. 2018. *Reinforcement Learning* (2nd ed.). The MIT Press.

[27] Ziyao Zhang, Liang Ma, Konstantinos Poularakis, Kin K Leung, and Lingfei Wu. 2019. DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN. In *IEEE International Conference on Communications (ICC 2019)*. IEEE, 1–7.