The Power of One Secret Agent

Tami Tamir
School of Computer Science, The Interdisciplinary Center (IDC), Herzliya, Israel
tami@idc.ac.il

Abstract

I am a job. In job-scheduling applications, my friends and I are assigned to machines that can process us. In the last decade, thanks to our strong employee committee, and the rise of algorithmic game theory, we are getting more and more freedom regarding our assignment. Each of us acts to minimize his own cost, rather than to optimize a global objective.

My goal is different. I am a secret agent operated by the system. I do my best to lead my fellow jobs to an outcome with a high social cost. My naive friends keep doing the best they can, each of them performs his best-response move whenever he gets the opportunity to do so. Luckily, I am a charismatic guy. I can determine the order according to which the naive jobs perform their best-response moves. In this paper, I analyze my power, formalized as the *Price of a Traitor* (PoT), in cost-sharing scheduling games – in which we need to cover the cost of the machines that process us.

Starting from an initial *Nash Equilibrium* (NE) profile, I join the instance and hurt its stability. A sequence of best-response moves is performed until I vanish, leaving the naive jobs in a new NE. For an initial NE assignment, S_0, the PoT measures the ratio between the social cost of a worst NE I can lead the jobs to, starting from S_0, and the social cost of S_0. The PoT of a game is the maximal such ratio among all game instances and initial NE assignments.

My analysis distinguishes between instances with unit- and arbitrary-cost machines, and instances with unit- and arbitrary-length jobs. I give exact bounds on the PoT for each setting, in general and in symmetric games. While it turns out that in most settings my power is really impressive, my task is computationally hard (and also hard to approximate).

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Job scheduling games, Cost sharing, Equilibrium inefficiency

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.32

1 Introduction

I am a job. In job-scheduling applications, my friends and I are assigned to machines that can process us. The authorities that assign us to machines like to analyze the way we are assigned. They treat us as instances of combinatorial optimization problems, and our assignment became a major discipline in operations research. In the old days, we were all controlled by a centralized scheduler who assigned us in a way that achieves an effective use of the system's resources, or a target quality of service [20]. In the last decade, thanks to our strong employees committee, and also the rise of algorithmic game theory, we are getting more and more freedom regarding our assignment. Many modern systems provide service to multiple strategic users, whose individual payoff is affected by the decisions made by other users of the system. As a result, non-cooperative game theory has become an essential tool in the analysis of our assignment [21, 15, 24, 4, 12, 3]. Each of us has strategic considerations and acts to minimize his own cost, rather than to optimize any global objective. Practically, this means that we *choose* a machine instead of being assigned to one by a centralized scheduler.

9th International Conference on Fun with Algorithms (FUN 2018).
Editors: Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe; Article No. 32; pp. 32:1–32:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

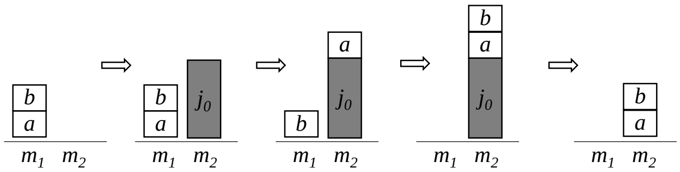

Figure 1 A simple example of a traitor BR-sequence with PoT= 2.

My goal is different, I am not the regular job you are used to analyze. Already in my childhood I was a problematic kid and my parents were invited regularly to school to discuss my behavior[1]. Recently, I started to work as a secret agent, operated by the system. My mission is to join a stable assignment of other jobs, perturb its stability, and lead a sequence of best-response moves, whose outcome is as poor as possible. When I'm done, I vanish, leaving the other jobs in a new stable profile, whose cost is hopefully higher. My naive friends keep doing the best they can, each of them performs his best-response move whenever he gets the opportunity to do so. Luckily, I am a charismatic guy; I can determine the order according to which the naive jobs deviate.

In this paper, I analyze my power, formalized as the *Price of a traitor* (PoT), in cost-sharing scheduling games. In these games every job has a subset of the machines on which it can be assigned, and the cost of every utilized machine is shared by the job assigned to it, where the share is proportional to the load generated by the jobs. My goal is to lead the jobs into a stable assignment in which the total cost of utilized machines is maximal. Before diving into the details, let me demonstrate my mission on a small example.

Example 1: Consider an instance with two machines m_1 and m_2 of costs 1 and 2 respectively. Assume that two naive jobs of length 1 are assigned on m_1 (see leftmost assignment in Figure 1). The cost of each of them in this initial profile is $1/2$. Assume that my length is $3+\epsilon$, and I appear and assign myself on m_2 (I am Job 0 - the gray guy in the figure). Since $2/(4+\epsilon) < 1/2$ each of the naive jobs will benefit from joining me. So they join me one after the other. Once we are all on m_2, I vanish. The jobs are left on the more expensive machine (rightmost assignment in Figure 1), and their assignment is stable, since they each pay 1, and a unilateral deviation to m_1 will also result in this cost. My mission is completed with a NE whose cost is doubled.

1.1 Preliminaries

An instance of a cost-sharing game with a traitor (CST) is given by a tuple $G = \langle \mathcal{J}, \mathcal{M}, \{M_j\}_{j\in\mathcal{J}}, p_0\rangle$, where $\mathcal{M}$ is a set of m machines, and $\mathcal{J}$ is a set of k *naive* jobs. Not all machines are feasible to all jobs. For each $j \in \mathcal{J}$, the machines that may process Job j are given by the set $M_j \subseteq \mathcal{M}$. Every job $j \in \mathcal{J}$ has processing time p_j which is independent of the machine on which it is assigned. Every machine $i \in \mathcal{M}$ has an activation cost, $c(i)$. The last component of the tuple specifies my length - the processing time of the *traitor*. Throughout this paper, I am denoted Job 0.

Every job is a player, where the strategy space of Job j is the set of machines in M_j. A profile of a CST game is a vector $S = \langle s_0, s_1, \ldots, s_k\rangle \in ((\mathcal{M} \cup \{\perp\}) \times M_1 \times \ldots \times M_k)$,

[1] Enthusiastic fans of the conference *FUN with algorithms* may recognize me as a bully job in [23].

describing the machines selected by the jobs. My strategy, s_0, is in $\mathcal{M} \cup \{\perp\}$, meaning that I can go to any machine and also be away, in which case $s_0 = \perp$. A profile in which $s_0 = \perp$ is denoted a *traitor-free* profile. For a machine $i \in \mathcal{M}$, the *load* on i in S, denoted $L_i(S)$, is the total processing time of the jobs assigned to machine i in S, that is, $L_i(S) = \sum_{\{j|s_j=i\}} p_j$. When S is clear from the context it is omitted.

A machine i is *utilized* in a profile S if $L_i(S) > 0$. The cost of a utilized machine is covered by the jobs assigned to it, where the share is proportional to the load generated by the jobs. Formally, the cost of Job j in the profile S is $cost_j(S) = c(s_j) \cdot \frac{p_j}{L_{s_j}(S)}$. This cost-sharing scheme fits the commonly used proportional cost-sharing rule for weighted players, (e.g., [21, 1, 11]).

Consider a game G. For a profile S, a job j, and a strategy $s'_j \in M_j$, let (S_{-j}, s'_j) denote the profile obtained from S by replacing the strategy of Job j by s'_j. That is, the profile resulting from a migration of Job j from machine s_j to machine s'_j. A profile s is a *pure Nash equilibrium* (NE) if no job can benefit from unilaterally deviating from his strategy in S to another strategy; i.e., for every job j and every strategy $s'_j \in M_j$ it holds that $cost_j((S_{-j}, s'_j)) \geq cost_j(S)$. This paper considers only *pure* strategies. Unlike mixed strategies, pure strategies may not be random or drawn from a distribution.

Given a profile S, the *best response* (BR) of Job j is $BR_j(S) = \arg\min_{s'_j \in M_j} cost_j(S_{-j}, s'_j)$; i.e., a machine i such that Job j's cost will be minimized if he is assigned to machine i, fixing the assignment of all other jobs. If there are several such machines, each of them is considered a best-response. *Best-Response Dynamics* (BRD) is a local-search method where in each step some player is chosen and plays his BR.

A naive job j is said to be *suboptimal* in a profile S if he can reduce his cost by migrating to another machine, i.e., if $s_j \notin BR_j(S)$. Given an initial profile S_0, a *traitor BR-sequence* from S_0 is a sequence of profiles $\langle S_0, S_1, \ldots S_T \rangle$ in which for every $t = 0, 1, \ldots$, either there exists a naive job j such that $S_{t+1} \in (S_{t_{-j}}, BR_j(S_t))$, or $S_{t+1} = (S_{t_{-0}}, s'_0)$. In other words, either a naive job performs a BR move or I perform a move of my choice – even if it is not beneficial for me. I am interested in traitor BR-sequences in which both S_0 and S_T are traitor-free NEs. The stability of S_0 is perturbed once I arrive and select some machine. Formally, in S_0 my strategy is $\perp$, and no naive job is suboptimal. Then, $S_1 = (S_{0_{-0}}, s'_0)$ for $s'_0 \in \mathcal{M}$. The last profile in a traitor BR-sequence is also traitor-free, that is $s_0(S_T) = \perp$. For a profile S_0, let TNE(S_0) be the set of Nash equilibria reachable from S_0 via a traitor BR-sequence. If my departure leaves the naive jobs in a non-stable profile, they will keep forming BR-moves until they converge to a NE (by [2] this will surely happen).

The social cost of a profile S is the total cost of resources utilized in S, which is equal to the total cost of the players. Formally, $cost(S) = \sum_{j \in \mathcal{J} \cup \{0\}} cost_j(S) = \sum_{i \in \cup_j s_j} c(i)$. Note that I pay my part in utilizing a machine that I share with others – this is essential also to keep my reliability among the naive jobs. However, the fact that the final NE in the sequence is traitor-free guarantees that I cannot force a very expensive outcome by selecting an expensive machine for myself.

Let NE(G) be the set of Nash equilibria in a CST game G. Being a weighted cost-sharing game with singleton strategies, it is well known that NE(G) $\neq \emptyset$ and that BRD converges to a NE [2]. Recall that TNE(S_0) is the set of traitor-free Nash equilibria reachable from a traitor-free NE S_0 via a traitor BR-sequence.

The *Price of a Traitor* in a game G, denoted PoT(G), is defined as the worst ratio, among all initial traitor-free NE profiles S_0, between the social cost of a NE in TNE(S_0) and the social cost of S_0. I.e.,

$$PoT(G) = \sup_{S_0 \in \mathrm{NE}(G)} \max_{S \in \mathrm{TNE}(S_0)} \frac{cost(S)}{cost(S_0)}.$$

For a class of games $\mathcal{G}$, the price of a traitor with respect to $\mathcal{G}$ is defined as the worst-case PoT over all games in $\mathcal{G}$. That is, $\mathrm{PoT}(\mathcal{G}) = \sup_{G \in \mathcal{G}}\{\mathrm{PoT}(G)\}$.

It is well known that NE profiles may be sub-optimal. Let $OPT(G)$ denote the minimal possible social cost of a feasible assignment of $\mathcal{J}$, i.e., $OPT(G) = \min_S cost(S)$. The inefficiency incurred due to self-interested behavior is quantified according to the *price of anarchy* (PoA) [15, 19] and *price of stability* (PoS) [1] 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, $\mathrm{PoA}(G) = \max_{S \in NE(G)} cost(S)/OPT(G)$, and $\mathrm{PoS}(G) = \min_{S \in NE(G)} cost(S)/OPT(G)$.

The following observation bounds my power for any game instance.

▶ **Observation 1.** *For every game G, $1 \le PoT(G) \le \frac{PoA(G)}{PoS(G)}$.*

Proof. For every initial NE profile, S_0, it holds that $cost(S_0) \ge OPT(G) \cdot \mathrm{PoS}(G)$. Also, for every $S \in \mathrm{TNE}(S_0)$, it holds that $cost(S) \le OPT(G) \cdot \mathrm{PoA}(G)$. Therefore,

$$\mathrm{PoT}(G) \le \max_{S \in \mathrm{TNE}(S_0)} \frac{cost(S)}{cost(S_0)} \le \frac{OPT(G) \cdot \mathrm{PoA}(G)}{OPT(G) \cdot \mathrm{PoS}(G)} = \frac{\mathrm{PoA}(G)}{\mathrm{PoS}(G)}.$$

Also, since $S_0 \in \mathrm{TNE}(S_0)$, it holds that $\mathrm{PoT}(G) \ge 1$. ◀

Related work: I am not a young job. I participated in many assignments in my life, and I always tried to analyze the performance of these assignments. In addition, I'm trying to follow the huge effort done by researchers in analyzing our assignments. Before the rise of algorithmic game theory, most of the study dealt with achieving a global objective of the assignment such as load balancing, minimizing our total completion time, or the makespan (corresponding to the maximal cost of some job) [20].

In the last decade, game-theoretic analysis became an important tool for analyzing our assignments, as many other systems in which a set of resources is shared by selfish users. *Congestion games* 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 [21, 1]. CST games are congestion games with singleton strategies, in which each resource has an activation cost that is shared by the players using it according to some sharing mechanism. A generalized, traitor-free, model of this game, in which the processing times of jobs depend on the machines they are assigned to was studied in [17, 2].

Best-Response dynamics corresponds to actual dynamics in real life applications. They are therefore starring in the study of non-cooperative game theory [1, 13, 8]. The important questions are whether BRD converges to a NE, if one exists [17, 12]; what is the converges time [5, 6, 22, 13]; and what is the quality of the solution [8]. The paper [22] studies the complexity of equilibria in a wide range of cost sharing games.

Other related work deal with games in which some of the players are not selfish. For example, in the Stackelberg model [14, 10, 7], a fraction of the jobs are selfish, while the rest are willing to obey a centralized authority. A Stackelberg strategy assigns the controllable jobs, trying to minimize the inefficiency caused by the others.

Table 1 The non-restricted jobs and their initial profile in the CST game constructed in the reduction.

Job j	p_j	M_j	$S_0(j)$	$cost_0(j)$
a	1	$\{m_1, m_2, m_3\}$	m_1	1/4
b	1	$\{m_2, m_4\}$	m_2	1/2
c	2	$\{m_3, m_5\}$	m_5	2/5
d	2	$\{m_3, m_5\}$	m_5	2/5
$1, \ldots, n$	a_j	$\{m_1, m_2\}$	m_1	$a_j/4$

Games in which some players are adversarial were defined and study in the areas of Cryptography [16, 18] and Rational Synthesis [9]. However, the goals and the allowed actions of the malicious players in these games are different, and their analysis is not relevant to my power.

2 Unit-cost Machines

In this section I study my power in an environment of unit-cost machines. The cost of a profile is simply the number of utilized machines. My goal is therefore to activate as many new machines, and keep them utilized also after my departure. Unfortunately, it turns out that my ambition exceeds my ability: in order to achieve my goal, I need to solve an NP-hard problem. Moreover, the reduction below presents an instance for which (i) $cost(S_0) = 4$, (ii) for every $S_T \in TNE(S_0)$ it holds that $cost(S_T) \in \{4, 5\}$, and (iii) an NP-hard problem should be solved in order to lead the jobs to a profile of cost 5. This implies that it is unlikely to have an algorithm that approximates my potential damage with ratio better than 4/5, and thus, my mission is APX-hard.

▶ **Theorem 2.** *My task is APX-hard even with a constant number of unit-cost machines.*

Proof. I show a reduction from the Partition problem. The input is a set I of positive numbers $\{a_1, a_2, ..., a_n\}$ such that $\forall j,\ 0 < a_j < 1$ and $\sum_j a_j = 2$. The goal is to decide whether there exists a subset $I_1 \subset I$, such that $\sum_{j \in I_1} a_j = \sum_{j \in I \setminus I_1} a_j = 1$. Given an instance of Partition, consider the CST game and initial profile depicted in Figure 2(a). The game is played on $\mathcal{M} = \{m_1, m_2, m_3, m_4, m_5\}$, where $\forall i, c(m_i) = 1$. The are $n + 8$ naive jobs. Four jobs of length 1 are *restricted*. Each of them is restricted to go to a different single machine, m_1, m_2, m_4 or m_5. These are the gray jobs in the figure. Since $|M_j| = 1$ for each of these jobs, they will not participate in the BR-sequence. The restricted jobs guarantee that the cost of every profile is at least 4. The lengths, possible strategies, and initial assignments of the other jobs are given in Table 1. My length is $p_0 = 2 + \epsilon$. Note that the last n jobs are originated from the Partition instance. Let J_I denote this set, whose total length is 2.

The initial profile, S_0, depicted in Figure 2(a) is indeed a NE, as jobs can only migrate to machines with a lower or equal load. My goal is to utilize m_3 and keep it utilized after I vanish. As I show, I must be able to solve the Partition problem in order to do it.

▶ **Claim 3.** *I can lead the jobs to a NE on* 5 *machines if and only if a partition exists.*

Proof. Assume first that a partition exists. Let J_A be a set of jobs $J_A \subset J_I$ such that $\sum_{j \in J_A} a_j = 1$, and let $J_B = J_I \setminus J_A$. Here is a traitor BR-sequence that ends with a NE on 5 machines: First, I'll migrate to m_2 and let the jobs in J_A perform BR. Recall that my length is $2 + \epsilon$. The loads on m_1 and m_2 are 4 and $4 + \epsilon$, respectively. Since $\frac{a_j}{4} > \frac{a_j}{4+\epsilon+a_j}$,

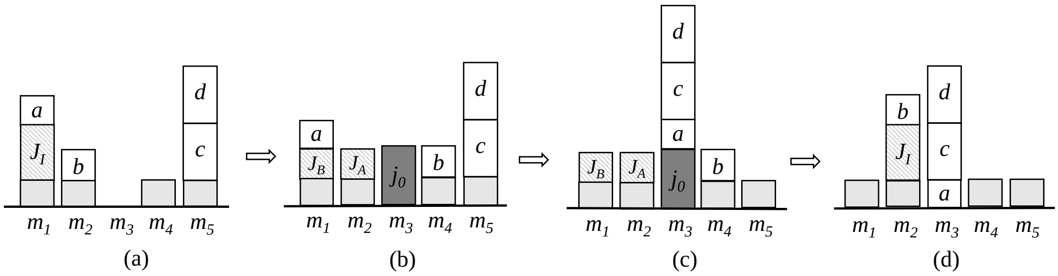

Figure 2 The CST game constructed in the reduction from Partition. (a) The initial profile S_0, (b) The profile before Job a performs BR, (c) The profile after c and d join us on m_3, and (d) the final NE if a Partition exists.

the jobs in J_A will move to m_2. Once the jobs of J_A are all on m_2, I'll migrate to m_4, and let Job b perform BR. He will join me, since $\frac{1}{3} > \frac{1}{4+\epsilon}$. Then, I'll move to m_3. The profile at this time-point is depicted in Figure 2(b). Let's analyze the possible strategies of Job a: If he stays on m_1 or move to m_2 his cost will be $1/3$, while if he joins me on m_3 his cost will be $1/(3+\epsilon)$. Thus, joining me on m_3 is his BR. This is exactly what I wanted - now I can attract additional jobs to this machine! After Job a joins me, I will let Jobs c and d perform BR. These guys are required in order to keep Job a on m_3 after I leave. Job c currently pays $2/5$. He will join us since $2/5 > 2/(5+\epsilon)$. Job d will clearly follow since $2/3 > 2/(7+\epsilon)$. The profile at this time-point is depicted in Figure 2(c). Stay tuned, we are getting closer to the end of our sequence. Next, I will let the jobs in J_B move, and join their friends in J_A on m_2; Job b will also join them. My mission is now completed - I can vanish, leaving the naive jobs in the profile depicted in Figure 2(d). This profile is a traitor-free NE: Jobs c and d will not return to m_5 since their cost will increase to $2/3$. Job a is staying with them since his cost would be $1/5$ also on m_2. The jobs of J_I are clearly happy together, and all other four jobs are restricted. The cost of this traitor-free NE is 5.

Assume now that a partition of I does not exist. I argue that I have no chance to keep m_3 utilized. First, note that Jobs c and d will not join me on m_3 if it's only me there, since $2/(4+\epsilon) > 2/5$. Next, let's analyze the conditions for Job a to join me on m_3. In order for m_3 to be his BR, the load on each of m_1 and m_2 must be less than my length, $2+\epsilon$. In any assignment, the jobs of J_I are partitioned such that for some $0 \leq \alpha \leq 2$, jobs of total length α are on m_1 and jobs of total length $2-\alpha$ are on m_2. However, for a small enough ϵ, we have that $\max(1+\alpha, 3-\alpha) \leq 2+\epsilon$ only for $\alpha = 1$. Therefore, if I has no partition, I will not be able to attract anyone to join me on m_3, and exactly 4 machines will be utilized in any traitor-free NE. ◀

◀

2.1 PoT in Games with Unit-length Jobs

In this simplest setting, of unit-cost machines and unit-length jobs, my power is very limited. The price of anarchy in this setting is k. PoA$=k$ is achieved by an instance in which one machine can process all the jobs, but in the NE each job is assigned on a different machine that is only capable to process him. The price of stability in this setting is 1 since an optimal assignment is stable - no job will utilize a new machine. Thus, by Observation 1, my potential power is k. Unfortunately, independent of S_0, I will never be able to lead the naive jobs to a more expensive profile. Formally,

▶ **Theorem 4.** *For $\mathcal{G} = \{$CST games with unit-cost machines and unit-length jobs$\}$, it holds that $PoT(\mathcal{G}) = 1$.*

Proof. Let S_0 be an initial NE. Along any traitor BR-sequence, no naive job activates a new machine, since activating a new machine costs 1, which is the maximal cost of a job in any profile. Assume that I move to a new machine and someone joins me. Each of us now pays 1/2. Such a migration is beneficial only if the other guy was on a machine by himself, implying that some machine was abandoned when he joined me. Moreover, additional jobs may join us, meaning that additional machines may be abandoned. Therefore, whenever I activate a new machine, if someone joins me, then the cost of at least one machine is saved, and if no one joins me then the total cost of the machines for the other guys does not change. Therefore, I will never be able to increase the number of machines that accommodate naive jobs. ◀

2.2 PoT in Games with Arbitrary-length Jobs

In games with unit-cost machines and arbitrary-length jobs, I can do much better. My power varies depending on k and m, and is equal to the price of anarchy [2].

▶ **Theorem 5.** *Let $\mathcal{G} = \{$CST games on unit-cost machines$\}$. (i) For $\mathcal{G}_1 \subseteq \mathcal{G}$ with $k < m$, $PoT\ (\mathcal{G}_1) = k$, (ii) For $\mathcal{G}_2 \subseteq \mathcal{G}$ with $m \leq k \leq 2m-2$, $PoT\ (\mathcal{G}_2) = m-1$, (iii) For $\mathcal{G}_3 \subseteq \mathcal{G}$ with $k \geq 2m-1$, $PoT\ (\mathcal{G}_3) = m$.*

Proof. (i) Assume that $k < m$. Clearly, $cost(S_0) \geq 1$ and in any NE profile the naive jobs may need to cover the cost of at most k machines. Thus, $PoT \leq k$. For the lower bound, given m and k such that $k < m$, consider a game G on $\mathcal{M} = \{m_0, \ldots, m_{m-1}\}$. My length is $p_0 = 2^k$, and for $1 \leq j \leq k$, Job j has length $p_j = 2^j$ and $M_j = \{m_0, m_j\}$. In the initial profile, S_0, the naive jobs are all together on m_0. This is clearly a NE, since a deviating job will need to pay for a new machine.

Here is a traitor BR-sequence that will lead us to a NE on k machines: starting from S_0, I will move to m_k. Since $2^k > \sum_{j=1}^{k-1} 2^j$, poor Job k needs to pay a bit more than 1/2. Since we have the same length, he will gladly join me to share the cost of m_k. I will then move to m_{k-1}. It is now the turn of Job $k-1$ to contribute a bit more than half of the load on m_0, and join me on m_{k-1}. Well, I'm sure you can now complete the sequence by yourself. Eventually, I will be together with Job 1 on m_1, while m_0 is empty and each of $m_2, \ldots, m_k$ is assigned only one job. This is the right time for me to vanish. The resulting profile is a TNE. The only alternative of each naive job is returning to m_0; however, m_0 is empty so it does not attract anyone. Since $cost(S_0) = 1$, and in the final NE the naive jobs are on k machines, we have that $\mathrm{PoT}(G) = k$.

(ii) Assume that $m \leq k \leq 2m-2$. Let me show you that $\mathrm{PoT} = m-1$. The lower bound is similar to the case in which $k < m$. My length is $p_0 = 2^k$, while for $1 \leq j \leq m-1$, Job j has length $p_j = 2^j$ and $M_j = \{m_0, m_j\}$. For $m \leq j \leq k$, Job j has length $p_j = \epsilon$ and his capable machines are $\{m_0, m_{m-1}\}$. In S_0 they are all assigned to m_0. As in case (i), I can lead the jobs to a profile in which they are assigned on $m-1$ machines, by attracting them, one by one starting from the longest job to their 'second' machine. Since $cost(S_0) = 1$, The PoT in this game is $m-1$.

For the upper bound, assume by contradiction that there is a game $G \in \mathcal{G}_2$ such that $\mathrm{PoT}(G) > m-1$. Since S_0 uses at least one machine and any profile uses at most m machines, in an instance with $\mathrm{PoT} > m-1$ it must be that $cost(S_0) = 1$ and some NE costs m. Since there is just one active machine in S_0, this machine is capable for all naive jobs. Denote this machine m_a. Assume that $S_T \in TNE(S_0)$ and $cost(S_T) = m$. It must be that there is at least one naive job on every machine. Since $k \leq 2m-2$, by the pigeonhole principle, there are at least two machines that are used by exactly one naive job. Let $m_b \neq m_a$ be a machine that is used only by some Job j. Thus, $cost_j(S_T) = 1$. Since there is at least one naive job on m_a and $m_a \in M_j$, by deviating to m_a, Job j can reduce his cost, contradicting the assumption that S_T is a NE, and thus, contradicting the assumption that $\mathrm{PoT}(G) > m-1$.

(iii) Assume that $k \geq 2m-1$. Let me present a game $G \in \mathcal{G}_3$ in which $cost(S_0) = 1$ and some NE in $\mathrm{TNE}(S_0)$ uses m machines, thus, $\mathrm{PoT}(G) = m$. This is clearly a tight bound as $cost(S_0) \geq 1$ and any schedule uses at most m machines. Given k, m, consider a game G on $\mathcal{M} = \{m_0, \ldots, m_{m-1}\}$. My length is $p_0 = 2^{2m-2}$, and for $1 \leq j \leq 2m-2$, Job j has length $p_j = 2^j$ and $M_j = \{m_0, m_{\lceil j/2 \rceil}\}$. Let J_R be the set of jobs $\{j \geq 2m-1\}$. Since $k \geq 2m-1$, the set J_R is not empty. The jobs of J_R have total length 1, and are restricted to m_0. In the initial profile, which is clearly a NE, all naive jobs are on m_0.

Here is a traitor BR-sequence that will lead the naive jobs to a NE on m machines: starting from S_0, I will assign myself on m_{m-1}. The load on m_0 is $\sum_{i=0}^{2m-2} 2^i = 2^{2m-1} - 1$. Poor Job $2m-2$, whose length is 2^{2m-2}, needs to pay a bit more than $1/2$. Since we have the same length, he will gladly join me to share the cost of m_{m-1}. The remaining load on m_0 is $\sum_{i=0}^{2m-3} 2^i = 2^{2m-2} - 1$. It is now the turn of Job $2m-3$ to contribute a bit more than half of the load on m_0. He will gladly join us on m_{m-1}. I will then move to m_{m-2} and attract the next pair of long jobs on m_0 to join me one after the other. The sequence continues - in turn, I attract the pair with the largest length to their 'second' machine. Eventually, only jobs from J_R, of total length 1, remain on m_0. At this time point, I will vanish. The resulting profile uses m machines and is a NE. The jobs of J_R have no alternative strategy, and the only alternative of the other naive jobs is returning to m_0. However, their current cost is either $1/3$ or $2/3$, so they prefer it over returning to m_0 and share it with J_R. Since $cost(S_0) = 1$, we get that $\mathrm{PoT}(G) = m$. ◀

3 Arbitrary-cost Machines

CST games with arbitrary-cost machines and unit-cost jobs fit the classic model of fair cost-sharing with singleton strategies. For games without a traitor it is known that the PoA is k and the PoS is $\mathcal{H}_k$ were $\mathcal{H}_0 = 0$, and $\mathcal{H}_i = 1 + 1/2 + \ldots + 1/i$. As I show, my power is quite limited, and moreover - my mission is computationally hard. On the other hand, as detailed in Section 4, when jobs may have arbitrary lengths, then my power equals the PoA already in symmetric games.

Let me start with the hardness result.

▶ **Theorem 6.** *My task is APX-hard, even with unit-length jobs and if for every naive job j, $|M_j| \leq 4$.*

Proof. I show a reduction from the maximum 3-bounded 3-dimensional matching problem (3DM-3). The input to the 3DM-3 problem is a set of triplets $U \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = n$. The number of occurrences of every element of $X \cup Y \cup Z$ in U is at most 3. The number of triplets is $|U| \geq n$. The desired output is a 3-dimensional matching in U of maximal cardinality; i.e., a subset $U' \subseteq U$, such that every element in $X \cup Y \cup Z$ appears at most once in U', and $|U'|$ is maximal. It is known that 3DM-3 is APX-hard.

Given an instance of 3DM-3, construct the following CST game with unit length jobs. $\mathcal{J} = \{x_1, x'_1 \ldots, x_n, x'_n, y_1, y'_1, \ldots, y_n, y'_n, z_1, z'_1, \ldots, z_n, z'_n\}$, that is, $3n$ pairs of jobs, one pair for every element of $X \cup Y \cup Z$. Let $\mathcal{M} = M_X \cup M_Y \cup M_Z \cup M_U$, where each of M_X, M_Y, M_Z is

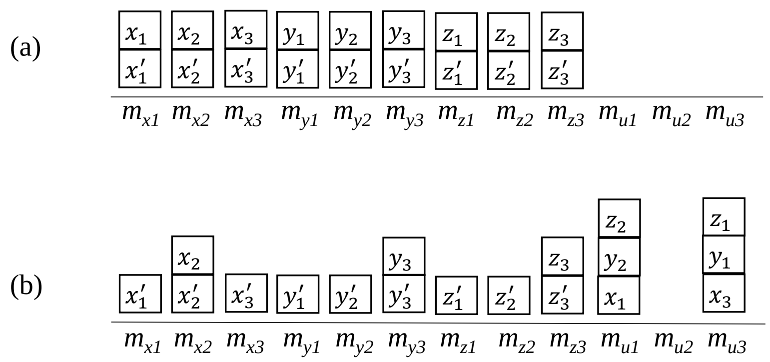

Figure 3 The CST game constructed for $n = 3$ and $U = \{\{x_1, y_2, z_2\}, \{x_1, y_3, z_3\}, \{x_3, y_1, z_1\}\}$. (a) The initial profile S_0, (b) The NE corresponding to the matching $\{\{x_1, y_2, z_2\}, \{x_3, y_1, z_1\}\}$.

a set of n *element-machines*, one machine per element, and M_U is a set of $|U|$ *triplet-machines*, one machine per triplet.

Every machine in M_X costs $3 + \epsilon$, every machine in $M_Y \cup M_Z$ costs $2 + \epsilon$, and every machine in M_U costs 3. The feasible machines for the naive jobs are as follows: For the pair x_i and x_i', Job x_i can choose between m_{x_i} and any machine of a triplet he belongs to, while Job x_i' is restricted to go to machine m_{x_i}. Formally, $M_{x_i} = \{m_{x_i}\} \cup \{m_{u_\ell} | x_i \in u_\ell\}$, and $M_{x_i'} = \{m_{x_i}\}$. Similarly, for the pair y_j and y_j', $M_{y_j} = \{m_{y_j}\} \cup \{m_{u_\ell} | y_j \in u_\ell\}$, and $M_{y_j'} = \{m_{y_j}\}$, and for the pair z_k and z_k', $M_{z_k} = \{m_{z_k}\} \cup \{m_{u_\ell} | z_k \in u_\ell\}$, and $M_{z_k'} = \{m_{z_k}\}$.

In the initial assignment, S_0, every pair is assigned on his dedicated machine. See an example in Figure 3(a). It is easy to verify that S_0 is a NE. The cost for every job corresponding to an X-element is $(3 + \epsilon)/2$, the cost for every other job is $(2 + \epsilon)/2$. Any migration of a naive job is associated with an activation of a triplet-machine and is therefore not beneficial. It holds that $cost(S_0) = 7n + 3n\epsilon$.

▶ **Claim 7.** *I can lead to a NE whose cost is $cost(S_0) + 3w$ if and only if a matching of size w exists.*

Proof. Let $W = \{u_1, \ldots, u_w\}$ be a 3-dim matching of size w. The traitor BR-sequence I will lead from S_0 consists of w iterations, in each of them I attract the elements of one triplet to their triplet-machine. Assume $\langle x_i, y_j, z_k \rangle = u_\ell \in W$. After I move to m_{u_ℓ}, I offer Job x_i to perform a BR-move. His current cost, on m_{x_i}, is $(3 + \epsilon)/2$, and he can reduce it to $3/2$ by joining me. All other triplet-machines that are capable to process him are empty, and therefore, joining me is his BR. Once he joins me, I offer y_j to perform a BR-move. Since $(2 + \epsilon)/2 > 3/3$, he would join us. Next, z_k will join us since $(2 + \epsilon)/2 > 3/4$. I will then move to attract the next triplet to their triplet-machine. After w such iterations, I will vanish. The resulting profile (see Figure 3(b)) is a traitor-free NE - the jobs assigned to the w triplet-machines each pays $3/3 = 1$ while returning to their element-machines will cost them $(3 + \epsilon)/2$ or $(2 + \epsilon)/2$. The other jobs are either restricted to their machine (jobs of type x_i', y_j' or z_k'), or can move to an empty triplet-machine - which is not beneficial.

For the other side of the reduction assume that there is a traitor BR-sequence that ends in a NE S_T whose cost is $cost(S_0) + 3w$. For every element-machine there is one job (the 'prime'-job) who is restricted to it. Also, all element-machines are utilized in S_0; therefore, in order to achieve cost $cost(S_0) + 3w$, exactly w triplet-machines are utilized in S_T. I claim that each such machine is assigned all its corresponding triplet. If u_ℓ is assigned only two jobs, then the cost for each of them is 1.5. Since at least one of the two jobs is a Y-job or a Z-job, he can migrate from m_{u_ℓ} to join his pair in S_0 for cost $1 + \epsilon/2$. Therefore, S_T is stable only if w machines are assigned their corresponding triplets - inducing a matching of size w. ◀

◀

3.1 PoT in Games with unit-length jobs

In order to bound my power in this setting, let us consider a stronger model, in which the jobs are allowed to perform *better*-response moves, and not only best-response ones. An *upgraded traitor* has the ability to select the next job to deviate and also his next strategy, as long as it is better than his current strategy. My power in this model is at least as high as in the regular model, since every BR-sequence is also a better-response one. The upper bound for the PoT in CST games with unit-length jobs is valid also for the stronger model, while the lower bound in my analysis below is achieved already by a traitor BR-sequence.

Let $\mathcal{G} = \{\text{CST games with unit-length jobs}\}$. Let $G \in \mathcal{G}$ and let S_0 be an initial profile in G, such that $\text{PoT}(\mathcal{G})$ is achieved by G starting from S_0. The proof of the following lemma is based on the fact that the set $\mathcal{M}$ can be tailored to include machines that can only accommodate one specific job, and I can attract the jobs to these machines.

▶ **Lemma 8.** *W.l.o.g., in the worst traitor better-response sequence from S_0, all the initial machines are emptied, and every job migrates exactly once.*

Proof. Let S_T be the most expensive profile in $\text{TNE}(S_0)$. Assume by contradiction that there is a machine m_a that was utilized in S_0 and is not empty in S_T. Assume that $L_a(S_T) = \ell$. Consider a game G' in which $\mathcal{M}' = \mathcal{M} \cup \{m'_1, \dots m'_\ell\}$. For $1 < z \leq \ell$, define $c(m'_z) = 2c(m_a)/z - \epsilon$. For m'_1 define $c(m'_1) = c(m_a)$. Let $j_1, j_2, \dots, j_\ell$ be the jobs on m_a in S_T according to the order they joined m_a. It is possible that some of them were on m_a in S_0, in which case their enumeration is arbitrary. In G', for every $1 \leq z \leq \ell$ define $M'_{j_z} = M_{j_z} \cup \{m'_z\}$. Clearly, for all z, $cost_{j_z}(S_T) = c(m_a)/\ell$. I claim that I can lead G' from S_0 to a NE of cost $cost(S_0) + \sum_{z=1}^{\ell} c(m'_z)$. Thus, G' has a higher PoT. The traitor better-response sequence from S_0 in G' will be identical to the sequence in G with the following suffix: Before I vanish, I will migrate to machine m'_ℓ. Since $c(m'_\ell)/2 < c(m_a)/\ell$, joining me is attractive for j_ℓ. I will continue in a similar way to evacuate m_a. Note that $c(m'_1) = c(m_a)$. The machine m'_1 is essential, since it is important to make sure that j_1 also leaves m_a: this guarantees that the resulting profile is a NE - for all $1 \leq z \leq \ell$, we have $c(m'_z) \leq c(m_a)$, and therefore none of them has an incentive to return to m_a and attract the other jobs back after I'm gone. Also, since m'_z is only capable to process Job j_z, no other job is affected. The above extension of the traitor better-response sequence can be applied for any machine which is utilized in S_0 but not emptied along sequence.

Using a similar extension of the sequence, I can show that there exists a worst traitor better-response sequence from S_0, in which every utilized machine accommodates exactly one naive job. Finally, let me show that there exists a worst sequence in which every naive job j migrates exactly once - from machine $s_j(S_0)$ to some new machine: By the above, there exists a sequence in which all the machines that were active in S_0 are empty in S_T and every utilized machine accommodates a single job. Therefore, every job migrates at least once. Assume by contradiction that there are naive jobs who migrate more than once. Let j be the naive job who performed the last before-last migration. By the choice of j, after his before-last migration there were migrations only to final destinations, and according to the properties above, these migrations are into new dedicated machines - each accommodating a single job.

Assume that in his before-last migration, Job j moved from m_a to m_b. Let ℓ_a and ℓ_b denote the loads on m_a and m_b, respectively, before the migration of Job j from m_a to m_b. The migration is beneficial, therefore, $c(m_a)/\ell_a > c(m_b)/(\ell_b + 1)$. Given that m_b is about to be evacuated, there exists some job on m_b who will be the first to migrate to some dedicated machine m'. His move would be beneficial, so his cost on m' will be less than $c(m_b)/(\ell_b + 1)$. Define a game G' in which $M'_j = M_j \cup \{m'\}$. Consider the sequence in which before the migration of job j I move to m' and then let Job j perform a BR move. Joining me will be j's best-response. In G' I need to permute the dedicated machines allowed for each of the jobs currently on M_b - such that it will be emptied as before, maybe in a different order. This permutation however does not hurt the total cost of the machines activated due to the jobs leaving m_b. Therefore, there exists a sequence in which the before-last migration of j is saved, and the total cost of machines utilized is not hurt. The above process can be repeated as long as there are jobs migrating more than once – to end up with a sequence fulfilling the properties stated in the lemma. ◀

Based on the above characterization, the PoT can be bounded as follows.

▶ **Theorem 9.** *For $\mathcal{G} = \{$CST games with unit-length jobs $\}$, it holds that $PoT(\mathcal{G}) = 2H_k - 1$.*

Proof. By Lemma 8, the PoT is achieved by emptying one by one the machines in S_0, where every job is attracted to a new machine. Thus, for every machine m_i, the load on m_i reduces during the traitor BR-sequence from $L_i(S_0)$ to 0. A naive job j that leaves his machine m_a when the load on it is ℓ will be attracted to join me on a new machine only if its cost is less than $\frac{2c(m_a)}{\ell}$. Also, if $\ell = 1$ then I can attract j to a machine of cost at most $c(m_a)$, as otherwise, j will return to m_a after I'm gone, and will also attract the other jobs back to it. For $\ell > 1$, the new machines have cost lower than $\frac{2c(m_a)}{\ell} \leq c(m_a)$, and therefore, the trapped jobs will not have an incentive to return to m_a after I vanish. The total cost of machines I will utilize in order to empty m_a is therefore less than $c(m_a) + \sum_{\ell=2}^{L_a(S_0)} \frac{2c(m_a)}{\ell} = c(m_a)(2\mathcal{H}_{L_a(S_0)} - 1)$. Summing over all the machines in S_0, we get that for the final NE S_T, $cost(S_T) < \sum_{i|L_i(S'_0)>0} c(i)(2H_{L_i(S'_0)} - 1)$. For at least one machine, $L_i(S_0) \leq k$, implying that $cost(S_T) < cost(S_0)(2H_k - 1)$.

For the lower bound, let me describe a CST game in which I can lead the jobs to a NE whose cost is arbitrarily close to $cost(S_0)(2H_k - 1)$. The game is played on $m = k+1$ machines, $\mathcal{M} = \{0, 1, \ldots, k\}$. The cost of machine m_0 is $1 + \epsilon$, for $1 \leq i < k$, we have $c(m_i) = \frac{2}{k-i+1}$, and $c(m_k) = 1$. There are k unit-length jobs, where for $1 \leq j \leq k$, $M_j = \{m_0, m_j\}$. In the initial profile, S_0, all the jobs are on m_0. Since the cost for each naive job is $\frac{1+\epsilon}{k}$ and the cheapest empty machine has cost $\frac{2}{k}$, S_0 is a NE. Here is a traitor BR-sequence I can initiate: First, I appear on m_1, whose cost is $\frac{2}{k}$. Joining me is beneficial and possible for Job 1. Once he migrates and joins me, I move further to m_2 and let Job 2 perform best-response. His current cost is $\frac{1+\epsilon}{k-1}$ and I offer him a cheaper alternative. I continue to attract the jobs one after the other until eventually, m_0 is empty, m_i accommodates Job i, for all $1 \leq i \leq k-1$, and m_k is shared by Job k and myself. My mission is completed. The resulting profile is a NE: the naive players have no incentive to activate m_0, since each of them has current cost at most 1.

The cost of this NE is $\sum_{i=1}^{k} c(m_i) = \sum_{i=1}^{k-1} \frac{2}{k-i+1} + c(m_k) = 2(H_k - 1) + 1 = 2H_k - 1$. Since $cost(S_0) = 1 + \epsilon$, the PoT is arbitrarily close to $2H_k - 1$. ◀

4 Symmetric Games

In a symmetric game, all the players have the same set of feasible machines. W.l.o.g., for all j, $M_j = \mathcal{M}$. It is well-known that in symmetric games, all the players use the same strategy in every NE. Indeed, if two naive players use different strategies, then at least one of them would benefit from joining the other. It is also known that PoA$= k$ and PoS=1 in this settings [1], where the high PoA is achieved even with unit-length jobs.

In this section I show that with unit-length jobs or with unit-cost machines I have no power, that is, I cannot lead the players to a NE worse than S_0. I then suggest an efficient way to increase my power: I consider the lightest relaxation of the unit-length condition, and show that allowing me to have an arbitrary length (while all other jobs have unit-length), is sufficient to achieve PoT=PoA$= k$. I then provide a tight bound for my power with arbitrary-length jobs and arbitrary-cost machines.

The first theorem follows trivially from the fact that in symmetric games all the players use a single machine in every NE.

▶ **Theorem 10.** *For $\mathcal{G} = \{$symmetric CST games with unit-cost machines$\}$, it holds that $PoT(\mathcal{G}) = 1$.*

Next, let me show that I cannot be of any help also with unit-length jobs.

▶ **Theorem 11.** *For $\mathcal{G} = \{$symmetric CST games with unit-length jobs$\}$, it holds that $PoT(\mathcal{G}) = 1$.*

Proof. Since the game is symmetric, in S_0 all the jobs are on the same machine, say m_0. Assume w.l.o.g., that $cost(S_0) = c(m_0) = 1$. For $k = 1$, I will be able to attract the single naive job only to a machine whose cost is less than 2. However, once I vanish, he would return to m_0. Therefore, for every $S_T \in TNE(S_0)$, $cost(S_T) = cost_1(S_T) \leq 1 = cost(S_0)$.

Assume next that $k > 1$. Clearly, I am the only job who may initiate the use of a machine whose cost is more than 1. In order to end up with a more expensive profile, some job must join me on an expensive machine. Assume by contradiction that there exists a traitor BR-sequence in which I attract someone to join me on an expensive machine. Let m_a be the first expensive machine in which a job j joins me. When job j migrates, since $k > 1$, apart from m_a, there is at least one active machine m_b utilized by jobs in $\mathcal{J} \setminus \{0, j\}$. Since m_a is the first expensive machine to accommodate a naive job, it must be that $c(m_b) \leq 1$. Therefore, Job j has an alternative strategy, m_b, in which his cost would be at most $1/2$, contradicting the assumption that his BR-move is to join me on m_a. We conclude that naive jobs will only migrate to machines of cost at most 1. Since they will end-up on a single machine, we get that also for $k > 1$, $PoT = 1$. ◀

To increase my power in symmetric games with unit-length jobs, I asked my operators to increase my processing time. Gladly, this works above and beyond everyone's expectations:

▶ **Theorem 12.** *For $\mathcal{G} = \{$symmetric CST games with unit-length jobs and arbitrary-length traitor$\}$, it holds that $PoT(\mathcal{G}) = k$.*

Proof. The upper bound follows from Observation 1 and the fact that in cost sharing symmetric games $PoA \leq k$ [15]. The lower bound is a generalization for arbitrary k of Example 1. Consider an instance with two machines m_1 and m_2 of costs 1 and k respectively. Assume that k unit-length jobs are assigned on m_1. Assume now that I appear and assign myself on m_2. My length is $k^2 - 1 + \epsilon$, thus, a job that joins me would pay $k \cdot \frac{1}{k^2+\epsilon}$, which is less than $\frac{1}{k}$, his current cost on m_1. The other jobs will follow, and once they are all on m_2, I will be gone, leaving them on in a traitor-free NE of cost k. ◀

Table 2 My power in various environments. In entries marked by [$\star$], the PoT is lower than the PoA/PoS-bound and the $PoA = k$. In all other entries, $PoT = PoA$.

Jobs \ Machines	unit-cost		arbitrary-cost	
	general	symmetric	general	symmetric
unit-length	1 [$\star$]	1	$\Theta(\mathcal{H}_k)$ [$\star$]	1 [$\star$]
arbitrary-length	$\min(m, k)$	1	k	$\frac{\sum_j p_j}{\max_j p_j}$

Let's consider next instances with arbitrary-length jobs and arbitrary-cost machines. Let $L(\mathcal{J}) = \sum_{j \in \mathcal{J}} p_j$ be the total length of the naive jobs, and let $\alpha(\mathcal{J}) = max_{j \in \mathcal{J}} p_j / L(\mathcal{J})$. Theorem 12 analyzes the case $\alpha(\mathcal{J}) = 1/k$. The following theorem generalizes it for arbitrary $\alpha(\mathcal{J})$.

▶ **Theorem 13.** *For $\mathcal{G} = \{symmetric\ CST\ games\}$, it holds that $PoT(\mathcal{G}) = PoA(\mathcal{G}) = 1/\alpha(\mathcal{J})$.*

Proof. Let me first bound the PoA. Assume that Job 1 determines the value of α, and that $OPT = 1$. Let S be a NE profile. Since the game is symmetric, all the jobs are on a single machine, and the cost of Job 1 is $\alpha \cdot cost(S)$. He can deviate to the machine of cost 1 utilized in the optimal profile, therefore, in order for S to be a NE, it must hold that $\alpha \cdot cost(S) \leq 1$, implying $cost(S) \leq 1/\alpha$.

The lower bound is a generalization of Example 1. Given a set of naive jobs $\mathcal{J}$, let $L = L(\mathcal{J}), \alpha = \alpha(\mathcal{J})$. Assume that Job 1 determines the value of α. Consider a symmetric CST game, on two machines, where $c(m_1) = 1$ and $c(m_2) = \frac{1}{\alpha} = \frac{L}{p_1}$. Let S_0 be the initial stable profile in which all the jobs are on m_1. Assume now that I appear and assign myself on m_2. My length is $\frac{L^2}{p_1} - p_1 + \epsilon$. Note that if Job 1 joins me, his cost would be $\frac{L}{p_1} \cdot \frac{p_1}{\frac{L^2}{p_1} + \epsilon}$ which is less than $\frac{p_1}{L}$, his current cost on m_1. It is easy to see that the other jobs will follow. Once they are all on m_2, I will be gone. The resulting profile has cost $c(m_2) = \frac{1}{\alpha}$. The cost for a job of length p_j is $\frac{L}{p_1} \cdot \frac{p_j}{L} \leq 1$, thus, no one will migrate back to m_1 and this profile is a traitor-free NE. ◀

5 Conclusions and Plans for My Retirement

Being the evil guy is not an easy task, but a rewarding one. My power is summarized in Table 2. I'm a bit disappointed from my limited power in instances with unit-length jobs, which is significantly lower than the PoA/PoS bound. However, if you run a system that processes arbitrary-length jobs and would like to boost your revenue, you should definitely hire me! If you deal with symmetric jobs then you will greatly enjoy my services if you process arbitrary-length jobs on arbitrary-cost machines.

I am exploring several ways to increase my power. One clear direction is to employ additional secret agents to work with me. I want to analyze the power of several traitors, who coordinate their moves trying to lead the naive jobs to a poor outcome. In this general setting, the number of traitors is γk for some fraction γ. Fooling the naive jobs by a bunch of secret agents could be really fun and rewarding!

I would also like to devise algorithms that calculate, for a given initial profile, a traitor BR-sequence with high PoT. In this paper I proved that the problem is NP-hard, but I believe that there are interesting classes of instances for which it is possible to come up with an optimal sequence, or at least an approximated one. Another interesting problem is to consider the power of a traitor in other congestion games. Specifically, after my retirement, I hope to volunteer in networks, and be in charge of routing messages. After gaining the trust of other players there, I will challenge myself harming the social cost in network formation games.

Alternatively, I may enter the world of congestion games – in which the cost associated with using a resource increases with the load on it. It seems that a totally different approach is required in such games, because I will no longer attract naive players to join me, but to get away from me. In general, almost every congestion game becomes more interesting when a single or multiple traitors are involved.

References

1. 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.
2. G. Avni and T. Tamir. Cost-sharing scheduling games on restricted unrelated machines. *Theoretical Computer Science*, 646:26–39, 2016.
3. V. Bilò and C. Vinci. On the impact of singleton strategies in congestion games. In *Proc. 25th Annual European Symposium on Algorithms*, pages 17:1–17:14, 2017.
4. Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2011.
5. E.Even-Dar, A.Kesselman, and Y.Mansour. Convergence time to nash equilibria. In *Proc. 30th Int. Colloq. on Automata, Languages, and Programming*, pages 502–513, 2003.
6. A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. 36th ACM Symp. on Theory of Computing*, pages 604–612, 2004.
7. A. Fanelli, M. Flammini, and L. Moscardelli. Stackelberg strategies for network design games. In *Proc. of the 3rd International Conference on Algorithmic Game Theory*, pages 222—233, 2010.
8. M. Feldman, Y. Snappir, and T. Tamir. The efficiency of best-response dynamics. In *Proc. of the 10th International Symposium on Algorithmic Game Theory (SAGT)*, 2017.
9. D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *The 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 190–204, 2010.
10. D. Fotakis. Stackelberg strategies for atomic congestion games. *Theory of Computing Systems*, 47(1):218–249, 2010.
11. Vasilis Gkatzelis, Konstantinos Kollias, and Tim Roughgarden. Optimal cost-sharing in general resource selection games. *Operations Research*, 64(6):1230–1238, 2016.
12. T. Harks and M. Klimm. On the existence of pure nash equilibria in weighted congestion games. *Math. Oper. Res.*, 37(3):419–436, 2012.
13. Samuel Ieong, Robert McGrew, Eugene Nudelman, Yoav Shoham, and Qixiang Sun. Fast and compact: A simple class of congestion games. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, 2005.
14. Yannis A. Korilis, Aurel A. Lazar, and Ariel Orda. Achieving network optima using stackelberg routing strategies. *IEEE/ACM Trans. Netw.*, 5(1):161–173, 1997.
15. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.
16. Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *Advances in Cryptology - CRYPTO 2006*, pages 180–197, 2006.

17 I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13(1):111–124, 1996.

18 Shien Jin Ong, David C. Parkes, Alon Rosen, and Salil Vadhan. Fairness with an honest minority and a rational majority. In Omer Reingold, editor, *Theory of Cryptography*, pages 36–53, 2009.

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

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 V. Syrgkanis. The complexity of equilibria in cost sharing games. In *WINE*, volume 6484, pages 366–377. Springer, 2010.

23 Tami Tamir. Scheduling with bully selfish jobs. In *Proceedings of the 5th International Conference on Fun with Algorithms*, FUN'10, pages 355–367, 2010.

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