On Designing Truthful Mechanisms for Online Scheduling[*]

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

[1] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, via **Ponte Don Melillo**, I-84084 Fisciano (SA), Italy, e-mail: {`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 $\mathrm{Q}||\mathrm{C}_{\max}$ involving *selfish agents*, considered by Archer and Tardos in [FOCS 2001], where jobs must be scheduled on m related machines, each of them owned by a different *selfish agent*.
We present a general technique for transforming competitive online algorithms for $\mathrm{Q}||\mathrm{C}_{\max}$ into truthful online mechanisms with a small loss of competitiveness.
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. A "direct" use of such natural algorithms to construct truthful mechanisms yields only trivial upper bounds for the case of two machines.
Finally, we consider mechanisms *with verification* introduced by Nisan and Ronen [STOC 1999] for offline scheduling problems. We present the first constant-competitive online truthful mechanism with verification for *any* number of machines.

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 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".

[*] Work supported by the European Project IP-FP6-015964 Aeolus – Algorithmic Principles for Building Efficient Overlay Computers.

Indeed, 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 to the protocol 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 globally optimal.

Let us consider the following basic routing/scheduling problem (first introduced by Nisan and Ronen [9]). We want to route *one* packet of size w over one of m parallel links with different speeds: link i takes w/s_i units of time to process this packet and the value of s_i is known to agent i only (this is the speed of his/her link). Why should agent i report s_i, instead of a different value r_i? Given that $1/s_i$ represents the cost for agent i of processing a packet of unit size, reporting a "very high" value r_i could let the algorithm to select some other link to route the packet. In this way agent i would have a benefit, since he/she will have no cost, but the resource allocation produced by the algorithm could be sub 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 agent 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) given to the agents. Selfish agents are supposed to be rational and thus will deviate from the truth-telling strategy (in the previous case, 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).

1.1 The Problem

Offline Selfish Scheduling. 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. The version of the problem where the number m of the machines is fixed is commonly denoted as $\mathrm{Q}_m||\mathrm{C}_{\max}$.

We study the selfish version of the $\mathrm{Q}||\mathrm{C}_{\max}$ problem considered by Archer and Tardos [2]. In this model each machine i is owned by a selfish agent and the corresponding speed s_i is known to that agent only. We call $t_i \overset{\text{def}}{=} 1/s_i$ the *type* of agent i. Agent i reports to the algorithm a value b_i, not necessarily equal to t_i, and the algorithm computes a schedule S that minimizes the makespan with respect to the values reported by the agents.

Each agent i is rational and he/she has his/her own valuation $v_i(X)$ for each possible schedule X. Intuitively, $v_i(X)$ represents how much user i likes schedule X. More specifically, a schedule X that assigns a total amount of work w_i to machine i is valuated by agent i of type t_i as $v_i(X)$, where

$$v_i(X) \overset{\text{def}}{=} -w_i \cdot t_i,$$

that is, the opposite of the completion time of machine i.

We remark that agent i selects the value b_i to be reported to the algorithm in such a way to have the algorithm output a schedule that he/she likes more.

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) to the each agent i in order to let him/her truthfully report his/her speed.

Formally, a *mechanism* is a pair $M = (A, P)$, where $P = (P_1, \ldots, P_m)$, and A is a scheduling algorithm. Algorithm A gets in input the list of jobs to process J and the types $b = (b_1, \ldots, b_m)$ reported by the agents, not necessarily equal to the true types and computes a schedule $A(b, J)$. Moreover, each agent i receives a payment equal to $P_i(b, J)$. Obviously, each agent i decides his/her strategy in such a way to maximize the resulting net profit or *utility* defined as

$$u_i^M(b, J) \overset{\text{def}}{=} P_i(b, J) + v_i(A(b, J)).$$

Each agent *knows* both algorithm A and the corresponding payment function P_i.

A mechanism is said to be *truthful with dominant strategies* (or simply *truthful*) if the payments P 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 $b_{-i} \overset{\text{def}}{=} (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m)$ 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 in this case 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 (i.e., the work and the payment assigned to agent i depend only on his own bid b). If M guarantees that the utility for all agents that report their true type is non-negative, then we say that the mechanism enjoys the *voluntary participation* property.

Online Selfish Scheduling. In the *online* version of $\mathrm{Q}||\mathrm{C}_{\max}$, 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 the first job appears, each agent declares her type and we denote by $b = (b_1, \ldots, b_m)$

the vector of declared types. We remark that declared types cannot be changed during the processing of the algorithm. An *online mechanism* for Q||C$_{\max}$ is a pair $M = (A, P)$ where A is an online algorithm for Q||C$_{\max}$and P is a sequence of payment functions P_i^k, for $i = 1, \ldots, m$ and $k > 0$ such that

- $w_i^A(b, J^k)$ is the sum of the sizes of the jobs assigned to machine i by the solution computed by A on input J^k and the vector b of declared types;
- $P_i^k(b, J^k)$ is the *non-negative* payment assigned to agent i after the k–th job is arrived and it has been assigned to a machine by algorithm A.

Observe that the mechanism is not allowed to ask money back from the agents. The total payment received by agent i after k jobs are processed is equal to $P_i(b, J^k) = \sum_{j=1}^{k} P_i^j(b, J^j)$.

The property of truthfulness is naturally extended to the online setting.

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 the sequence of jobs 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 $b_i = t_i$.*

Mechanisms With Verification. We also study the online version of a different model of mechanisms, proposed by Nisan and Ronen [9]. 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 work, the mechanism can verify whether the machine lied declaring to be *faster* and, if so, the machine receives no payment. These mechanisms are usually termed mechanisms *with verification*. Mechanisms that always provide an agent the associated payment are sometimes called mechanisms *without verification* or simply mechanisms.

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 (see Definition 2) then there exist payment functions P such that (A, P) is truthful. Under mild assumptions on A, it is possible to define the payment functions to guarantee also the voluntary participation property. They also gave a monotone optimal (exponential-time) algorithm and a $(3+\varepsilon)$-approximate randomized (polynomial-time) monotone algorithm for Q||C$_{\max}$. Andelman *et al* [1] provided an elegant technique for turning any ρ-approximate algorithm for Q$_m$||C$_{\max}$ into a $\rho(1+\varepsilon)$-approximate monotone mechanism. As a result, given any polynomial-time $(1+\varepsilon)$-approximate algorithm for this problem, one can obtain a $(1+\varepsilon)$-approximate 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. A 3-approximate truthful mechanism for any number of machines has been presented by Kovacs [8].

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, for any $\varepsilon > 0$, no deterministic truthful mechanism can be $(2 - \varepsilon)$-approximate for $m \geq 2$ machines. Nisan and Ronen also introduced mechanisms with verification and gave a polynomial-time $(1 + \varepsilon)$-approximate truthful mechanism for any fixed number of unrelated machines whose execution times are bounded by some constant. This result also holds for any (non-constant) number of related machines [4]. Truthful mechanisms with verification for related machines have been characterized in [4].

1.3 Our Contribution

A central question in (algorithmic) mechanism design is how to translate approximation/online algorithms into approximation/online mechanisms. 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. The following question is of interest: given an algorithm A of approximation/competitive ratio ρ, can we obtain a monotone algorithm $\overline{A}$ with the same approximation/competitive ratio? In this paper, we try to give answers to this question using the $\mathrm{Q}||\mathrm{C}_{\max}$ problem as case of study.

We first consider online mechanisms for $\mathrm{Q}_2||\mathrm{C}_{\max}$. We show that any online ρ-competitive algorithm A can be turned into a $\overline{\rho}$-competitive online monotone algorithm $\overline{A}$ such that $\overline{\rho} \leq \max\{\rho \cdot t, 1 + 1/t\}$, for every $t \geq 1$. Actually, we prove a stronger result since algorithm A needs to be ρ-competitive only for the case of *identical speeds* (Theorem 4). In particular, the "monotonization" of the greedy algorithm [3] yields an online mechanism whose competitive ratio is at most $1 + \sqrt{7}/2 < 1.823$ (Corollary 1).

Concerning the issue of designing new online monotone algorithms and/or adapting existing ones, we observe that there is a common idea in the design of several approximation/online algorithms that is used also in the Vickrey auction (see e.g. [9]): 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. A similar negative result applies to all algorithms which assign the first job to the fastest machine. These results show that, if one wants to obtain non-trivial upper bounds for two machines, then one has to design "unnatural" algorithms.

We also consider the case of an arbitrary number of machines. First of all, observe that all our lower bounds given for the case $m = 2$ also apply to $\mathrm{Q}_m||\mathrm{C}_{\max}$, for any $m > 2$. As for the upper bounds, in Sect. 5 we present a 12-competitive

[3] This algorithm, also known in the literature as ListScheduling, assigns the current job J_k to the machine that minimizes the completion time of J_k.

online mechanism *with verification* for *any number* of machines. This is the first constant-competitive truthful online algorithm for a non-constant number of machines, albeit with verification.

Notation. Throughout the paper s_i will denote the speed of the i^{th} machine, t_i is its type (i.e., $t_i = 1/s_i$) and b_i is the type reported by agent i to the mechanism. We denote by $\mathsf{cost}(X, (s, J))$ the cost of the schedule X of the jobs in J with respect to the speed vector s and we denote by $\mathsf{opt}(s, J)$ the cost of an optimal schedule of jobs in J with respect to the speed vector $(s$.

All definitions of game theoretic concepts in the rest of the paper are to be intended for the $\mathrm{Q}||\mathrm{C}_{\max}$ problem.

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*

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

where $w_i^A((b_i, b_{-i}), J)$ is the work 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*

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

Next, we translate the above result into the online setting. We will use this 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 exist payment functions P such that (A, P) is truthful. Moreover, there exist payment functions P_i^k such that $P_i^k((b_i, b_{-i}), J^k) \ge 0$ for all J, k and (b_i, b_{-i}).*

Proof Sketch. We only prove the last part of the theorem. The remaining of the proof can be obtained from proof of Theorem 1 in [2].

Define $P_i^{k+1}((b_i, b_{-i}), J^{k+1})$ as

$$P_i^{k+1}((b_i, b_{-i}), J^{k+1}) \overset{def}{=} P_i((b_i, b_{-i}), J^{k+1}) - P_i((b_i, b_{-i}), J^k).$$

Observe that, since we do not allow to reassign jobs, it holds that, for every b_i,

$$w_i^A((b_i, b_{-i}), J^k) \leq w_i^A((b_i, b_{-i}), J^{k+1}).$$

Thus, by Eq. (1) we have that

$$P_i^{k+1}((b_i, b_{-i}), J^{k+1}) = b_i \cdot w_i^A((b_i, b_{-i}), J^{k+1}) - b_i \cdot w_i^A((b_i, b_{-i}), J^k) \geq 0.$$

□

3 Online Monotonization

In this section, we give a general technique for transforming an online algorithm for $Q_2||C_{\max}$ into an online *monotone* algorithm for $Q_2||C_{\max}$. Based on this transformation, we present an online truthful mechanism whose competitive ratio is about 1.823.

Let A be an online algorithm for the $Q_2||C_{\max}$ problem. The basic idea to obtain a monotone online algorithm from A is to distinguish two cases: if a machine is significantly faster than the other we assign all jobs to that machine; if, instead, machines speeds are "almost the same", we run algorithm A to produce a fixed schedule that does depends only on the machine indexes and not on their speeds. The algorithm template in Figure 1 implements this idea.

```
Algorithm $t$-A-mon$((s_1, s_2), J)$
 1. $s_{\max} := \max\{s_1, s_2\}$; $s_{\min} := \min\{s_1, s_2\}$;
 2. if $s_{\max}/s_{\min} \leq t$ then
        run online algorithm $A((1,1), J)$;
 1. else assign every job to machine of speed $s_{\max}$;
```

Fig. 1. An online monotone algorithm for two machines.

Theorem 3. *For every $t > 1$ and for every online algorithm A for $Q_2||C_{\max}$, algorithm t-A-mon is an online* monotone *algorithm for $Q_2||C_{\max}$.*

Proof. Observe that when algorithm t-A-mon starts, depending on the machine speeds and on the parameter t it selects to process all the jobs either with the online algorithm A or with the online algorithm that assigns all jobs to the fastest machine. Thus t-A-mon is an online algorithm too.

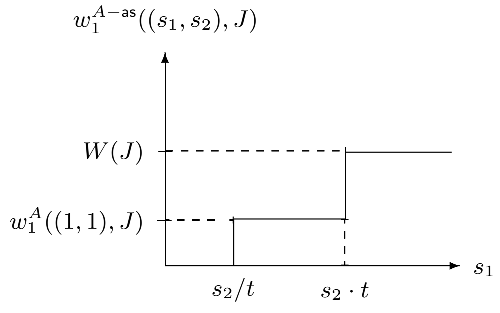

Fig. 2. The curve of the work $w_1^{t-\mathsf{A-mon}}((s_1,s_2),J)$ assigned by algorithm t-$\mathsf{A\text{-}mon}$ to machine 1.

Let $w_i^{t-\mathsf{A-mon}}((s_1,s_2),J)$ denote the work assigned to machine i by t-$\mathsf{A\text{-}mon}$ on input J and (s_1,s_2). Also let $W(J)=\sum_{a=1}^{|J|} J_a$ be the sum of all jobs' sizes. Observe that, by definition of t-$\mathsf{A\text{-}mon}$, we have that

$$w_1^{t-\mathsf{A-mon}}((s_1,s_2),J)=\begin{cases} w_1^A((1,1),J) & \text{if } s_1 \le s_2 \text{ and } s_1 \ge s_2/t,\\ w_1^A((1,1),J) & \text{if } s_1 > s_2 \text{ and } s_1 \le s_2\cdot t,\\ 0 & \text{if } s_1 \le s_2 \text{ and } s_1 < s_2/t,\\ W(J) & \text{if } s_1 \ge s_2 \text{ and } s_1 > s_2 \cdot t.\end{cases} \tag{2}$$

Notice that, since $t>1$, we have $s_2/t < s_2$. From the above equation we obtain the curve in Figure 2 that gives the work allocated to machine 1 by the algorithm. The figure clearly implies the monotonicity w.r.t. machine 1. The very same argument shows the monotonicity w.r.t. machine 2. Hence the theorem follows.

Theorem 4. *Let A be an online algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ which is ρ-competitive for the special case where machines have identical speeds. Then, for every $t>1$, the resulting online algorithm t-$\mathsf{A\text{-}mon}$ is $\overline{\rho}$-competitive for $\mathrm{Q}_2||\mathrm{C}_{\max}$, where $\overline{\rho}=\max\{\rho\cdot t, 1+1/t\}$.*

Proof. Assume that the speeds of the two machines are $s_1=1$ and $s_2=r\ge 1$. We first observe that assigning all jobs to the fastest machine yields a solution of cost at most $1+1/r$ times the cost of an optimal solution. Therefore, if $r>t$, algorithm t-$\mathsf{A\text{-}mon}$ is $(1+1/t)$-competitive. If $r\le t$, instead, algorithm t-$\mathsf{A\text{-}mon}$ runs algorithm A and computes a solution X_A whose makespan is at most ρ times the cost of an optimal scheduling of the same set of jobs J on two machines with identical speeds, that is $\mathsf{cost}(X_A,((1,r),J)) \le \rho\cdot\mathsf{opt}((1,1),J)$. We next show that $\mathsf{cost}(X_A,((1,r),J)) \le \rho\cdot r\cdot\mathsf{opt}((1,r),J)$.

First observe that for any scheduling X we have that

$$\mathsf{cost}(X,((1,1),J)) \ge \mathsf{cost}(X,((1,r),J)) \ge \frac{\mathsf{cost}(X,((1,1),J))}{r}.$$

Moreover, if X^* denotes an optimal solution for the instance $((1,s),J)$, then we have that

$$\mathsf{opt}((1,s),J) \geq \frac{\mathsf{cost}(X^*,((1,1),J))}{s} \geq \frac{\mathsf{opt}((1,1),J)}{s}.$$

From the above inequality and using the fact that the algorithm A is ρ-competitive, we obtain

$$\begin{aligned} \mathsf{cost}(X_A,((1,r),J)) &\leq \mathsf{cost}(X_A,((1,1),J)) \\ &\leq \rho \cdot \mathsf{opt}((1,1),J) \\ &\leq \rho \cdot r \cdot \mathsf{opt}((1,r),J). \end{aligned} \tag{3}$$

Suppose now that machine speeds are s_1 and $s_2 \geq s_1$. Using a simple rescaling argument, we have that

$$\mathsf{cost}(X_A,((s_1,s_2),J)) = \frac{\mathsf{cost}(X_A,((1,s_2/s_1),J))}{s_1}$$

and

$$\mathsf{opt}((s_1,s_2),J) = \frac{\mathsf{opt}((1,s_2/s_1),J)}{s_1}.$$

Then, using Eq. (3), we have that algorithm$\mathsf{A\text{-}mon}$ is $(\rho \cdot r)$-competitive. Since we are in the case $r \leq t$, the theorem follows.

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

Proof. The greedy algorithm on is 3/2-competitive for the special case of $\mathrm{Q}_2||\mathrm{C}_{\max}$ where machines have identical speeds [7]. From Theorem 4, using the greedy algorithm in the algorithm t-$\mathsf{A\text{-}mon}$ we obtain a monotone algorithm whose competitive-ratio is at most $\max\{3t/2, 1+1/t\}$, where $t > 1$ can be chosen arbitrarily. In particular, for $t = \frac{1+\sqrt{7}}{3}$, the competitive ratio is equal to $\frac{1+\sqrt{7}}{2} \simeq 1.823$.

4 On Building Online Monotone Algorithms

In this section we show that a large class of "natural" monotone algorithms, including most of the known algorithms for scheduling, cannot achieve a competitive ratio smaller than 2. This lower bound implies that, for $m = 2$, these algorithms cannot improve over the trivial 2-approximation monotone algorithm that assigns all the jobs to the machine that declares to be faster.

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). *A scheduling algorithm A is* best-first *if it always assigns the first job to the fastest machine.*

In addition, for each s_1 and $s_2 \geq s_1$, it is natural to treat speed vectors $s' = (s_1, s_2)$ and $s'' = (\alpha s_2, \alpha s_1)$ as essentially the same instance: by rescaling, and reindexing machines we can reduce both of them to the instance $(1, s_2/s_1)$. Hence, an algorithm is supposed to produce the same solution for all the three instances. We thus consider the following class of algorithms:

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

We prove now that each algorithm which is either best-first or symmetric cannot be less than 2-competitive.

Theorem 5. *No online monotone best-first algorithm can be better than 2-competitive. This holds even for the case of two jobs and two machines.*

Proof. By contradiction, let A be a best-first, monotone and $(2-\gamma)$-competitive algorithm, for some $\gamma > 0$. Consider instance (s, J), where $J = (1, 1+\varepsilon)$, for some $\varepsilon > 0$, and let $s_1 = 1$, $s_2 = 1+\varepsilon$, and $s_i = \varepsilon$ for $3 \leq i \leq m$. 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.

Consider now a new instance (s', J), where s' is equal to s except for $s'_2 = 1-\varepsilon$. We observe that for this instance algorithm A 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 work to machine 2 not greater than 1. Thus, also the second job is assigned to machine 1. However, this implies that $\mathsf{cost}(A(s', J) = 2+\varepsilon$, while the optimum has cost $1+\varepsilon$. For ε sufficiently small, this contradicts the hypothesis that A is $(2-\gamma)$-competitive.

Theorem 6. *No online monotone symmetric algorithm can be better than 2-competitive. This holds even for the case of two jobs and two machines.*

Proof. We prove the theorem for $m = 2$. The extension to $m > 2$ is straightforward. Let us assume by contradiction that A is a monotone, symmetric, and $(2-\gamma)$-competitive algorithm, for some $\gamma > 0$. Consider the three instances $((1, 1+\varepsilon), J)$, $((1+\varepsilon, 1), J)$ and $((1, 1), J)$, where $J = (1, 1+\varepsilon)$, for some $\varepsilon > 0$. It can be easily seen that for ε sufficiently small ($\varepsilon < \frac{\gamma}{2-\gamma}$) with respect to all the three instances algorithm A cannot allocate both the jobs to the same machine, otherwise it contradicts the hypothesis that it is $2-\gamma$-competitive.

Thus, for each instance algorithm A can output one of two possible solutions:

solution	machine 1	machine 2
SOL_1	$1+\varepsilon$	1
SOL_2	1	$1+\varepsilon$

Consider now the solutions produced on input the instance $((1, 1 + \varepsilon), J)$. We distinguish two cases:

- $A((1, 1 + \varepsilon), J) = SOL_1$
  Since A is symmetric, from Definition 4 it holds that
  $$w_1^A((1 + \varepsilon, 1), J) = w_2^A((1, 1 + \varepsilon), J) = 1. \tag{4}$$
  Observe that $w_2^A((1, 1), J) = 1$: indeed, since A is monotone, we have that $w_2^A((1, 1), J) \le w_2^A((1, 1+\varepsilon), J) = 1$; since A is $(2-\gamma)$-competitive, $w_2^A((1, 1), J) > 0$. Thus, algorithm A on input $((1, 1), J)$ must give in output the solution SOL_1.
  Moreover, by the monotonicity of A, it must also hold that $w_1^A((1+\varepsilon, 1), J) \ge w_1^A((1, 1), J) = 1 + \varepsilon$. contradicting Eq.4.
- $A((1, 1 + \varepsilon), J) = SOL_2$
  Observe that algorithm A assigns the job J_1 to machine 1 and, since reassignment of jobs is not allowed, it will assign this job to the same machine even if we consider the sequence of jobs $J^1 = (J_1)$. By the monotonicity of algorithm A we have that $w_2^A((1, 1), J_1) \le w_2^A((1, 1 + \varepsilon), J_1) = 0$ which implies that $w_1^A((1, 1), J_1) = 1$. Again, by the monotonicity of algorithm A, $w_1^A((1 + \varepsilon, 1), J_1) = 1$.
  Consider now the assignment of the job J_2 with respect to the speed vector $(1 + \varepsilon, 1)$. This job cannot be assigned to machine 1, otherwise the solution returned by A would have cost equal to $2+\varepsilon$ and the competitive ratio would be $\frac{2+\varepsilon}{1+\varepsilon} > 2 - \gamma$, contradicting the hypothesis that A is $(2 - \gamma)$-competitive. Therefore, it must be the case that $w_1^A((1 + \varepsilon, 1), J) = 1$ and $w_2^A((1 + \varepsilon, 1), J) = J_2 = 1 + \varepsilon$, contradicting the hypothesis that A is symmetric.

5 Online Mechanisms with Verification

In this section we consider online mechanisms with *verification* [9, 4]. In these mechanisms the payment to an agent can be provided after the corresponding machine terminates; in this case, the mechanism can compute the payment as a function of such finish time(s). In the online setting, once machine j releases a job J_i, the mechanism observing its release time $r(J_i)$ can compute the time taken by the machine to process the job and compare it with the type reported by the agent at the beginning of the processing: if the agent declared to be faster than it really is ($b_j < s_j$), the mechanism recognizes it lied and assigns no payment to the agent. However, agent j could still declare to be slower (i.e., $b_j > s_j$), release all jobs accordingly (i.e., $r(J_i) = J_i/b_j$) and be not caught by the mechanism.

In [4] we show that truthful mechanisms with verification allow to use algorithms which satisfy a weaker form of monotonicity:

Definition 5 (roughly monotone algorithm [4][4]). *An algorithm A is* roughly monotone *if, for every job sequence J, for every i, for every s_{-i} it holds that*

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

The following result has been proved for offiline algorithms/mechanisms for Q||C$_{\max}$. Its extension to the online case is straightforward and it uses the same arguments as in the proof of Theorem 2.

Theorem 7 (essentially due to [4]). *If $M = (A, P)$ is an online truthful mechanism with verification then A is roughly monotone. Moreover, for every roughly monotone, constant competitive algorithm A, there exists a payment function P such that (A, P) is an online truthful mechanism with verification satisfying the voluntary participation.*

5.1 Online Mechanisms for Arbitrary Number of Machines

In [3] an 8-competitive algorithm Assign-R has been given for the online Q||C$_{\max}$ problem. Let s be the speed vector and let J be the list of jobs already scheduled.

The algorithm receives an extra parameter Λ and for each new job assigns it to the *least capable machine*, that is, the slowest machine such that the cost of the resulting assignment stays below 2Λ. The following lemma is a reformulation of the result proved in [3].

Lemma 1 (essentially due to [3]). *For every speed vector s and for every $\Lambda \geq \mathsf{opt}(s, J)$, algorithm* Assign-R *does not fail in assigning any newly arrived job in J. Moreover, if algorithm* Assign-R *fails in assigning a job J_k, then $\mathsf{opt}(s, J) \geq \mathsf{opt}(s, J^k) > \Lambda$.*

Then, if the optimum is known in advance we can run algorithm Assign-R with $\Lambda = \mathsf{opt}(s, J)$ and obtain a 2-competitive assignment. Using a simple doubling technique (see e.g. [5]) one can obtain an algorithm Assign-R which, starting with $\Lambda = 1$, doubles the value of Λ each time Assign-R(s, Λ) fails in assigning a job J_i. In this case a new instance of Assign-R with a new parameter $\Lambda' = 2\Lambda$ is run to assign job J_k and jobs that possibly arise after it. (We continue doubling the value of Λ until it is possible to assign J_k to some machine.) Each instance of Assign-R computes a new assignments independently from the assignments computed by the previous instances. This technique can increment the competitive ratio of the algorithm by at most a factor of 4: a factor of 2 is due to the work assigned in all the previous phases except for the last; another factor of 2 is due to the approximation of Λ.

Theorem 8 (due to [3]). *Algorithm* Assign-R *is at most 8-competitive.*

[4] The conference version of [4] uses the term 'weakly monotone' in place of 'roughly monotone' employed in the full version.

Observe that algorithm Assign-R is *not* roughly monotone. In fact, since jobs are assigned to least capable machines, a machine can receive no jobs to process when it declares its real speed but it can receive some jobs when it declares to be slower. To avoid this, we have to guarantee that if a machine j receives a positive work all machines faster than j receive positive work too. In the following, we show how to modify algorithm Assign-R in order to obtain a roughly monotone algorithm for $\mathrm{Q}||\mathrm{C}_{\max}$ having a constant competitive ratio.

```
Algorithm Monotone-Assign-R($s, \Lambda$):
/* $s_1 \leq s_2 \cdots \leq s_m$; */
initialize $w'_i := 0$ and $w''_i := 0$ for every machine $i$;
 1. upon arrival of new job $J_k$ do begin
 2. let $j$ be the slowest machine such that
                         $((w''_j + J_k)/s_j \leq 2\Lambda)$;
 3. if there exists a machine $l$ faster than $j$ with $w'_l = 0$
 4.          let $l^*$ be the fastest machine with $w'_{l^*} = 0$;
 5.          assign $J_k$ to machine $l^*$ and set $w'_{l^*} = J_k$;
 6. else assign $J_k$ to machine $j$ and set $w''_j := w''_j + J_k$;
 7. end.
```

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

We partition jobs in two sets: the *real* jobs that are jobs assigned to machines using algorithm Assign-R, and the *ghost* jobs that are assigned to machines according to a different rule, to guarantee that the roughly monotonicity condition holds. For each machine j let w'_j and w''_j be the sums of weights of ghost and real jobs assigned to this machine, respectively.

Algorithm Monotone-Assign-R (see Fig. 3) receives a threshold Λ. In assigning a new job, 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 (step 2). Then, two cases can occur:

1. if there exists a machine faster than j that has received no work yet, job J_k is assigned to the fastest of such machines and it is considered as a ghost job, that is it will not be considered by algorithm Assign-R (step 5);
2. if all machines faster than j have been assigned at least one job, then job J_k is assigned to machine j and it is considered as a real job (step 6).

Lemma 2. *For every speed vector s and for every $\Lambda \geq \mathsf{opt}(s, J)$, 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_k, then* $\mathsf{opt}(s, J) \geq \mathsf{opt}(s, J^k) > \Lambda$.

Proof. Let J' denote the set of ghost jobs, and $J'' \overset{def}{=} J \backslash J'$ be the set of real jobs. We remark that the partition in ghost and real jobs depends on the algorithm.

Notice that algorithm Monotone-Assign-R can fail only in step 2, if it cannot find any machine that can schedule next job in time not greater than 2Λ. Observe that in this step the algorithm considers only the work due to real jobs already assigned to machines. Thus, we can restrict our analysis only to real jobs.

Real jobs are assigned according to algorithm Assign-R, without considering ghost jobs. Hence, by Lemma 1, if $\Lambda \geq \mathsf{opt}(s, J) \geq \mathsf{opt}(s, J'')$ we have that algorithm Assign-R never fails in allocating jobs in J'', and this implies that algorithm Monotone-Assign-R never fails in allocating jobs in J.

If, instead, Monotone-Assign-R fails in allocating a job J_k, then Assign-R fails as well in allocating all real jobs in J^k. Therefore, by Lemma 1 $\mathsf{opt}(s, J^k) \geq \Lambda$. Since $\mathsf{opt}(s, J) \geq \mathsf{opt}(s, J^k)$ the lemma follows.

Using a doubling technique as in [5] 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 the assignemnt made for a particular value of Λ, is independent from the assignments computed for smaller values of Λ).

Theorem 9. *Algorithm* Monotone-Assign-R *is at most* 12*-competitive.*

Proof. Let J' denote the set of ghost jobs, and $J'' = J \setminus J'$ be the set of real jobs. Moreover, let $\Lambda(s, J)$ denote the last value for which Monotone-Assign-R does not fail. By Lemma 1 we have that $\Lambda(s, J) \leq 2\mathsf{opt}(s, J)$.

Observe that algorithm Monotone-Assign-R assigns real jobs in J'' according to the algorithm Assign-R and, by Theorem 8 we have that for each j the time necessary to process all the real jobs assigned to machine j is

$$w''_j / s_j \leq 8\mathsf{opt}(s, J'') \leq 8\mathsf{opt}(s, J).$$

Moreover, the algorithm Monotone-Assign-R assigns at most one ghost job to machine j, having weight w'_j. Notice that this job is assigned to machine j while using the bound $\Lambda' \leq \Lambda(s, J)$ if there exists at least a machine slower than j that can finish to process this job, and all the jobs previously assigned to it, in at most $2\Lambda'$. Thus, we have that

$$w'_j / s_j \leq 2\Lambda' \leq 2\Lambda(s, J) \leq 4\mathsf{opt}(s, J).$$

Let X be the cost of the solution computed by Monotone-Assign-R(s, J). We can state that

$$\mathsf{cost}(X, (s, J)) \leq \max_{1 \leq j \leq m} \{\frac{w'_j + w''_j}{s_j}\} \leq 12 \cdot \mathsf{opt}(s, J)$$

and the Theorem follows.

Theorem 10. *Algorithm* Monotone-Assign-R *is roughly monotone.*

Proof. Consider two instances (s, J) and (s', J) such that $J = J_1, J_2, \cdots$ is a sequence of jobs and $s = (s_j, s_{-j})$ and $s = (s'_j, s_{-j})$ are speed vectors with $s'_j < s_j$. With a little abuse of notation we denote by $w_j(s, J)$ the work assigned by algorithm Monotone-Assign-R to machine j on input the instance (s, J) and by $\Lambda(s, k)$ the value of Λ for which Monotone-Assign-R allocates job J_k with respect to the speed vector s. To prove the Theorem we have to show that for each job sequence J, if $w_j(s, J) = 0$ then $w_j(s', J) = 0$. As a matter of fact, we will prove a stronger result. In fact, we will prove by induction on k that $\Lambda(s, k) = \Lambda(s', k)$ and that Monotone-Assign-R produces the same allocation for the two instances.

The base step $k = 1$ is trivial. Let $l \neq j$ be the machine that receive job J_1 with respect to s. This means that there exists no machine (in particular j) that can process this job in time $\Lambda(s, 0)$ and l is the fastest machine that can process it in time not greater than $2\Lambda(s, 0)$. Obviously, both these two properties are still true if we reduce speed of machine j from s to s'.

Suppose now by inductive hypothesis that $\Lambda(s, k-1) = \Lambda(s', k-1)$ and the allocations of the jobs in J^{k-1} computed with respect to s and s' are equal. Let $l \neq j$ be the machine that receive job J_k with respect to s. We distinguish two cases. If $\Lambda(s, k) = \Lambda(s, k-1)$ then either j cannot process job J_k in time $2\Lambda(s, k)$ or it can but l is faster than j and it has no work. In both cases the same allocation will be chosen even if we reduce the speed of machine j. If $\Lambda(s, k) > \Lambda(s, k-1)$, instead, we have that no machine is able to process job J_k in time $\Lambda(s, k)$, and thus $J_k/s_j > \Lambda(s, k)$, but there exists a machine l, distinct from j, that is able to process the job within time $2\Lambda(s, k)$. Obviously, this is still true if we reduce speed of machine j. Thus $\Lambda(s, k) = \Lambda(s', k)$. Moreover, the algorithm does not assign the job J_k to machine j since either it is too slow to process it or there exists a machine faster than j that can process it and it received no job in previous steps. In both the cases the algorithm will compute the same assignment if we reduce the speed of machine j. This concludes the proof.

By combining the previous theorems with Theorem 7 we obtain the following:

Corollary 2. *There exists an online truthful mechanism with verification which is* 12*-competitive for any number of machines.*

5.2 Lower Bounds for Mechanisms With Verification

We conclude by observing that the lower bound we proved in Theorem **??** applies also to mechanisms with verification. Indeed, the same proof holds if we replace 'monotone' with 'roughly monotone' and, for sequences of just one job, roughly monotone algorithms are also monotone. We can thus argue, as in the proof of Theorem **??**, that the first job must be allocated to the same machine when its speed is increased from 1 to $r > 1$. The rest of the proof goes according to the same line and thus the following holds:

Corollary 3. *No online truthful mechanism with verification can be less than* $\rho(r)$*-competitive, where* r *is the maximum ratio between machine speeds and*

$\rho(r) = \min\{r, 1 + 1/r\}$. *Hence, if no assumption is made on* r *then no such mechanism can be less than* ϕ*-competitive. These bounds hold even for the case of two jobs and two machines.*

Observe that the greedy algorithm provides a matching upper bound for the case of two machines (see Table **??**):

Corollary 4. *There exists an online truthful mechanism with verification for the problem* $\mathrm{Q}_2||\mathrm{C}_{\max}$ *which is* ϕ*-competitive. If the speeds ratio* r *is at least* ϕ*, then the mechanism is* $(1 + (1/r))$*-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. The power of verification for one-parameter agents. In *International Colloquium on Automata, Languages, and Programming (**ICALP**)*, volume 3142 of *LNCS*, 2004.
5. Y. Azar. *Online load balancing.* Springer, 1998. In Online algorithms - the state of the art, pag. 178-195.
6. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM J. on Computing*, 9(1):91–103, 1980.
7. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
8. A. Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *Annual European Symposium on Algorithms (**ESA**)*, volume 3669 of *LNCS*, pages 619–627, 2005.
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.