On Designing Truthful Mechanisms for Online Scheduling*

Vincenzo Auletta[1], Roberto De Prisco[1,2], and Paolo Penna[1], and Giuseppe Persiano[1]

[1] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, via **S. Allende** 2, I-84081 Baronissi (SA), Italy
{auletta, robdep, penna, giuper}@dia.unisa.it
[2] Faculty Group at Akamai Technologies, Cambridge, MA, USA

Abstract. We study the *online* version of the scheduling problem involving *selfish agents* considered by Archer and Tardos [FOCS 2001]: jobs must be scheduled on m parallel related machines, each of them owned by a different *selfish agent*.

Our study focuses on general techniques to translate approximation/competitive algorithms into equivalent approximation/competitive *truthful mechanisms*. Our results show that this translation is more problematic in the online setting than in the offline one. For $m = 2$, we develop an offline and an online "translation" technique which, given *any* ρ-approximation/competitive (polynomial-time) algorithm, yields an $f(\rho)$-approximation/competitive (polynomial-time) mechanism, with $f(\rho) = \rho(1 + \varepsilon)$ in the offline case, for every $\varepsilon > 0$. By contrast, one of our lower bounds implies that, in general, online ρ-competitive algorithms cannot be turned into $\rho(1+\varepsilon)$-competitive mechanisms, for some $\varepsilon > 0$ and every $m \geq 2$.

We also investigate the issue of designing new online algorithms from scratch so to obtain efficient competitive mechanisms, and prove some lower bounds on a class of "natural" algorithms. Finally, we consider the variant introduced by Nisan and Ronen [STOC 1999] in which machines can be *verified*. For this model, we give a $O(1)$-competitive online mechanism for *any* number of machines and prove that some of the above lower bounds can be broken.

1 Introduction

Optimization problems dealing with resource allocation are classical algorithmic problems and they have been studied for decades in several models. Typically, algorithms are evaluated by comparing the (measure of) the solutions they return to the best possible one. In particular, one tries to estimate the loss of

* Work supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO).

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 3–17, 2005.

performance due to the lack of computational resources (*approximation ratio*) or to the lack of information (*competitive ratio*).

In both settings, the underlying hypothesis is that the input is (eventually) available to the algorithm (either from the beginning in off-line algorithms or during its execution in on-line algorithms). This assumption cannot be considered realistic in the context of modern networks like the Internet where certain information regarding the resources are not directly available to the "protocol". Indeed, since the resources are owned/controlled/used by different *self-interested* entities (e.g., corporations, autonomous systems, etc.). Each of these entities, or *selfish agents*, hold some *private information* which is needed in order to compute an optimal resource allocation (e.g., routing the traffic over the Internet requires routers of different autonomous systems to exchange information on which routers can process traffic faster). Each agent can possibly *misreport* his/her piece of information if this leads the system to compute a solution that is more beneficial for him/her. This, in spite of the fact that such a solution may *not* be not globally optimal.

The field of *Mechanism Design* is the branch of Game Theory and Microeconomics that studies how to design complex auctions, also termed *mechanisms*, which guarantee that no agents has an incentive in misreporting his/her piece of information. Loosely speaking, a mechanism is a pair $M = (A, P)$, where A is an algorithm computing a solution, and $P = (P^1, \ldots, P^n)$ is the vector of payment functions (see Sect. 1.1 for a formal definition). Selfish agents are suppose to be rational and thus will deviate from the truth-telling strategy (in our problem, to report $r_i = s_i$) only if a better one exists. Therefore, one seeks for *truthful* mechanisms, that is, mechanisms that guarantee that every agent i can maximize his/her net profit or *utility* by playing the truth-telling strategy (see Sect. 1.1).

In this work we consider the *online* version of a basic scheduling/routing problem involving *selfish agents*, first addressed by Archer and Tardos [2]. We will investigate the approximation/competitive ratio of truthful mechanisms for this problem. Our goal is to quantify the (further) loss of optimality due to the combination of selfish agents with the online setting. Central to our study is the existence of general techniques that allow to translate ρ-approximation/competitive algorithms into a $f(\rho)$-approximation/online mechanisms, for some function $f(\cdot)$.

1.1 The Problem

Offline Selfish Version. Consider the problem of scheduling jobs on related machines ($\mathrm{Q}||\mathrm{C}_{\max}$): We are given a set of m machines with speed $s_1, s_2, \ldots, s_m$ and a set of n jobs of size $J_1, J_2, \ldots, J_n$. We want to assign every job to a machine so to minimize the *makespan*, that is, the maximum over all machines of w_i/s_i, where w_i is the sum of the job weights assigned to machine i. When the set of machines m is fixed, this problem version is commonly denoted to as $\mathrm{Q}_m||\mathrm{C}_{\max}$.

We study the selfish version of the $Q||C_{\max}$ problem in which each machine i is owned by a selfish agent and the corresponding speed s_i is known to that agent only. In particular, any schedule S that assigns load w_i to machine i is valuated by agent i as $v^i(S)$, where

$$v^i(S) \stackrel{\text{def}}{=} -w_i/s_i,$$

that is, the opposite of the completion time of machine i. Intuitively, $v^i(S)$ represents how much user i likes solution S. This model has been first considered by Archer and Tardos [2].

We stress that our goal is to compute a solution S which minimizes the makespan with respect to the *true* machine speeds $s_1, \ldots, s_m$. Hence, we need to provide some incentive (e.g., a payment P^i) to the each agent i in order to let him/her truthfully report his/her speed. Formally, a *mechanism* is a pair $M = (A, P_A)$, where $P_A = (P_A^1, \ldots, P_A^m)$, and A is a scheduling algorithm. Each agent i reports its type b_i which is not necessarily the true type $t_i \stackrel{\text{def}}{=} 1/s_i$. Algorithm A gets in input the reported types $b = (b_1, \ldots, b_m)$, and each agent i receives a payment equal to $P_A^i(b, J)$. Obviously, each agent i wants to maximize the resulting net profit or *utility* defined as

$$u_i^M(b, J) \stackrel{\text{def}}{=} P_A^i(b, J) + v^i(A(b, J)).$$

Each agent *knows* both algorithm A and the payment function P_A^i.

A mechanism is said to be *truthful with dominant strategies* (or simply *truthful*) if the payments P_A and the algorithm A guarantee that no agent obtains a larger utility when reporting $b_i \neq t_i$, independently of the other agents' reported types; that is, for all J, for all reported types $b_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m)$ of all the agents except i, and for all possible declarations b_i of agent i, it holds that

$$u_i^M((t_i, b_{-i}), J) \geq u_i^M((b_i, b_{-i}), J),$$

where the writing (x, b_{-i}) denotes the vector $(b_1, \ldots, b_{i-1}, x, b_{i+1}, \ldots, b_m)$. We stress that no agent i has any advantage from knowing the true speeds t_{-i} of the other agents: indeed, the utility of agent i does *not* depend on the speeds of the other agents (the work/payment assigned to machine/agent i depend on the agent bids b only). If M guarantees that the utility is non-negative for all agents i that report their true type, then we say that the mechanism enjoys the *voluntary participation* property.

Online Selfish Version. In the *online* version of $Q||C_{\max}$, jobs arrive one-by-one and must be scheduled upon their arrival. Moreover, jobs cannot be reallocated. For any (possibly infinite) sequence of jobs $J = J_1 J_2 \cdots$, we let J^k denote the prefix $J_1 J_2 \cdots J_k$ of the first k jobs, for $1 \leq k \leq |J|$. Before any job appears, each agent declares her type and we denote by $b = (b_1, \ldots, b_m)$ the vector of declared types. An *online mechanism* for $Q||C_{\max}$ is a pair $M = (A, P)$ where P is a sequence of payment functions P_i^k, for $i = 1, \ldots, m$ and $k > 0$ such that

- The algorithm A is an online algorithm for $Q||C_{max}$; we denote by $w_i^A(b, J^k)$ the sum of the job sizes assigned to machine i by the solution computed by A on input J^k and vector b of declared types.
- When the k-th jobs arrives, it is assigned by A to a machine and each agent i receives *non-negative* payment $P_i^k(b, J^k)$. That is, we are not allowed to ask money back from the agents.

The total payment received by agent i after k jobs is equal to $P_i(b, J^k) = \sum_{j=1}^{k} P_i^j(b, J^j)$.

Definition 1 (online truthful mechanism). *We say that an online mechanism is* truthful *with respect to* dominant strategies *if for any prefix J^k of J, for all b_{-i}, and for all types t_i, the function $u_i^M((b_i, b_{-i}), J^k)$ is maximized for $b_i = t_i$.*

Verifiable Machines. We also study the online version of the model proposed by Nisan and Ronen [9] of verifiable machines. Here the payment for each job is awarded after the job is released by the machine (we stress that a machine cannot release a job assigned to it before the job has been executed). Intuitively, if a machine has received positive load, the mechanism can verify whether the machine lied declaring to be *faster* and, if so, the machine receives no payment.

1.2 Previous Results

Archer and Tardos [2] have characterized the (offline) algorithms A for $Q||C_{max}$ for which there exist payment functions P such that (A, P) is a truthful mechanism. In particular they show that if an algorithm A is monotone (that is, it satisfies $w_i^A((b_i', b_{-i}), J) \leq w_i^A((b_i, b_{-i}), J)$, for all $b_i' > b_i$) then there exists a payment function P such that (A, P) is truthful. Under mild assumptions on A, it is possible to define the payment function to guarantee voluntary participation. They also gave a monotone optimal (exponential-time) algorithm for $Q||C_{max}$ and a $(3 + \varepsilon)$-approximate randomized (polynomial-time) monotone algorithm. In [4] we gave a $(4 + \varepsilon)$-approximate deterministic (polynomial-time) monotone algorithm for $Q_m||C_{max}$. Recently and independently from this work, Andelman *et al* [1] provided an elegant technique for turning any ρ-approximation algorithm for $Q_m||C_{max}$ into a $\rho(1 + \varepsilon)$-approximation monotone mechanism. As a result, given any polynomial-time $(1 + \varepsilon)$-approximation algorithm for this problem, one can obtain a $(1 + \varepsilon)$-approximation mechanism running in polynomial time. They indeed settle the approximation guarantee of the $Q_m||C_{max}$ by obtaining a fully polynomial-time approximation scheme which is monotone. Moreover, they provide a 5-approximation truthful mechanism for the $Q||C_{max}$ problem, i.e., for any number of machines.

Nisan and Ronen [9] considered the case of unrelated machines and gave a randomized 7/4-approximate truthful mechanism for two machines and a deterministic m-approximate truthful mechanism for any number of machines. Moreover, they proved that no deterministic truthful mechanism can be $(2 - \varepsilon)$-

approximate for $m \geq 2$ machines. Nisan and Ronen also considered the case of verifiable unrelated machines and gave a polynomial-time $(1+\varepsilon)$-approximate truthful mechanism for any fixed number of machines. For the case of verifiable related machines (that is $Q||C_{\max}$), in [6], we characterized the algorithms A for which there exist payment functions P such that (A, P) is a truthful mechanism. Based on this we developed a polynomial-time $(1+\varepsilon)$-approximate truthful mechanism for the offline version of $Q||C_{\max}$.

1.3 Our Contribution

A central question in (algorithmic) mechanism design is to translate approximation/online algorithms into approximation/online mechanisms: given an algorithm A of approximation/competitive ratio ρ, can we obtain a monotone algorithm $\overline{A}$ with the same approximation/competitive ratio? A general approach to the design of approximation/competitive mechanisms might be that of developing general "monotonization" techniques: starting from any ρ-approximation/competitive algorithm A, transform A into a monotone algorithm $\overline{A}$ with approximation/competitive ratio $\overline{\rho}$ depending on ρ. We first consider the $Q_2||C_{\max}$ problem for which we provide the following two general results:

Offline Case: Every polynomial-time ρ-approximation algorithm can be transformed into a *monotone* polynomial-time $(\rho+\varepsilon)$-approximation algorithm $\overline{A}$, for every $\varepsilon > 0$ (Theorem 3). This result is a special case of the one obtained independently by Andelman *et al* [1]: indeed, their monotonization technique extends our result to any fixed number of machines.

Online Case: Given an online ρ-competitive algorithm A, for every $t > 0$, it is possible to obtain an online monotone algorithm $\overline{A}_t$ whose competitive ratio $\overline{\rho}$ satisfies $\overline{\rho} \leq \max\{\rho \cdot t, 1 + 1/t\}$ (Theorem 5). Moreover, the same bound holds if A is a ρ-competitive algorithm (only) for *identical speeds*. The "monotonization" of the greedy algorithm [1] thus yields and online mechanism whose competitive ratio is at most $1 + \sqrt{7}/2 < 1.823$ (Corollary 2).

It is natural to ask whether the loss of performance due to our "monotonization" for the online setting is really necessary, and whether (some of the) existing algorithms could preserve their competitive guarantee (after being turned into a monotone one).

We first show a general lower bound on *monotone online* algorithms. Consider the problem restricted to instances for which $s_{\max}/s_{\min} = r$, for any $r > 0$. Then, no such algorithm can be less than $\rho(r)$-competitive, with $\rho(r) \geq \min\{r, 1+1/r\}$ (Theorem 6). This gives a general lower bound of $\phi \simeq 1.62$, which also holds for sequences of two jobs (Corollary 3). At least for such sequences our technique is optimal: indeed, since the greedy algorithm is 1-competitive, our method yields a ϕ-competitive online algorithm (simply choose $t = \phi$).

[1] This algorithm, also known in the literature as ListScheduling, assigns the current job J_i to the machine that minimize the completion time of J_i.

An underlying implicit assumption in designing scheduling algorithms is that, for the same set of jobs, speed vectors $s = (s_1, s_2)$ and $s_\alpha = (\alpha s_2, \alpha s_1)$ lead to the same solution (modulo a machine re-indexing). We show that this (apparently natural) way of proceeding must necessarily lead to online monotone algorithms whose competitive ratio is not smaller than 2. In particular, we isolate two pathological facts that, each of them alone, prevent from having a non-trivial competitive ratio (see Theorems 7-8): (i) the first job is always assigned to the fastest machine, and (ii) solution for (s_1, s_2) is isomorphic (modulo a index exchange) to that for (s_2, s_1).

It is worth observing that the *lack of information* plays a central role both in the *online* and in the *selfish* setting of the problem. In the online setting we do not know the "future;" when dealing with "selfish" agents we do not know part of the input. Our results (see Table 1) show that the combination "online+selfish" makes the $Q_2||C_{\max}$ problem harder than both the offline with selfish agents and the online (without selfish agents) versions. In particular, for $\sqrt{2} < r \leq \phi$, it holds that (i) r is a lower bound for any online monotone algorithm (i.e., any mechanism), while (ii) there is an upper bound $\rho \leq 1 + 1/(r+1) < r$ provided by the greedy for the online case (without selfish agents).

Table 1. Our and previous results for the case of two machines: all lower bounds also apply to exponential-time algorithms, while upper bounds are obtained via polynomial-time ones

	Offline		**Online**	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Non Selfish	1 [trivial]	$1+\varepsilon$ [8]	$1+1/(r+1)$ [folklore]	$1+1/(r+1)$, for $r \leq \phi$ [3–greedy] $1+1/r$, for $r > \phi$ [3–greedy]
Selfish	1 [trivial]	$1+\varepsilon$ [Cor. 1] or [1]	$\min\{r, 1+1/r\}$ [Thm. 6]	$1+\sqrt{7}/2 < 1.823$ [Thm. 5 and Cor. 2]

All our lower bounds also apply to $Q_m||C_{\max}$, for any $m > 2$. As for the upper bounds, in Sect. 6 we present a 12-competitive algorithm for *any number* of *verifiable* machines. This is the first constant-competitive truthful online algorithm for any number of machines ($Q||C_{\max}$).

The ability to "verify" machines has been proved to yield better approximation mechanisms in the offline case for other scheduling problems [9, 6]. The results here show that the same happens also for the online version of $Q_m||C_{\max}$, for any $m \geq 2$. By contrast, the results by Andelman *et al* [1] imply that in the *offline* setting verification does not help for $Q_m||C_{\max}$, for any $m \geq 2$.

Due to lack of space, some of the proofs are omitted in this extended abstract. We refer the interested reader to the full version of this work [5].

Notation. Throughout the paper s_i will denote the speed of the i-th machine, t_i its type (i.e., $t_i = 1/s_i$) and b_i the type reported by agent i.

2 Characterization of Online Truthful Mechanisms

For the offline case, Archer and Tardos [2] characterized the class of algorithms that can be used as part of a truthful mechanism. More precisely, we have the following definition and theorem.

Definition 2 (monotone algorithm). *An algorithm A is* monotone *if, for every i, for every J, for every b_{-i}, for every b_i and $b'_i > b_i$ it holds that*

$$w_i^A((b'_i, b_{-i}), J) \leq w_i^A((b_i, b_{-i}), J),$$

where $w_i^A((b_i, b_{-i}), J)$ is the load assigned to machine i when J is the job sequence and agents report types (b_i, b_{-i}).

Theorem 1 (offline characterization [2]). *A mechanism $M = (A, P)$ is truthful if and only if A is monotone. Moreover, for every monotone algorithm A, there exist payment functions P such that (A, P) is truthful and satisfies voluntary participation if and only if $\int_0^\infty w_i^A((u, b_{-i}), J)\, du < \infty$ for all i, J, and b_{-i}. In this case, we can take the payments to be*

$$P_i((b_i, b_{-i}), J) = b_i \cdot w_i^A((b_i, b_{-i}), J) + \int_{b_i}^{\infty} w_i^A((u, b_{-i}), J) du. \tag{1}$$

Next, we translate the result above into the online setting. We will use the characterization to obtain our upper and lower bounds.

Theorem 2 (online characterization). *An online mechanism $M = (A, P)$ is truthful if and only if A is an online monotone algorithm. Moreover, for every online monotone algorithm A, there exists a payment function P such that (A, P) is truthful. In addition, there exist payment functions P_i^k such that $P_i^k((b_i, b_{-i}), J^k) \geq 0$ for all J, k and (b_i, b_{-i}).*

3 Monotonization Techniques

3.1 Offline Monotonization

In this section we give a general technique for transforming any ρ-approximate algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$ into an offline $(\rho+\varepsilon)$-approximate *monotone* algorithm $\overline{A}$. Essentially, our monotonization technique goes thorough two steps: (i) we first consider an algorithm A_γ which is noting but A running over speeds rounded to the closest power of γ, and (ii) we inspect the solutions of A_γ by varying only one of the two machine speeds over a polynomial number of values: indeed, considering only instances $(1, \gamma^i)$ will guarantee the monotonicity.

In the sequel we let A be any algorithm satisfying the following two properties:

$$w_1^A((s_{\min}, s_{\max}), J) \leq w_2^A((s_{\min}, s_{\max}), J), \tag{2}$$

$$A((s_{\min}, s_{\max}), J) = A((1, s_{\max}/s_{\min}), J). \tag{3}$$

This is without loss of generality since any offline algorithm which violates any of the two conditions above can be easily modified without any loss in the approximation guarantee.

Theorem 3. *For algorithm A and every $\varepsilon > 0$, there exists a monotone algorithm $\overline{A}$ such that, if A is a (polynomial-time) ρ-approximation algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$, then algorithm $\overline{A}$ is a monotone (polynomial-time) $(\rho + \varepsilon)$-approximation algorithm.*

Corollary 1. *For every $\varepsilon > 0$, there exists a polynomial-time $(1 + \varepsilon)$-approximation mechanism for $\mathrm{Q}_2||\mathrm{C}_{\max}$.*

Remark 1. Recently and independently from this work, the above result has been improved in [1]. The authors provided a more general technique for obtaining a monotone algorithm $\overline{A}$ for the $\mathrm{Q}_m||\mathrm{C}_{\max}$ problem. In particular, Corollary 1 can be improved so to obtain a monotone FPTAS for this problem version. Moreover, for the case $m = 2$, their technique essentially leads to the same algorithm as the one proposed here.

3.2 Online Monotonization

The basic idea is to output a "fixed" allocation that ignores the machine speeds as long as they are "almost the same": this allocation is based on the machine indexes only. As soon as one machine becomes significantly faster than the other, we assign all jobs to that machine. The algorithm template in Figure 1 implements this idea.

Algorithm A-asymmetric
1. fix a threshold $t > 1$;
2. $s_{\max} := \max\{s_1, s_2\}$; $s_{\min} := \min\{s_1, s_2\}$;
3. if $s_{\max}/s_{\min} \leq t$ then
   run online algorithm A on machine speeds $s'_1 = s'_2 = 1$;
1. else assign every job to machine of speed $s_{\max}$;

Fig. 1. An online monotone algorithm for two machines

Theorem 4. *For every $t > 1$ and for every online algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$, algorithm A-asymmetric is an online* monotone *algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$.*

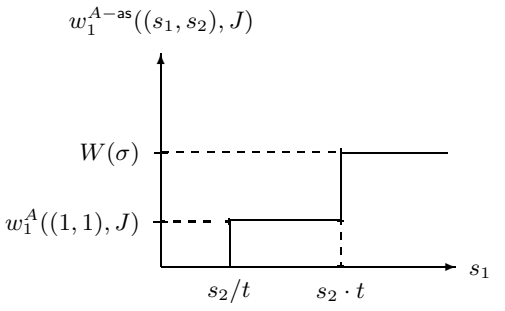

Fig. 2. The work curve $w_1^{A-\mathsf{as}}((\cdot, s_2), J)$ of algorithm A-asymmetric

Proof. The algorithm A-asymmetric is clearly an online algorithm since the choice of which strategy to use is done based on the machine speeds, which do not change during the online phase (i.e., when jobs arrive).

Let $w_i^{A-\mathsf{as}}((s_1, s_2), J)$ denote the work assigned to machine i by A-asymmetric on input J and speeds (s_1, s_2), for $i = 1, 2$. Also let $W(J) = \sum_{a=1}^{|J|} J_a$. Observe that, by definition of A-asymmetric, we have that

$$w_1^{A-\mathsf{as}}((s_1,s_2),J) = \begin{cases} w_1^A((1,1),J) & \text{if } s_1 \le s_2 \text{ and } s_1 \ge s_2/t, \\ w_1^A((1,1),J) & \text{if } s_1 > s_2 \text{ and } s_1 \le s_2 \cdot t, \\ 0 & \text{if } s_1 \le s_2 \text{ and } s_1 < s_2/t, \\ W(J) & \text{if } s_1 \ge s_2 \text{ and } s_1 > s_2 \cdot t. \end{cases} \tag{4}$$

Notice that, since $t > 1$, we have $s_2/t < s_2$. From the above equation we obtain the allocation curve in Figure 2, which clearly implies the monotonicity w.r.t. machine 1.

By using the same argument, we can prove the monotonicity of the function $w_2^{A-\mathsf{as}}((s_1, \cdot), J)$. This completes the proof.

Theorem 5. *For every ρ-competitive online algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$, and for every $t > 1$, algorithm A-asymmetric is ρ^{as}-competitive algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ for $\rho^{as} = \max\{\rho \cdot t, 1 + 1/t\}$.*

Corollary 2. *There exists an online monotone algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ whose competitive ratio is $\frac{1+\sqrt{7}}{2} \simeq 1.823$.*

Proof. Let us consider the greedy algorithm A_{gr} whose competitive ratio on two machines of identical speed is 3/2 [8]. Then, from Theorem 5 we have that algorithm A_{gr}-asymmetric has competitive-ratio bounded from above by $\max\{3t/2, 1 + 1/t\}$. We minimize this quantity by choosing $t > 1$ such that $3t/2 = 1 + 1/t$. This corresponds to $t = \frac{1+\sqrt{7}}{3}$, thus yielding a competitive ratio equal to $\frac{1+\sqrt{7}}{2} \simeq 1.823$.

4 Lower Bound for Online Selfish Scheduling

In this section, we provide a general lower bound for online $Q||C_{\max}$ with selfish agents. This result proves that the selfish online version of this problem is more difficult than the corresponding version of the problem with no selfish agents, even for two machines.

Theorem 6. *For every $m \geq 2$ and every $r > 1$, no monotone online algorithm can be less than ρ_r-competitive, where $\rho_r = \min\{r, 1 + 1/r\}$. This holds even for two jobs.*

Proof. By contradiction, let A be an online monotone ρ-competitive algorithm on m machines, for $\rho < \min\{r, 1 + 1/r\}$. Let $J = (J_1, J_2) = (1, r)$ and let $s = (1, \ldots, 1)$. Observe that $A(s, J)$ cannot allocate two jobs on the same machine otherwise A would produce a solution of cost $1 + r$, while the optimum costs r, contradicting the hypothesis that A is ρ competitive. Without loss of generality, assume $w_1^A = 1$ and $w_2^A = r$.

Suppose now that speed of machine 1 is increased to r. Since A is monotone also with respect to the sequence J^1, then it must be the case $w_1^A((r, s_{-1}), J^1) = w_1^A(s, J^1) = 1$. Since we do not allow jobs to be reassigned, we have to consider only two cases:

($w_1^A((r, s_{-1}), J) = 1 + r$.) In this case, $(1 + 1/r)/\mathsf{opt}((r, s_{-1}), J) = 1 + 1/r$, thus contradicting the hypothesis that A is ρ-competitive.

($w_1^A((r, s_{-1}), J) = 1$.) This gives $r/\mathsf{opt}((r, 1), J) = r$, contradicting the hypothesis that A is ρ-competitive.

Hence the theorem follows.

Corollary 3. *No monotone online algorithm for $Q_2||C_{\max}$ can be less than ϕ-competitive. This holds even for two jobs, in which case the bound is tight since there exists a ϕ-competitive online monotone algorithm.*

Proof. The lower bound follows from Theorem 6 by taking $r = \phi = 1+1/r$. As for the upper bound, consider algorithm A_{gr}-asymmetric with $t = \phi$. For sequences of two jobs A_{gr} is 1-competitive. Theorem 5 thus implies a competitive ratio $\rho \leq \max\{\phi, 1 + 1/\phi\} = \phi$.

5 On Building Online Monotone Algorithms

Apparently, a good way to obtain online monotone algorithms is to guarantee that faster machines receive more work. In particular, when dealing with the case of only one job, a natural (optimal) solution is to assign it to the fastest machine. This is also what a direct use of the so called *Vickery auction* [10] would give for our problem. (These so called "sealed bid" auctions compute a solution only based on the agents' bids – see e.g. [9, 2].) This motivates the following definition:

Definition 3 (best-first algorithm). *An algorithm A is* best-first *if the first job is always assigned to the fastest machine.*

In addition, it is natural to treat speeds (s_1, s_2) and $(\alpha s_2, \alpha s_1)$ as essentially the same instance: by rescaling, and reindexing machines we reduce both of them to $(1, s_2/s_1)$. Hence, the algorithm is supposed to produce the same solution. We thus consider the following class of algorithms:

Definition 4 (symmetric algorithm). *An algorithm A is* symmetric *if, for any two speed vectors s and s' such that, for a permutation π, $s' = \pi(s)$ it holds that, for all i, $w_i^A(s, J) = w_{\pi(i)}^A(s', J)$.*

A simple argument shows that any monotone algorithm which is best-first *and* symmetric cannot be less than 2-competitive, even for $m = 2$. There are, however, algorithms which are best-first though not symmetric or vice versa. Does any of these give a better performance? The next two results prove that the answer to this question is no.

Theorem 7. *For every $m \geq 2$, no online monotone best-first algorithm for $\mathrm{Q}_m||\mathrm{C}_{\max}$ can be better than 2-competitive. This holds even for two jobs.*

Proof. By contradiction, let A be a best-first, monotone and $(2-\gamma)$-competitive algorithm, for some $\gamma > 0$. Consider $J = (1, 1+\varepsilon)$, for some $\varepsilon > 0$, and let $s_1 = 1$, $s_2 = 1+\varepsilon$ and $s_i = \varepsilon$, with $3 \leq i \leq m$. Notice that, since A is $(2-\gamma)$-competitive and best-first, it is possible to take ε sufficiently small so that A assigns the first job to machine 2 and the second job to machine 1.

Suppose now that speed of machine 2 is reduced to $1 - \varepsilon$. We observe that A, on input J and $(1, 1-\varepsilon)$ assigns no jobs to machine 2. In fact, since it is best-first, it assigns the first job to machine 1. Moreover, since it is monotone, it has to assign a load to machine 2 not greater than 1. Thus, also the second job is assigned to machine 1. However, this implies that the solution computed by A has cost $2 + \varepsilon$, while the optimum has cost $1 + \varepsilon$. For ε sufficiently small, this contradicts the hypothesis that A is $(2 - \gamma)$-competitive.

Theorem 8. *For every $m \geq 2$, no online monotone symmetric algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ can be less than 2-competitive. This holds even for two jobs.*

Proof. We prove the theorem for $m = 2$. The extension to $m > 2$ is straighforward. Let us assume by contraddiciton that A is a monotone, symmetric, and $(2 - \gamma)$-competitive algorithm, for some $0 < \gamma < 1$. Consider $J = (1, 1 + \varepsilon)$, for some $\varepsilon > 0$ and let $s_1 = 1$ and $s_2 = 1 + \varepsilon$. For sufficiently small ε, algorithm A cannot allocate two jobs on the same machine. We thus have two possible solutions for algorithm A:

solution	machine 1 $s_1 = 1$	machine 2 $s_2 = 1 + \varepsilon$
SOL_1	$1 + \varepsilon$	1
SOL_2	1	$1 + \varepsilon$

Let ε be such that $\frac{2}{1+\varepsilon} > 2-\gamma$, that is, $\varepsilon < \frac{\gamma}{2-\gamma}$. We distinguish two cases:

($A((1,1+\varepsilon),J) = SOL_1$.) By monotonicity of A, $w_2^A((1,1),J) \leq w_2^A((1,1+\varepsilon),J) = 1$. If $w_2^A((1,1),J) = 0$, then we have a solution of cost $2+\varepsilon$, thus implying that A must be at least $(2+\varepsilon)/(1+\varepsilon)$-competitive. For our choice of ε, this would contradict the hypothesis that A is $(2-\gamma)$-competitive. Thus, $A((1,1),J)$ must coincide with solution SOL_1. Again, by monotonicity, it must hold $w_1^A((1+\varepsilon,1),J) \geq w_1^A((1,1),J) = 1+\varepsilon$. This contradicts the hypothesis that A is symmetric: indeed, from Definition 4 it holds that $w_2^A((1,1+\varepsilon),J) = w_1^A((1+\varepsilon,1),J) = 1+\varepsilon$, thus implying $w_1^A((1,1+\varepsilon),J) = 1$.

($A((1,1+\varepsilon),J) = SOL_2$.) Let us consider the allocation produced by A w.r.t. the first job only, that is, $J^1 = J_1 = 1$. Observe that, since jobs cannot be reassigned, it must hold $w_1^A((1,1+\varepsilon),J) = 1 = w_1^A((1,1+\varepsilon),J_1)$. Since A must be monotone also w.r.t. J^1, it holds that $w_2^A((1,1),J_1) \leq w_2^A((1,1+\varepsilon),J_1) = 0$. This implies $w_1^A((1,1),J_1) = J_1 = 1$. By monotonicity, $w_1^A((1+\varepsilon,1),J_1) = 1$. When the second job arrives, algorithm A can assign it to one of the two machines. If $w_1^A((1+\varepsilon,1),J) = J_1 + J_2 = 2+\varepsilon$, then A cannot be $(2-\gamma)$-competitive because of our choice of ε. Therefore, it must be the case that $w_1^A((1+\varepsilon,1),J) = 1$ and $w_2^A((1+\varepsilon,1),J) = J_2 = 1+\varepsilon$. This contradicts the hypothesis that A is symmetric: indeed, we have $w_1^A((1,1+\varepsilon),J) = 1 \neq w_2^A((1+\varepsilon,1),J_1) = 1+\varepsilon$.

Remark 2. Observe that our monotonization technique for offline algorithms requires the algorithm to be "monotonized" to be *both* best-first and symmetric. Thus, we implicity require the resulting algorithm to be best-first and symmetric as well.

6 Online Mechanisms with Verification

In this section we consider online mechanisms with *verification* [6]: the payments to an agent can be provided after the corresponding machine terminates; in this case, the mechanism can compute the payments as a function of such finish time(s). In the online setting, once machine j releases a job J_i, the mechanism observes a release time $r(J_i)$. However, machine j could declare to be slower (i.e., $b_j > s_j$) and release all jobs accordingly (i.e., $r(J_i) = J_i/b_j$).

In [6] we provide a sufficient condition to design truthful mechanisms:

Definition 5 (weakly monotone algorithm [6]). *An algorithm A is* weakly monotone *if, for every job sequence J, for every i, for every s_{-i} it holds that*

$$w_i^A((s_i,s_{-i}),J) = 0 \Rightarrow \forall s_i' < s_i, w_i^A((s_i',s_{-i}),J) = 0.$$

We will make use of the following result:

Theorem 9 ([6]). *An algorithm A admits a payment function p such that $M = (A, p)$ is truthful for the case of verifiable machines if and only if A is weakly monotone.*

We first observe that the greedy algorithm is weakly monotone. Therefore, we have the following result on the "power" of verification for the $\mathrm{Q}_2||\mathrm{C}_{\max}$ problem:

Theorem 10. *Let us consider the $\mathrm{Q}_2||\mathrm{C}_{\max}$ problem. There exists two functions $UB_v(\cdot)$ and $LB(\cdot)$, such that (i) no truthful mechanism can be less than $LB(r)$-competitive if machines cannot be verified, (ii) there is an $UB_v(r)$-competitive truthful mechanism for the case of verifiable machines, and (iii) if r satisfies $\sqrt{2} < r \leq \phi$, then $UB_v(r) < LB(r)$.*

Proof. Consider r such that $\sqrt{2} < r \leq \phi$, thus implying $r < 1 + 1/r$. Theorem 6 implies that no online monotone algorithm can be less than $LB(r)$-competitive, with $LB(r) = \min\{r, 1 + 1/r\} = r$. On the contrary, if verification is allowed, then the greedy algorithm is weakly monotone. Theorem 9 thus implies that its competitive ratio ρ_{gr} satisfies (see Table 1)

$$UB_v(r) \leq \rho_{gr} \leq 1 + 1/(r+1).$$

For $r > \sqrt{2}$, it holds that $r > 1 + 1/(r+1)$, thus implying $UB_v(r) < LB(r)$.

In [3] an 8-competitive algorithm Assign-R has been given. The algorithm assumes that the optimum $\mathsf{opt}(s, J)$ is known in advance and assigns a new job to the *least capable machine*, that is, the slowest machine such that the cost of the resulting assignment stays below $\Lambda \stackrel{def}{=} 2 \cdot \mathsf{opt}(s, J)$. A simple doubling technique is then used to remove this assumption at the cost of losing a factor of 4 in the approximation.

A simple observation shows that algorithm Assign-R is *not* weakly monotone. We next modify it so to obtain a weakly-monotone algorithm having a constant competitive ratio for the $\mathrm{Q}||\mathrm{C}_{\max}$ problem, i.e., for any (even non-constant) number of machines.

Algorithm Monotone-Assign-R(s, Λ):
/* $s_1 \leq s_2 \cdots \leq s_m$; */
initialize $w'_j := 0$ and $w''_j := 0$ for every machine j;
1. upon arrival of new job J_i do begin
2. let l be the slowest machine such that

$$((w''_l + J_i)/s_l \leq 2\Lambda) \wedge ((w'_l > 0) \vee (w'_{l+1} > 0));$$

3. assign J_i to machine l;
4. if $w'_l > 0$ then $w''_l := w''_l + J_i$ else $w'_l := J_i$; end.

Fig. 3. An online weakly monotone algorithm for any number of machines

Algorithm Monotone-Assign-R (see Fig. 3) receives a threshold Λ. In assigning the k^{th} job to a machine, the algorithm considers the slowest machine i for which the makespan of the resulting schedule, computed considering only the *real* jobs, does not exceed 2Λ. Then two cases are possible:

1. At least one machine faster than j has not received any load yet. Then job k is assigned to the fastest such machine and is considered a *ghost* job.
2. All machines faster than j have been assigned at least one job. In this case, job k is assigned to machine j and is considered a *real* job.

Lemma 1. *For every speed vector s and for every $\Lambda \geq 2 \cdot \mathsf{opt}(s, J)$, algorithm* Monotone-Assign-R *does not fail in assigning any newly arrived job in J. Moreover, if algorithm* Monotone-Assign-R *fails in assigning a job J_i, then $\mathsf{opt}(s, J) \geq \mathsf{opt}(s, J^i) \geq \Lambda$.*

Proof. Let J' denote the set of jobs that Monotone-Assign-R assigns to a machine which is currently empty, and $J'' \stackrel{def}{=} J \setminus J'$. Jobs in J'' are assigned according to algorithm Assign-R. Hence, if Monotone-Assign-R fails, then Assign-R fails as well. Therefore, $\mathsf{opt}(s, J^i) \geq \Lambda$ and the lemma follows.

Using a doubling technique (see e.g. [7]) one can obtain an algorithm Monotone-Assign-R which, starting from $\Lambda = 1$, doubles the value of Λ each time Monotone-Assign-R(s, Λ) fails: in this case we assign J_i, and jobs that possibly arise subsequently, by running Monotone-Assign-R with a new parameter $\Lambda' = 2\Lambda$. (We continue doubling the value of Λ until it is possible to assign J_i to some machine.) Notice that every time we double the value of Λ, we ignore the assignment performed in the previous phases (i.e., for smaller values of Λ).

Theorem 11. *Algorithm* Monotone-Assign-R *is at most* 12*-competitive.*

Proof. Let J' denote the set of jobs that Monotone-Assign-R assigns to a machine which is currently empty, and $J'' \stackrel{def}{=} J \setminus J'$. Let $\Lambda(s, J)$ denote the last value for which Monotone-Assign-R does not fail. Algorithm Monotone-Assign-R assigns jobs in J'' as algorithm Assign-R. Moreover, each machine has at most one extra job from J'. Hence, given the values w'_j and w''_j defined as in algorithm Monotone-Assign-R (see Fig. 3), we have $w'_j \leq \Lambda$ and $w''_j \leq \Lambda$. From Lemma 1 we obtain $w''_j \leq 8 \cdot \mathsf{opt}(s, J')$ and $w'_j \leq 4 \cdot \mathsf{opt}(s, J'')$. Hence, at each time step, the cost C of the solution satisfies $C \leq \max_{1 \leq j \leq m}\{w'_j + w''_j\} \leq 12 \cdot \mathsf{opt}(s, J)$.

Theorem 12. *Algorithm* Monotone-Assign-R *is weakly-monotone.*

Proof. Given the speed vector s, let $s' = (s'_i, s_{-i})$ with $s'_i < s_i$. We denote by $\Lambda(s, i)$ the value of Λ for which Monotone-Assign-R allocates job J_i. We will prove by induction on i that $\Lambda(s, i) = \Lambda(s', i)$ and that Monotone-Assign-R produces the same allocation. The base step $i = 1$ is trivial. As for the inductive step, since J_i is not allocated to machine with speed s_i, let l be the index of the machine to which J_i is allocated to. If $(w''_l + J_i)/s_l \leq \Lambda(s, i-1)$, then, by inductive hypothesis,

the same holds with respect to s', thus implying that J_i is also allocated to machine l on input s'. Clearly, in this case, $\Lambda(s', i) = \Lambda(s', i-1) = \Lambda(s, i)$. Otherwise, let $l(s)$ and $l(s')$ denote the index of the machine to which job J_i is assigned to on input s and s', respectively. In the two cases, we must increase the corresponding threshold up to a value such that $(w''_{l(s)} + J_i)/s_{l(s)} \leq \Lambda(s, i)$ and $(w''_{l(s')} + J_i)/s_{l(s')} \leq \Lambda(s', i)$. Hence, $\Lambda(s', i) = \Lambda(s, i)$ and $l(s') = l(s)$. (The latter equality follows from Step 2 in Monotone-Assign-R(s, Λ)). By inductive hypothesis, the allocation of J^{i-1} is the same, thus implying that also job J_i is allocated to the same machine.

Finally, using the payment functions for weakly monotone algorithms of [6], we can obtain the following:

Corollary 4. *The* Q$||$C$_{\max}$ *problem with verifiable machines admits an online truthful polynomial-time mechanism which is* 12*-competitive.*

Acknowledgements. We are grateful to the authors of [1] for providing us with a copy of their work.

References

1. N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Annual Symposium on Theoretical Aspects of Computer Science (**STACS**)*, volume 3404 of *LNCS*, pages 69–82, 2005.
2. A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the IEEE Symposium on Foundations of Computer Science (**FOCS**)*, pages 482–491, 2001.
3. J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.
4. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Annual Symposium on Theoretical Aspects of Computer Science (**STACS**)*, volume 2996 of *LNCS*, pages 608–619. Springer, 2004.
5. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. On designing truthful mechanisms for online scheduling. Technical report, European Project CRESCCO, http://www.ceid.upatras.gr/crescco/, 2004.
6. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. The power of verification for one-parameter agents. In *International Colloquium on Automata, Languages, and Programming (**ICALP**)*, volume 3142 of *LNCS*, 2004.
7. Y. Azar. *Online load balancing.* Springer, 1998. In Online algorithms - the state of the art, pag. 178-195.
8. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
9. N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (**STOC**)*, pages 129–140, 1999.
10. W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, pages 8–37, 1961.