Race Scheduling Games

Shaul Rosner and Tami Tamir

School of computer Science. The Interdisciplinary Center (IDC), Herzliya, Israel.
E-mail: shaul.rosner@post.idc.ac.il, tami@idc.ac.il

Abstract

Job scheduling on parallel machines is a well-studied singleton congestion game. We consider a variant of this game in which the jobs are partitioned into *competition sets*, and the goal of every player is to minimize the completion time of his job *relative to his competitors*. Specifically, the primary goal of a player is to minimize the *rank* of its completion time among his competitors, while minimizing the completion time itself is a secondary objective. This fits environments with strong competition among the participants, in which the relative performance of the players determine their welfare.

We define and study the corresponding *race scheduling game* (RSG). We show that RSGs are significantly different from classical job-scheduling games, and that competition may lead to a poor outcome. In particular, an RSG need not have a pure Nash equilibrium, and best-response dynamics may not converge to a NE even if one exists. We identify several natural classes of games, on identical and on related machines, for which a NE exists and can be computed efficiently, and we present tight bounds on the equilibrium inefficiencies. For some classes we prove convergence of BRD, while for others, even with very limited competition, BRD may loop. Among classes for which a NE is not guaranteed to exist, we distinguish between classes for which, it is tractable or NP-hard to decide if a given instance has a NE.

Striving for stability, we also study the Nashification cost of RSGs, either by adding dummy jobs, or by compensating jobs for having high rank. Our analysis provides insights and initial results for several other congestion and cost-sharing games that have a natural 'race' variant.

1 Introduction

Two men are walking through a forest. Suddenly they see a bear in the distance, running towards them. They start running away. But then one of them stops, takes some running shoes from his bag, and starts putting them on. "What are you doing?" says the other man. "Do you think you will run faster than the bear with those?" "I don't have to run faster than the bear," he says. "I just have to run faster than you."

In job-scheduling applications, jobs are assigned on machines to be processed. Many interesting combinatorial optimization problems arise in this setting, which is a major discipline in operations research. A centralized scheduler should assign the jobs in a way that achieves load balancing, an effective use of the system's resources, or a target quality of service [25].

Many modern systems provide service to multiple strategic users, whose individual payoff is affected by the decisions made by others. As a result, non-cooperative game theory has become

an essential tool in the analysis of job-scheduling applications (see e.g., [23, 2, 13, 16, 4], and a survey in [28]. Job-scheduling is a weighted congestion game [26] with *singleton* strategies, that is, every player selects a single resource (machine).

In traditional analysis of congestion games, the goal of a player is to minimize his cost. We propose a new model denoted *race games* that fits environments with strong competition among the participants. Formally, the players form *competition sets*, and a player's main goal is to do well relative to his competitors. The welfare of a player is not measured by a predefined cost or utility function, but relative to the performance of his competitors. This natural objective arises in many real-life scenarios. For example, in cryptocurrency mining, one needs to be the first miner to build a block. It does not matter how fast a miner builds a block, as long as she is the first to do so. Similarly, when buying event tickets from online vendors, the time spent in the queue is far less important than what tickets are available when it is your turn to buy. Participants' ranking is crucial in numerous additional fields, including auctions with a limited number of winners, where, again, the participants' rank is more important than their actual offer, transplant queues, sport leagues, and even submission of papers to competitive conferences.

In this paper we study the corresponding *race scheduling game* (RSG, for short). We assume that the jobs are partitioned into *competition sets*. The primary goal of a job is to minimize the *rank* of its completion time among its competitors, while minimizing the completion time itself is a secondary objective. As an example, consider a running competition. In order to be qualified for the final, a runner should be faster than other participants in her heat. The runners' ranking is more important than their finish time.

Unfortunately, as we show, even very simple RSGs may not have a NE. We therefore focus on *potentially* more stable instances. In many real-life scenarios, competition is present among agents with similar properties. For example, there is a competition among companies that offer similar services; in sport competitions, the participants are categorized by their sex and age group, or by their weight. Some of our results consider games in which competing players are *homogeneous*. Specifically, we assume that all the jobs in a competition set have the same length.

Our results highlight the differences between RSGs and classical job-scheduling games. We identify classes of instances for which a stable solution exists and can be computed efficiently, we analyze the equilibrium inefficiency, and the convergence of best-response dynamics. We distinguish between different competition structure, and between environments of identical and related machines. In light of our negative results regarding stability existence, we also study the problem of *Nashification*. The goal of Nashification is, given an instance of RSGs, to turn it into an instance that has a stable solution. This is done either by adding dummy jobs, or by compensating jobs for having high rank. We believe that this 'race' model fits many natural scenarios, and should be analyzed for additional congestion and cost-sharing games.

2 Model and Preliminaries

A *race scheduling game* (RSG) is given by $G = \langle \mathcal{J}, \mathcal{M}, \{p(j)\}\ \forall j \in \mathcal{J}, \{d_i\}\ \forall i \in \mathcal{M}, S\rangle$, where $\mathcal{J}$ is a set of n *jobs*, $\mathcal{M}$ is a set of m *machines*, $p(j)$ is the *length* of job j, d_i is the *delay* of machine i, and S is a partition of the jobs into *competition sets*. Specifically, $S = \{S_1, \ldots, S_c\}$ such that $c \leq n$, $\cup_{\ell=1}^{c} S_\ell = \mathcal{J}$, and for all $\ell_1 \neq \ell_2$, we have $S_{\ell_1} \cap S_{\ell_2} = \emptyset$. For every job $j \in S_\ell$,

the other jobs in S_ℓ are denoted the *competitors* of j. Let n_ℓ denote the number of jobs in S_ℓ.

Job j is controlled by Player j whose strategy space is the set of machines $\mathcal{M}$. A profile of a RSG is a schedule $s = \langle s_1, \ldots, s_n \rangle \in \mathcal{M}^n$ describing the machines selected by the players[1]. For a machine $i \in \mathcal{M}$, the *load* on i in s, denoted $L_i(s)$, is the total length of the jobs assigned on machine i in s, that is, $L_i(s) = \sum_{\{j|s_j=i\}} p(j)$. When s is clear from the context, we omit it. It takes $p(j) \cdot d_i$ time-units to process job j on machine i. As common in the study of job-scheduling games, we assume that all the jobs assigned on the same machine are processed in parallel and have the same completion time. Formally, the completion time of job j in the profile s is $C_j = L_{s_j}(s) \cdot d_{s_j}$. Machines are called identical if their delays are equal.

Unlike classical job-scheduling games, in which the goal of a player is to minimize its completion time, in race games, the goal of a player is to do well relative to its competitors. That is, every profile induces a ranking of the players according to their completion time, and the goal of each player is to have a lowest possible rank in its competition set. Formally, for a profile s, let $C^s_{S_\ell} = \langle C^s_{\ell_1}, \ldots, C^s_{\ell_{n_\ell}} \rangle$ be a sorted vector of the completion times of the players in S_ℓ. That is, $C^s_{\ell_1} \leq \ldots \leq C^s_{\ell_{n_\ell}}$, where $C^s_{\ell_1}$ is the minimal completion time of a player from S_ℓ in s, etc.. The *rank* of Player $j \in S_\ell$ in profile s, denoted $rank_j(s)$ is the rank of its completion time in $C^s_{S_\ell}$. If several players in a competition set have the same completion time, then they all have the same rank, which is the corresponding median value. For example, if $n_\ell = 4$ and $C^s_{S_\ell} = (7, 8, 8, 13)$ then the players' ranks are $\langle 1, 2.5, 2.5, 4 \rangle$, and if all players in S_ℓ have the same completion time then they all have rank $(n_\ell+1)/2$. Note that, independent of the profile, $\sum_{j \in S_\ell} rank_j(s) = n_\ell(n_\ell + 1)/2$. For a profile s and a job $j \in S_\ell$, let $N_{low}(j, s)$ be the number of jobs from S_ℓ whose completion time is lower than $C_j(s)$, and let $N_{eq}(j, s)$ be the number of jobs from S_ℓ, whose completion time is $C_j(s)$. Note that for $j \in N_{eq}(j, S)$. We have,

Observation 2.1 $rank_j(s) = N_{low}(j, s) + \frac{1+N_{eq}(j,s)}{2}$.

The primary objective of every player is to minimize its rank. The secondary objective is to minimize its completion time. Formally, Player j prefers profile s' over profile s if $rank_j(s') < rank_j(s)$ or $rank_j(s') = rank_j(s)$ and $C_j(s') < C_j(s)$. Note that classic job-scheduling games are a special case of RSGs in which the competition sets are singletons; thus, for every job j, in every profile, s, we have $rank_j(s) = 1$, and the secondary objective, of minimizing the completion time is the only objective.

A machine i is a *best response* (BR) for Player j if, given the strategies of all other players, j's rank is minimized if it is assigned on machine i. Best-Response Dynamics (BRD) is a local-search method where in each step some player is chosen and plays its best improving deviation (if one exists), given the strategies of the other players.

The focus in game theory is on the *stable* outcomes of a given setting. The most prominent stability concept is that of a Nash equilibrium (NE): a profile such that no player can improve its objective by unilaterally deviating from its current strategy, assuming that the strategies of the other players do not change. Formally, a profile s is a NE if, for every $j \in \mathcal{J}$, s_j is a BR for Player j.

Some of our results consider RSGs with *homogeneous* competition sets. We denote by $\mathcal{G}_h$ the corresponding class of games. Formally, $G \in \mathcal{G}_h$ if, for every $1 \leq \ell \leq c$, all the jobs in

[1] In this paper, we only consider *pure* strategies.

S_ℓ have the same length, p_ℓ. The following example summarizes the model and demonstrates several of the challenges in analyzing RSGs.

Example: Consider a game $G \in \mathcal{G}_h$ on $m = 3$ identical machines, played by $n = 9$ jobs in two homogeneous competition sets. S_1 consists of four jobs having length 4, and S_2 consists of five jobs having length 3 (to be denoted 4-jobs and 3-jobs, respectively). All the machines have the same unit-delay. Fig. 1 presents four profiles of this game. The completion times are given above the machines and the jobs are labeled by their ranks. Consider the jobs of S_2 in Profile (a). Their completion times are $C_{S_2}^{(a)} = (7, 12, 12, 12, 12)$. Thus, the 3-job on M_2 has rank 1, and the four jobs on M_3 all have rank $\frac{2+3+4+5}{4} = 3.5$. We show that Profile (a) is a NE. Consider first deviations of a 4-job from Profile (a): a migration of a 4-job from M_1 to M_2 is not beneficial, as it leads to profile (b) in which the rank of the 4-jobs on M_2 is 3.5. This is higher than 3 – the rank of the migrating job in Profile (a). Other deviations of a 4-job are clearly not beneficial. Consider next deviations of a 3-job. A migration from M_3 to M_2 is not beneficial, as it leads to profile (c) in which the rank of the 3-jobs on M_2 is 4.5. This is higher than 3.5 – the rank of the deviating job in Profile (a). Other deviations of a 3-job are clearly non beneficial. Thus, Profile (a) is a NE. This example demonstrates that race games are significantly different from classical job-scheduling games. In particular, a beneficial migration may increase the completion time of a job. For example, the migration of a 3-job that leads from Profile (c) to Profile (a) increases the completion time of the deviating job from 10 to 12, but reduces its rank from 4.5 to 3.5. Moreover, simple algorithms that are known to produce a NE schedule for job-scheduling games without competition need not produce a NE in race games. In our example, Profile (d) is produced by the *Longest Processing Time* (LPT) rule. It is not a NE since the 3-job on M_1 can reduce its rank from 5 to 4 by migration to either M_2 or M_3.

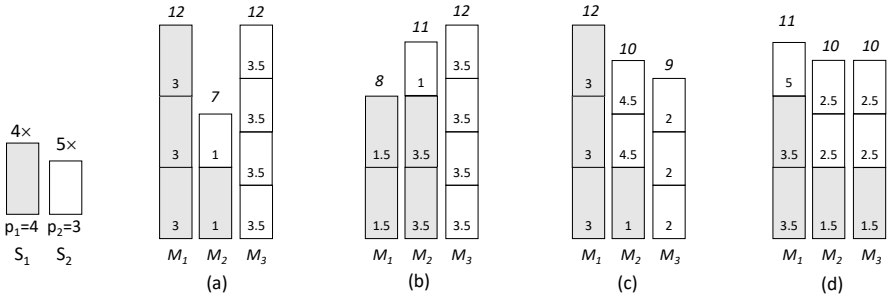

Figure 1: Jobs are labeled by their ranks. (a) A NE profile. (b) and (c) Deviations from the NE are harmful. (d) An LPT schedule.

The *social cost* of a profile s, denoted $cost(s)$ is the *makespan* of the corresponding schedule. That is, the maximal completion time of a job, given by $max_i L_i(s) \cdot d_i$. A *social optimum* of a game G is a profile that attains the lowest possible social cost. We denote by $OPT(G)$ the cost of a social optimum profile; i.e., $OPT(G) = \min_s max_i L_i(s) \cdot d_i$.

It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of the society as a whole. We quantify the inefficiency incurred due to self-interested behavior according to the *price of anarchy* (PoA) [23, 24] and *price of stability* (PoS) [3, 27] measures. The PoA is the worst-case inefficiency of a pure Nash equilibrium, while the

PoS measures the best-case inefficiency of a pure Nash equilibrium. Formally,

Definition 2.1 *Let $\mathcal{G}$ be a family of games, and let G be a game in $\mathcal{G}$. Let $\Upsilon(G)$ be the set of pure Nash equilibria of the game G. Assume that $\Upsilon(G) \neq \emptyset$.*

- *The* price of anarchy *of G is the ratio between the* maximal *cost of a PNE and the social optimum of G. That is, $PoA(G) = \max_{s \in \Upsilon(G)} cost(s)/OPT(G)$. The* price of anarchy *of the family of games $\mathcal{G}$ is $PoA(\mathcal{G}) = sup_{G \in \mathcal{G}} PoA(G)$.*

- *The* price of stability *of G is the ratio between the* minimal *cost of a PNE and the social optimum of G. That is, $PoS(G) = \min_{s \in \Upsilon(G)} cost(s)/OPT(G)$. The* price of stability *of the family of games $\mathcal{G}$ is $PoS(\mathcal{G}) = sup_{G \in \mathcal{G}} PoS(G)$.*

2.1 Related Work

There is wide literature on job scheduling on parallel machines. The minimum makespan problem corresponds to the centralized version of our game in which all jobs obey the decisions of one utility. This is a well-studied NP-complete problem. For identical machines, the simple greedy List-scheduling (LS) algorithm [19] provides a $(2 - \frac{1}{m})$-approximation to the minimum makespan problem. A slightly better approximation-ratio of $(\frac{4}{3} - \frac{1}{3m})$ is guaranteed by the Longest Processing Time (LPT) algorithm [20] and A PTAS is given in [21]. For related machines, with various speeds, LS algorithm provides a $\theta(m)$-approximation [12], and a PTAS is given in [22].

Congestion games [26] consist of a set of resources and a set of players who need to use these resources. Players' strategies are subsets of resources. Each resource has a latency function which, given the load generated by the players on the resource, returns the cost of the resource. In *singleton* congestion games players' strategies are single resources. In *weighted* congestion games, each player j has a *weight* $p(j)$, and its contribution to the load of the resources he uses as well as its cost are multiplied by $p(j)$ [9].

The special case of symmetric weighted singleton congestion games corresponds to the setting of job-scheduling: the resources are machines and the players are jobs that need to be processed by the machines. A survey of results of job-scheduling games appears in [28]. For identical machines, it is known that LPT-schedules are NE schedules [18], and that the price of anarchy, which corresponds to the makespan approximation, is $2 - \frac{2}{m+1}$ [17]. For uniformly related machines, the price of anarchy is bounded by $\frac{\log m}{\log \log \log m}$ [13]. For two machines, a bound of $\frac{1+\sqrt{5}}{2}$ is given in [23]. Additional related work on job-scheduling games deal with cost functions that depend on the internal order of jobs, e.g., [11, 7], or a cost function based on both the load on the machine and its activation cost [15].

Other related work studies additional models in which players' objective involves *social preferences*. In standard game theoretic models, players' objective is to maximize their own utility, while in games with social preferences, players have preferences over vectors of all players' utilities. For example, [29] studies a model in which the *mental state* of a player is a score based on all players' utilities, and in a mental equilibrium, players can not deviate and improve this score. The main difference from our setting is that in their model, optimizing one's utilization has the highest priority, thus, every NE is also a mental equilibrium, which is not the case in race games. Other models that capture preferences based on emotions such as

empathy, envy, or inequality aversion are presented and studied in [6, 10, 14]. A lot of attention has been given to such models in behavioral game theory. We are not aware of previous work that analyzes competition in the framework of congestion games. Other social effect, such as altruism and spite were studied, e.g., in [1, 5, 8].

2.2 Our Results

We show that competition dramatically impacts job-scheduling environments that are controlled by selfish users. RSGs are significantly different from classical job-scheduling games; their analysis is misleading, and known tools and techniques fail even on simple instances. We start by analyzing RSGs on identical machines. We show that a RSG need not have a NE, and deciding whether a game instance has a NE is an NP-complete problem. This is valid even for instances with only two pairs of competing jobs and two machines, and for instances with homogeneous competition sets. Moreover, even in cases where a NE exists, BRD may not converge. On the other hand, we identify several non-trivial classes of instances for which a NE exists and can be calculated efficiently. Each of these positive results is tight in a sense that a slight relaxation of the class characterization results in a game that may not have a NE. Specifically, we present an algorithm for calculating a NE for games with unit-length jobs, for games in $\mathcal{G}_h$ with a limited number of competition sets and machines, or with limited competition-set size, and games in $\mathcal{G}_h$ in which the job lengths form a divisible sequence (e.g., powers of 2).

We then provide tight bounds on the equilibrium inefficiency with respect to the minimum makespan objective. For classical job-scheduling, it is known that $PoS = 1$ and $PoA = 2 - \frac{2}{m+1}$ [17]. We show that for RSGs on identical machines, $PoS = PoA = 3 - \frac{6}{m+2}$. This result demonstrates the 'price of competition'. The fact that $PoS > 1$ implies that even if the system has full control on the initial job assignment, the best stable outcome may not be optimal. Moreover, since $PoA = PoS$, in the presence of competition, having control on the initial job assignment may not be an advantage at all.

For related machines, we start with a negative result showing that even the seemingly trivial case of unit-length jobs is tricky, and a NE may not exist, even if all jobs are in a single competition set. For this class of games, however, it is possible to decide whether a game has a NE, and to calculate one if it exists. Without competition, for unit-jobs and related machines, a simple greedy algorithm produces an optimal schedule. Moreover, $PoA = PoS = 1$. We show that for RSGs with unit jobs and related machines, $PoS = PoA = 2$. We then move to study games on related machines and arbitrary-length jobs. Striving for positive results, we focus on two machines and homogeneous instances. We present an algorithm for calculating a NE, and prove that any application of BRD converges to a NE. We then bound the equilibrium inefficiency for arbitrary competition structure. Specifically, for RSGs on two related machines, $PoS = PoA = 2$. The PoS lower bound is achieved already with homogeneous competition sets. Note that for classical job-scheduling game on two related machines, it holds that $PoS = 1$ and $PoA = \frac{1+\sqrt{5}}{2}$ [23], thus, again, we witness the harmful effects of a competition.

In light of the negative results regarding equilibrium existence, we discuss possible strategies of the system to modify a RSG instance or the players' utilization, such that the resulting game has a NE. We consider two approaches for *Nashification*. The first is addition of dummy jobs, and the second is compensation of low-rank players. Our hardness results imply that min-cost

Nashification is also hard. For both approaches, we present tight bounds on the Nashification cost, in general and for unit-jobs on related machine.

We conclude with a discussion of additional congestion games whose 'race' variant is natural and interesting. We show that some of our results and techniques can be adopted to other games, and suggest some directions for future work.

3 Identical Machines - Equilibrium Existence

In this section we assume that all the machines have the same unit-delay, that is, for all $i \in \mathcal{M}, d_i = 1$. The following example demonstrates that even very simple RSGs may not have a NE. Consider an instance with two machines and two competing jobs of lengths $p_1 < p_2$. If the jobs are on different machines, then the long job has a higher completion time and can reduce its rank by joining the short one, so they both have the same completion time and therefore the same rank. If the jobs are on the same machine, then the short job can reduce its rank by escaping to the empty machine. Thus, no profile is a NE.

We now show that it is NP-hard to decide whether a given RSG has a NE, even if there are only two pairs of competing jobs, and all other jobs are singletons aiming only at minimizing their completion times.

Theorem 3.1 *Given an instance of RSG, it is NP-complete to decide whether the game has a NE profile.*

Proof: Given a profile, s, verifying that it is a NE can be done in polynomial time by considering the jobs one after the other, and checking, for each job, whether its current machine is its best-response to the other jobs' assignment.

The hardness proof is by a reduction from the partition problem. An instance of Partition consists of a set A of k integers $a_1, \ldots, a_k$ that sums up to $2B$ for some integer B. The goal is to decide whether A has a partition into two disjoint sets A' and A'' such that $\sum_{j \in A'} a_j = \sum_{j \in A''} a_j = B$. Given an instance of Partition, we construct the following instance of RSG. There are $m = 2$ identical machines and $n = 2k + 2$ jobs. Two jobs have length $B + 1$, k jobs have length $\epsilon < \frac{1}{k}$, and k jobs are originated from the Partition elements, each having length a_j. All the jobs form a single competition set, that is $S_1 = \mathcal{J}$.

We show that G has a NE if and only if a partition of A exists. Assume first that a partition exists. Consider the schedule depicted in Fig. 2(a). Each machine is assigned one job of length $B + 1$, half of the ϵ-jobs, and a set of jobs corresponding to A' or A''. Since $\sum_{j \in A'} a_j = \sum_{j \in A''} = B$, the load on the machines is balanced, and all the jobs have the same rank. Any migration of a job will increase its completion time and as a result also its rank, thus, Profile (a) is a NE.

Assume that a partition of A does not exist. We show that no NE exists. First, consider a schedule in which the two long jobs are assigned on the same machine, say M_2. The load on M_2 is at least $2B + 2$. The load on M_1 is less than $2B + 1$. If there are additional jobs on M_2, they will clearly benefit from migrating to M_1. Consider therefore the schedule depicted in Fig. 2(b), in which the two long jobs are the only jobs on M_2. Their rank is $n - \frac{1}{2}$. A migration to M_1 will increase the completion time of the deviating job, however, it will share the high

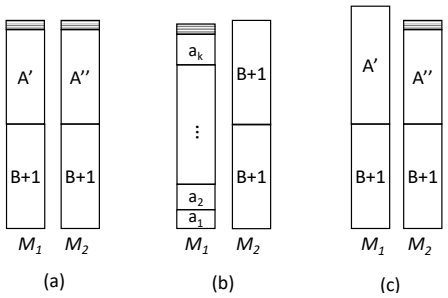

Figure 2: Possible profiles of G. (a) A NE schedule if a Partition exists. No NE if a partition does not exists and the long jobs are (b) together or (c) separated.

completion time with $n-1$ jobs. Thus, such a migration will reduce its rank to $\frac{n}{2}+1$. Thus, no profile in which the two long jobs are on the same machine is a NE.

Consider next a schedule in which there is one long job on each of M_1 and M_2. Since A has no partition, the A-jobs split are between the two machines such that one machine, say M_1, processes a subset A' of total length at least $B+1$, while M_2 processes a subset A'' of total length at most $B-1$. Since the total length of the ϵ-jobs is less than 1, these jobs will join M_2 (see Fig. 2(c)). Note that in Profile (c) there are at most $k+1$ jobs on M_1 and at least $k+1$ jobs on M_2. Thus, deviating from M_1 to M_2 is rank-reducing.

We conclude that G has a NE if and only if a partition exists. ∎

The above hardness result refers to a case in which $S_1 = \mathcal{J}$, that is, all the jobs compete with each other. The jobs of length $B+1$ are required to show that the hardness holds even if there are hardly any competitions. Specifically, assume that S_1 and S_2 each consist of a single long job and a single ϵ-job, and except for these two pairs, all other competition sets are singletons. Thus, the only goal of $n-4$ jobs is to minimize their completion time. The analysis is valid for this game as well: If a partition exists, then Profile (a) is a NE. If a partition does not exist, then profile (b), as well as every profile in which the long jobs are together, is not a NE since long jobs will benefit from joining their competitors. Also, profile (c) as well as every profile in which the long jobs are separated, is not a NE, since the long job on the loaded machine will benefit from joining its competitor.

Corollary 3.2 *Given an instance of RSG, with only two pairs of competing jobs, it is NP-hard to decide whether the game has a NE profile.*

Hoping for positive results, we turn to consider the class $\mathcal{G}_h$ of RSGs with homogeneous competition sets. Recall that $G \in \mathcal{G}_h$ if, for every $1 \leq \ell \leq c$, all the jobs in S_ℓ have the same length, p_ℓ. Unfortunately, as demonstrated in Fig. 3, games in this class, even with only three sets and three machines, may not admit a NE. Moreover, as demonstrated in Fig. 4, even if a NE exists, it may be the case that BRD does not converge.

The next natural question is whether there is an efficient way to decide, given a game $G \in \mathcal{G}_h$, whether G has a NE. We answer this question negatively:

Theorem 3.3 *Given an instance of RSG with homogeneous competition sets, it is NP-complete to decide whether the game has a NE profile.*

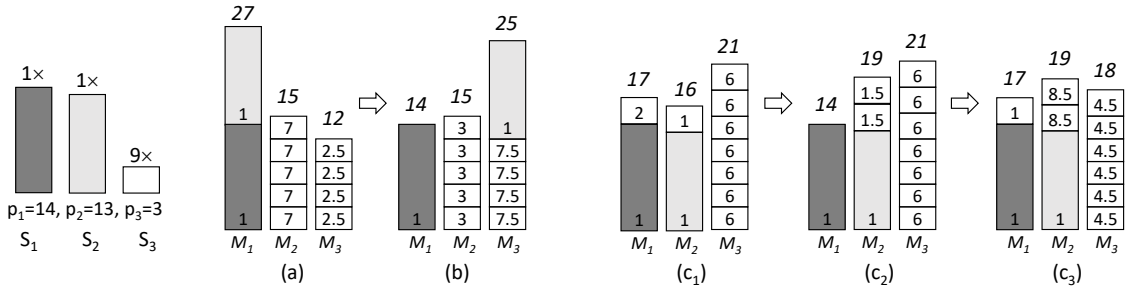

Figure 3: An example of a RSG with homogeneous competition sets, that has no NE. Jobs are labeled by their ranks. Profiles (a)-(b) show that big jobs must be on different machines. Profiles $(c_1) - (c_2) - (c_3) - (c_1)$ loop when big jobs are on different machines.

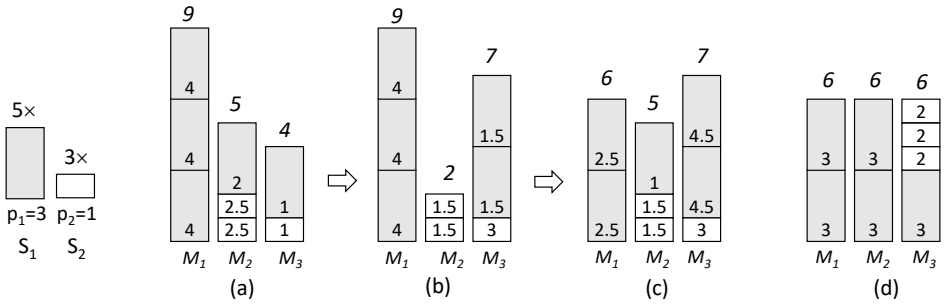

Figure 4: An example of a RSG with homogeneous competition sets in which $c = 2$, $p_1|p_2$, and BRD may loop (profiles (a)-(b)-(c)-(a)). A NE exists (profile (d)). Jobs are labeled by their ranks.

Proof: Given a profile s, verifying that it is a NE can be done in polynomial time by considering the jobs one after the other, and checking for each job whether its current machine is its best-response to the other jobs' assignment.

The hardness proof is by a reduction from the partition problem. An instance of Partition consists of a set A of k integers $a_1, \ldots, a_k$ that sums up to $2B$ for some integer B. The goal is to decide whether A has a partition into two disjoint sets A' and A'' such that $\sum_{j \in A'} a_j = \sum_{j \in A''} a_j = B$. Given an instance of Partition, we construct the following instance of RSG with homogeneous competition sets. There are $m = 4$ identical machines and $n = k + 13$ jobs divided into $k + 2$ competition sets. S_1 consists of 4 jobs of length 9, S_2 consists of 9 jobs of length 2, and for every $1 \leq j \leq k$, the j-th element in A defines the set S_{j+2} that has a single job of length $\frac{a_j}{4B}$. Note that the total length of jobs originated from A is $1/2$. We show that G has a NE if and only if a partition exists.

Assume first that a partition exists. Consider the profile (a) depicted in Fig 5(a): M_1 processes two jobs of length 9, M_4 processes 7 jobs of length 2, and each of M_2 and M_3 processes one job of length 9, one job of length 2, and all jobs corresponding to A' or A''. We show that profile (a) is a NE. Jobs corresponding to A' or A'' are in singleton competition sets and do not have a load-reducing migration, and therefore do not have a beneficial migration. A migration of a job from S_1 will make the target machine most loaded, and does not increase

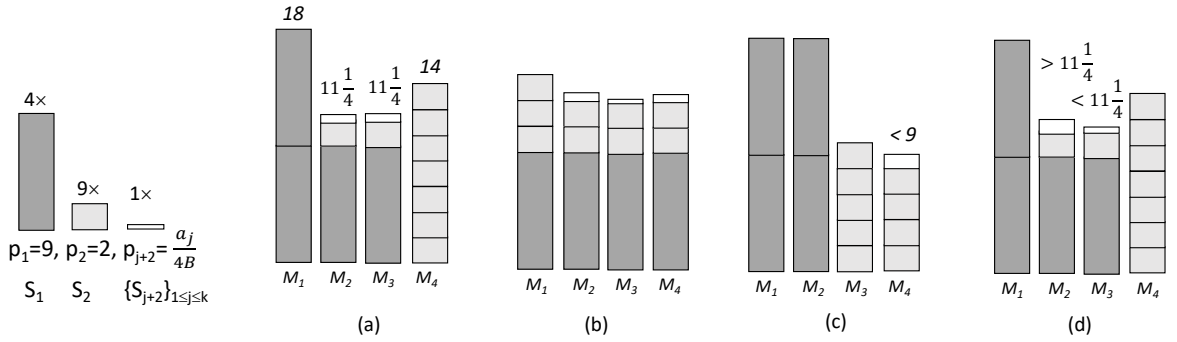

Figure 5: (a) A NE profile if a partition exists. $(b)-(d)$ Non-stable profiles if a partition does not exist.

the number of jobs sharing the maximal rank with the deviating job, therefore, it is never beneficial. For jobs from S_2, M_2 and M_3 have the lowest completion time, and leaving them is not beneficial. A migration from M_4 will make the deviating job be in the sole most loaded machine processing jobs from s_2, with fewer jobs from S_2 sharing the max-rank. Specifically, it will complete at time 20 or $13\frac{1}{4}$, while the other jobs from S_2 have completion times at most 12. Therefore, jobs from S_2 do not have a rank-reducing migration as well, and profile (a) is a NE.

Assume that a partition of A does not exist. We show that the game has no NE. Consider first profiles in which 3 or 4 of the 9-jobs are assigned on the same machine, say M_1. It must be that at least one of the other machines has load less than 9, implying that escaping from M_1 is beneficial. Consider next profiles in which the four long jobs of S_1 are assigned on different machines (Fig. 5(b)). If there is only one most-loaded machine, then the 9-job on it can benefit from joining any of the other machines. If there are two most-loaded machines, then there must be a machine with only one 2-job, and some 2-job has a beneficial migration.

If the long jobs are assigned in two pairs, say on M_1 and M_2 (Fig. 5(c)), then there is at least one machine with load less than 9, implying, that each of the 9-jobs on M_1 and M_2 can benefit from migrating to that machine, reducing its rank from 2.5 to 2.

We are left with profiles in which two 9-jobs are assigned together, say on M_1, and there is a single 9-job on two other machines, say M_2 and M_3 (Fig. 5(d)). In order to avoid a cost-reducing beneficial migration of the 2-jobs, there must be seven 2-jobs on M_4 and one 2-job on each of M_2 and M_3. Since the jobs corresponding to A are in singleton competition sets and only aim at minimizing their completion time, they are all on M_2 and M_3. Given that there is no partition, M_2 and M_3 are not balanced. Assume w.l.o.g., that $L_2 > L_3$. Thus, $L_3 < 11\frac{1}{4}$, implying that $L_3 + 2 < L_4$. Thus, there is a beneficial migration of a job of length 2 from M_2 to M_3. After such a migration regardless of the migration of jobs corresponding to A there is a beneficial migration of a job from M_4 to M_2, that will be followed by a beneficial migration from M_3 to M_4. This sequence of three migrations brings us back to a profile from which some 2-job has a beneficial migration.

We conclude that G has a NE if and only if a partition exists. ∎

In light of the above negative results, we would like to characterize instances in which a NE is guaranteed to exist. One such class includes instances of unit-length jobs and arbitrary

competition sets.

Theorem 3.4 *If all jobs have the same length, then a NE exists and can be computed efficiently*

Proof: We assume w.l.o.g., that all jobs have unit length. The following algorithm produces a NE.

Algorithm 1 Calculating a NE Schedule of a game of unit-jobs

1: let $L = \lfloor \frac{n}{m} \rfloor$ and $h = n\%m$.
2: assign the jobs set after set, fill the machines one after the other. The first h machines are filled to capacity $L+1$, the last $m-h$ machines are filled to capacity L.
3: **for** $i = h$ to $m-1$ **do**
4: let S_ℓ be the last set assigned on M_i.
5: **if** $n_\ell(i) \leq n_\ell(i+1)$ **then**
6: move one job $j \in S_\ell$ from M_i to M_{i+1}.
7: **else**
8: break
9: **end if**
10: **end for**

By definition of h and L we have that $(L+1)h + L(m-h) = n$, thus, all jobs are assigned. Consider a set S_ℓ. If all the jobs of S_ℓ are assigned on high machines (with load $L+1$), or all the jobs of S_ℓ are assigned on low machines (of load L), then the jobs of S_ℓ are stable. The schedule produced in line 2 is not stable only if jobs from the last set assigned on a high machine can increase the number of jobs sharing the higher completion time with them. Specifically, if $n_\ell(h) \leq n_\ell(h+1)$. If this is the case, one job is migrated to M_{h+1}, and the same test is applied on the suffix of machines from M_{h+1} to M_m. Based on the above, it is easy to show by induction on i, for $h \leq i \leq m-1$, that all the jobs on $M_1, \ldots, M_i$ are stable. Finally, if the final load on M_m is L, then jobs on M_m are clearly stable, and if the final load on M_m is $L+1$ then sets on M_m are either solely on M_m or have more members on M_m - as otherwise, the load of M_m would not increase to $L+1$. Thus, the algorithm produces a NE. □ ■

Another class for which we have a positive result considers instances of only two competition sets and three machines. It is tight in light of the no-NE example given in Fig. 3, in which there are three sets on three machines.

Theorem 3.5 *If $G \in \mathcal{G}_h$ has $c = 2$ and $m = 3$, then a NE exists and can be computed efficiently.*

Proof: For $\ell = 1, 2$, let $P_\ell = n_\ell \cdot p_\ell$, and let $P = P_1 + P_2$. We distinguish between two cases:

Case 1: For both $\ell, P_\ell > P/3$. Assign a minimal number of jobs of total length at least $\frac{P}{3}$ from S_1 on M_1. Assign a minimal number of jobs of total length at least $\frac{P}{3}$ from S_2 on M_3. Assign the remaining jobs on M_2. For $i \in \{1, 2, 3\}$, let L_i^0 denote the load on M_i after this initial assignment. Note that if a job from S_1 is moved from M_1 to M_2 and a job from S_2 is moved from M_3 to M_2 then the resulting load on M_2 will be $L_2 > \frac{P}{3}$, since by their construction $L_1^0 - p_1 < \frac{P}{3}$ and $L_3^0 - p_2 < \frac{P}{3}$. Assume w.l.o.g., that $p_2 < p_1$. Now, as long as it is beneficial, let jobs from M_3 migrate to M_2. Let L_i^* denote the load on M_i at the end of the process. First note that $L_2^* \leq L_3^*$, since $L_2^0 \leq \frac{P}{3} < L_3^0$ and there are more jobs from S_2 on M_3,

therefore, a migration from M_3 to M_2 is beneficial only if it does not result in M_2 being more loaded. If some job from M_3 migrates to M_2 then we have a NE: any job from M_1 migrating to M_i for $i \in \{2,3\}$ will cause $L_1^* < \frac{P}{3} < L_i^*$. Given that there are more jobs from S_1 on M_1, such a migration is not beneficial. Also, no job can benefit from leaving M_2, which is the least loaded machine.

If no jobs migrates from M_3 to M_2, let jobs migrate from M_1 to M_2 in the same way. After the last such migration, by the same arguments, is not beneficial for jobs from S_3 to migrate to M_2 - and also since $L_1^* \geq L_2^*$ there can be no beneficial migration from M_3 to M_1. Since M_2 is the least loaded it is also not beneficial for any job to leave it. Therefore, we have a NE.

Case 2: For some i, $P_i \leq P/3$. Assume w.l.o.g., that $P_2 \leq P/3$. Assign all the jobs of S_2 on M_3, then greedily assign jobs from S_1 on a lightly loaded machine, breaking ties in favor of smaller indexed machines. Let L_i^0 denote the load on M_i after this initial assignment. Note that for any two machines $|L_{i_1}^0 - L_{i_2}^0| \leq p_1$. Proceed as follows, distinguishing between three cases:

1. M_3 is a least loaded machine. Since M_1 and M_2 process only jobs of S_1, and since any migration into M_3 will make M_3 the most loaded machine, we have a NE.

2. M_3 is the most loaded machine. Since the most loaded machine has load at least $\frac{P}{3}$, and $P_2 \leq \frac{P}{3}$, there must be at least one job from S_1 on M_3. Also, $L_1 = L_2$ since otherwise if w.l.o.g $L_1 > L_2$ then $L_3 > L_1 = L_2 + p_1$. This contradicts the assignment of the last job from S_1 on M_3, as placing it on M_2 is better. Once we move a job from S_1 from M_3 to M_1 we have a NE since all jobs of S_2 are on M_3 which is now the least loaded machine. The jobs on M_1 have the highest cost, but they cannot benefit from migrating, since they share the high rank with more competitors.

3. $L_2^0 < L_3^0 \leq L_1^0$. We migrate jobs from M_3 into M_2 as long as $L_3 - L_2 \geq 2p_2$. Since $L_1^0 - L_2^0 = p_1$, we end up with $L_1^* - L_2^* \leq p_1$ and $L_1^* - L_3^* \leq p_1$. Moreover, the order between the machines' loads has not changed. Thus, we have a NE. □

■

Classical job-scheduling games are race games with singleton competition sets. A NE may not exist even if there is just one pair of competing jobs. Also, it is NP-hard to decide if a NE exists even if there are only singletons and two competing pairs (see Theorem 3.1). For games with homogeneous competition sets, in which there are only singleton and pairs, we have positive news.

Theorem 3.6 *If $G \in \mathcal{G}_h$, and for all ℓ, $|S_\ell| \leq 2$ then a NE exists and can be computed efficiently.*

Proof: We present an algorithm for calculating a NE profile. For $k = 1, 2$, let A_k denote the set of competition sets with k jobs.

Algorithm 2 Calculating a NE Schedule of a game $G \in \mathcal{G}_h$ with singleton and doubleton sets

1: for every set ℓ, let P_ℓ denote the total length of jobs in S_ℓ.
2: sort the sets such that $P_1 \geq P_2 \geq \ldots \geq P_c$.
3: **for** $\ell = 1$ to c **do**
4: schedule S_ℓ on a least loaded machine
5: **end for**
6: while there exists $S_\ell \in A_2$ assigned to M_a such that $L_a - 2p_\ell = L_b$ for a least loaded machine M_b, move one job from S_ℓ to M_b.

We show that the resulting assignment is a NE. It is well known that when LPT completes, the gap between the loads on any two machines is at most the length of the shortest job on the more loaded machine. Consider a job $j \in S_\ell$ assigned on machine M_a. If j has no competitors, then by the above property, the load on every other machine is at least $L_a - p(j)$, and thus, no beneficial migration exists for j. Assume that $j \in S_\ell$ has a competitor, and both jobs are on M_a. By the above LPT property, every less loaded machine has load at least $L_a - 2p_\ell$. A migration into a machine with load higher than $L_a - 2p_\ell$ will make the target machine more loaded, and is therefore not beneficial. Migrating into a machine whose load is exactly $L_a - 2p_\ell$ is the only possible beneficial migration of j. Both jobs in S_ℓ keep their 1.5 rank, with a reduced completion time. The loads on M_a and M_b become perfectly balanced. Note that every machine can be a target of at most one such migration, thus, after at most $m/2$ migration a NE is reached. □

■

3.1 Homogeneous Competition sets with Divisible Lengths

Let $\mathcal{G}_{div}$ be the class of RSGs with homogeneous competition sets in which the job lengths form a divisible sequence. Formally, let $p_1 > p_2 > \ldots > p_c$ denote the different job lengths in $\mathcal{J}$, then $S_\ell = \{j | p(j) = p_\ell\}$, and for every $\ell_1 > \ell_2$, it holds that $p_{\ell_1} | p_{\ell_2}$. For example, if all job lengths are powers-of-2 and $S_\ell = \{j | p(j) = 2^{c-\ell+1}\}$ then $G \in \mathcal{G}_{div}$.

As demonstrated in Fig. 4, BRD may not converge to a NE even if $G \in \mathcal{G}_{div}$ and $c = 2$. Nevertheless, we prove that a NE can be computed directly for any game $G \in \mathcal{G}_{div}$.

Theorem 3.7 *If $G \in \mathcal{G}_{div}$, then a NE exists and can be computed efficiently.*

Proof: We provide an algorithm for calculating a NE assignment. The algorithm assigns the jobs from largest to smallest. It proceeds in iterations, where jobs of S_ℓ are assigned in iteration ℓ. The following simple observation is based on the fact $p_{\ell_2} | p_{\ell_1}$ for any $\ell_2 \geq \ell_1$, and is valid for any assignment of the jobs in non-increasing order.

Claim 3.8 *Let s be any assignment of the jobs in $S_1, \ldots, S_{\ell-1}$. For any two machines M_a, M_b, it holds that $p_\ell | (L_a(s) - L_b(s))$.*

Proof: The jobs assigned in s are all larger than the jobs than S_ℓ. Since $p_{\ell_1} | p_\ell$ for any $\ell_1 > i$, there exist $x_a, x_b \in \mathbb{N}$ such that $L_a(s) = x_a \cdot p_\ell$ and $L_b(s) = x_b \cdot p_\ell$. Therefore, $L_a(s) - L_b(s) = (x_a - x_b) \cdot p_\ell$ where $x_a - x_b \in \mathbb{N}$. □ ■

Now let J_a be the set of jobs assigned already, and let $J_u = \mathcal{J} \setminus J_a$ be the set of unassigned jobs. Initially, $J_a = \emptyset$ and $J_u = \mathcal{J}$. For a set S_ℓ, let u_ℓ be the number of unassigned jobs

from S_ℓ. For simplicity, we do not include the updates of u_ℓ and the sets J_a and J_u in the description below, and assume that their value correspond to the current profile. Similarly, L_a is the current load on machine a. Recall that for a set of jobs A, we denote by $P(A)$ the total length of jobs in A. In particular, $P(J_u)$ is the total length of unassigned jobs.

Algorithm 3 Calculating a NE Schedule of a game in $\mathcal{G}_{div}$

1: **for** $\ell = 1$ to c **do**
2: select a machine with minimal load, denote it $M_{\ell 1}$.
3: assign $\min(n_\ell, n_\ell^1)$ jobs from S_ℓ on $M_{\ell 1}$, where n_ℓ^1 is the smallest integer satisfying $\sum_{k \neq \ell 1} \max(L_{\ell 1} + n_\ell^1 \cdot p_\ell - L_k, 0) \geq P(J_u) - n_\ell^1 \cdot p_\ell$.
4: **while** $u_\ell > 0$ and some machine has load less than $L_{\ell 1} - p_\ell$ **do**
5: let M_a be a machine with minimal load.
6: assign $\min(u_\ell, (L_{\ell 1} - L_a)/p_\ell - 1)$ jobs from S_ℓ on M_a.
7: **end while**
8: **while** $u_\ell > 0$ **do**
9: assign a job from S_ℓ on a lightly loaded machine with the most jobs from S_ℓ.
10: **end while**
11: **end for**

For every ℓ, the jobs of S_ℓ are assigned in iteration ℓ. Every iteration consists of at most three phases. In the first phase (lines 2-3), the algorithm assigns jobs of S_ℓ on a machine, $M_{\ell 1}$, that will accommodate a maximal number of jobs from S_ℓ, and will be the most loaded machine accommodating jobs from S_ℓ. In the second phase(lines 4-7), the algorithm assigns jobs of S_ℓ greedily, on a least loaded machine, until its load is $L_{\ell 1} - p_\ell$. In the optional third phase (lines 8-10), performed after all the machines have load at least $L_{\ell 1} - p_\ell$, the algorithm assigns jobs greedily, iteratively adding a single job from S_ℓ on a lightly loaded machine with a maximal number of jobs from S_ℓ.

In line 3, the number of jobs assigned on $M_{\ell 1}$ is set to be the minimum between n_ℓ - the number of jobs in S_ℓ, and n_ℓ^1, where n_ℓ^1 is determined in the following way: For an integer $x > 0$, assume x jobs from S_ℓ are placed on $M_{\ell 1}$. The resulting load on $M_{\ell 1}$ is $L_{\ell 1} + xp_\ell$. For every machine $k \neq \ell 1$, it is now possible to add on M_k jobs of total load $\max(0, L_{\ell 1} + xp_\ell - L_k)$ without making M_k more loaded than $M_{\ell 1}$. n_ℓ^1 is selected to be the minimal x such that the available capacity on the other machines, if none of them is allowed to become more loaded than $M_{\ell 1}$, is sufficient to accommodate the remaining jobs – whose total length is given by $P(J_u) - xp_\ell$.

Let s_ℓ be the profile after all jobs from S_ℓ are assigned. Let $s = s_c$ be the profile produced by the algorithm. The assignment of jobs from S_ℓ in the second phase, implies the following.

Observation 3.9 *If there are additional machines accommodating jobs from S_ℓ whose load in s_ℓ equals $L_{\ell 1}(s_\ell)$, then for every machine M_a, $L_a(s_\ell) \geq L_{\ell 1}(s_\ell) - p_\ell$.*

In order to show that s is stable we will prove the following property of s:

Claim 3.10 *For every $1 \leq \ell \leq c$, the machine $M_{\ell 1}$ is the most loaded machine in s among the machines accommodating jobs from S_ℓ.*

Proof: $M_{\ell 1}$ is the first machine assigned jobs from S_ℓ. It is assigned $min(n_\ell, n_\ell^1)$ jobs from S_ℓ where $\sum_{k \neq \ell 1} \max(L_{\ell 1} + n_\ell^1 \cdot p_\ell - L_k, 0) \geq P(J_u) - n_\ell^1 \cdot p_\ell$. Note first that if $min(n_\ell, n_\ell^1) = n_\ell$,

then $M_{\ell 1}$ is the only machine accommodating jobs from S_ℓ and the claim clearly holds. If $min(n_\ell, n_\ell^1) = n_\ell^1$, then the choice of n_ℓ^1 guarantees that the unassigned jobs can fit into the other machines without exceeding capacity $L_{\ell 1} + n_\ell^1 \cdot p_\ell$ is sufficient to accommodate all the unassigned jobs. Claim 3.8 together with the fact that jobs are assigned in non-increasing length order, guarantees that when the remaining jobs are assigned, no machine will become more loaded than $M_{\ell 1}$. ■

We show that s is a NE. Assuming for contradiction it is not, and let S_ℓ be the competition set with minimal p_ℓ such that a job $j \in S_\ell$ has a beneficial migration.

By Claim 3.10, $M_{\ell 1}$ it is the most loaded machine with a job $j \in S_\ell$. By the algorithm it also has the most jobs from S_ℓ. Additionally, from Observation 3.9 any migration of a job $j \in S_\ell$ from $M_{\ell 1}$ to a machine M_a will make M_a the most loaded machine with a job from S_ℓ. This means no job $j \in S_\ell$ in $M_{\ell 1}$ has a load-reducing migration to any machine M_a. We show that there is no load-increasing rank reducing migration as well: by the choice of n_ℓ^1 if there is any other machine M_b with $L_b = L_a + p_\ell$ and $L_b < L_{\ell 1}$ then there is a job $j_2 \in S_{\ell_2} \neq S_\ell$ with $p_{j_2} < p_j$ assigned to M_b, and then j_2 has a beneficial migration to M_a, which contradicts the choice of p_ℓ.

Given that $M_{\ell 1}$ is not involved, and that a migration to a more loaded machine is never beneficial, any beneficial migration is between machines M_a and M_b with $L_b < L_a < L_{\ell 1}$. The greedy choice of machines in the algorithm along with line 6 in the algorithm ensure that if $L_a > L_{\ell 1} - p_\ell$ then $L_b \geq L_{\ell 1} - p_\ell$. This means, by Observation 2.1, that a migration from M_a to M_b is not beneficial, as $L_{\ell 1} \leq L_b + p_\ell$ and $M_{\ell 1}$ has the most jobs from S_ℓ. If $L_a = L_{\ell 1} - p_\ell$ then, by the choice of $M_{\ell 1}$, the load $L_b > L_a - p_\ell$. This implies that a migration of a job j from M_a to M_b is load increasing - and by the greedy choice of initial machine, the number of jobs in a machine with lower load at least stays the same.

Therefore, no beneficial migration exists and s is a stable profile.

Remark: Algorithm 3 assumes that jobs from different competition sets have different lengths. A more relaxed definition of $\mathcal{G}_{div}$ allows several competition sets with the same job-length. Formally $p_1 \geq p_2 \geq \ldots \geq p_c$ is the divisible sequence of job-lengths (possibly with repetitions). The algorithm can be modified to fit such instances by calculating, for each competition set S_ℓ, how many machines will end up having load $L_{\ell 1}$ at the end of the algorithm, and assigning an extra job to at most that number of machines during the second step of the algorithm, instead of the third step. This resolves the case in which two machines may have load- gap exactly p_ℓ and migrating between them may be beneficial.

■

4 Identical Machines - Equilibrium Inefficiency

In this section we analyze the equilibrium inefficiency of RSGs with respect to the objective of minimizing the maximal cost of a player (equivalent to the makespan of the schedule). For the classical job-scheduling game, the Price of Anarchy is known to be $2 - \frac{2}{m+1}$ for m identical machines, and the Price of Stability is known to be 1. We show that competition causes higher inefficiency. Specifically, both the PoA and the PoS are $3 - \frac{6}{m+2}$. We prove below the upper bound for the PoA and the lower bound for the PoS. Additionally we describe, given $m \geq 3$ and $\epsilon > 0$, a game G, with homogeneous competition sets, for which $PoA(G) = 3 - \frac{6}{m+2} - \epsilon$.

Given that we prove a matching PoS bound, this proof is less interesting. However, it can serve as a warm-up for the lower bound PoS proof, which is more involved.

Theorem 4.1 *For every $m \geq 3$ and $\epsilon > 0$, there exists a game G on m identical machines for which $PoA(G) = 3 - \frac{6}{m+2} - \epsilon$.*

Proof: Given $m \geq 3$ and $\epsilon > 0$, we present a game $G \in \mathcal{G}_h$ for which $PoA(G) = 3 - \frac{6}{m+2} - \epsilon$. Let $n = (m-1)m + 3$. The jobs are partitioned into two competition sets. S_1 consists of $(m-1)m$ jobs of length 1, and S_2 consists of 3 jobs of length $m - \delta$, where δ is a small constant that depends on ϵ and m.

Consider a schedule s, in which one machine processes the three jobs of S_2 and each of the other $m-1$ machines processes m unit jobs. Fig. 6(a) presents the schedule for $m = 5$. We show that s is a NE: Since the three jobs of S_2 are assigned on one machine, they all have the same rank. Migrating to a less loaded machine will result in load $2m - \delta$, leaving the two competitors with load $2m - 2\delta$, thus the rank of a migrating job would be hurt. The jobs of S_1 are balanced and do not have a beneficial migration as well. The cost of s determined by the completion time of the machine processing the jobs of S_2 and is $3(m - \delta)$.

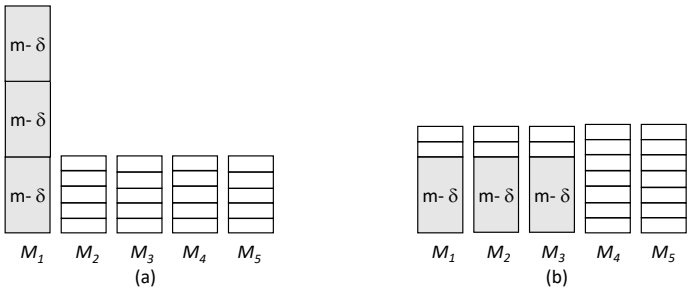

Figure 6: A tight example for $m = 5$. (a) a NE profile. (b) an optimal profile.

If $m > 3$, then an optimal profile assigns the 3 long jobs on 3 different machines, and balances the load on the machines with the unit-jobs (see in Fig. 6(b)). Specifically, there are two unit-jobs on each of the three machines with jobs from S_2, and $m+2$ unit-jobs on each of the other $m-3$ machines. Note that $3 \cdot 2 + (m-3)(m-2) = m(m-1) = |S_1|$. The cost of an optimal assignment is $m + 2$. The PoA is therefore $\frac{3m-3\delta}{m+2} = 3 - \frac{6}{m+2} - \epsilon$.

If $m = 3$, then the cost of an optimal assignment is $m + 2 - \delta$. If $m = 2$ then an optimal profile assigns two jobs of length $2 - \delta$ on one machine, and a single job of length $2 - \delta$ together with two unit-jobs on the second machine. The cost of an optimal assignment is $m + 2 - 2\delta$, thus, the choice of δ for a given ϵ is a bit different.

Note that the optimal profile is also a NE: The three long jobs have the same cost $m+2-\delta$, and thus the same rank. Any deviation will increase their cost and rank. The unit-jobs on the first 3 machines have lower rank than the other unit-jobs, but a deviation of a unit-job whose cost is $m + 2$, will increase its cost and only reduce the number of jobs sharing the maximal completion time with the deviating job. □ ■

Theorem 4.2 *If G is a RSG on m identical machines that has a NE, then $PoA(G) \leq 3 - \frac{6}{m+2}$.*

Proof: Let s be any NE of G. Let M_i be a machine with the highest load, that is $cost(s) = L_i(s)$, and let j be a shortest job on M_i. Assume $j \in S_\ell$. If there are at most two jobs on M_i then let j' be a longest job on M_i. Clearly, $OPT(G) \geq p(j')$ and $cost(s) \leq 2p(j')$, implying $PoA(G) \leq 2$. Otherwise, $p(j) \leq cost(s)/3$.

Claim 4.3 *For every machine $i' \neq i$, it holds that $L_{i'}(s) \geq \frac{1}{3}cost(s)$.*

Proof: We first show that $L_{i'} \geq L_i - 2p(j)$. Assume by contradiction that for some machine i', $L_{i'} < L_i - 2p(j)$. Assume that job j migrates to machine i'. Before the migration, j's rank is maximal among S_ℓ. After the migration, its completion time reduces to $L_{i'} + p(j) < L_i - p(j)$. If there are jobs from S_ℓ on M_i, their completion time after j's departure is $L_i - p(j)$, so the migration reduces j's rank, contradicting the stability of s. If j has no competitors on machine i, then the migration is clearly beneficial, since j reduces its completion time and can only improve its rank. Given that $p(j) \leq cost(s)/3$, we conclude, $L_{i'} \geq L_i - 2p(j) \geq L_i - \frac{2}{3}cost(s) = \frac{1}{3}cost(s)$. □ ■

Using the above claim, we can bound the social optimum:

$$OPT(G) \geq \frac{\sum_j p(j)}{m} = \frac{\sum_i L_i(s)}{m} \geq \frac{cost(s) + (m-1)cost(s)/3}{m} = \frac{(m+2)cost(s)}{3m}.$$

Thus, $PoA(G) = \frac{cost(s)}{OPT} \leq \frac{3m}{m+2} = 3 - \frac{6}{m+2}$. ■

Theorem 4.4 *For every $m \geq 3$ and $\epsilon > 0$, there exists a RSG G on m identical machines such that G has a NE, and $PoS(G) \geq 3 - \frac{6}{m+2} - \epsilon$.*

Proof: Given $m \geq 3$ and $\epsilon > 0$, we describe a RSG G such that $\text{PoS}(G) = 3 - \frac{6}{m+2} - \epsilon$. Let $E = \{\delta_1, \delta_2, \delta_3\} \cup \{\epsilon_i | 1 \leq i \leq m(m-1)\}$ be a set of $3 + m(m-1)$ small positive numbers such that $\delta_1 < \delta_2 < \delta_3 \leq \frac{\epsilon m}{3}$, $\delta_1 + \delta_2 > \delta_3$, $\sum_{i>0} \epsilon_i < \frac{1}{4}$, and any subset of E with any coefficient in $\{-1, +1\}$ for each element, has a unique sum. That is, $\forall A_1, A_2 \subseteq \{\delta_1, \delta_2, \delta_3, \epsilon_1, \ldots, \epsilon_{m(m-1)}\}$ such that $A_1 \neq A_2$, and any $\gamma_k \in \{-1, +1\}$ we have $\sum_{k \in A_1} \gamma_k \cdot k \neq \sum_{k \in A_2} \gamma_k \cdot k$.

The set of jobs consists of $1 + m(m-1)$ competition sets, $S_0, \ldots, S_{m(m-1)}$:

1. S_0 consists of three jobs, where j_0^i for $i = 1, 2, 3$ has length $m - \delta_i$.
2. For $\ell = 1, \ldots, m(m-1)$, the set S_ℓ consists of two jobs: j_ℓ^1 of length $\frac{1}{4} - \epsilon_\ell$, and j_ℓ^2 of length $\frac{3}{4} + \epsilon_\ell$. Note that $p(j_\ell^1) + p(j_\ell^2) = 1$.

The PoS analysis is based on the fact that in every NE, the three long jobs of S_0 are assigned on the same machine, while an optimal assignment is almost balanced. We first restrict the possible assignments of the jobs in S_ℓ for all $\ell \geq 0$.

Observation 4.5 *In every NE of G, $\forall \ell \geq 1$, the two jobs of S_ℓ are assigned on the same machine.*

Proof: Let s be an assignment in which j_ℓ^1 and j_ℓ^2 are on different machines. Given that every subset of E has a different sum, one of j_ℓ^1 and j_ℓ^2 is assigned on a machine with a higher load. The rank of this job in s is 2. By joining its competitor, its rank will reduce to 1.5. ■

Consider now the three jobs of S_0.

Claim 4.6 *In every NE, the jobs of S_0 are assigned on the same machine.*

Proof: Let s be an assignment in which j_0^1, j_0^2 and j_0^3 are assigned on three different machines.

Given that every subset of E has a different sum, the completion time of some job from S_0 is higher than the completion time of its competitors. This job has rank 3 and it can reduce its rank to 2.5 by joining either of its competitors. Thus, s is not a NE.

Consider next an assignment s in which two jobs from S_0 are on one machine, and the third job is on a different machine. W.l.o.g., assume that j_0^1 and j_0^2 are assigned on M_1, and j_0^3 is assigned on M_2. Clearly, there is no NE in which there is another job on M_1, since any small job can benefit from moving from M_1 to a less loaded machine. By Observation 4.5, the jobs of $S_\ell, \forall \ell \geq 1$ are assigned together. Thus, a total load of $m(m-1)+m-\delta_3 = m^2-\delta_3$ is split among the $m-1$ machines $M_2, \ldots, M_m$. Since $m-1 \nmid m^2$, there are two machines M_a, M_b such that $L_a(s) > L_b(s) - \frac{1}{2}$, and M_a accommodates some set S_ℓ. Therefore, j_ℓ^1, whose size is a bit less than $\frac{1}{4}$, can reduce its rank from 1.5 to 1 by migrating to M_b. We conclude that s is not a NE. ∎

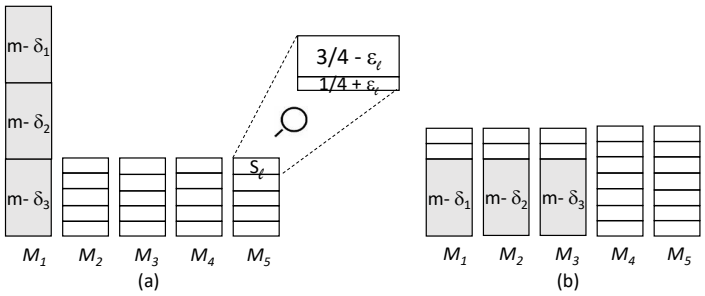

Figure 7: A tight example for $m = 5$. (a) The only NE profile. (b) an optimal profile.

By Claim 4.6, in every NE, s, the jobs of S_0 are assigned on the same machine, say M_1. Thus, the cost of every NE is at least $L_1(s) = 3m - \sum_{i=1}^{3} \delta_i$. We show that a NE exists. An example for $m = 5$ is given in Fig. 7(a). Assign the jobs of S_0 on M_1. Distribute all remaining jobs such that for all $\ell \geq 1$ the jobs of S_ℓ are on the same machine, and there are m such sets assigned on each machine other than M_1. For all $a \geq 2$ we have $L_a = m$. This assignment is a NE since any migration of a job j from S_ℓ with $\ell \geq 1$ will increase its rank from 1.5 to 2, and any migration of a job j_0^x from S_0, will end up with load at least $2m - \delta_x$ which is more than $2m - \delta_y - \delta_z$, the remaining load on M_1. Thus, such a migration increases the rank of the deviating job from 2 to 3 and is not beneficial.

We turn to describe an optimal assignment. The total load of the jobs is $m(m-1)+3m-\sum_{i=1}^{3}\delta_i = m(m+2) - \sum_{i=1}^{3}\delta_i$. The maximal length of a job is less than m, and there are $3 \leq m$ long jobs. The remaining jobs can be arranged in unit-length pairs. Thus, an optimal assignment is almost perfectly balanced (up to a gap of δ_3), where the most loaded machine has load $\frac{m(m+2)}{m} = m+2$. The resulting PoS is $\frac{3m-\sum_{i=1}^{3}\delta_i}{m+2} < 3 - \frac{6}{m+2} - \epsilon$. ∎

5 Related Machines

In this section we consider RSGs played on related machines. Recall that d_i is the processing delay of machine M_i. Thus, it takes $p(j) \cdot d_i$ time units to process a job of length $p(j)$ on

M_i. For a profile s, $C_i(s) = L_i(s) \cdot d_i$ is the completion time of M_i, and the cost of every job assigned on it.

5.1 Unit-length Jobs

For classical job-scheduling games with unit-jobs and related machines the picture is simple and well understood. For a profile s, let $C_i^+(s) = (L_i(s) + 1) \cdot d_i$ denote the completion time of M_i if one more job would be assigned on it. A simple greedy algorithm that assigns the jobs sequentially, each on a machine minimizing C_i^+, is known to produce a Nash equilibrium profile that also minimizes the makespan. Moreover, every best-response sequence converges to an optimal schedule, thus, without competition, $PoA = PoS = 1$.

Surprisingly, as we show, even this simple setting of RSGs with unit-length jobs may not have a NE. Consider a game with $n = 5$ unit jobs, that form a single competition set. Assume there are three machines with delays $1, 1 + \epsilon$ and $1 + 2\epsilon$. First note that a NE profile must fulfil $L_1 \geq L_2 \geq L_3$, as otherwise, it is clearly beneficial to deviate from a slow machine to a less loaded faster machine. Also note that if $L_1 = 4$, then one of the slower machines is empty and a deviation from M_1 to the empty machine is beneficial. The remaining load vectors are $\{\langle 3, 2, 0\rangle, \langle 3, 1, 1\rangle, \langle 2, 2, 1\rangle\}$. As demonstrated in Fig. 8, none of the corresponding profiles is a NE. Profile (c) is the output of a greedy algorithm. However, a job on M_2 can reduce its rank from 4.5 to 4 by a migration to M_1 (Profile (a)). Once it migrates, it is beneficial for the job on M_3 to join M_2 (Profile (b)), and a migration from M_1 to M_3 brings us back to Profile (c).

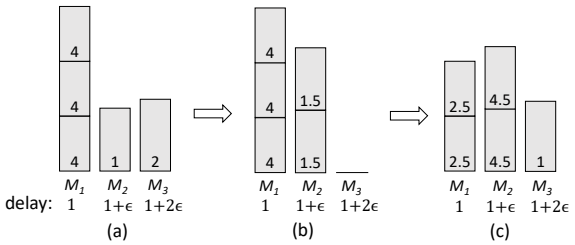

Figure 8: No NE of a RSG with five competing unit-jobs on three related machines. The profiles loop (a)-(b)-(c)-(a). Jobs are labeled by their ranks.

While a NE may not exist, this class of instances is somewhat simpler. We show that it is possible to decide efficiently whether a given game instance has a NE and to compute one if it exists. Recall that the same task is NP-hard for games in $\mathcal{G}_h$ even on identical machines.

Theorem 5.1 *Let G be a game with unit-jobs on related machines in which all jobs are in the same competition set ($S_1 = \mathcal{J}$). It is possible to decide efficiently whether G has a NE and to compute one if it exists.*

Proof: We present an algorithm that returns a NE if one exists. The set Λ includes the machine to which a job already migrated.

Algorithm 4 Attempt to find a NE of a game with a single competition set of unit jobs.

greedily assign each job on a most loaded machine among those minimizing C_i^+.
$\Lambda = \emptyset$
while $\Lambda \neq \mathcal{M}$ **do**
  let M_a be a highest load machine among the highest cost machines not in Λ.
  if there is a beneficial migration out of M_a **then**
    let M_b be a lowest cost machine among the most loaded machines, such that a migration from M_a to M_b is beneficial.
    migrate a job from M_a to M_b.
    add M_b to Λ.
  else
    break
  end if
end while
A NE exists **if and only if** the current profile is a NE.

We show that if a NE exists then it is reached by the algorithm. The algorithm only halts once there is no beneficial migration from the highest cost machine that was not previously migrated to. Therefore, if the final profile is not a NE then the only possible beneficial migration is from a lower cost machine, or from a machine $M_a \in \Lambda$ – meaning a machine that had a job added on it.

Since the initial assignment is greedy, and the migrating job chosen in the algorithm is always from a highest cost machine not in Λ, any migration once the algorithm completes will make the target machine have higher cost than any machine not in Λ. Therefore for any two machines such that $M_a \in \Lambda$ and $M_b \notin \Lambda$, we have $C_a \geq C_b$. This means if there is a beneficial migration from any machine M_z not in Λ then there is a beneficial migration from a highest cost machine not in Λ, which is a contradiction to the algorithm stopping.

Therefore, if the instance reached by the algorithm is not a NE we can assume the only beneficial migrations are from machines in Λ. Note that the first machine added to Λ is the sole highest cost machine after the migration, and therefore $|\Lambda| \geq 2$. Additionally, as machines in Λ have a bigger load, a job will only migrate to a machine M_b during the algorithm if $C_b^+ < C_z, \forall M_z \in \Lambda$. This means the only beneficial migrations are of jobs from machines $M_z \in \Lambda$ with $C_z > min_{i:M_i \in \Lambda} C_i$.

Since the machines in Λ were chosen as most loaded, and since migrations from a machine in Λ to a machine $M_b \notin \Lambda$ is cost-decreasing, such migrations will remain possible regardless of migrations among machines in Λ or among machines not in Λ.

Once no such migrations are beneficial, by the choice of machines added to Λ, beneficial migrations from some machine $M_a \in \Lambda$ to M_z exist, as a job from the machine added immediately after M_z to Λ benefits from a migration to M_z. Once the migration is performed the game returns to an earlier stage in the algorithm, where a job from the highest cost machine $M_b \notin \Lambda$ has a beneficial migration to M_z. We know this migration is beneficial as it happened during the algorithm. Therefore, beneficial migrations always exist, implying that a NE does not exist. □ ■

The above positive result may lead one to expect that it would be possible to modify an

instance slightly in order to get a game in which a NE exists. In Section 6 we discuss the Nashification of RSGs with unit-jobs by adding dummy jobs, and show that, unfortunately, given n jobs and m related machines, there is no constant k such that a game of $n+k$ jobs on this set of machines is guaranteed to have a NE.

For the equilibrium inefficiency of games with unit-jobs or related machines, we show the following tight bounds.

Theorem 5.2 *If G is a game on related machines and unit-jobs, for which a NE exists, then $PoA(G) < 2$. Also, for every $\epsilon > 0$ there exists a game for which $PoS(G) = 2 - \epsilon$.*

Proof: Let s be a NE profile of G. Let M_1 be a machine with maximal completion time, and let M_2 be a machine for which $(L_i+1)\cdot d_i$ is minimal. If $(L_2+1)\cdot d_2 \geq L_1 \cdot d_1$ then we show that s is optimal: Assume that a better profile, s', exists. It must be that $L_1(s') < L_1(s)$, thus, for some machine M_a, it holds that $L_a(s') > L_a(s)$. Therefore, $L_a(s') \cdot d_a \geq (L_a(s)+1) \cdot d_a \geq (L_2(s)+1) \cdot d_2 \geq L_1(s) \cdot d_1$, contradicting the assumption that s' is better than s.

Assume that $(L_2+1)\cdot d_2 < L_1 \cdot d_1$. Let $j \in S_1$ be a job on M_1. If j has no competitors on M_1, then migrating to M_2 is beneficial. Thus, in order for s to be a NE, there are at least two jobs from S_1 on M_1, therefore, $L_1 \geq 2$. Assume that j migrates to M_2. Its competitors on M_1 will reduce their completion time to $(L_1-1)\cdot d_1$ and j will reduce its completion time to $(L_2+1)\cdot d_2$. If $(L_2+1)\cdot d_2 < (L_1-1)\cdot d_1$, then j's rank is reduced. If $(L_2+1)\cdot d_2 = (L_1-1)\cdot d_1$, then j's rank may reduce or remain the same and its completion time is reduced. In both cases the migration is beneficial. Given that s is a NE, it must be that $(L_2+1)\cdot d_2 > (L_1-1)\cdot d_1$. Also, by the choice of M_2, and given that $(L_2+1)\cdot d_2 < L_1 \cdot d_1$, it must be that $OPT(G) \geq (L_2+1)\cdot d_2$, since in any profile $s' \neq s$, at least one machine processes more jobs than in s. We conclude that

$$PoA(G) \leq \frac{cost(s)}{OPT(G)} \leq \frac{L_1 \cdot d_1}{(L_2+1)\cdot d_2} \leq \frac{L_1 \cdot d_1}{(L_1-1)\cdot d_1} < \frac{L_1}{L_1-1} \leq 2.$$

The last inequality follows from the fact that $L_1 \geq 2$.

The lower bound on the PoS, is achieved already on two machines, and is detailed in the proof of Theorem 5.5. □ ■

5.2 Variable-length Jobs

Our negative results for unit-jobs are clearly valid for variable-length jobs, even with homogeneous competition sets. We are still able to come up with some good news for two machines. We present a linear-time algorithm for calculating a NE, show that any BRD sequence converges to a NE, and provide tight bounds on the equilibrium inefficiency.

Theorem 5.3 *If $m = 2$ and $G \in \mathcal{G}_h$ then G has a NE, and a NE can be calculated efficiently.*

Proof: We present an algorithm for finding a NE on two machines. For every $1 \leq \ell \leq c$, let $P_\ell = n_\ell \cdot p_\ell$. For any profile s, let $\Gamma(s)$ be the set of competition sets fully assigned on M_1, such that $S_\ell \in \Gamma(s)$ iff $(L_1(s) - \lceil n_\ell/2 \rceil\, p_\ell) \cdot d_1 > (L_2(s) + \lceil n_\ell/2 \rceil\, p_\ell) \cdot d_2$. Note that $S_\ell \in \Gamma(s)$ iff a majority of the jobs in S_ℓ benefit from migrating to M_2.

Algorithm 5 Calculating a NE Schedule of a game $G \in \mathcal{G}_h$ on two related machines

1: initialize s to be the profile in which all the jobs are on M_1.
2: **while** $C_1(s) > C_2(s)$ and $\Gamma(s) \neq \emptyset$ **do**
3: let $S_\ell \in \Gamma(s)$ be a competition set with maximal P_ℓ in $\Gamma(s)$.
4: **while** some job $j \in S_\ell$ has a beneficial migration from M_1 to M_2 **do**
5: move j from M_1 to M_2
6: **end while**
7: update $\Gamma(s)$ according to the current profile s.
8: **end while**
9: **if** $C_1(s) \neq C_2(s)$ **then**
10: **while** profile is not a NE **do**
11: move the smallest job that has a beneficial migration.
12: **end while**
13: **end if**

We show the algorithm halts. It is easy to see that it is never beneficial for a job to migrate out of the machine with a lower completion time. Consider the first part of the algorithm (lines 2-6), for every set S_ℓ activated in this iteration, except for the last one, all the jobs of S_ℓ move from M_1 to M_2. Let S_z be the last set activated in the first part, then, by definition of $\Gamma(s)$, a majority of the jobs from S_z have already moved to M_2.

If, after the first part of the algorithm, $C_1(s) > C_2(s)$, then for any set S_ℓ with jobs in M_1, either $\ell = z$, and no job from S_z have a beneficial migration, or all of the jobs of S_ℓ are on M_1, and less than half of them have a beneficial migration to M_2. Therefore, M_1 will remain the machine with the highest completion time, and the best-response sequence will halt.

If, after the first part of the algorithm, $C_2(s) > C_1(s)$, it is clearly not beneficial for a job from S_z to move back to M_1. We show that less than half of the jobs from any other set, S_ℓ, on M_2 may migrate back to M_1. Recall that in the first part of the algorithm, sets are chosen in decreasing P_ℓ. Thus, $P_\ell \geq P_z$. Before the migration of jobs from S_z, M_1 has a higher completion time, and more than half of the jobs in S_z moved to M_2 before it became the more loaded machine. Thus, $C_2(s) - C_1(s)$ is less than $d_2 \cdot P_z/2$. Thus, at most $\left\lfloor \frac{n_\ell - 1}{2} \right\rfloor$ jobs from S_ℓ have a beneficial migration back to M_1. This implies that M_2 remains the machine with the highest completion time, and the best-response sequence halts. □ ■

Theorem 5.4 *If $m = 2$ and $G \in \mathcal{G}_h$ then BRD converges to a NE.*

Proof: Assume that BRD is performed starting from an arbitrary profile. It is easy to see that a migration from M_a to M_b is never beneficial if $L_a \cdot d_a \leq L_b \cdot d_b$ before the migration. Therefore, the only migrations in the BRD are from the machine with the higher completion time. We denote by a *switching migration*, a beneficial migration of a job $j \in S_\ell$ from M_a to M_b such that $L_a \cdot d_a > L_b \cdot d_b$ but $(L_a - p_i) \cdot d_a < (L_b + p_i) \cdot d_b$, that is, the target of the migration becomes the machine with the higher completion time. Note that a job $j \in S_\ell$ that performs a switching migration has the maximal rank in S_ℓ before the migration, and also the maximal rank in S_ℓ after the migration. The migration is beneficial since the number of jobs from S_ℓ on M_b after the migration is higher than their number on M_a before the migration.

Assume by contradiction that BRD does not halt. Since the number of profiles is finite, this implies that BRD loops. Let $C_{max} = L_b \cdot d_b$ denote the maximal cost of a machine during the BRD loop, where M_b can be either the fast or the slow machine. Let t be the first time

in which C_{max} is achieved during the BRD loop. Since a migration out of the machine with lower completion time is never beneficial, C_{max} is a result of a switching migration into M_b, say of $j \in S_\ell$.

Since BRD loops, a job from S_ℓ migrates back to M_a after time t. We claim that such a migration cannot be beneficial. Before the switching migration, $C_j = (L_a(t) + p_j) \cdot d_a$. The switching migration implies that jobs from S_ℓ have lower rank when their completion time is $C_{max} = L_b(t) \cdot d_b$ compared to their rank (with fewer competitors) on M_a with load $L_a(t) + p_j$. Therefore, a migration of $j' \in S_\ell$ to M_a after time t is beneficial only if the load on M_a is less than $L_a(t)$. However, this implies that the load on M_b is more than $L_b(t)$, contradicting the choice of C_{max} as the maximal cost during the BRD loop. □ ■

Theorem 5.5 *If G is a game on two related machines, for which a NE exists, then $PoA(G) < 2$. Also, for every $\epsilon > 0$ there exists a game $G \in \mathcal{G}_h$ for which $PoS(G) = 2 - \epsilon$.*

Proof: Let s be a NE profile of G. If $C_1 = C_2$ then s is optimal. Assume w.l.o.g., that $C_1 > C_2$, that is, $L_1 \cdot d_1 > L_2 \cdot d_2$. Let p be the minimal length of a job on M_1. The stability of s implies that $(L_1 - p) \cdot d_1 < (L_2 + p) \cdot d_2$. Also, for any profile, if the load on M_1 is at most $L_1 - p$ then the load on M_2 is at least $L_2 + p$, implying that the cost of any profile, and in particular the optimal one is higher than $(L_1 - p) \cdot d_1$. If there is more than one job in M_1 then $p \leq L_1/2$, so the POA is less than $\frac{L_1 \cdot d_1}{(L_1/2) \cdot d_1} = 2$. Otherwise there is only one job, of length p, on M_1. Given that s is a NE, this job can not benefit from joining M_2, thus, $P \cdot d_2 > p \cdot d_1$ where P is the total length of all jobs in the instance. Since the job of length p must be processed on a single machine, if there exists a better assignment, it must be that $d_2 < d_1$. The cost of a perfectly balanced assignment is $\frac{P \cdot d_1 d_2}{d_1 + d_2} > \frac{p \cdot d_1^2}{d_1 + d_2} > \frac{p \cdot d_1^2}{2d_1} > \frac{p \cdot d_1}{2}$. Thus, $OPT(G) > cost(s)/2$.

For the lower bound on the PoS, we show that if $d_1 \neq d_2$, then the bound is achieved already with unit-length jobs: given $\epsilon > 0$, consider a game G with two machines having delays $d_1 = 1 - \frac{\epsilon}{2}$ and $d_2 = 1$, and two competing jobs. In the optimal profile, the jobs are assigned on different machines and $C_{max} = 1$. However, this profile is not stable as the job on the slow machine will join its competitor. The best NE assigns the two jobs on the fast machine. Its completion time is $2(1 - \frac{\epsilon}{2})$, implying $PoS(G) = 2 - \epsilon$.

If $d_1 = d_2$, consider a game $G \in \mathcal{G}_h$ with $p_1 = 1, n_1 = 2$ and $p_2 = \delta, n_2 = 1$, where $\frac{2}{1+\delta} = 2 - \epsilon$. Assume w.l.o.g., $d_1 = d_2 = 1$. For this game, $OPT(G) = 1 + \delta$. The only stable profiles are those in which every competition set is assigned on a different machine, thus $PoS(G) = \frac{2}{1+\delta}$. ■

6 Nashification of Race Scheduling Games

In this section we discuss possible strategies of a centralized authority to change the instance or compensate players such that the resulting game has a NE. The first approach we analyze is *addition of dummy jobs*. The cost of such an operation is proportional to the total length of the dummy jobs, as this corresponds to the added load on the system. By Theorem 3.1, it is NP-hard to identify whether Nashification with budget 0 is possible. Thus, the min-budget problem is clearly NP-hard. We present several tight bounds on the required budget.

Theorem 6.1 *Let G be a RSG on m identical machines. Let $p_{max} = max_j p(j)$ be the maximal length of a job in $\mathcal{J}$. It is possible to Nashificate G by adding dummy jobs of total length at*

most $(m-1)p_{max}$*. Also, for every* m *and* $\epsilon > 0$ *there exists a game* G *for which jobs of total length* $(m-1)p_{max} - \epsilon$ *are required to guarantee a NE.*

Proof: We first prove that jobs of total length $(m-1)p_{max}$ are sufficient. We assign the jobs on the machines in an arbitrary NE assignment with respect to their completion time, ignoring the competition sets, e.g., by LPT rule. Any such assignment fulfills, for any two machines M_a and M_b, that $|L_a - L_b| \leq p_{max}$, as otherwise jobs on M_a have a beneficial migration.

Let L_{max} be the maximal load of a machine. For every machine i with load less than L_{max}, we can add a dummy job of length $L_{max} - L_i$ to $\mathcal{J}$ and assign it on M_i. The resulting assignment is perfectly balanced, and therefore, also a NE. The total length of the dummy jobs is at most $(m-1)p_{max}$.

To prove the analysis is tight, given m, ϵ, consider an instance of m machines and a single competition set with two jobs of lengths p and $\epsilon' = \epsilon/(m-1)$. As mentioned in the introduction, this instance has no NE. The only way to guarantee a NE in an extended instance, is by assigning the two jobs on the same machine on or different machines having the same load. In order to avoid a beneficial migration of the ϵ'-job, all the machines must have load at least $p - \epsilon'$. Therefore, an addition of dummy jobs of total length $(m-1)(p-\epsilon') = (m-1)p_{max} - \epsilon$ is inevitable. ∎

For related machines and unit-jobs we showed in Section 5.1 that a game may not have a NE with a single competition set. It is tempting to believe that for such simple instances, Nashification may be achieved by an addition of a constant number of dummy jobs. Our next result shows that $m-2$ dummies may be required, and always suffice.

Theorem 6.2 *For any RSG on* m *related machines and a single competition set of unit-jobs, it is possible to achieve a NE by adding at most* $m-2$ *dummy jobs to the instance. Also, for every* m *there exists a RSG with a single competition set of unit jobs on* m *machines that requires* $m-2$ *dummy jobs to be added in order for a NE to exist.*

Proof: We describe an algorithm that produces a NE by adding at most $m-2$ dummy jobs to the single competition set. The algorithm first assigns the jobs greedily on the machines, and performs a single migration from a machine achieving maximal cost into a most loaded machine among the fastest ones. Then, while the profile is not stable, the algorithm identifies a potential best-response migration of a job j. If this migration increases the cost of job j, then it is performed. However, if the migration is cost-reducing, then job j does not migrate, and a new dummy job is added on the target machine.
Recall that $C_i^+ = (L_i + 1) \cdot d_i$.

Algorithm 6 Nashification of a game G with a single competition set of unit jobs

1: assign the jobs greedily, every job is added to a machine minimizing C_i^+.
2: let M_{high} be a machine such that $C_{high} = \max_i C_i$
3: let M_{fast} be a most loaded machine among the machines having minimal delay. That is, $d_{fast} = \min_i d_i$ and $L_{fast} = \max_{i|d_i=d_{fast}} L_i$.
4: **if** $C_{high} > C_{fast}$ **then**
5: migrate a job from M_{high} to M_{fast}
6: **end if**
7: **while** profile is not a NE **do**
8: let M_a and M_b be machines such that a migration from M_a to M_b is beneficial.
9: **if** $C_a \leq C_b^+$ **then**
10: migrate a job from M_a to M_b. // perform a cost-increasing migration.
11: **else**
12: add a job to M_b. // do not perform a cost-reducing/preserving migration.
13: **end if**
14: **end while**

By the condition in line 7, the algorithm halts with a NE, therefore, we need to show that it halts and that at most $m-2$ jobs are added. A beneficial migration of a job from M_a to M_b may be cost-reducing, that is, $C_a > C_b^+$, or not. In the latter case, the migrating job reduces its rank since its high completion time is now shared with more competitors. A job is added whenever a migration is not cost-increasing. Since the number of consequent cost-increasing migrations is bounded, a new job must be added if the sequence of beneficial migrations is long enough.

After lines 4-5 are performed, M_{fast} is a highest cost machine. This implies that the only beneficial migrations from M_{fast} are cost-reducing, However, such migrations never happen in the algorithm, since a new job is added to the target machine instead. Therefore, M_{fast} remains the highest cost machine throughout the algorithm. Additionally, a migration to a higher cost machine is never beneficial, so it is never beneficial to migrate into M_{fast}. Let C_{fast}^0 denote the completion time of M_{fast} before the while loop. The greedy assignment implies that when the while loop begins, adding two jobs on any machine will make it the most costly machine. Therefore, if one dummy job is added on every machine M_a such that $C_a^+ < C_{fast}^0$, a NE is reached – as any migration will make the target machine the highest cost machine, without increasing the number of jobs sharing the max-rank (as the current highest cost machine is also a fastest one). Therefore, at most $m-1$ dummy jobs will be added.

We prove that an addition of $m-2$ dummy jobs is sufficient. If, when beginning the loop, there are less than $m-1$ machines M_a such that $C_a^+ < C_{fast}^0$ then, by the above, at most $m-2$ dummies will be added. Otherwise, let M_z be a next fastest machine after M_{fast}, that is, $d_z \leq d_i$ for any $M_i \neq M_{fast}$. We show that no dummy job is added on M_z, and that at most $m-2$ jobs in total are assigned on $\mathcal{M} - \{M_{fast}, M_z\}$. We distinguish between two cases:

1. C_z is maximal among the machines in $\mathcal{M} \setminus \{M_{fast}\}$. In this case a migration into a machine M_b is beneficial only if $C_b^+ < C_z$. Therefore, if a dummy job is added on each machine M_i such that $C_i^+ < C_z$, a NE is reached. Since there are at most $m-2$ such machines, at most $m-2$ dummy jobs are required.

2. C_z is not maximal among the machines in $\mathcal{M} \setminus \{M_{fast}\}$. If the instance is not a NE before the while loop, then there is a cost-increasing beneficial migration into M_z from some machine M_a with $C_a > C_z$. After this migration, no dummy job will be added on M_z, and at most one dummy job will be added on M_a, since once one dummy job is added on M_a, its load returns to its level after the greedy assignment, and it is not attractive as a target for a cost-reducing/preserving migration.

For the lower bound, given m, consider a game with $m+4$ unit-jobs and m machines having the following delays: $d_1 = 0.31, d_2 = 0.4, d_3 = 1$, and for all $4 \leq i \leq m, d_i = 0.5 + i\epsilon$.

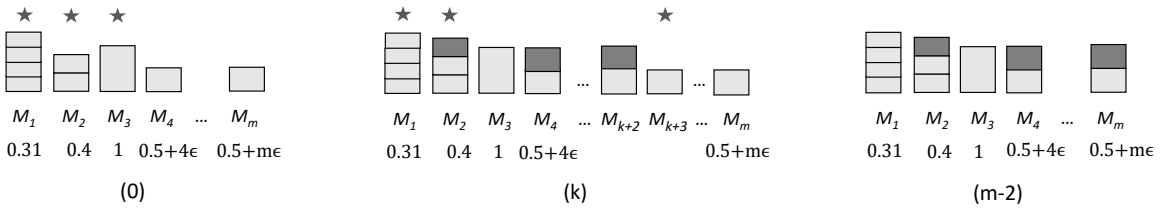

Figure 9: A game for which an addition of $m-2$ jobs is inevitable for Nashification. A BRD loop exists on the starred machines. Profile (0) is a dummy-free profile fulfilling simple stability constrains. Profile (k) fulfills the simple stability constraints after k dummy jobs are added. Profile $(m-2)$ is a NE with $m-2$ dummy jobs.

In any NE, M_1 must have load at least 4, as otherwise, the highest cost is achieved by $M_a \neq M_1$, and a migration from M_a into M_1 increases the number of jobs with highest rank and is beneficial. On the other hand, if $L_1 > 4$, then jobs on M_1 have a beneficial migration into some slower machine. Therefore, in every NE, M_1 has load exactly 4. By similar arguments, M_2 has load at least 2: there are m jobs on the $m-1$ machines $\mathcal{M} \setminus \{M_1\}$. If $L_2 < 2$, then some machine $i > 2$ has load at least 2 and thus, a higher completion time. A deviation from M_i into M_2 is beneficial. Fig. 9(0) presents a profile that fulfils these constraints: $L_1(0) = 4, L_2(0) = 2$, and $L_i(0) = 1$ for $i > 2$. Profile (0) is not a NE: the job assigned on M_3 has a beneficial migration into M_2, to be followed by a migration from M_1 to M_3, and a migration from M_2 to M_1. If the constraints are fulfilled with M_2 having load higher than 2, then one of the other machines is empty, and a job from M_1 has a beneficial migration into it. Now, a job from M_2 has a beneficial migration into M_1 and our BRD loop comes to life.

After k dummy jobs are added, the stability constraints are extended to show they must be on M_2 and the $k-1$ fastest machines among $M_4, \ldots, M_m$. On the other hand, a BRD cycle on M_1, M_2 and M_{k+3} exists (Fig. 9(k)). In order to avoid all these cycles, and get a NE (Fig. 9$(m-2)$), a dummy job must be added on M_2 and every M_i, $4 \leq i \leq m$. Therefore, $m-2$ jobs are required to reach a NE. ∎

A different approach to achieve a NE, is *Nashification by payments*. The cost of a job is $C_j - \gamma_j$ where γ_j is a *compensation* given to the job by the system. A deviating job, will lose the compensation currently suggested to it. The goal is to achieve a NE, while minimizing $\sum_j \gamma_j$. For example, with two competing jobs of length 1 and $1+\epsilon$ on two identical machines, by setting $\gamma_2 = \epsilon$, the optimal schedule is a NE.

Theorem 6.3 *For any RSG G on identical machines, it is possible to achieve a NE with total compensation less than P, where $P = \sum_{j \in \mathcal{J}} p(j)$. Also, for every m and $\epsilon > 0$ there is a game G for which total compensation $P - \epsilon$ is required to achieve a NE.*

Proof: Given a game G, let s be any NE profile of G assuming no competition. Let $C_{min}(s)$ be the minimal completion time of a machine in s. For every job j set $\gamma_j = C_j(s) - C_{min}(s)$. The stability of s implies that $\gamma_j \leq p_j$, as otherwise, job j has a beneficial migration to a machine with load $C_{min}(s)$. With the compensation, for all jobs, $C_j(s) - \gamma_j = C_{min}(s)$, thus, in every competition set, all the jobs have the same rank. Any deviation will lead to cost at least $C_{min}(s) + p(j)$ and no compensation, and can only reduce the number of jobs sharing the max-rank. Thus, s is a NE.

For the lower bound, given $\epsilon > 0$, and an integer x, consider the following game on $m = x+1$ machines. Let $\delta = \frac{\epsilon x}{x+1}$. There are x competition sets, each with a single job of lengths 1 and a single job of length $\frac{\delta}{x^2}$. Note that $P = x + \frac{\delta}{x}$. In every assignment, some machine, say M_0, has load at most δ/x. For every set, if the small job is on a machine with load at least 1, then it should be compensated for not escaping to M_0. If the small job is on M_0, then the long job should be compensated for not joining it. Thus, a total compensation of at least $x(1 - \delta/x) = P - \epsilon$ is required. ∎

7 Conclusions and Directions for Future Work

Our paper suggests a new model for analyzing environments with strong competition. *Race games* are congestion games in which players' welfare depends on their relative performance. The main objective of a player is to perform well relative to his competitors, while minimizing his cost is a minor objective. A profile is a NE if no player can improve her rank, or reduce her cost while keeping her rank.

We analyzed job-scheduling race games on parallel machines. Having an additional constraint for stability, race games are less stable than classical load-balancing games, thus our results for general games are mostly negative. In particular, for all the classes of instances we considered, we showed that $PoS = PoA$, while the same competition-free game has a lower PoA and $PoS = 1$. Practically, it means that competition may lead to a poor outcome even if the system can control the initial players' strategies. Striving for stability, we also studied the cost of Nashification, by either adding dummy jobs to the instance, or compensating jobs for having high rank. While in the general case, Nashification may involve balancing all the machines or jobs' cost, in some natural classes it can be achieved in cheaper ways. Min-cost Nashification of a given instance is NP-complete. We leave open the corresponding approximation problem.

Race games can be studied in various additional settings. In fact, every congestion game in which players are associated with a utility has its 'race' variant. We list below several examples. Additional questions may refer to the structure of the competition-sets, for example, competition sets may overlap, or may be defined according to the players' strategy space (symmetric competition sets). The study of coordinated deviations is another intriguing direction. In the presence of competition, coalitions may be limited to include only members of different competition sets. On the other hand, temporal collaboration, may be fruitful even for competing players. Thus, there are many different interesting variants of coordinated deviations in race games.

7.1 Additional Race Congestion Games

We describe below several natural games whose 'race' version is interesting and different from their classical min-cost game. For each of these games, it is desired to analyze the impact of various competition structures. We provide examples and simple initial results that are either based on or extend some of the results in this paper.

Semi-symmetric Matroid Race Game: Consider a race congestion game with m resources (the set $\mathcal{M}$). The strategy space of player j is any subset of $p(j) \leq m$ resources. A profile of the game is given by specifying, for every player j, the set $E(j)$ of resources allocated to j such that $|E(j)| = p(j)$. Note that this is a special class of matroid games, in which all the resources are feasible to all players, but each player has a different demand. The cost of a player is the total load on the resources he uses. We present an algorithm for computing a NE for any partition of the players to competition sets (not necessarily homogeneous). On the other hand, if different resources may have different cost functions, then a NE may not exist even in the simple case of $p(j) = 1$, since the resulting game is equivalent to a RSG on related machines and unit-jobs, discussed in Section 5.1. It would be interesting to study additional variants, possibly with restricted sets of resources, and limited structure of competition sets.

Algorithm 7 Calculating a NE profile of a semi-symmetric matroid race game

1: **for** $j = 1$ to n **do**
2: let $E(j)$ be a set of $p(j)$ lightly loaded resources
3: **end for**
4: **while** there are two resources a and b, such that $L_a = L + 1, L_b = L$, and there exist a set S_ℓ such that $1 \leq n_\ell(a) \leq n_\ell(b)$, and there exists a player $j \in S_\ell$ such that $a \in E(j)$ and $b \notin E(j)$ **do**
5: $E(j) = E(j) \cup \{b\} \setminus \{a\}$.
6: **end while**

We show that the algorithm terminates with a NE. Let $T = \sum_j p(j)$, $L = \lfloor \frac{T}{m} \rfloor$ and $h = T\%m$. Since the jobs are assigned greedily on lightly loaded resources. lines 1-3 produce a feasible assignment in which h resources have load $L + 1$ and $m - h$ resources have load L. We show that the loop in lines 4-6 halts, and then show that it halts with a NE. For a profile s, let $\phi(s) = \sum_{i \in \mathcal{M}} \sum_{\ell=1}^{c} n_\ell(i)^2$. We show that ϕ is a potential function that increases with every swap performed in line 5. Clearly, the contribution of resources in $\mathcal{M} \setminus \{a, b\}$ does not change. For a and b, the change in ϕ is $\Delta(\phi) = (n_\ell(a) - 1)^2 + (n_\ell(b) + 1)^2 - n_\ell(a)^2 - n_\ell(b)^2 = 2(n_\ell(b) - n_\ell(a) + 1) > 0$. The last inequality follows from the migration condition $n_\ell(a) \leq n_\ell(b)$.

Consider the profile after the algorithm terminates. We show that no player has a beneficial deviation. Let $j \in S_\ell$. First note that it is never beneficial to replace a resource a whose load is X by a resource b whose load is at least X. Such a deviation will increase the load on b to be at least $X + 1$, and therefore j's cost would increase by at least 1. It also increases the cost of its competitors on the target resource by 1, but can never reduce j's rank. Combine the above with the fact that all loads are exactly C or $C + 1$, we conclude that the only beneficial deviations replace a resource, a, whose load is $C + 1$ by a resource, b, whose load is at most C. Such a deviation does not change j's cost; it reduces by one the cost of a's users, and increases by one the cost of b's users. Thus, such a deviation reduces j's rank if and only if

$n_\ell(a) \leq n_\ell(b)$. Note that j itself contributes to $n_\ell(a)$, so the number of competitors that are hurt is strictly larger than the number of competitors that benefit from j's migration. Since this is exactly the condition detected in line 4, no beneficial deviation exists when the algorithm terminates. Finally, note that the maximal value of $\phi(s)$ is $O(n^2)$, therefore the convergence is within polynomial time.

Routing Games: In this classical congestion game, every player needs to select a path from his source vertex to his target vertex. Edges are associated with non-decreasing latency functions. The special case of parallel link network corresponds to job-scheduling, thus, with weighted players, there is a very simple game with no-NE for two players. Moreover, no NE exists even in a symmetric game with a single competition set, unweighted players and, uniform latency edges. Consider for example 5 players and a network consisting of three disjoint parallel $s-t$ paths, of 4,5, and 6 edges. It can be seen that this game has no NE. The analysis is similar to the analysis of the RSG on 3 related machines (Fig. 8). It would be interesting to identify classes of routing game for which a NE exists and can be computed. In this setting, it is also interesting to study the cost of Nashification by performing changes in the network.

Cost-Sharing Games: In cost-sharing games, players want to use resources that are used by other players, as it reduces the player's share in the resource cost. However, players may avoid sharing a resource since their competitors may also benefit from this sharing. The study of race cost-sharing games, and in particular network formation games, can quantify the impact of competition in games with positive congestion effect.

References

[1] A. Anagnostopoulos, L. Becchetti, B. Keijzer, and G. Schäfer. Inefficiency of Games with Social Context. *Theor. Comp. Sys.* 57(3):782 —804, 2015.

[2] N. Andelman, M. Feldman, and Y. Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289 – 317, 2009.

[3] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

[4] G. Avni and T. Tamir. Cost-sharing scheduling games on restricted unrelated machines. *Theoretical Computer Science*, 646:26 – 39, 2016.

[5] V. Bilò, A. Celi, M. Flammini, and V. Gallotti. Social Context Congestion Games. In *Proc. 18th SIROCCO*, pages 282-–293, 2011.

[6] R. J. Aumann. Rule-Rationality versus Act-Rationality. *Discussion Paper Series* DP497, The Federmann Center for the Study of Rationality, the Hebrew University, Jerusalem, 2008.

[7] Y. Azar, K. Jain, and V. Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 323–332, Philadelphia, PA, USA, 2008.

[8] P.A. Chen, B. de Keijzer, D. Kempe, and G. Schäfer. The robust price of anarchy of altruistic games. In *Proc. 7th WINE*, pages 383-–390, 2011.

[9] K. Bhawalkar, M. Gairing, and T. Roughgarden. Weighted Congestion Games: The Price of Anarchy, Universal Worst-Case Examples, and Tightness. *ACM Trans. Econ. Comput.* 2(4), Article 14, 2014.

[10] G. E. Bolton and A. Ockenfels. Erc: A theory of equity, reciprocity, and competition. *The American Economic Review*, 90(1):166–193, 2000.

[11] I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos, and L. Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2011.

[12] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[13] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *ACM Trans. Algorithms*, 3(1):4:1–4:17, 2007.

[14] E. Fehr and K. M. Schmidt. A theory of fairness, competition, and cooperation. *The Quarterly Journal of Economics*, 114(3):817–868, 1999.

[15] M. Feldman and T. Tamir. Conflicting congestion effects in resource allocation games. *Journal of Operation Research*, 60(3):529–540, 2012.

[16] A. Fiat, H. Kaplan, M. Levi, and S. Olonetsky. Strong price of anarchy for machine load balancing. In *Proc. 34th Int. Colloq. on Automata, Languages, and Programming*, 2007.

[17] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, 19(3):312–320, 1979.

[18] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spiraklis. The Structure and Complexity of Nash Equilibria for a Selfish Routing Game. In *Proc. 29th Int. Colloq. on Automata, Languages, and Programming*, pages 510–519, 2002.

[19] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[20] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

[21] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM*, 34(1):144–162, 1987.

[22] K. Jansen, K. M. Klein, , and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 72, pages 1–13, 2016.

[23] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[24] C. H. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.

[25] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

[26] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[27] A. S. Schulz and N. Stier Moses. On the performance of user equilibria in traffic networks. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 86–87, Philadelphia, PA, USA, 2003.

[28] B. Vöcking. *Algorithmic Game Theory*, chapter 20: Selfish Load Balancing. Cambridge University Press, 2007.

[29] E. Winter, L. Méndez-Naya, and I. García-Jurado. Mental equilibrium and strategic emotions. *Management Science*, 63(5):1302–1317, 2017.