# Resource-Aware Cost-Sharing Mechanisms with Priors

VASILIS GKATZELIS, Drexel University
EMMANOUIL POUNTOURAKIS, Drexel University
ALKMINI SGOURITSA, University of Liverpool

In a decentralized system with $m$ machines, we study the selfish scheduling problem where each user strategically chooses which machine to use. Each machine incurs a cost, which is a function of the total load assigned to it, and some cost-sharing mechanism distributes this cost among the machine's users. The users choose a machine aiming to minimize their own share of the cost, so the cost-sharing mechanism induces a game among them. We approach this problem from the perspective of a designer who can select which cost-sharing mechanism to use, aiming to minimize the price of anarchy (PoA) of the induced games.

Recent work introduced the class of *resource-aware* cost-sharing mechanisms, whose decisions can depend on the set of machines in the system, but are oblivious to the total number of users. These mechanisms can guarantee low PoA bounds for instances where the cost functions of the machines are all convex or concave, but can suffer from very high PoA for cost functions that deviate from these families.

In this paper we show that if we enhance the class of resource-aware mechanisms with some prior information regarding the users, then they can achieve low PoA for a much more general family of cost functions. We first show that, as long as the mechanism knows just two of the participating users, then it can assign special roles to them and ensure a constant PoA. We then extend this idea to settings where the mechanism has access to the probability with which each user is present in the system. For all these instances, we provide a mechanism that achieves an expected PoA that is logarithmic in the expected number of users.

CCS Concepts: • **Theory of computation** → **Quality of equilibria**; **Algorithmic game theory**.

Additional Key Words and Phrases: cost-sharing, resource-aware, price of anarchy, scheduling games

**ACM Reference Format:**
Vasilis Gkatzelis, Emmanouil Pountourakis, and Alkmini Sgouritsa. 2021. Resource-Aware Cost-Sharing Mechanisms with Priors. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC '21), July 18–23, 2021, Budapest, Hungary.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3465456.3467650

## 1 INTRODUCTION

In this paper we revisit a classic selfish scheduling problem: in a large decentralized system with a set $M$ of machines and a set $\mathcal{N}$ of registered users, each day some subset of these users enter the system seeking to process some task. Each user assigns their task to one of the machines, generating a cost that depends on the machine's total load, and the cost of each machine is then charged to its users, through some cost-sharing mechanism. The users' goal is to minimize their own share of the cost, so they strategically assign their task to the machine that would yield the smallest cost share. However, their cost share depends on the congestion of each machine, and thus on the strategic choices of all the other users currently in the system, giving rise to a game.

The need to better understand these games and to evaluate the efficiency of their outcomes lies at the heart of Algorithmic Game Theory, and some of the first seminal papers in this literature


*EC '21, July 18–23, 2021, Budapest, Hungary*

ACM ISBN 978-1-4503-8554-1/21/07…$15.00
https://doi.org/10.1145/3465456.3467650

analyzed the *price of anarchy* (PoA) of such games, i.e., the extent to which the performance of their Nash equilibria approximates the optimal performance. Much of this work, e.g., in congestion games and network formation games, assumed that the users share the cost *equally*, in accordance with the Shapley value cost-sharing mechanism (e,g., see Chapters 18 and 19, respectively, from [27]). However, it soon became clear that the equal-sharing policy can lead to highly inefficient outcomes, even in very simple instances [2]. As a result, subsequent work focused on the design of alternative, more sophisticated, cost-sharing mechanisms, with the goal of reducing the PoA.

The first to study the extent to which a designer can reduce the PoA using improved cost-sharing mechanisms were Chen et al. [7]. One of their main goals was to analyze mechanisms that are *stable* (i.e., guarantee the existence of pure Nash equilibria in the games they induce) and *decentralized* (i.e., have limited information regarding the overall state of the system). Taking the need for decentralization to an extreme, they focused on the class of *oblivious* cost-sharing mechanisms[1], which decide how to share the cost of each machine among its users without using *any* information regarding the set of other users or machines that are present in the system. After providing a precise characterization of stable mechanisms for network formation games (where the resources that the agents use have *constant* cost-functions), they systematically analyzed their performance. Building on this work, von Falkenhausen and Harks [28] and then Christodoulou et al. [9] considered more general classes of cost functions. Among other results, von Falkenhausen and Harks [28] showed that no oblivious cost-sharing mechanism can guarantee a PoA bound better than linear function in the number of agents, even for instances with concave cost functions.

Motivated by this limitation of oblivious mechanisms, subsequent work introduced the model of *resource-aware mechanisms* [9, 13]. Compared to oblivious mechanisms, resource-aware ones are more informed: their decision regarding how to share the cost of a machine can also depend on the set of other machines that are available in the system. Using this additional information, Christodoulou et al. [9] managed to overcome the limitations of oblivious mechanisms and design resource-aware mechanisms that achieve a constant PoA for convex and concave cost functions. On the negative side, they showed that there exists a class of seemingly simple cost functions for which no resource-aware cost-sharing mechanism can achieve a PoA better than $O(\sqrt{n})$.

These negative results suggest that it may be impossible for resource-aware mechanisms to achieve a constant PoA for interesting cost functions beyond convex and concave. However, although resource-aware mechanisms are more informed than oblivious ones, they are still severely limited in terms of what they know about the users in the system. In this paper we enhance resource-aware mechanisms with some prior information regarding the users in the system, and we show that this is sufficient for us to design cost-sharing mechanisms that achieve low PoA for a very broad class of cost functions.

### 1.1 Our Results

Our main results show that, using only a limited amount of prior information regarding the set of users in the system, resource-aware cost-sharing mechanisms can guarantee a low price of anarchy for a very wide class of selfish scheduling problems.

*Cost functions.* In contrast to prior work in cost-sharing mechanisms, which was mostly restricted to either convex or concave cost functions, our positive result applies to a much larger class of functions. Specifically, we consider any instance where the cost functions of the machines satisfy a mild condition regarding how fast they can grow. We call a cost function *bounded* if it satisfies the condition that $c(\ell+1)/c(\ell) = O(1)$ for all $\ell > 0$, i.e., that the relative jump in the cost function can be upper bounded by some constant. Although the class of bounded functions does not capture

[1] Also known as *uniform* mechanisms.

extreme examples of cost functions such as $c(\ell) = \ell^\ell$, where $c(\ell+1)/c(\ell) > \ell$, it captures the vast majority of functions that may characterize the cost incurred by some machine as a function of its load. For example, it includes all polynomial and even exponential cost functions. Note that this class also contains highly complicated functions that may not have a closed-form expression.

*Games with two known users.* We first consider resource-aware mechanisms that are oblivious to the set of users in the system, with the exception of just two users. The main idea behind our proposed mechanism is to assign special roles to these two users, referred to as *enforcers*, and carefully incentivize them to enforce an approximately efficient assignment in equilibrium. Using this approach we manage to guarantee a constant PoA for instances with any combination of bounded cost functions.

**Theorem:** For any class of scheduling games with two known users and bounded cost functions, there exists a stable resource-aware cost-sharing mechanism that achieves a constant PoA.

*Games with stochastic user arrivals.* We then extend this idea to the case where each user enters the system with some probability $p$, and the mechanism knows $p$ but not the realization. To achieve a good PoA bound for this class of instances we assign the role of the enforcer to more users, depending on the value of $p$, and guarantee an expected PoA that is at most logarithmic in the expected number of users.

**Theorem:** For any class of scheduling games with i.i.d. arrivals[2] and bounded cost functions, there exists a stable resource-aware cost-sharing mechanism that achieves an expected PoA of $O(\log(\tilde{n}))$, where $\tilde{n} = p|\mathcal{N}|$ is the expected number of users.

*Technical obstacles.* Designing efficient cost-sharing mechanisms for such a wide family of instances is quite demanding: the structure of the optimal assignment can change, depending on the actual number, $n$, of users in the system, but the mechanism is oblivious to this number. So, how can the mechanism approximate the optimal solution without knowing $n$? Prior work focused on the case of concave or convex cost functions, and designed mechanisms leveraging the fact that the corresponding optimal assignments are reasonably "well behaved": for concave costs there always exists an optimal solution where all the jobs are assigned to a *single* machine, and for convex costs an optimal assignment can be reached using a simple *greedy* solution (e.g., [8, 9]). However, we cannot expect to find such convenient structural properties when dealing with the vast family of bounded functions, because the optimal assignment can change radically as a function of $n$.

To deal with this fundamental obstacle, we propose a novel solution: rather than trying to implement the optimal assignment in equilibrium, we instead seek to implement a "well behaved" alternative assignment implied by an *online algorithm*. This algorithm assigns jobs to machines using a predetermined assignment sequence which is *independent of the total number of jobs, n*. We prove that this algorithm has a constant competitive ratio and then carefully design our cost-sharing mechanisms aiming to implement the outcome of this algorithm in equilibrium, thus inheriting a good approximation guarantee. We believe this technique may be of independent interest.

### 1.2 Related Work

Our work extends the recent literature that uses resource-aware cost-sharing mechanisms to achieve low PoA in different classes of games. Christodoulou and Sgouritsa [13] were the first to study this family of mechanisms[3], focusing on the class of network formation games (like Chen et al. [7] did for the family of oblivious mechanisms). Unlike the scheduling games that we study in

[2]We also extend this result to hold even if each bidder $i$ arrives with a different probability, $p_i$.

[3]In their paper, Christodoulou and Sgouritsa [13] refer to these mechanisms as *universal* instead of resource-aware.

this paper, network formation games take place over a graph: each agent is associated with a vertex of the graph and needs to use a path connecting that vertex to a designated sink-vertex, and each edge of the graph corresponds to a resource with a constant cost function. One can think of the games in our paper as the special case where the graph has just two vertices (a source and a sink) and several parallel edges: each edge corresponds to one of the machines, and every agent needs to choose one of these edges in order to get from the source to the sink. From this perspective, our games are more restricted in terms of the users' strategy space, but quite more general in terms of the classes of cost functions. Christodoulou and Sgouritsa [13] showed that when the graph is outerplanar, then resource-aware mechanisms can outperform oblivious ones, but they also proved that an analogous separation is not possible for general graphs. In subsequent work, Christodoulou et al. [9] designed resource-aware mechanisms for the same class of scheduling games that we study in this paper, and were able to achieve a constant PoA for instances with convex and concave cost functions. In a recent paper, Christodoulou et al. [8] extended many of these results to more general graphs, beyond parallel links, including directed acyclic or series parallel graphs with convex or concave cost functions on the edges.

The assumption that the cost-sharing mechanism may have additional prior information regarding the users was also part of the model studied by Christodoulou and Sgouritsa [13] for the case of network formation games. Specifically, rather than assuming that the source vertex of each agent is chosen adversarially, they assumed that it is drawn from a distribution over all vertices. The cost-sharing mechanism is aware of this stochastic process, so they designed a mechanism that leverages this information to achieve a constant PoA. Following-up on this work Christodoulou et al. [11] extended the constant PoA to also include Bayesian Nash equilibria.

Ensuring that a cost-sharing mechanism is stable can be quite demanding, so characterizations of stable mechanisms can be very useful. Building on the impressive characterization of stable oblivious mechanisms by Chen et al. [7], Gopalakrishnan et al. [16] provided a characterization for the set of stable oblivious cost-sharing mechanisms. They proved that these mechanisms correspondend to the class of generalized weighted Shapley values. Leveraging this characterization, Gkatzelis et al. [15] analyzed this family of cost-sharing protocols and showed that the unweighted Shapley value achieves the optimal price of anarchy guarantees for a large family of network cost-sharing games.

Other papers on the design and analysis of cost-sharing protocols include Harks and von Falkenhausen [19], who focused on capacitated facility location games, Marden and Wierman [22] who considered a utility maximization model, and Harks et al. [17], who considered a model that imposes some constraints over the portions of the cost that can be shared among the agents. Also, Harks and Miller [18] studied the performance of several cost-sharing protocols in a setting, where each player can declare a different demand for each resource.

Finally, there are several other models in which cost-sharing has played a central role. For example, Moulin and Shenker [26] focused on participation games, while Moulin [25] and Mosk-Aoyama and Roughgarden [23] studied queueing games. Caragiannis et al. [6] recently also pointed out some connections between cost-sharing mechanisms and the literature on coordination mechanisms, which started with the work of Christodoulou et al. [10] and led to several papers focusing on decentralized scheduling policies for machine scheduling games [1, 3–5, 12, 14, 20, 21]. Just like the research on cost-sharing mechanisms, most of the work in coordination mechanisms studies how the price of anarchy varies with the choice of local scheduling policies on each machine (i.e., the order in which to process jobs assigned to the same machine).

## 2 PRELIMINARIES

We analyze the scheduling games that arise in a decentralized system with a set $M = \{1, 2, \ldots, m\}$ of $m$ machines and a set $N = \{1, 2, \ldots, n\}$ of $n$ users. Each user owns a job and needs to schedule it

on one of the machines. Each machine $j \in M$ is characterized by a cost function $c_j : \mathbb{N} \to \mathbb{R}$, where $c_j(\ell)$ is the cost that the machine would incur for processing a total of $\ell$ jobs. The cost function satisfies $c_j(0) = 0$ and it is non-decreasing.

The strategy profile, $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, of a scheduling game is a schedule, where $s_i$ corresponds to the machine that player $i$ chooses for her job. We use $S_j(\mathbf{s}) = \{i \in N : s_i = j\}$ to denote the set of players who scheduled their jobs on machine $j$ in profile $\mathbf{s}$, and $\ell_j(\mathbf{s}) = |S_j(\mathbf{s})|$ to denote the load on machine $j$ in $\mathbf{s}$. Therefore, the cost of machine $j$ in this schedule is $c_j(\ell_j(\mathbf{s}))$, and the overall generated cost is $C(\mathbf{s}) = \sum_{j \in M} c_j(\ell_j(\mathbf{s}))$. For notational simplicity, apart from $c_j(\ell_j(\mathbf{s}))$, we also use $c_j(\mathbf{s})$ to denote the cost of machine $j$ in $\mathbf{s}$, since $j$'s load is directly implied by $\mathbf{s}$.

**Cost-Sharing Mechanisms.** A *cost-sharing mechanism* is a protocol that determines the cost of each agent using a machine. Formally, a cost-sharing mechanism $\Xi$ defines at each schedule $\mathbf{s}$ a nonnegative cost share $\xi_{ij}(\mathbf{s})$ for each $j \in M$ and $i \in S_j(\mathbf{s})$. Since the machine that $i$ uses, i.e., $s_i$, is implied by $\mathbf{s}$, we also denote this cost share as $\xi_i(\mathbf{s})$. In any schedule $\mathbf{s}$, the cost of each machine $j$ must be fully covered by the agents using it, so $\sum_{i \in S_j(\mathbf{s})} \xi_i(\mathbf{s}) \geq c_j(\ell_j(\mathbf{s}))$. We use $\hat{C}(\mathbf{s}) = \sum_{j \in M} \sum_{i \in S_j(\mathbf{s})} \xi_i(\mathbf{s})$ to denote the overall cost suffered by the users in $\mathbf{s}$. Since the agents pay at least the cost they generate, we have $\hat{C}(\mathbf{s}) \geq C(\mathbf{s})$ for every profile $\mathbf{s}$. If there exists some profile for which this inequality is strict, i.e., the cost suffered by the users is greater than the cost that they generated, then we say that the cost-sharing mechanism uses *overcharging*.

**Resource-Aware Mechanisms.** In the class of *resource-aware* cost-sharing mechanisms, the value of the cost-share $\xi_{ij}(\mathbf{s})$ for each $i \in S_j(\mathbf{s})$ can depend on the set $S_j(\mathbf{s})$ of agents using that machine, on the set of machines $M$, and their cost functions, but not on the set $N \setminus S_j(\mathbf{s})$ of agents using other machines. In this paper we enhance this class of mechanisms with some prior stochastic information regarding the set $N \setminus S_j(\mathbf{s})$, which enriches the set of cost-sharing functions that we can implement, allowing us to achieve improved performance guarantees.

**Pure Nash Equilibrium (PNE).** A tuple $(N, M, \mathbf{c}, \Xi)$ of a set of agents, a set of machines and their cost functions $\mathbf{c} = (c_j)_{j \in M}$, and a cost-sharing mechanism, defines a scheduling game $G$. The goal of every user in this game is to choose a machine that minimizes her own share of the cost, determined by $\Xi$. A strategy profile $\mathbf{s}$ is a *pure Nash equilibrium* (PNE) of this game $G$ if for every player $i \in N$, and every strategy $s_i' \in M$

$$\xi_i(\mathbf{s}) \;=\; \xi_i(s_i, \mathbf{s}_{-i}) \;\leq\; \xi_i(s_i', \mathbf{s}_{-i}),$$

where $\mathbf{s}_{-i}$ denotes the profile of strategies for all agents other than $i$. In other words, in a PNE $\mathbf{s}$ no agent can decrease her cost share by unilaterally deviating from machine $s_i$ to $s_i'$ if all the other agents' choices remain fixed.

**Stability.** In accordance with prior work, we restrict our attention to *stable* cost-sharing mechanisms, i.e., ones that induce games possessing at least one PNE.

**Price of Anarchy (PoA).** To measure the performance of a cost-sharing mechanism in a given game, $G$, we evaluate the total cost $\hat{C}(\mathbf{s})$ suffered by the users in the worst equilibrium $\mathbf{s}$, and compare it to the minimum total cost they could suffer. If we let $Eq(G)$ be the set of all PNE of $G$ and $F(G)$ denote the set of all its feasible schedules, then the *price of anarchy* (PoA) of game $G$ is

$$\mathrm{PoA}(G) = \frac{\max_{\mathbf{s} \in \mathrm{Eq}(G)} \hat{C}(\mathbf{s})}{\min_{\mathbf{s}^* \in F(G)} C(\mathbf{s}^*)}.$$

Rather than evaluating the performance of cost-sharing mechanisms on a single game, we evaluate them on large classes of games. A class of scheduling games, $\mathcal{G}$, is defined by a tuple $(\mathcal{N}, \mathcal{C}, \Xi)$, which comprises a universe of players $\mathcal{N}$, a universe of cost functions $\mathcal{C}$, and a cost sharing mechanism $\Xi$. An instance of a scheduling game $G \in \mathcal{G}$ consists of some subset of

users $S \subseteq \mathcal{N}$, a set $M$ of machines with cost functions from $C$, and the cost sharing mechanism $\Xi$. The *worst-case price of anarchy* of mechanism $\Xi$ for a class of games $\mathcal{G}$ is then defined as $\text{PoA}(\mathcal{G}) = \sup_{G \in \mathcal{G}} \text{PoA}(G)$. We also consider settings where the subset of agents, $S$, is drawn from $\mathcal{N}$ based on some distribution $P$. In that case, we evaluate the *expected price of anarchy* of $\Xi$ as

$$\text{ExpectedPoA}(\mathcal{G}) = \sup_{M, \mathbf{c} \in C^{|M|}} \left\{ \mathop{\mathbb{E}}_{S \sim P} [\text{PoA}((S, M, \mathbf{c}, \Xi))] \right\}.$$

In other words, given an adversarial choice of machines $M$ using cost functions from $C$, we evaluate the expected PoA over the randomness of $P$ in defining the subset of agents $S$.

**Classes of Cost Functions.** We say that a cost function is *bounded* if $c(\ell + 1)/c(\ell) = O(1)$ for all $\ell > 0$. Another class of functions that plays an important role in prior work is that of *capacitated constant* cost functions. That is, functions such that $c(\ell) = c$ when $\ell \leq t$ and $c(\ell) = \infty$ when $\ell > t$, for some positive constants $c$ and $t$. Note that, although these cost functions are not bounded, one of our first results shows that we can achieve a small PoA for them as well, as long as their capacity, $t$, is at least 4. Finally, a *4-step* function is a step function whose segments have length at least 4. In other words, the value of a 4-step function does not change more than once within any interval of length 4 in its domain. Note that capacitated constant functions with capacity at least 4 are a special case of a 4-step function. Also, it is easy to verify that for any bounded cost function $c'$, there exists a 4-step function $c$ such that $c(\ell) \geq c'(\ell)$ and $c(\ell)/c'(\ell) = O(1)$ for all $\ell > 0$[4]. This means that we can always approximate a bounded cost function using a 4-step function, so in the rest of the paper we assume that the cost functions are all 4-step functions.

**Global Ordering.** Our mechanisms, as well as many mechanisms in the related work (e.g. [8, 9, 24]), use a *global ordering* $\pi$ over the universe $\mathcal{N}$ of players in deciding how to distribute the cost. Although the externality of the users in the games that we study is symmetric (e.g., they all cause the same marginal increase in the cost of a machine), the mechanism needs to share the cost unevenly among them to achieve a good PoA[5]. The global ordering provides a consistent way for the mechanism to differentiate between these users. To ensure that no fairness concerns arise from the asymmetry introduced by these mechanisms, we assume that this global ordering can change periodically in a predetermined way, thus providing a symmetric treatment of the users over time.

## 3 ONLINE SCHEDULING ALGORITHM

The main obstacle that resource-aware mechanisms face in approximating the optimal solution is that they do not know the number $n$ of agents that are present in the system. Since the optimal solution can change radically as a function of $n$, how can the cost-sharing mechanism try to approximate it without knowing the value of $n$?

Rather than trying to implement the optimal assignment as an equilibrium, the main idea behind our solution is to instead implement a much more "well behaved" allocation that, in turn, closely approximates the cost of the optimal assignment. Specifically, we define an online algorithm, called Delayed-OPT, which sequentially assigns jobs to machines using a predetermined order, without knowing the value of $n$. We show that this algorithm has a constant competitive ratio and then we design cost-sharing mechanisms aiming to implement the outcome of this algorithm in equilibrium.

If $A(n)$ is the outcome of the online algorithm and $\text{OPT}(n)$ is the optimal allocation (i.e. the feasible schedule with the minimum social cost) when the total number of jobs is $n$, then the competitive ratio is equal to $\max_n \{C(A(n))/C(\text{OPT}(n))\}$. To simplify the description of the Delayed-OPT

[4]To verify this fact, note that given a bounded function $c'$, we can define a 4-step function $c$ such that for every $k \in \mathbb{N}$, if $\ell \in [4k-3, 4k]$ then $c(\ell) = c'(4k)$. Clearly, $c(\ell) \geq c'(\ell)$ for all $\ell > 0$. Also, since $c'$ is bounded, this means that for every $\ell$ we have $c(\ell)/c'(\ell) \leq c'(\ell+4)/c'(\ell) = O(1)$.

[5]It is well known that the PoA is linear in the number of agents if we share the cost equally [2].

online algorithm, without loss of generality we normalize the costs functions. That is, all costs are multiplied by the same constant such that the minimum non-zero cost is equal to 1.For each $k \in \mathbb{N}$, let $a_k = \max\{q \in \mathbb{N} : C(\mathrm{OPT}(q)) < 2^k\}$ be the largest number of jobs such that the optimal social cost for scheduling these jobs remains less than $2^k$ (Figure 1 shows two examples for capacitated constant cost functions). Using this definition, let $\ell^*_{jk}$ denote the number of jobs assigned to machine $j$ in the optimal allocation when the total number of jobs is $a_k$.

When the $q^{\text{th}}$ job arrives, the Delayed-OPT finds the smallest value of $k$ such that for some machine $j \in M$ the number of jobs, $\ell_j$, assigned to it so far is less than $\ell^*_{jk}$. Then, among all such machines, the algorithm assigns this job to the one that has the smallest index[6]. The algorithm then increments the value of $\ell_j$ by one and moves on to the next job. A formal description of the Delayed-OPT algorithm is provided as Algorithm 1, below, and two examples of the induced assignment are provided in Figure 1.

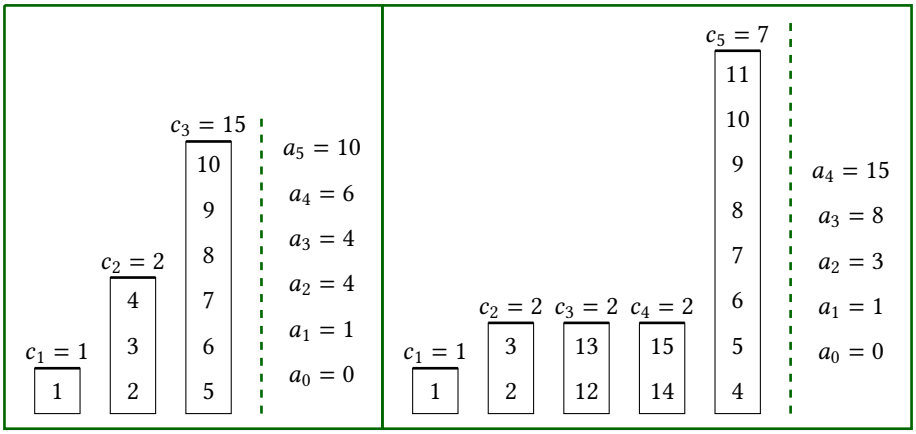


Fig. 1. These figures depict machines with capacitated constant cost functions. Figure (a) shows three machines whose cost is 1, 2, and 15 for any load up to 1, 3, and 6, respectively (and the cost becomes infinite for any load beyond that). Similarly, Figure (b) shows five machines whose cost is 1, 2, 2, 2, and 7 for any load up to 1, 2, 2, 2, and 8, respectively. In both figures, the $a_k$ values are given on the right, and each number inside the machines represents a job with the number indicating the order of their arrival. The figures show how the Delayed-OPT algorithm assigns the jobs to the machines, e.g. the first job is assigned to the first machine in both cases.

Lemma 3.1. *If $q \le a_{k'}$ for some $k' \in \mathbb{N}$, then the value of $k$ computed by the algorithm in the iteration corresponding to the $q^{th}$ job satisfies $k \le k'$.*

Proof. Assume that this is not the case. This would mean that in that iteration of the algorithm, for every machine $j \in M$ we have $\ell_j \ge \ell^*_{jk'}$. Summing over all $j \in M$, this would yield

$$\sum_{j \in M} \ell_j \;\ge\; \sum_{j \in M} \ell^*_{jk'} \;=\; a_{k'}.$$

But, since $q = \sum_{j \in M} \ell_j + 1$, this contradicts the fact that $q \le a_{k'}$. □

[6]We assume that the machines have some arbitrary, but fixed, ordering indicated by their indices.

**ALGORITHM 1:** DELAYED-OPT Online Algorithm

$q \leftarrow 0$ // Initialize counter for the number of jobs
$\ell_j \leftarrow 0$ for each $j \in M$ // Initialize all loads to zero
**while** *there exist more jobs* **do**
  $q \leftarrow q + 1$
  $k \leftarrow \min\{k \in \mathbb{N} \mid \exists j \in M : \ell_j < \ell^*_{jk}\}$
  $j \leftarrow \arg\min\{j \in M \mid \ell_j < \ell^*_{jk}\}$
  $\ell_j \leftarrow \ell_j + 1$ // Assign job to first machine that has not reached target load

We now proceed to show that the competitive ratio of this algorithm is less than 4.

THEOREM 3.2. *The competitive ratio of the DELAYED-OPT algorithm is less than 4.*

PROOF. Let $k \in \mathbb{N}$ be the minimum value such that $n \leq a_k$. The load that the algorithm assigns on any machine $j$ is no more than $\max_{k' \leq k}\{\ell^*_{jk'}\}$. As a result, the cost of the DELAYED-OPT algorithm for $n$ jobs is

$$C(A(n)) \leq \sum_{k' \leq k} C(\mathrm{OPT}(a_{k'})) < \sum_{k' \leq k} 2^{k'} < 2^{k+1},$$

while the optimal cost is $C(\mathrm{OPT}(n)) \geq C(\mathrm{OPT}(a_{k-1} + 1)) \geq 2^{k-1}$, leading to a competitive ratio of less than $2^{k+1}/2^{k-1} = 4$. □

The following lemma will be useful in the next sections.

LEMMA 3.3. *If $c_j(\ell) = c_j(\ell + \ell')$ for some machine $j$ and some loads $\ell, \ell' > 0$, then right after the iteration that the DELAYED-OPT algorithm assigns the $\ell^{th}$ job at machine $j$, it assigns the next $\ell'$ jobs at the same machine.*

PROOF. Suppose that in the iteration that the DELAYED-OPT algorithm assigns the $\ell^{\text{th}}$ job at machine $j$ it computes $k$ to be the smallest value such that there exists a machine $j'$ with $\ell_{j'} < \ell^*_{j'k}$. Since the algorithm assigns the current job to machine $j$, at this iteration $\ell_j < \ell^*_{jk}$ and $j$ has the smallest index among machines that satisfy this inequality.

Moreover, since the cost functions are all non-decreasing, the cost of machine $j$ is the same for all loads between $\ell$ and $\ell + \ell'$, which means that $\ell^*_{jk} \geq \ell + \ell'$. To better see this suppose on the contrary that $\ell^*_{jk} < \ell + \ell'$. The allocation that assigns another job to machine $j$ has the same cost with current optimal allocation, i.e. $C(\mathrm{OPT}(a_k)) = C(\mathrm{OPT}(a_k + 1))$. This is a contradiction to the definition of $a_k$ that needs to satisfy that $C(\mathrm{OPT}(a_k)) < C(\mathrm{OPT}(a_k + 1))$.

Overall, in the next iteration, $\ell_j < \ell^*_{jk}$ and $j$ should be the smallest index that satisfies this inequality, otherwise this wouldn't be true in the previous iteration. Therefore, the DELAYED-OPT algorithm assigns the next job to machine $j$ and by induction it should assign all the following jobs until the load of machine $j$ becomes $\ell + \ell'$. Figure 1 shows such examples. □

## 4 RESOURCE-AWARE MECHANISM FOR GAMES WITH TWO KNOWN USERS

In this section we consider resource-aware mechanisms that are oblivious to the set of users in the system, with the exception of just two users. Formally, we consider classes of games such that for every game $G$ in this class, the set of agents, $S$, always contains two known agents. Note that the set $S$ is otherwise totally unrestricted and can also contain an adversarially chosen subset of the agents from $\mathcal{N}$, so this class of games is quite general. In fact, since the optimal allocation may very heavily depend on the total number of agents that participate in the game, the aforementioned

restriction is seemingly benign. In what follows, we propose a resource-aware mechanism that assigns a special role to the two known agents, leading to very efficient equilibria for any bounded cost function. In fact we show that the assignment of every Nash equilibrium in the induced game is the same as the outcome of the Delayed-OPT algorithm when scheduling $|S|$ jobs.

As a warm-up, we first consider the games whose cost functions are drawn from the class of capacitated cost functions, and then we go on to extend our result beyond this class.

### 4.1 Warm-up: A Class of Capacitated Constant Functions

In order to more clearly capture the intuition behind how our proposed mechanism works, we first focus on games whose cost functions are capacitated constant, with a capacity of at least 4. That is, for every machine $j$ we have $c_j(\ell) = c_j$ when $\ell \leq t_j$ and $c_j(\ell) = \infty$ otherwise, where $c_j > 0$ and $t_j \geq 4$ are constants[7]. Note that these cost functions are actually not bounded, since they jump from some constant to infinity when their capacity is exceeded, so this section also shows that our positive results can even be extended to cost functions beyond the class of bounded ones.

Before presenting our protocol, we make an important observation, that can be derived directly from Lemma 3.3, regarding the allocation of the Delayed-OPT algorithm when the machines have capacitated constant cost functions.

Observation 1. *For any instance involving a set $M$ of machines with capacitated constant cost functions, there exists an ordering of the machines in $M$ such that the Delayed-OPT algorithm fills up machine $j$ up to its capacity before assigning any job to any machine $j'$ that is later in the ordering.*

Let $D = \{1, 2\}$ be the set of the two agents, called *enforcers*, who are guaranteed to participate, and let $R = S \setminus D$ be the rest of the agents, which we call *regular* agents. Also, given some set of agents $S'$, let $h(S')$ be the first (highest priority) agent in $S'$ according to a global ordering $\pi$. For simplicity we assume that the machines are renamed according to the ordering implied by the Delayed-OPT algorithm (Observation 1), and let $Z_j = \sum_{k \leq j} c_k$ be the sum of the costs of the first $j$ machines in this ordering. Note that the value of $Z_j$ is *strictly* increasing with $j$ by our convention that $c_j > 0$ for all $j$. Finally, to define the protocol we also use an arbitrarily small positive value $\varepsilon_j$ for each machine $j$ to be used as a special charge for enforcers in some cases; $\varepsilon_j$ values are strictly decreasing values, i.e. $\varepsilon_j > \varepsilon_{j+1}$.

**Brief description of the protocol.** The enforcers are charged with the small value $\varepsilon_j$ for using machine $j$ only in two cases: i) if they are together in $j$ along with at least one regular agent (if there were no regular agent, the enforcers should cover the cost of the machine) and the load of machine $j$ doesn't exceed its capacity $t_j$, ii) if the enforcer is alone in $j$ and the load of machine $j$ exceeds its capacity $t_j$. In any other case they pay $Z_j$. Regarding the regular agents, the highest priority regular agent always pays a non-zero charge. More specifically, if machine $j$'s load doesn't exceed $t_j$, the highest priority regular agent pays the cost of $j$, $c_j$, if the machine is full (i.e. its load equals $t_j$) and there is no enforcer in $j$; otherwise, meaning when $j$'s load is less than $t_j$ or there is an enforcer in $j$, the highest priority regular agent pays $Z_j$. The rest of the regular agents are charged with 0 if $j$'s load doesn't exceed $t_j$. If $j$'s load exceeds $t_j$, then everybody is charged with infinity.

[7]The assumption of $c_j > 0$ for all $j$ is w.l.o.g. because if there are machines with zero cost, we may charge everybody with 0, unless the machine load exceeds its capacity, in which case everybody is charged with infinity. In both the Delayed-OPT algorithm and any Nash equilibrium those machines are firstly occupied up to their capacity and then other machines are used resulting in a PoA equal to the one that ignores those machines.

*Protocol.* Given a strategy profile $\mathbf{s}$, the cost share of any enforcer $i \in D$ for using machine $j$ is

$$\xi_i(\mathbf{s}) = \begin{cases} \varepsilon_j & \text{if } \ell_j(\mathbf{s}) \leq t_j \text{ and } D \subset S_j(\mathbf{s}) \\ \varepsilon_j & \text{if } \ell_j(\mathbf{s}) > t_j \text{ and } D \cap S_j(\mathbf{s}) = \{i\} \\ Z_j & \text{otherwise.} \end{cases}$$

The cost share of any regular agent $i \in R$ for using machine $j$ is

$$\xi_i(\mathbf{s}) = \begin{cases} 0 & \text{if } \ell_j(\mathbf{s}) \leq t_j \text{ and } i \neq h(S_j(\mathbf{s}) \cap R) \\ c_j & \text{if } \ell_j(\mathbf{s}) = t_j, D \cap S_j(\mathbf{s}) = \emptyset \text{ and } i = h(S_j(\mathbf{s}) \cap R) \\ Z_j & \text{if } \ell_j(\mathbf{s}) = t_j, D \cap S_j(\mathbf{s}) \neq \emptyset \text{ and } i = h(S_j(\mathbf{s}) \cap R) \\ Z_j & \text{if } \ell_j(\mathbf{s}) < t_j \text{ and } i = h(S_j(\mathbf{s}) \cap R) \\ \infty & \text{otherwise.} \end{cases}$$

The main idea behind this protocol is that in the equilibrium if agents use some machine, all machines with lower indices should be full (i.e. its load equals its capacity). As we mentioned above, $Z_j$ values are strictly increasing. As a result if some agent is charged $Z_j$ in machine $j$, she prefers to deviate to a non-full machine (non-full means that its load is less that its capacity) with smaller index. Such an agent exists when the machine is not full or when an enforcer is using it. However, there is no such agent when the machine is full with only regular agents, where the importance of enforcers comes in place as we explain next. We note here that it is crucial to keep the budget balance in full machines without enforcers so that we do not lose in efficiency too much.

If a machine that is not used by the Delayed-OPT algorithm is full with only regular agents, enforcers are going to disrupt them and push them to machines with lower indices. The reason is because $\varepsilon_j$ values are decreasing, so enforcers prefer to occupy machines with higher indices. So, if an enforcer deviates to a full machine $j$, the load of that machine will exceed capacity and the enforcer will be charged $\varepsilon_j$.

The cases where the charges of the enforcers are high ($Z_j$) are crucial in order to guarantee stability as we show in Theorem 4.2.

Theorem 4.1. *The PoA for the class of capacitated constant cost functions, assuming two enforcers, is constant.*

Proof. It is sufficient to show that the social cost of any Nash equilibrium is constant away from the cost induced by the Delayed-OPT algorithm, which in turn is constant away from the cost of the optimal allocation.

In fact we show that under any pure Nash equilibrium, the allocation is the same with the outcome of the Delayed-OPT algorithm; that is for any used machine $r$, all prior machines $j < r$ are fully used. Then, it is easy to check that, regarding the overcharging, each enforcer may "cause" some regular agent to pay at most the cost of the outcome of the Delayed-OPT algorithm and each enforcer itself may pay some arbitrarily small value $\varepsilon_j$.[8]

For the sake of contradiction suppose that in some Nash equilibrium there exist machines $j < r$ such that machine $r$ is used and machine $j$ is not full. Let $r$ be the largest possible such index.

- If $r$ is not full, or if it is full and has at least one enforcer, there exists an agent paying $Z_r$ and if he deviates to $j$ he should pay at most $Z_j < Z_r$, so he has an incentive to deviate (Figure 2).
- If $r$ is full with only regular agents, there exists an enforcer in an earlier machine $j' < r$ paying at least $\varepsilon_{j'}$. That enforcer has an incentive to deviate to $r$ where he will pay $\varepsilon_r < \varepsilon_{j'}$ (Figure 3).

[8]There is no Nash equilibrium where the enforcers are charged more than some $\varepsilon_j$, unless there is no regular agent where we again have the same overcharging.

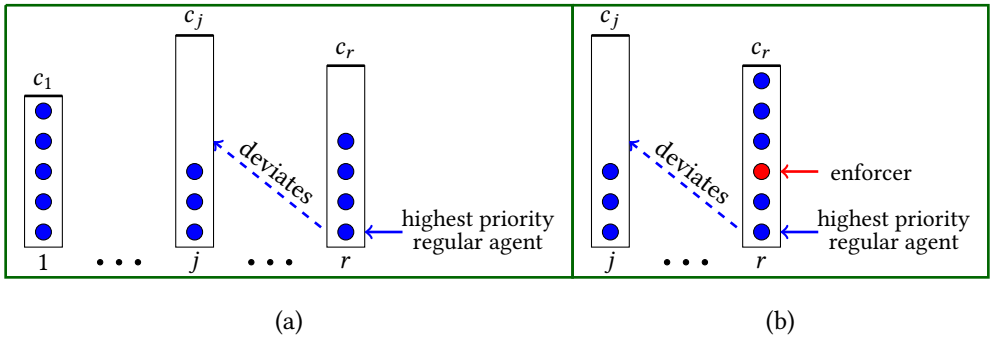


Fig. 2. In this figure we assume that machine $j$ is not full and there exists a non empty machine $r$, with $r > j$ (where $r$ is the maximum such index). If machine $r$ is either not full (a) or has an enforcer (b), then there is always a regular agent from $r$ that prefers to deviate to $j$.

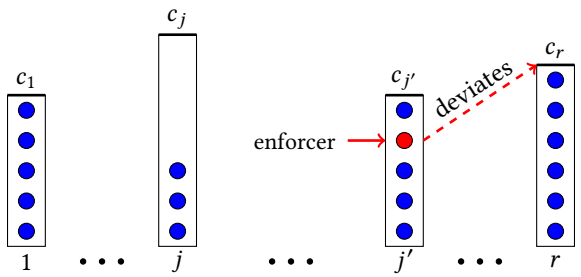


Fig. 3. In this figure we assume that machine $j$ is not full and there exists a non empty machine $r$, with $r > j$ (where $r$ is the maximum such index). If machine $r$ is full with only regular agents, then any enforcer prefers to deviate to $r$.

In both cases there exists an agent with an incentive to deviate to another machine which is a contradiction to our assumption that this is a Nash equilibrium.

□

Theorem 4.2. *The protocol for the class of capacitated constant cost functions, assuming two enforcers, is stable.*

Proof. In order to show stability, we create a strategy profile that is an equilibrium for any set of agents $S \subseteq \mathcal{N}$ as long as $D \subseteq S$.

Let $n$ be the number of agents in the system, where $n-2$ of them are regular agents, since there exist two enforcers. Suppose that $r$ machines are occupied based on the Delayed-OPT algorithm, with the first $r-1$ machines being fully occupied and machine $r$ having $n_r \leq t_r$ agents. The strategy profile we create depends on the value of $n_r$.

*Case of* $n_r \leq 2$. In this case, we create a strategy profile where the enforcers use the last full machine $r-1$ (unless $r = 1$, meaning that there is no regular agent, and the enforcers use machine 1 which is a Nash equilibrium). The regular agents are placed according to the outcome of the Delayed-OPT algorithm such that in the last machine $r$ the *lowest* priority agents are placed (Figure 4 (a)). Next we show that nobody has an incentive to deviate from this strategy profile and therefore it is stable.

The enforcers are currently charged with $\varepsilon_{r-1}$ and if they deviate to any previous machine $j$ with $j < r-1$ they will be charged with $\varepsilon_j > \varepsilon_{r-1}$. Moreover, if they unilaterally deviate to $r$ they will be charged $Z_r > \varepsilon_{r-1}$ because they will be the only enforcer there. Deviating to any other machine $j$ with $j > r$ will result in an even higher charge, since the enforcer will be alone there. Overall, enforcers have no incentive to deviate.

From the regular agents' perspective, nobody has an incentive to deviate to a full machine $j' < r$ because its load then will exceed its capacity resulting in infinity charges. Additionally, no agent currently located to some machine $j \leq r$ has an incentive to deviate to an empty machine $j' > r$, because he is currently charged at most $Z_j$ and if he deviates to $j'$, he will be charged $Z_{j'} > Z_j$. The last case to check is if an agent currently located to some machine $j < r$, has an incentive to deviate to $r$. Note that if he deviates to $r$, the machine will still not be full and he will be the highest priority agent, as in $r$ we allocated the lowest priority agents; therefore, he will be charged $Z_r > Z_j$, where $Z_j$ is the maximum he may currently be charged.

*Case of $n_r > 2$.* In this case, we create a strategy profile where the enforcers use that last machine $r$. The regular agents are placed according to the outcome of the DELAYED-OPT algorithm such that in the last machine $r$ the *lowest* priority agents are placed (Figure 4 (b)). Similar arguments hold in this case in order to show that nobody has an incentive to deviate from this strategy profile.

More specifically the enforcers are currently charged with $\varepsilon_r$ and any deviation will result in a charge of either $\varepsilon_j$ with $j < r$ or $Z_j$ with $j > r$, which are both strictly greater than $\varepsilon_r$.

Regarding the regular agents, as before, nobody wants to deviate to a full machine or to a machine $j > r$. Any agent currently using some machine $j < r$ is charged with at most $Z_j$ and if he deviated to machine $r$ he would pay at least $Z_r > Z_j$ because he would be the highest priority agent in $r$. □

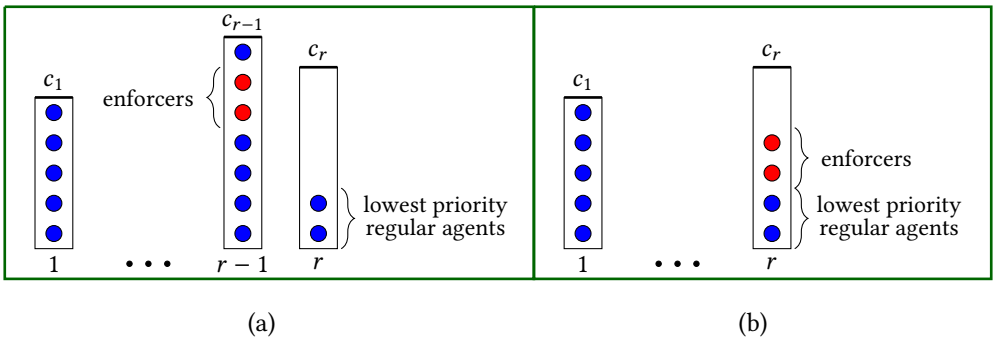


Fig. 4. This figure shows the stable outcomes when the DELAYED-OPT algorithm allocates in the last machine (a) at most two agents and (b) more than two agents.

In the appendix we give some intuition on the need of the restrictions we assume here, namely the need of at least two enforcers and the capacities being at least 4 . Both restrictions are important in order to guarantee stability.

### 4.2 Bounded Cost Functions

We now extend the result of Section 4.1 to the class of bounded cost functions. For simplicity, we focus on the class of 4-step cost functions which naturally generalize the capacitated cost functions considered above; as we discussed in Section 2, any bounded cost function can be approximated by a 4-step cost function, so our results directly extend to bounded cost functions as well.

A key difference between the segments of 4-step functions and the capacitated constant functions is that having a single job in a segment may have two meanings in the respective machines with capacitated constant functions: it may be considered as i) having a single job in the machine with capacitated constant function corresponding to that segment or ii) having an overload in the machine with capacitated constant function corresponding to the previous segment of the step function. In order to overcome this ambiguity we slightly change our protocol in order to handle those two cases consistently and get the same results.

Let agents $D = \{1, 2\}$ be the two agents/enforcers who are guaranteed to participate, and let $R = S \setminus D$ be the *regular* agents. Before describing the protocol we need to give some further definitions; Figure 5 gives some intuition for some of the following definitions.

#### 4.2.1 *Preliminaries.*

We first provide an alternative definition of a 4-step function. In the rest of the paper, we will be assuming that all the machine cost functions are 4-step functions.

*Definition 4.3.* A function $c$ is called *4-step function* if the following are true: there are steps of lengths $t(1), t(2), \cdots \geq 4$ such that for all $k$ and all $x \in [1 + \sum_{k'=1}^{k-1} t(k'), \sum_{k'=1}^{k} t(k')]$ we have that

$$c(x) = c\left(\sum_{k'=1}^{k} t(k')\right) = \tilde{c}(k).$$

That is, the cost function increases only when an extra step needs to be used. If any number of jobs between one and $t(1)$ are undertaken by this machine, the cost is $\tilde{c}(1)$. Then if one more job is added the cost jumps to $\tilde{c}(2)$ and then the cost for $t(1) + 1$ up to $t(1) + t(2)$ jobs remains $\tilde{c}(2)$, and so forth. Note that trivially all functions on natural numbers are 1-step functions.

**Length and cost of a segment.** According to Definition 4.3, we define segment $k$ of machine $j$ to be the $k^{th}$ step of machine $j$'s cost function $c_j$ and has length $t_j(k)$ and cost $\tilde{c}_j(k)$.

**Last used segment $\varkappa_j(\ell_j(\mathbf{s})) = \varkappa_j(\mathbf{s})$.** For each machine $j$ and profile $\mathbf{s}$, we denote by $\varkappa_j(\mathbf{s})$ the last segment that is used in machine $j$ under $\mathbf{s}$. It holds that $c_j(\mathbf{s}) = \tilde{c}_j(\varkappa_j(\mathbf{s}))$.

**Machine's excess $w_j(\ell_j(\mathbf{s})) = w_j(\mathbf{s})$.** For each machine $j$ and profile $\mathbf{s}$, we denote by $w_j(\mathbf{s})$ the number of jobs occupying the last segment of machine $j$ under $\mathbf{s}$ if that segment is not filled to capacity. If the number of jobs fill the last segment to capacity then we set $w_j(\mathbf{s})$ to 0. More formally,

$$w_j(\mathbf{s}) = \begin{cases} \ell_j(\mathbf{s}) - \sum_{k=1}^{\varkappa_j(\mathbf{s})-1} t_j(k) & \text{if } \ell_j(\mathbf{s}) > \sum_{k=1}^{\varkappa_j(\mathbf{s})-1} t_j(k) \\ 0 & \text{otherwise} \end{cases}$$

**Segment order $\phi$.** Lemma 3.3 implies that the Delayed-OPT algorithm fills up a segment up to its capacity before assigning any job to any other segment. Therefore, a priority order on the segments can be derived according to the Delayed-OPT algorithm that assigns for every machine $j$ and segment $k$ a number $\phi_j(k)$. The function $\phi$ respects the order of the machine step costs, i.e. it is strictly monotone.

**Definition of $Z_j(k)$.** Similarly to the case of capacitated constant functions, we define

$$Z_j(k) = \sum_{j'} \max_{k' : \phi_{j'}(k') \leq \phi_j(k)} \tilde{c}_{j'}(k'),$$

which is the aggregate cost of all machines if all segments up to $\phi_j(k)$ in the priority order are occupied. W.l.o.g. the $Z_j(k)$ values are *strictly* increasing according to the order defined by $\phi$. This is by assuming non-zero costs which is w.l.o.g. according to footnote 7, and additionally if two

consecutive segments have the same cost, we may assume that they are merged into a single segment.

**Definition of $\varepsilon_j(k)$.** We also use an arbitrarily small positive value $\varepsilon_j(k)$ for each segment $k$ of machine $j$ to be used as a special charge for enforcers in some cases; $\varepsilon_j(k)$ values are strictly decreasing values according to the order $\phi$, i.e. if $\phi_j(k) > \phi_{j'}(k')$ then $\varepsilon_j(k) < \varepsilon_{j'}(k')$.

**First two machines.** We further distinguish two machines 1 and 2 to be the first and the second machines, respectively, to be used by the DELAYED-OPT algorithm.

**Highest priority agents $h(S')$.** Given some set of agents $S'$, $h_i(S')$ is the $i^{th}$ agent in $S'$ according to the global ordering $\pi$.

*Protocol.* Given a strategy profile $\mathbf{s}$ we next define the cost shares of the agents. For simplicity, we drop the dependency on the load and on $\mathbf{s}$ since there is no ambiguity. The cost share of any enforcer $i \in D$ using machine $j$ is

$$\xi_i(\mathbf{s}) = \begin{cases} \varepsilon_j(\varkappa_j) & \text{if } w_j \neq 1 \text{ and } D \subset S_j \\ \varepsilon_j(\varkappa_j - 1) & \text{if } w_j = 1, D \cap S_j = \{i\} \text{ and } \varkappa_j > 1 \\ Z_j(\varkappa_j) & \text{otherwise.} \end{cases}$$

The cost share of any regular agent $i \in R$ using machine $j$ is

$$\xi_i(\mathbf{s}) = \begin{cases} c_j & \text{if } w_j = 0, D \cap S_j = \emptyset \text{ and } i = h_1(S_j \cap R) \\ Z_j(\varkappa_j) & \text{if } w_j = 0, D \cap S_j \neq \emptyset \text{ and } i = h_1(S_j \cap R) \\ Z_j(\varkappa_j) & \text{if } w_j = 1 \text{ and } i \in \{h_1(S_j \cap R), h_2(S_j \cap R)\} \\ Z_j(\varkappa_j) & \text{if } w_j \notin \{0, 1\} \text{ and } i = h_1(S_j \cap R) \\ 0 & \text{otherwise.} \end{cases}$$

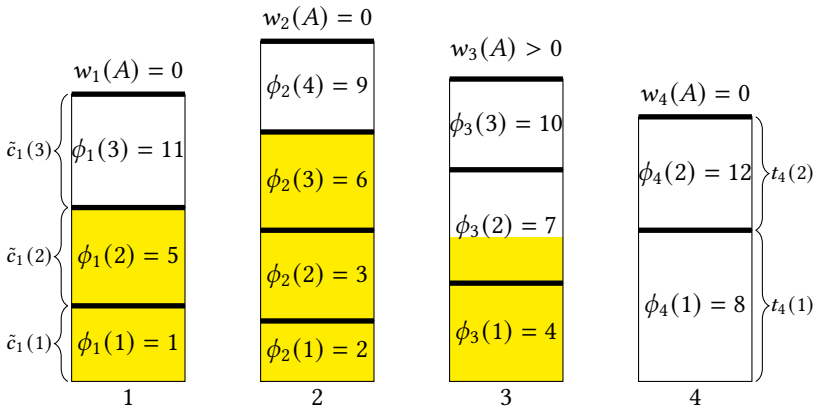


Fig. 5. This figure shows an example of the first four machines in the order that are used by the DELAYED-OPT algorithm. The cost functions belong to the class of 4-step functions. In the figure, $\tilde{c}_1(k)$ is the cost of machine 1 when the $k^{th}$ step/segment is used but not the $(k+1)^{th}$, and $t_4(k)$ is the length of the $k^{th}$ step/segment of machine 4. The $\phi_j(k)$ values show the order that the DELAYED-OPT algorithm fills the segments. In this example, the allocation $A$ of the DELAYED-OPT algorithm fully uses the first 6 segments and also part of the $7^{th}$ segment. The excess of all machines but the third are 0 and machine 3 has positive excess since its $2^{nd}$ segment is not fully used.

Theorem 4.4. *The PoA for the class of 4-step cost functions, assuming two enforcers, is constant.*

Proof. We will show that the total cost of any induced pure Nash equilibrium, assuming *two* enforcers, is constant away the total cost induced by the Delayed-OPT algorithm which in turns is a constant approximation to the cost of the optimal allocation. In order to show this, we will show that any pure Nash equilibrium $\mathbf{s}$ has the same allocation with the Delayed-OPT algorithm allocation $A$. This means that for any machine $j$ the load in $\mathbf{s}$ and $A$ are the same, i.e. $\ell_j(\mathbf{s}) = \ell_j(A)$.

Claim 1. *For any Nash equilibrium* $\mathbf{s}$, $\ell_j(\mathbf{s}) = \ell_j(A)$ *for all* $j$.

Proof. For the sake of contradiction suppose that there exists some Nash equilibrium $\mathbf{s}$ with different allocation than $A$. Then there should be a machine $r$ with $\ell_r(\mathbf{s}) > \ell_r(A)$. If there are many machines with more load in $\mathbf{s}$ than in $A$, we choose $r$ to be the one with the maximum $\phi_r(\varkappa_r(\mathbf{s}))$.

For any machine $j$, with $\ell_j(\mathbf{s}) \leq \ell_j(A)$, it holds that the last segment of machine $j$ under $A$ precedes the last segment of machine $r$ under $\mathbf{s}$ according to segment order $\phi$, i.e. $\phi_j(\varkappa_j(A)) < \phi_r(\varkappa_r(\mathbf{s}))$, meaning that overall $\phi_r(\varkappa_r(\mathbf{s}))$ is the maximum among used segments under $\mathbf{s}$. The reason is that, if $l$ is the last machine used by the Delayed-OPT algorithm, the excess of all other machines different than $l$ is 0 under $A$, and therefore if $r \neq l$, $\varkappa_r(\mathbf{s})$ is not used in $A$; by the definition of $\phi$ order, $\phi_j(\varkappa_j(A)) < \phi_r(\varkappa_r(A) + 1) \leq \phi_r(\varkappa_r(\mathbf{s}))$. If $r = l$, it trivially holds that $\phi_j(\varkappa_j(A)) < \phi_r(\varkappa_r(A)) \leq \phi_r(\varkappa_r(\mathbf{s}))$.

Next we show that under $\mathbf{s}$ either a regular agent or an enforcer has an incentive to deviate leading to a contradiction.

- If there is at least one enforcer in machine $r$, or $\varkappa_r(\mathbf{s})$ is not full, i.e. $w_r(\mathbf{s}) \neq 0$, the highest priority regular agent in $r$, $h_1(S_r(\mathbf{s}) \cap R)$, is paying $Z_r(\varkappa_r(\mathbf{s}))$. If this agent deviated to any machine $j$, with $\ell_j(\mathbf{s}) < \ell_j(A)$ (there exists at least one because $\ell_r(\mathbf{s}) > \ell_r(A)$), the total load on that machine would be at most $\ell_j(A)$ and therefore the agent's payment would be at most $Z_j(\varkappa_j(A)) < Z_r(\varkappa_r(\mathbf{s}))$.
- If machine $r$ has only regular agents and 0 excess, i.e. $w_r(\mathbf{s}) = 0$, there exists an enforcer in some machine $j' \neq r$ that is charged with at least $\varepsilon_{j'}(\varkappa_{j'}(\mathbf{s}))$. If he deviated to machine $r$, the excess of that machine would become 1 and he would be the only enforcer in machine $r$, therefore, he would be charged with $\varepsilon_r(\varkappa_r(\mathbf{s})) < \varepsilon_{j'}(\varkappa_{j'}(\mathbf{s}))$, where the inequality holds because $\phi_r(\varkappa_r(\mathbf{s}))$ is the maximum among used segments under $\mathbf{s}$.

□

□

Theorem 4.5. *The protocol for the class of 4-step cost functions, assuming two enforcers, is stable.*

Due to space limitations we refer the reader to the appendix for the proof of the theorem.

## 5 RESOURCE-AWARE MECHANISM FOR GAMES WITH STOCHASTIC ARRIVALS

In this section we study the case of stochastic arrivals, where each agent $i$ appears in the system with probability $p_i$ and the mechanism has access to $\mathbf{p} = (p_1, p_2, \ldots, p_{|N|})$. Let $S$ be the random set of the arriving agents and $M$ be a set of machines whose cost functions are from the class of 4-step functions. We design a cost sharing scheme with the goal of minimizing the expected price of anarchy defined as follows:

$$\text{ExpectedPoA}(\mathcal{G}) = \sup_{M, \mathbf{c} \in C^{|M|}} \left\{ \mathop{\mathbb{E}}_{S \sim P} [\text{PoA}((S, M, \mathbf{c}, \Xi))] \right\}.$$

Our main theorem (Theorem 5.4) bounds the expected price of anarchy of our protocol in relation to the expected number of arriving agents $\tilde{n} = \mathbb{E}_{S \sim \mathbf{p}}[|S|]$. For the sake of simplicity in this section we

prove the case of identical agents that is $p_i = p$ for all $i$. The proof for the general case (Theorem 5.6) can be found in the appendix. We show that for the case of independently arriving agents there exists a protocol using $3 + \lfloor 3\frac{\log(p|\mathcal{N}|)}{-\log(1-p)} \rfloor$ enforcers that achieves an expected price of anarchy of at least

$$\text{ExpectedPoA}(\mathcal{G}) = O\left(\log\left(\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[|S|]\right)\right) = O(\log \tilde{n})\,.$$

*Protocol.* The protocol is similar to the one we defined with the guaranteed enforcers. However rather than using the two guaranteed agents as the enforcers we choose an appropriate set of enforcers using the distributional information we have. In order to guarantee stability for any number of enforcers we adjust the cost sharing protocol by adding two rest points for enforcers where they pay 0 share; we further slightly modify the cost shares of enforcers to include cases where many enforcers use the same machine.

Given the set of arriving agents $S$ and a strategy profile $\mathbf{s}$ we next define the cost shares of the agents. Let $D \subseteq S$ be the set of enforcers in $S$ and $R = S \setminus D$ be the set of regular agents in $S$. For simplicity, we drop the dependency on the load and on $\mathbf{s}$ since there is no ambiguity. The cost share of any enforcer $i \in D$ using machine $j$ is

$$\xi_i(\mathbf{s}) = \begin{cases} 0 & \text{if } j \in \{1,2\}, w_j \in \{0, t_j(\varkappa_j) - 1\} \text{ and } D \cap S_j = \{i\} \\ \varepsilon_j(\varkappa_j) & \text{if } w_j \neq 1, D \cap S_j \supset \{i\}, R \cap S_j \neq \emptyset \text{ and } i \in \{h_1(S_j \cap D), h_2(S_j \cap D)\} \\ \varepsilon_j(\varkappa_j - 1) & \text{if } w_j = 1, D \cap S_j = \{i\} \text{ and } \varkappa_j > 1 \\ Z_j(\varkappa_j) & \text{otherwise.} \end{cases}$$

The cost share of any regular agent $i \in R$ using machine $j$ is (the same as in Section 4.2)

$$\xi_i(\mathbf{s}) = \begin{cases} c_j & \text{if } w_j = 0, D \cap S_j = \emptyset \text{ and } i = h_1(S_j \cap R) \\ Z_j(\varkappa_j) & \text{if } w_j = 0, D \cap S_j \neq \emptyset \text{ and } i = h_1(S_j \cap R) \\ Z_j(\varkappa_j) & \text{if } w_j = 1 \text{ and } i \in \{h_1(S_j \cap R), h_2(S_j \cap R)\} \\ Z_j(\varkappa_j) & \text{if } w_j \notin \{0,1\} \text{ and } i = h_1(S_j \cap R) \\ 0 & \text{otherwise.} \end{cases}$$

We refer the reader to the appendix for the proof of the following theorem.

Theorem 5.1. *The protocol for the class of 4-step cost functions is stable for any number of enforcers.*

Next we continue with upper bounding the expected price of anarchy of our protocol. First we prove two important lemmas on how far the cost of any Nash equilibrium may be from the cost of the allocation $A$ of the Delayed-OPT algorithm, conditioned on the number of enforcers in the system. We distinguish two cases of having at least three enforcers or at most two enforcers in the system. In the first case, the proof is similar to the one of Theorem 4.4, but we now need at least three enforcers because based on the protocol at most two enforcers may pay 0 cost shares; those enforcers have no incentive to deviate to machines that are not used in $A$ and are full with regular agents.

Lemma 5.2. *If $d \geq 3$ enforcers arrive then the cost of the Nash equilibrium is no more than $d + 3$ times the cost of the allocation $A$ of the Delayed-OPT algorithm, by ignoring the arbitrarily small charges of $\varepsilon_j(k)$ values.*

Proof. Since we have two rest points for the enforcers (first case of the cost shares), meaning that at most two enforcers may pay 0 in any Nash equilibrium $\mathbf{s}$, if $d \geq 3$ enforcers appear in the system, then at least one of them must pay a non-zero share. Following the proof of Theorem 4.4

we can easily infer that $\mathbf{s}$ uses the exact same allocation as the Delayed-OPT algorithm (Claim 1). Next we need to bound the overcharging cost. Note that any cost share $Z_j(\varkappa_j(A))$ for some $j$ is no more than the cost of $A$. As a result we simply need to bound the number of agents charged with such a cost share. Let $d_j$ be the number of enforcers in machine $j$.

First, we examine the machines other than the last machine used by the Delayed-OPT algorithm. By definition, such machine $j$ will have zero excess, $w_j(A) = 0$. Therefore, if there are only regular agents there will be no overcharging. If there are only enforcers the overcharging is $d_j Z_j(\varkappa_j(A))$ which is at most $d_j$ times the total cost of $A$. If there is at least one enforcer and at least one regular agent, then we have at most one regular agent paying $Z_j(\varkappa_j(A))$ and either one enforcer is paying 0 or two enforcers are paying the arbitrarily small value $\varepsilon_j(\varkappa_j(A))$. In any case, the overcharging is at most $d_j$ times the total cost of $A$, by ignoring the $\varepsilon_j(\varkappa_j(A))$ values.

Second, let's consider the last machine $r$ used by the Delayed-OPT algorithm. At most two regular agents are charged with $Z_r(\varkappa_r(A))$ (if $w_r(A) = 1$). It is also possible that all the enforcers are charged with $Z_r(\varkappa_r(A))$ and therefore, the total overcharging in machine $r$ is at most $d_r + 2$ times the total cost of $A$, by ignoring again the $\varepsilon_r(\varkappa_r(A))$ values.

Overall, the overcharging is at most $d + 2$ times the total cost of $A$ and as a result, the total cost of $\mathbf{s}$ is no more than $d + 3$ times the total cost of $A$. □

Lemma 5.3. *If $r$ regular agents and $d \leq 2$ enforcers arrive then the cost of the Nash equilibrium $\mathbf{s}$ is no more than $r + d$ times the cost of the allocation $A$ of the Delayed-OPT algorithm, by ignoring the arbitrarily small charges of $\varepsilon_j(k)$ values.*

Proof. If the allocation of $\mathbf{s}$ is not the same as $A$, there must be some machine $j$ such that $\ell_j(\mathbf{s}) < \ell_j(A)$. This implies that any agent can deviate to this machine and pay at most $Z_j(\varkappa_j(A))$. As a result the cost of any agent (enforcer or regular) under $\mathbf{s}$ is no more than $Z_j(\varkappa_j(A))$ which is upper bounded by the total cost of $A$. Therefore, the total cost of $\mathbf{s}$ is no more than $r + d$ times the total cost of $A$. □

Next we upper bound the price of anarchy in the special case where the agents arrival probabilities are identical, that is $p_i = p$ for all $i \in \mathcal{N}$. Note that in this case the expected number of agents is $\tilde{n} = E_{S\sim\mathbf{p}}[|S|] = p|\mathcal{N}|$.

Theorem 5.4. *For independently arriving agents with identical probabilities $p$ and for the class of 4-step cost functions, there exists a protocol using $|D| = 3 + \lfloor 3\frac{\log(p|\mathcal{N}|)}{-\log(1-p)} \rfloor$ enforcers that achieves an expected price of anarchy of*

$$ExpectedPoA(\mathcal{G}) = O(\log(\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[|S|])) = O(\log \tilde{n})\,.$$

Before proceeding with the proof of Theorem 5.4, we state the following lemma. The proof of the lemma is in the appendix.

Lemma 5.5. *If we choose a set of enforcers $D$ such that $|D| = 3 + \lfloor 3\frac{\log(p|\mathcal{N}|)}{-\log(1-p)} \rfloor$ then*

$$\mathbb{P}[d \leq 2] \mathop{\mathbb{E}}_{S\sim\mathbf{p}}[|S| \mid d \leq 2] \leq 9\,,$$

*where $\mathbb{P}$ is the probability symbol and $d = |S \cap D|$ is a random variable depending on $\mathbf{p}$.*

Proof. (Theorem 5.4) Let $S$ be a random set of arriving agents, $G$ be the corresponding game, $Eq(G)$ be the set of Nash equilibria for $G$ and $A$ be the allocation of the Delayed-OPT algorithm for the set $S$. Moreover, let $d = |S \cap D|$ be a random variable depending on $\mathbf{p}$. Combining both Lemmas 5.2 and 5.3 we get that the expected ratio of the cost of the worst case equilibrium to the cost of the allocation of the Delayed-OPT algorithm is

$$\mathop{\mathbb{E}}_{S\sim\mathbf{p}}\left[\frac{\max_{\mathbf{s}\in Eq(G)}\hat{C}(\mathbf{s})}{C(A)}\right] = \mathbb{P}[d\le 2]\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[|S| \mid d\le 2] + \mathbb{P}[d\ge 3]\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[d+3 \mid d\ge 3]\,. \tag{1}$$

We can bound the second summand of Equation (1) as

$$\begin{aligned}\mathbb{P}[d\ge 3]\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[d+3 \mid d\ge 3] &\le \mathop{\mathbb{E}}_{S\sim\mathbf{p}}[d+3 \mid d\ge 3]\\ &\le \mathop{\mathbb{E}}_{S\sim\mathbf{p}}[d \mid d\ge 0] + 6 = p|D| + 6\,.\end{aligned} \tag{2}$$

Combining Lemma 5.5, Equation (1) and Equation (2) we get that

$$\begin{aligned}\mathop{\mathbb{E}}_{S\sim\mathbf{p}}\left[\frac{\max_{\mathbf{s}\in E(G)}\hat{C}(\mathbf{s})}{C(A)}\right] &\le p|D| + 15 \le \lfloor 3\log(p|\mathcal{N}|)\left(\frac{p}{-\log(1-p)}\right)\rfloor + 18\\ &\le 3\log(p|\mathcal{N}|)\left(\frac{p}{-\log(1-p)}\right) + 18\\ &\le 3\log(p|\mathcal{N}|) + 18 = 3\log\tilde{n} + 18\,,\end{aligned} \tag{3}$$

where the last inequality is due to $(\frac{p}{-\log(1-p)}) \le 1$ for $p \ge 0$. The fact that the cost of the DELAYED-OPT algorithm outcome is a constant approximation to the optimum cost completes the proof. □

Next we upper bound the price of anarchy when the agents arrival probabilities are not necessarily identical. For $p_i$ being the probability that agent $i$ arrives, $\tilde{n} = E_{S\sim\mathbf{p}}[|S|] = \sum_{i=1}^{|\mathcal{N}|} p_i$ is the expected number of agents. The proof of the theorem is in the appendix.

THEOREM 5.6. *For independently arriving agents with not necessarily identical probabilities and for the class of 4-step cost functions, there exists a protocol that achieves an expected price of anarchy of*

$$ExpectedPoA(\mathcal{G}) = O(\log(\mathop{\mathbb{E}}_{S\sim\mathbf{p}}[|S|])) = O(\log\tilde{n})\,.$$

## ACKNOWLEDGMENTS

The authors would like to thank Giorgos Christodoulou for helpful discussions during the initial stages of this project. The work of the first author was partially supported by NSF CAREER award CCF-2047907.

## REFERENCES

[1] Fidaa Abed and Chien-Chung Huang. 2012. Preemptive coordination mechanisms for unrelated machines. In *European Symposium on Algorithms*. Springer, 12–23.

[2] Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. 2008. The Price of Stability for Network Design with Fair Cost Allocation. *SIAM J. Comput.* 38, 4 (2008), 1602–1623.

[3] Yossi Azar, Lisa Fleischer, Kamal Jain, Vahab S. Mirrokni, and Zoya Svitkina. 2015. Optimal Coordination Mechanisms for Unrelated Machine Scheduling. *Operations Research* 63, 3 (2015), 489–500.

[4] Sayan Bhattacharya, Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2014. Coordination mechanisms from (almost) all scheduling policies. In *5th conference on Innovations in theoretical computer science*. ACM, 121–134.

[5] Ioannis Caragiannis. 2013. Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica* 66, 3 (2013), 512–540.

[6] Ioannis Caragiannis, Vasilis Gkatzelis, and Cosimo Vinci. 2017. Coordination Mechanisms, Cost-Sharing, and Approximation Algorithms for Scheduling. In *Web and Internet Economics - 13th International Conference, WINE 2017, Bangalore, India, December 17-20, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10660)*, Nikhil R. Devanur and Pinyan Lu (Eds.). Springer, 74–87.

[7] Ho-Lin Chen, Tim Roughgarden, and Gregory Valiant. 2010. Designing network protocols for good equilibria. *SIAM J. Comput.* 39, 5 (2010), 1799–1832.

[8] George Christodoulou, Vasilis Gkatzelis, Mohamad Latifian, and Alkmini Sgouritsa. 2020. Resource-Aware Protocols for Network Cost-Sharing Games. In *EC '20: The 21st ACM Conference on Economics and Computation, Virtual Event, Hungary, July 13-17, 2020*, Péter Biró, Jason D. Hartline, Michael Ostrovsky, and Ariel D. Procaccia (Eds.). ACM, 81–107.

[9] Giorgos Christodoulou, Vasilis Gkatzelis, and Alkmini Sgouritsa. 2017. Cost-Sharing Methods for Scheduling Games under Uncertainty. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*. 441–458.

[10] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. 2009. Coordination mechanisms. *Theor. Comput. Sci.* 410, 36 (2009), 3327–3336.

[11] George Christodoulou, Stefano Leonardi, and Alkmini Sgouritsa. 2019. Designing cost-sharing methods for bayesian games. *Theory of computing systems* 63, 1 (2019), 4–25.

[12] Giorgos Christodoulou, Kurt Mehlhorn, and Evangelia Pyrga. 2014. Improving the price of anarchy for selfish routing via coordination mechanisms. *Algorithmica* 69, 3 (2014), 619–640.

[13] George Christodoulou and Alkmini Sgouritsa. 2019. Designing Networks with Good Equilibria under Uncertainty. *SIAM J. Comput.* 48, 4 (2019), 1364–1396.

[14] Richard Cole, José R. Correa, Vasilis Gkatzelis, Vahab S. Mirrokni, and Neil Olver. 2015. Decentralized utilitarian mechanisms for scheduling games. *Games and Economic Behavior* 92 (2015), 306–326.

[15] Vasilis Gkatzelis, Konstantinos Kollias, and Tim Roughgarden. 2016. Optimal Cost-Sharing in General Resource Selection Games. *Operations Research* 64, 6 (2016), 1230–1238.

[16] Ragavendran Gopalakrishnan, Jason R. Marden, and Adam Wierman. 2014. Potential games are necessary to ensure pure Nash equilibria in cost sharing games. *Mathematics of Operations Research* (2014).

[17] Tobias Harks, Martin Hoefer, Anja Schedel, and Manuel Surek. 2021. Efficient black-box reductions for separable cost sharing. *Mathematics of Operations Research* 46, 1 (2021), 134–158.

[18] Tobias Harks and Konstantin Miller. 2011. The worst-case efficiency of cost sharing methods in resource allocation games. *Operations Research* 59, 6 (2011), 1491–1503.

[19] Tobias Harks and Philipp von Falkenhausen. 2014. Optimal cost sharing for capacitated facility location games. *Eur. J. Oper. Res.* 239, 1 (2014), 187–198.

[20] Nicole Immorlica, Li Erran Li, Vahab S. Mirrokni, and Andreas S. Schulz. 2009. Coordination mechanisms for selfish scheduling. *Theoretical Computer Science* 410, 17 (2009), 1589–1598.

[21] Konstantinos Kollias. 2013. Nonpreemptive coordination mechanisms for identical machines. *Theory of Computing Systems* 53, 3 (2013), 424–440.

[22] Jason R. Marden and Adam Wierman. 2013. Distributed welfare games. *Operations Research* 61, 1 (2013), 155–168.

[23] Damon Mosk-Aoyama and Tim Roughgarden. 2009. Worst-case efficiency analysis of queueing disciplines. In *International Colloquium on Automata, Languages and Programming*. Springer, 546–557.

[24] Hervé Moulin. 1999. Incremental cost sharing: Characterization by coalition strategy-proofness. *Social Choice and Welfare* 16, 2 (1999), 279–320.

[25] Hervé Moulin. 2008. The price of anarchy of serial, average and incremental cost sharing. *Economic Theory* 36, 3 (2008), 379–405.

[26] Hervé Moulin and Scott J. Shenker. 2001. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Economic Theory* 18, 3 (2001), 511–533.

[27] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani (Eds.). 2007. *Algorithmic Game Theory*. Cambridge University Press.

[28] Philipp von Falkenhausen and Tobias Harks. 2013. Optimal Cost Sharing for Resource Selection Games. *Mathematics of Operations Research* 38, 1 (2013), 184–208.