Race Scheduling Games

Shaul Rosner and Tami Tamir

School of Computer Science, The Interdisciplinary Center, Israel
`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 [20]. 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., [19,10,12,3], and a survey in [24]). Job-scheduling is a weighted congestion game [21] 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 an 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 $C^s_{S_\ell} = \langle 7, 8, 8, 13\rangle$ 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 $C_j(s)$. We have,

Observation 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 $C_j(s') < C_j(s)$.

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

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 ℓ, all the jobs in 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 $\frac{2+3+4+5}{4} = 3.5$. Profile (a) is a NE. For example, a deviation of a 4-job from M_1 to M_2 leads to Profile (b), and thus involves an increase in the rank of the deviating jobs from 3 to 3.5. It can be verified that other deviations are not beneficial either. 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.

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

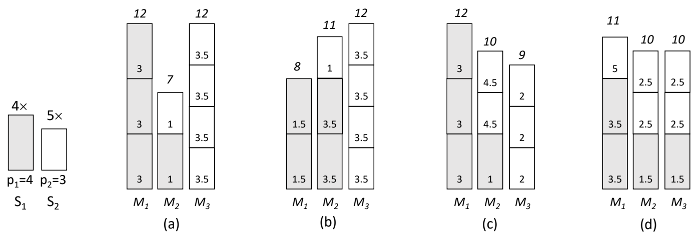

Fig. 1. Jobs are labeled by their ranks. (a) A NE profile. (b) and (c) Deviations from the NE are harmful. (d) An LPT schedule.

inefficiency incurred due to self-interested behavior according to the *price of anarchy* (PoA) [19] and *price of stability* (PoS) [2,23] 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.

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 entity. This is a well-studied NP-complete problem. For identical machines, the simple greedy List-scheduling (LS) algorithm [15] 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 [16], and A PTAS is given in [17]. For related machines, with various speeds, LS algorithm provides a $\theta(m)$-approximation [9], and a PTAS is given in [18].

Congestion games [21] 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 $p(j)$ [7].

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 [24]. For identical machines, it is known that LPT-schedules are NE schedules [14], and that the price of anarchy, which corresponds to the makespan approximation, is $2-\frac{2}{m+1}$ [13]. For uniformly related machines, the price of anarchy is bounded by $\frac{\log m}{\log\log\log m}$ [10]. For two machines, a bound of $\frac{1+\sqrt{5}}{2}$ is given in [19].

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, [25] 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 [4,6,11]. 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 unintuitive, and known tools and techniques fail even on simple instances. We start by analyzing RSGs on identical machines. We show that an RSG need not have a NE, and deciding whether a game instance has a NE is a 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 $PoA = 2 - \frac{2}{m+1}$ [13]. 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 $PoA = \frac{1+\sqrt{5}}{2}$ [19], 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 an 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. A full version that includes all the proofs is available in [22].

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.

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. 2, games in this class, even with only three sets and three machines, may not admit a NE. Moreover, as demonstrated in Fig. 3, even if a NE exists, it may be the case that a 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 2. *Given an instance of RSG with homogeneous competition sets, it is NP-complete to decide whether the game has a NE profile.*

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.

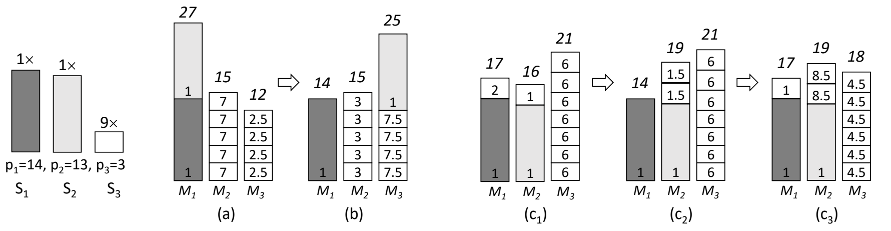

Fig. 2. An example of an RSG with homogeneous competition sets, that has no NE. Jobs are labeled by their ranks. Profiles (a)-(b) show that big jobs must be on different machines. Profiles $(c_1)-(c_2)-(c_3)-(c_1)$ loop when big jobs are on different machines.

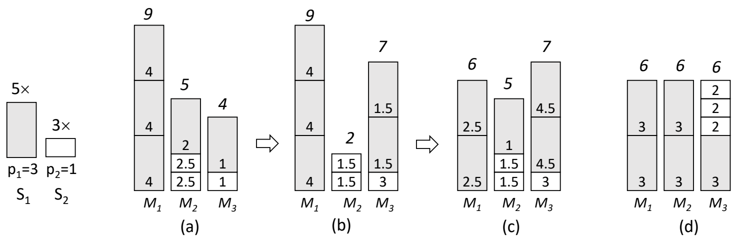

Fig. 3. An example of an RSG with homogeneous competition sets in which $c = 2$, $p_1|p_2$, and BRD may loop (profiles (a)-(b)-(c)-(a)). A NE exists (profile (d)). Jobs are labeled by their ranks.

Theorem 3. *If all jobs have the same length, then a NE exists and can be computed efficiently.*

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. 2, in which there are three sets on three machines.

Theorem 4. *If $G \in \mathcal{G}_h$ has $c = 2$ and $m = 3$, then a NE exists and can be computed efficiently.*

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, in the full version we show that it is NP-hard to decide if a NE exists even if there are only singletons and two competing pairs. For games with homogeneous competition sets, in which there are only singleton and pairs, we have positive news.

Theorem 5. *If $G \in \mathcal{G}_h$, and for all ℓ, $|S_\ell| \leq 2$ then a NE exists and can be computed efficiently.*

In search of more positive results, we turn to look at games with homogeneous competition sets with divisible lengths. Instances with divisible lengths arise often in applications in which clients can only select several levels of service. Moreover, naturally, in such settings, clients with similar service requirements compete with each other. 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. 3, 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 $G \in \mathcal{G}_{div}$. In the proof we provide an algorithm for computing a NE for instances in this class.

Theorem 6. *If $G \in \mathcal{G}_{div}$, then a NE exists and can be computed efficiently.*

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. In the full version [22], 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, the proof in the full version is less interesting. However, it can serve as a warm-up for the lower bound PoS proof, which is more involved.

Theorem 7. *If G is an RSG on m identical machines that has a NE, then $PoA(G) \leq 3 - \frac{6}{m+2}$. For every $m \geq 3$ and $\epsilon > 0$, there exists an RSG G on m identical machines such that G has a NE, and $PoS(G) \geq 3 - \frac{6}{m+2} - \epsilon$.*

Proof. The proof of the PoA upper bound is given in the full version [22]. We describe the lower bound on the PoS. Given $m \geq 3$ and $\epsilon > 0$, we describe an RSG G such that 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 $\ell \geq 0$.

Claim. In every NE, the three jobs of S_0 are assigned on the same machine, and for all $\ell \geq 1$, the two jobs of S_ℓ are assigned on the same machine.

By the above claim, the cost of every NE is at least the load incurred by the three jobs in S_0, that is, $3m - \sum_{i=1}^{3} \delta_i$. We show that a NE of this cost exists. An example for $m = 5$ is given in Fig. 4(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.

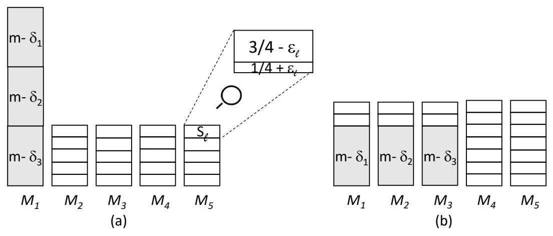

Fig. 4. A tight example for $m = 5$. (a) The only NE profile. (b) an optimal profile.

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. 5, 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).

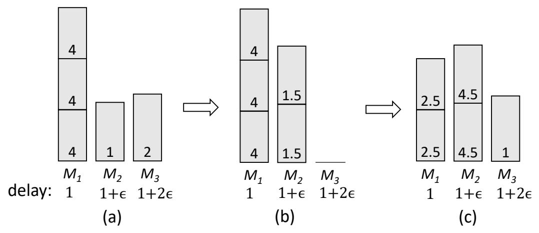

Fig. 5. No NE of an RSG with five competing unit-jobs on three related machines. The profiles loop (a)-(b)-(c)-(a). Jobs are labeled by their ranks.

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 8. *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.*

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 9. *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 $PoS(G) = 2 - \epsilon$.*

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

Theorem 11. *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 a 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 12. *If G is an RSG 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 $PoS(G) = 2 - \epsilon$.*

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 2, 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 13. *Let G be an 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.*

For related machines and unit-jobs we showed in Section 5.1 that a game may not have a NE even 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 14. *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 an 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. 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 $4 \le i \le m, d_i = 0.5 + i\epsilon$. Fig. 6 shows the behaviour of such an instance. A full description of this game, as well as an algorithm that produces a NE by adding at most $m-2$ dummy jobs is provided in the full version [22]. □

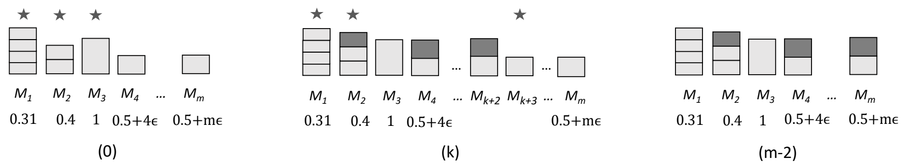

Fig. 6. A game for which an addition of $m-2$ jobs is inevitable for Nashification. A BRD loop exists on the starred machines. Profile (0) is a dummy-free profile fulfilling simple stability constrains. Profile (k) fulfills the simple stability constraints after k dummy jobs are added. Profile $(m-2)$ is a NE with $m-2$ dummy jobs.

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 15. *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.*

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. In the full version we list 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.

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. 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.
3. G. Avni and T. Tamir. Cost-sharing scheduling games on restricted unrelated machines. *Theoretical Computer Science*, 646:26 – 39, 2016.
4. R. J. Aumann. Rule-Rationality versus Act-Rationality. *Discussion Paper Series* DP497, The Hebrew University's Center for the Study of Rationality, 2008.
5. V. Bilò, A. Celi, M. Flammini, and V. Gallotti. Social Context Congestion Games. In *Proc. 18th SIROCCO*, pages 282-–293, 2011.
6. G. E. Bolton and A. Ockenfels. Erc: A theory of equity, reciprocity, and competition. *The American Economic Review*, 90(1):166–193, 2000.
7. 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.
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. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.
10. A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *ACM Trans. Algorithms*, 3(1):4:1–4:17, 2007.
11. E. Fehr and K. M. Schmidt. A theory of fairness, competition, and cooperation. *The Quarterly Journal of Economics*, 114(3):817–868, 1999.
12. A. Fiat, H. Kaplan, M. Levi, and S. Olonetsky. Strong price of anarchy for machine load balancing. In *Proc. 34th ICALP*, 2007.
13. G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, 19(3):312–320, 1979.
14. 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 ICALP*, pages 510–519, 2002.
15. R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.
16. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.
17. 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.
18. K. Jansen, K. M. Klein, , and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *Proc. 43rd ICALP*, pages 1–13, 2016.
19. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.
20. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2008.
21. R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
22. S. Rosner and T. Tamir. Race Scheduling Games. https://cs.idc.ac.il/~tami/Papers/RSG-full.pdf
23. A. S. Schulz and N. Stier Moses. On the performance of user equilibria in traffic networks. In *Proc. 43rd SODA,* pages 86–87, 2003.
24. B. Vöcking. *Algorithmic Game Theory*, chapter 20: Selfish Load Balancing. Cambridge University Press, 2007.
25. E. Winter, L. Méndez-Naya, and I. García-Jurado. Mental equilibrium and strategic emotions. *Management Science*, 63(5):1302–1317, 2017.