Theoretical Computer Science 406 (2008) 173–186

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Deterministic monotone algorithms for scheduling on related machines☆

Pasquale Ambrosio, Vincenzo Auletta*

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Universitá di Salerno, via Ponte Don Melillo, I-84084 Fisciano (SA), Italy

ARTICLE INFO

Keywords:
Algorithmic mechanism design
Monotone algorithms
Scheduling

ABSTRACT

We consider the problem of designing monotone deterministic algorithms for scheduling tasks on related machines in order to minimize the makespan. Several recent papers showed that monotonicity is a fundamental property to design truthful mechanisms for this scheduling problem.

We give both theoretical and experimental results. First of all we consider the case of two machines when speeds of the machines are restricted to be powers of a given constant $c > 0$. We prove that algorithm Largest Processing Time (LPT) is monotone for any $c \geq 2$ while it is not monotone for $c \leq 1.78$; algorithm List Scheduling (LS), instead, is monotone only for $c > 2$.

In the case of $m > 2$ machines we restrict our attention to the class of "greedy-like" monotone algorithms defined in [Vincenzo Auletta, Roberto De Prisco, Paolo Penna, Giuseppe Persiano, Deterministic truthful approximation mechanisms for scheduling related machines, in: Proceedings of 21st Annual Symposium on Theoretical Aspects of Computer Science. STACS '04, in: Lecture Notes in Computer Science, vol. 2996, Springer, 2004, pp. 608–619]. It has been shown that greedy-like monotone algorithms can be used to design a family of $2 + \varepsilon$-approximate truthful mechanisms. In particular, in [Vincenzo Auletta, Roberto De Prisco, Paolo Penna, Giuseppe Persiano, Deterministic truthful approximation mechanisms for scheduling related machines, in: Proceedings of 21st Annual Symposium on Theoretical Aspects of Computer Science. STACS '04, in: Lecture Notes in Computer Science, vol. 2996, Springer, 2004, pp. 608–619], the greedy-like algorithm Uniform is proposed and it is proved that it is monotone when machine speeds are powers of a given integer constant $c > 0$. In this paper we propose a new algorithm, called Uniform_RR, that is still monotone when speeds are powers of a given integer constant $c > 0$ and we prove that its approximation factor is not worse than that of Uniform. We also experimentally compare the performance of Uniform, Uniform_RR, LPT, and several other monotone and greedy-like heuristics.

1. Introduction

In this paper we consider the problem of designing deterministic monotone algorithms for scheduling tasks on related machines in order to minimize the makespan (i.e. the maximum completion time). A classical result of game theory [15,2], states that monotonicity is a necessary condition to design truthful (dominant strategies) mechanisms for problems with one-parameter agents, such as our scheduling problem.

☆ An extended abstract of this paper appeared in [Pasquale Ambrosio, Vincenzo Auletta, Deterministic monotone algorithms for scheduling on related machines, in: Revised Selected Papers of Second International Workshop on Approximation and Online Algorithms, WAOA '05, in: Lecture Notes in Computer Science, vol. 3351, Springer, 2005, pp. 267–280]. Work supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO).

* Corresponding author.
E-mail addresses: ambrosio@dia.unisa.it (P. Ambrosio), auletta@dia.unisa.it (V. Auletta).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.06.050

Mechanisms are a classical concept of the theory of non-cooperative games [18]. In these games there are several independent agents that have to work together in order to optimize a global objective function. However, each player has her own private valuation function, maybe different from the global objective function, and may lie if this can improve her valuation of the game output, even though this can produce a suboptimal solution. Non-cooperative games can be used to model problems that have to be solved in market environments where heterogeneous entities have to cooperate to compute some global function but they compete for "resources" (e.g. the autonomous systems that regulate the routing of traffic in Internet) [19].

The main idea of the *Mechanism Design* theory is to pay the agents to convince them to perform strategies that help the system to optimize the global objective function. A *Mechanism* $M = (A, P)$ is a combination of two elements: an algorithm A computing a solution and a payment function P specifying the amount of "money" the mechanism should pay to each agent. A mechanism is *Truthful with Dominant Strategies* (in what follows, simply truthful) if its payments guarantee that agents are not stimulated to lie, whatever strategies other agents perform.

Recently, mechanism design has been applied to several optimization problems arising in computer science and networking that have been (re-)considered in the context of non-cooperative games [6,20,19].

1.1. State of the art

The celebrated *VCG mechanism* [8,13,21] is the prominent technique to derive truthful mechanisms for optimization problems. However, this technique applies only to *utilitarian* problems, that are problems where the objective function is equal to the sum of the agents valuation functions (e.g., shortest path, minimum spanning tree, etc.). In the seminal papers by Nisan and Ronen [16,17] it is pointed out that VCG mechanisms do not completely fit in a context where computational issues play a crucial role. In fact, VCG mechanisms assume that it is possible to compute an optimal solution of the corresponding optimization problem, while several optimization problems exist that are NP-Hard.

Scheduling is a classical optimization problem that is not utilitarian, since we aim to minimize the *maximum* over all machines of their completion times and it is NP-Hard. Moreover, scheduling models important features of different allocation problems and routing problems in communication networks. Thus, it has been the first problem for which new techniques, not VCG based, have been introduced. Nisan and Ronen [16,20] consider the problem of task scheduling when each machine is owned by a different agent that declares the processing times of the tasks assigned to her machine and the algorithm has to compute the scheduling based on the values declared by the agents. For the case of *unrelated* machines they provide an m-approximate truthful mechanism, where m is the number of machines.

The simpler variant of the task scheduling on *related* machines (in short $Q||C_{\max}$) is considered in [2]. In this case each machine i has a speed s_i and the processing time of a task is given by the ratio between the weight of the task and the speed of the machine. This version of the problem has a very interesting property: the valuation of an agent can be expressed as the product of the load assigned to its machine times a *parameter* (namely, the inverse of speed). We denote these problems as problems involving *One Parameter Agents*. A fundamental result, given in [15,2], shows that a mechanism $M = (A, P)$ for a problem with one-parameter agents is truthful if and only if algorithm A is monotone. Moreover, in [2] it is shown how to construct a payment function P_A such that if A is monotone then (A, P_A) is a truthful mechanism. It is also proved that the algorithm that computes the lexicographically minimal optimal solution is monotone. Intuitively, monotonicity means that increasing the speed of exactly one machine does not make the algorithm decrease the work assigned to that machine (see Section 2 for a formal definition). The results of [15,2] bring us back to "pure algorithmic problems" as all we need is to find a good algorithm for the original problem which also satisfies the additional *monotonicity* requirement. Recently, several results have been obtained for problems with one-parameter agents, even for other problems such as combinatorial auctions [1,5].

Several algorithms are known in the literature for $Q||C_{\max}$, but most of them are not monotone. Greedy algorithms were proposed by Graham in the '60s for the case of identical machines. He proves that the algorithm Largest Processing Time (LPT), which considers the tasks in non-increasing order by weight and assigns each task to the machine with the minimal completion time, has approximation ratio $(4/3 - 1/(3m))$ [12]; algorithm List Scheduling (LS), which considers tasks in the same order as given in input, is $(2 - 1/m)$-approximate [11]. Moreover, a PTAS can be constructed using LPT as a subroutine [12]: first assign the h largest jobs optimally, for any constant h, and then assign the remaining jobs using the algorithm LPT. The same algorithms can be also used for $Q||C_{\max}$. In particular, LPT is ϕ-approximate for two machines, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio, and $\frac{2m}{m+1}$-approximate for m machines [10]. List Scheduling, instead, is $O(\log m)$-approximate [3,7]. However, both these algorithms are not monotone [2].

Non-greedy techniques have been used to provide a PTAS for $Q||C_{\max}$ [14] and constant approximations for the online version of the problem [3,6]. However, all these algorithms are intrinsically not monotone.

The first non-trivial monotone algorithm for $Q||C_{\max}$ is given in [2], where a *randomized* 3-approximate mechanism for $Q||C_{\max}$ is presented that is *truthful in expectation*. In [4] a technique is provided to construct a family of $(2+\varepsilon)$-approximate monotone algorithms PTAS-Gc for $Q||C_{\max}$ starting from a monotone allocation algorithm that is "greedy-like" (i.e. its cost is within an additive factor of $O(t_{\max}/s_1)$ from the cost of LPT, where $t_{\max}$ is the largest task weight and s_1 is the smallest machine speed). The basic idea, derived by the PTAS of Graham, is to combine the optimal scheduling of the largest tasks with the schedule computed by a monotone "greedy-like" algorithm. In [4] the algorithm Uniform is also proposed which

is greedy-like and it is monotone in the particular case where machine speeds are *divisible*: that is, each speed belongs to a set $C = \{c_1, c_2, \ldots, c_p, \ldots\}$ such that for each i, c_{i+1} is a multiple of c_i. Thus, they obtain a family of deterministic truthful $(2 + \varepsilon)$-approximate mechanisms for the case of divisible speeds. This result, combined with payment functions of [2], implies the existence, for any fixed number of machines, of deterministic truthful $(4 + \varepsilon)$-approximate mechanisms for the case of *arbitrary* speeds, for any $\varepsilon > 0$.

1.2. Our results

In [4] two questions are left open. The first one is whether LPT is monotone in the particular case of divisible speeds; the second one is whether the algorithm Uniform is monotone also in the case of arbitrary speeds.

In this paper we try to answer both questions. With respect to the first question we give answers only for the case of 2 machines (see Section 3). We slightly modify the definition of divisible speeds of [4] and say that speeds are *c-divisible* if and only if they are powers of a given positive constant c. We prove that LPT is monotone for c-divisible speeds when $c \geq 2$ while it is not monotone when $c \leq 1.78$. Moreover, we prove some interesting properties of the schedules computed by LPT. We also prove that LS is monotone if $c > 2$. With respect to the second question, we prove that any "Uniform-like" algorithm is not monotone when speeds are not divisible. It is possible to modify the algorithm and obtain monotonicity but this implies a much weaker approximation factor (see Section 4). We also describe a new algorithm Uniform_RR, that is a variation of Uniform, and prove that it is monotone for divisible speeds and for each input it obtains an approximation factor not worse than Uniform (see Section 4.1).

We experimentally evaluate the performances of algorithms Uniform and Uniform_RR, comparing them to LPT and to several other "Uniform-like" heuristics. We run our experiments assuming arbitrary integer speeds chosen uniformly in a given range. Obviously, for the algorithms that are guaranteed to be monotone only in the divisible speeds case we round the speeds of the machines so that we can assume they are 2-divisible. The experiments show that algorithm Uniform_RR outperforms the other considered heuristics both with respect to the worst case and the average case approximation factor and it is very close to LPT; algorithm Uniform, instead, is very close to its theoretical upper bound (see Section 5). A somewhat surprising result is that best results are obtained by rounding speeds and using algorithm Uniform_RR for 2-divisible speeds. This means that the approximation induced by the rounding of the speeds is not significant in practice.

We also study the performance of the mechanism PTAS-Gc given in [4]. This mechanism uses a greedy-like algorithm as a subroutine to compute the scheduling. We measure the performance of the mechanism when algorithms Uniform and Uniform_RR are plugged into the mechanism. Our comparison is based on both the cost of the solution and the amount of money that the mechanism has to pay to the agents. The experiments point out that the cost of the solution computed by the mechanism based on Uniform is less than the cost of the solution computed by the mechanism based on Uniform_RR. With respect to the payments, instead, the two algorithms are equivalent: in fact, in more than half of our tests the two mechanisms pay exactly the same amount of money and there is no significant difference between the total amount of money paid on average. There are few instances where Uniform pays much more than Uniform_RR, while on all instances where Uniform_RR pays more than Uniform the difference is not significant.

2. The problem

In this section we formally define the $Q||C_{\max}$ problem. Consider m machines having speeds $s = \langle s_1, s_2, \ldots, s_m\rangle$, with $s_1 \leq s_2 \leq \cdots \leq s_m$, and n tasks of weights $\sigma = (t_1, t_2, \ldots, t_n)$. In what follows, we simply denote the j-th task with its weight t_j. A schedule is a mapping that associates each task to a machine. The amount of time to complete task j on machine i is t_j/s_i. The *work* of machine i, denoted as w_i, is given by the sum of the weights of the tasks assigned to i. The *load* (or completion time) of machine i is given by w_i/s_i. The cost of a schedule is the maximum load over all machines, that is, its *makespan*.

Given an algorithm A for $Q||C_{\max}$, $A(\sigma, s) = (A_1(\sigma, s), A_2(\sigma, s), \ldots, A_m(\sigma, s))$ denotes the solution computed by A on input the task sequence σ and the speed vector s, where $A_i(\sigma, s)$ is the load assigned to machine i. The cost of this solution is denoted by $\textsc{Cost}(A, \sigma, s)$. In what follows, we omit σ and s every time it is clear from the context. Following the standard notation of game theory, we denote by $s_{-i} = (s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_m)$ the vector of the speeds of all machines except machine i and we write $s = (s_{-i}, s_i)$.

Definition 1 (*Monotone Scheduling Algorithms*). A scheduling algorithm A is monotone iff for any machine i, keeping fixed the speeds of the other machines s_{-i}, the work assigned to machine i is not decreasing with respect to s_i, that is for any $s_i' > s_i''$ it holds that

$$w_i(s_{-i}, s_i') \geq w_i(s_{-i}, s_i'').$$

We denote by $\textsc{opt}(\sigma, s)$ the cost of the optimal solution. Without loss of generality, we assume that the optimal solution is *lexicographically minimal*. As shown in [2], an optimal algorithm that computes the lexicographically minimal solution is monotone.

An algorithm A is a c-approximate algorithm if, for every instance (σ, s), $\text{COST}(A, \sigma, s) \leq c \cdot \text{OPT}(\sigma, s)$. A *polynomial-time approximation scheme* (PTAS) for a minimization problem is a family $\mathcal{A}$ of algorithms such that, for every $\varepsilon > 0$ there exists a $(1+\varepsilon)$-approximation algorithm $A_\varepsilon \in \mathcal{A}$ whose running time is polynomial in the size of the input and in $1/\varepsilon$.

LARGEST PROCESSING TIME (LPT) and LIST SCHEDULING (LS) are two greedy algorithms widely used for $Q||C_{\max}$. LPT first sorts the tasks in nonincreasing order by weight and then process them assigning task t_j to machine i that minimizes $(w_i + t_j)/s_i$, where w_i denotes the work of machine i before task t_j is assigned; if more than one machine minimizing the above ratio exists then the machine with smaller index is chosen. LS uses the same rule as LPT to assign tasks to machines, but it processes the tasks in the same order as they appear in σ. For any fixed number of machines, there exists a PTAS for $Q||C_{\max}$ that assigns the h largest tasks optimally, for h large enough, and the remaining tasks with LPT [12].

We introduce now the class of *greedy-close* algorithms.

Definition 2 (*[4] Greedy-close Scheduling Algorithms*). A scheduling algorithm A is greedy-close if for any speed vector s and for any task sequence σ we have that

$$\text{COST}(A, \sigma, s) \leq \text{COST}(\text{LPT}, \sigma, s) + O\left(\frac{t_{\max}}{s_1}\right),$$

where $t_{\max}$ is the largest task of σ and s_1 is the smallest speed in s.

In [4] the monotone scheduling algorithm PTAS-Gc is provided that uses a greedy-close algorithm Gc as a subroutine. This algorithm splits the task sequence in three parts: the h largest tasks are scheduled optimally; remaining tasks are scheduled by LPT until it is possible to assign tasks without increasing the makespan obtained in the first phase; the remaining tasks are scheduled by Gc, independently from the schedule obtained in the first two phases. The following theorem holds.

Theorem 3 (*[4]*). *For any positive integer n and for any $\varepsilon > 0$, if* Gc *is a greedy-close algorithm, then there exists an integer $h > 0$ such that for all task sequences σ and all speed vectors s of length n,*

$$\text{COST}(\text{PTAS-Gc}, \sigma, s, h) \leq (2+\varepsilon)\text{OPT}(\sigma, s)$$

We say that speeds of the machines are c-divisible, for any constant $c > 0$, if and only if each speed is a power of c. We say that the $Q||C_{\max}$ problem is restricted to c-divisible speeds if speeds are c-divisible and each agent can declare only values that are powers of c.

3. Scheduling on two machines with c-divisible speeds

In this section we consider the case of two machines with c-divisible speeds and give upper and lower bounds on the values of c that guarantee the monotonicity of algorithms LPT and LS.

We start by showing two interesting properties of the schedules computed by LPT. We first observe that the scheduling computed by LPT is a Nash Equilibrium: if a task assigned to a machine (say i) is moved to another machine then it has a completion time that is not smaller than its completion time on machine i. This property has been first proved in [9] and it is very important in the context of dynamic systems since it implies that the system is in a stable state and no entity has an incentive to move from its state. Here, for the sake of clarity, we give a formal proof of it.

Claim 4. *Let $w_1, w_2, \ldots, w_m$ be the works assigned to the machines by* LPT. *For each task t_k, let $i(k)$ be the machine which t_k is assigned to. Then, for each $1 \leq j \leq m$, it holds that $\frac{w_j + t_k}{s_j} \geq \frac{w_{i(k)}}{s_{i(k)}}$.*

Proof. Suppose that our thesis is false and there exists a task t_k that is assigned to machine $i(k)$ and a machine j such that $\frac{w_j + t_k}{s_j} < \frac{w_{i(k)}}{s_{i(k)}}$. Without loss of generality we can assume that t_k is the smaller task assigned to $i(k)$ (otherwise we can replace t_k with another smaller task assigned to $i(k)$). Let $w'_{i(k)}$ and w'_j be the works assigned to machines $i(k)$ and j just before the algorithm assigns task t_k. Observe that $w'_j \leq w_j$ and $w'_{i(k)} = w_{i(k)} - t_k$. Thus, we have that

$$\frac{w'_j + t_k}{s_j} < \frac{w'_{i(k)} + t_k}{s_{i(k)}},$$

contradicting the hypothesis that the algorithm assigned task t_k to machine $i(k)$. □

Lemma 5. *For each speed vector s and for each sequence of tasks σ, the schedule computed by* LPT *on input s and σ is such that for any i, j, if $s_i \leq s_j/2$ then $w_i \leq w_j$, where w_i is the work assigned by the algorithm to machine i.*

Proof. Suppose by contradiction that $w_i > w_j$ and consider a task t assigned to machine i. We have that $\frac{w_j + t}{s_j} < \frac{2w_i}{s_j} \leq \frac{w_i}{s_i}$, contradicting the hypothesis that LPT assigned task t to machine i. □

Let $c(A) > 0$ be the smallest real number such that for each $c \geq c(A)$ the algorithm A is monotone when restricted to c-divisible speeds. We briefly describe now the argument that we use to lower bound $c(A)$. Consider two speed vectors $s = \langle s_1, s_2 \rangle$ and $s' = \langle s'_1, s'_2 \rangle$, where s' differs from s only for the speed of one machine (say machine i) and $s_i \leq s'_i$. For each sequence of tasks $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$, we divide the tasks in σ in the following four sets with respect to the allocations computed by A with respect to s and s':

- $T_i(\sigma)$, for $i = 1, 2$, is the set of tasks of σ that are assigned to machine i both with respect to s and s';
- $L(\sigma)$ is the set of tasks of σ that are assigned to machine 1 with respect to s and to machine 2 with respect to s';
- $R(\sigma)$ is the set of tasks of σ that are assigned to machine 2 with respect to s and to machine 1 with respect to s'.

In the following we omit the argument σ whenever it is clear from the context. It is easy to see that

$$w_1(s) = \frac{T_1 + L}{s_1}, \qquad w_2(s) = \frac{T_2 + R}{s_2}, \qquad w_1(s') = \frac{T_1 + R}{s'_1}, \qquad w_2(s') = \frac{T_2 + L}{s'_2}. \tag{1}$$

Theorem 6. *For any $c \geq 2$, the algorithm* LPT *is monotone when restricted to the case of two machines with c-divisible speeds.*

Proof. Suppose by contradiction that LPT is not monotone for c-divisible speeds. Then, there exist two speed vectors $s = \langle s_1 \leq s_2 \rangle$ and $s' = \langle s'_1 \leq s'_2 \rangle$, where s' has been obtained from s by increasing only one speed, and a sequence of tasks $\sigma = \langle \sigma', t \rangle$ such that the scheduling of the tasks in σ computed by LPT with respect to s and s' is not monotone. Without loss of generality assume that σ is the shortest sequence that LPT schedules in a not monotone way. This means that the schedule of σ' is monotone while the allocation of t destroys the monotonicity. We distinguish three cases.

First of all, consider the case $s'_2 \geq c \cdot s_2$. Since, by hypothesis, the schedule of σ' is monotone while the schedule of σ is not monotone we have that $w_2(\sigma', s) \leq w_2(\sigma', s')$ and $w_2(\sigma, s) > w_2(\sigma, s')$. By Eq. (1) it follows that

$$R(\sigma') \leq L(\sigma') < R(\sigma') + t. \tag{2}$$

Moreover, since t is the smallest task of σ we have that

$$R(\sigma') \geq L(\sigma')/2. \tag{3}$$

Observe now that if LPT on input s assigns task t to machine 2 then $\frac{T_1(\sigma')+L(\sigma')+t}{s_1} > \frac{T_2(\sigma')+R(\sigma')+t}{s_2}$, from which we obtain

$$T_2(\sigma') < \frac{s_2}{s_1}\left(T_1(\sigma') + L(\sigma') + t\right) - R(\sigma') - t. \tag{4}$$

Similarly, if LPT on input s' assigns task t to machine 1 then $\frac{T_1(\sigma')+R(\sigma')+t}{s'_1} \leq \frac{T_2(\sigma')+L(\sigma')+t}{s'_2}$, from which we obtain

$$T_2(\sigma') + L(\sigma') + t \geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t). \tag{5}$$

Substituting Eq. (4) in Eq. (5) we obtain that

$$\begin{aligned}
L(\sigma') &\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - T_2(\sigma') - t \\
&\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + R(\sigma') + t - t \\
&\geq \frac{2s_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + R(\sigma') + t - t \\
&= \frac{s_2}{s_1}T_1(\sigma') + \frac{s_2}{s_1}(2R(\sigma') + 2t - L(\sigma') - t) + (R(\sigma') + t) - t \\
&= \frac{s_2}{s_1}T_1(\sigma') + \frac{s_2}{s_1}(2R(\sigma') - L(\sigma')) + t\left(\frac{s_2}{s_1} - 1\right) + (R(\sigma') + t) \\
&\geq (R(\sigma') + t)
\end{aligned}$$

where the last inequality holds since by hypothesis $\frac{s_2}{s_1} \geq 1$ and, by Eq. (3), $R(\sigma') \geq L(\sigma')/2$. However, this contradicts Eq. (2) and therefore there is no instance σ for which the schedule computed by LPT is not monotone.

The case $s_2 \geq s'_1 \geq c \cdot s_1$ can be reduced to the previous case by observing that if there exists σ such that $w_1(\sigma, s) > w_1(\sigma, s')$ then it holds that $w_2(\sigma, s) < w_2(\sigma, s')$. However, the schedules computed by LPT with respect to s and s' are equal to the schedules computed with respect to speed vectors $(1, s_2/s_1)$ and $(1, s_2/s'_1)$, where $s_2/s'_1 < s_2/s_1$.

The case $s'_1 > s_2$ follows from Lemma 5. □

The previous theorem implies that a deterministic monotone algorithm exists for $Q||C_{\max}$ on two machines.

Corollary 7. *There exists a deterministic monotone algorithm for scheduling tasks on two related machines with approximation factor not greater than 2ϕ.*

Proof. Consider the sequence of tasks σ and the speed vector $s = (s_1, s_2)$. We first compute $s' = (s'_1, s'_2)$, where $s'_i = 2^{\lfloor \log s_i \rfloor}$ and then run LPT on σ and s'. By Theorem 6 the scheduling produced by this algorithm is monotone. The approximation factor of LPT on two machines is less than ϕ [10]. We have to add a factor of 2 for the rounding of the machine speeds. □

Table 1
Schedules of $\sigma = \langle y, x, x/2 + 2\varepsilon, x/2 - \varepsilon\rangle$ produced by LPT on input $s = \langle 1, c\rangle$ and $s' = \langle 1, c^2\rangle$

Speed	1	c
Tasks	x	y $x/2 + 2\varepsilon$ $x/2 - \varepsilon$
Speed	1	c^2
Tasks	$x/2 + 2\varepsilon$ $x/2 - \varepsilon$	y x

Notice that a better approximation factor is obtained by the simple algorithm that schedules all tasks on the fastest machine. However, this Corollary is interesting because it suggests a possible way to give monotone algorithms for the case of m machines: given an algorithm A that is monotone for c-divisible speeds, round down the speeds of the machines to powers of c and then use A to compute a scheduling with respect to the rounded speeds.

Intuitively, Theorem 6 states that LPT is monotone if a machine that wants to reduce its speed has to do it in a significant way (at least half in this case). It is interesting to study which is the value of c(LPT). Next Lemma gives a constructive lower bound on this value.

Lemma 8. *For any $c \leq 1.78$, the restriction of* LPT *to two machines and c-divisible speeds is not monotone.*

Proof. Consider the sequence of tasks $\sigma = \langle y \geq x \geq x/2 + 2\varepsilon \geq x/2 - \varepsilon\rangle$ and the speed vectors $s = \langle 1, c\rangle$ and $s' = \langle 1, c^2\rangle$. Without loss of generality we assume $c \geq 1$, otherwise we can simply scale the speeds down. Assume that LPT schedules tasks of σ as shown in Table 1 when receives in input s and s', respectively. Clearly, this schedule is not monotone since machine 2 receives a total load of $y + x + \varepsilon$ with speed c and a total load of $x + y$ with speed c^2. In the following we give conditions on y, x and ε in order to have LPT producing the schedules given in Table 1.

We observe that LPT on input s produces the schedule given in Table 1 when

$$\frac{y + x}{c} > x \tag{6}$$

and

$$\frac{y + x + \varepsilon}{c} < \frac{3}{2}x - \varepsilon. \tag{7}$$

Similarly, LPT, on input s', produces the schedule given in Table 1 when

$$\frac{y + x}{c^2} < x \tag{8}$$

and

$$\frac{y + x + x/2 - \varepsilon}{c^2} > x + \varepsilon. \tag{9}$$

By trivial computations it can be seen that for any $c \leq 1.78$ it is possible to choose y, x and ε so that previous inequalities hold. In particular, for $c = 1.78$ we can take $y = 113.5$, $x = 68$, $\varepsilon = 0.005$. □

The argument of the proof of Lemma 8 cannot be extended since for any $c \geq \frac{3+\sqrt{17}}{4}$ it is not possible to choose y, x and ε in order to satisfy Eqs. (6)–(9).

Theorem 9. *For any $c > 2$, the algorithm* LS *is monotone when restricted to the case of two machines with c-divisible speeds.*

Proof. The proof is similar to Theorem 6. Suppose by contradiction that LS is not monotone for c-divisible speeds and let $s = \langle s_1, s_2\rangle$ and $s' = \langle s_1, s_2'\rangle$ be two speed vectors that differ only on one speed and let $\sigma = \langle \sigma', t\rangle$ be a minimal length sequence of tasks such that the schedules of σ produced by LS on input s and s' are not monotone. Thus, the schedules of σ' are monotone while the assignment of task t destroys the monotonicity.

Consider first the case where $s_2' \geq c \cdot s_2$. In this case we can state that LS assigns t to machine 2 if it receives in input s, and it assigns t to machine 1 if it receives in input s'. Observe that Eq. (2) holds also in this case and we have that

$$\frac{T_1(\sigma') + R(\sigma') + t}{s_1} < \frac{T_2(\sigma') + L(\sigma') + t}{s_2'} \leq \frac{T_2(\sigma') + L(\sigma') + t}{cs_2}.$$

Algorithm UNIFORM
Input: a task sequence σ, speed vector $s = \langle s_1, s_2, \ldots, s_m \rangle$, with $s_1 \leq s_2 \leq \ldots \leq s_m$
(1) run algorithm LPT to allocate the task sequence σ on $S = \sum_{i=1}^{m} s_i$ identical virtual machines;
(2) order the virtual machines by nondecreasing load $l_1, \ldots, l_S$.
(3) Let $g := GCD(s_1, s_2, \ldots, s_m)$ and partition the virtual machines into g blocks $B_1, \ldots, B_g$, each consisting of S/g consecutive virtual machines. For $1 \leq i \leq g$ and $1 \leq k \leq S/g$, denote by B_{ik} the k-th virtual machine of the i-th block. Thus the virtual machine B_{ik} has load $l_{(i-1)\frac{S}{g}+k}$.
(4) For $1 \leq j \leq m$ let $k_j = \frac{1}{g}\sum_{l=1}^{j-1} s_l$; then machine j receives the load of virtual machines $B_{ik_j+1}, \ldots, B_{ik_j+\frac{s_j}{g}}$ for $1 \leq i \leq g$.

Fig. 1. Algorithm UNIFORM.

Thus, we have that

$$\begin{aligned} L(\sigma') &\geq \frac{cs_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - T_2(\sigma') - t \\ &\geq \frac{cs_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + R(\sigma') + t - t \\ &= R(\sigma') + t + (c-1)\frac{s_2}{s_1}T_1(\sigma') + \frac{s_2}{s_1}(R(\sigma') - L(\sigma') + t) + (c-1)\frac{s_2}{s_1}R(\sigma') + \left((c-2)\frac{s_2}{s_1} - 1\right)t, \end{aligned}$$

where the second inequality follows from Eq. (4). Observe now that, since $c > 2$ and $s_2 \geq s_1$, we have that $(c-1)\frac{s_2}{s_1}T_1(\sigma') \geq 0$, $(c-1)\frac{s_2}{s_1}R(\sigma') \geq 0$ and $\left((c-2)\frac{s_2}{s_1} - 1\right)t \geq 0$. Thus, we can conclude that

$$\left(1 + \frac{s_2}{s_1}\right)L(\sigma') \geq \left(1 + \frac{s_2}{s_1}\right)(R(\sigma') + t)$$

that contradicts Eq. (2).

The proofs of the cases where machine 1 increases its speed are the same as the proofs of the same cases in Theorem 6. □

4. UNIFORM-like algorithms

In this section we prove that algorithm UNIFORM, proposed in [4], is not monotone with respect to not divisible speeds. In all this section we assume that machine speeds are positive integers.

We start by giving a short description of the algorithm (the code of the algorithm is given in Fig. 1), while we refer the interested reader to [4] for a complete description. Algorithm UNIFORM works in two phases: first it uses LPT as a subroutine to compute a schedule of the tasks to $S = \sum_{i=1}^{m} s_i$ identical "virtual machines"; then, it assigns to each real machine i the work assigned to s_i virtual machines in such a way that each virtual machine is assigned to only one real machine. To guarantee the monotonicity of UNIFORM the mapping of the virtual machines to the real machines is such that $w_1/s_1 \leq w_2/s_2 \leq \cdots \leq w_m/s_m$. In particular, UNIFORM partitions the virtual machines into g blocks of the same size, where g is equal to $GCD(s_1, s_2, \ldots, s_m)$, in such a way that each virtual machine of block i has a work greater than any other machine in a block $j < i$. Finally, for each block i, for $i = 1, 2, \ldots, m$, it assigns s_i/g consecutive virtual machines to the real machine i, starting from the virtual machine with less work.

In [4] it is proved that UNIFORM is greedy-close and it is monotone in the particular case of divisible speeds. The question whether UNIFORM is monotone also when speeds are not divisible is left open. Theorem 10 gives a negative answer to this question.

Theorem 10. *Algorithm* UNIFORM *is not monotone with respect to not divisible speeds.*

Proof. We prove the Theorem by constructing an example where the allocation computed by algorithm UNIFORM is not monotone. Consider the task sequence $\sigma = \langle 2, 2, 2, 1, 1, 1 \rangle$ and the speed vectors $s = \langle 3, 8 \rangle$ and $s' = \langle 2, 8 \rangle$. Observe that on input (σ, s) algorithm UNIFORM partitions the virtual machines in only one block and assigns all the load to machine 2 (see Fig. 2(a)): thus, we have a work equal to 0 for machine 1 and a work equal to 9 for machine 2. On input (σ, s'), instead, algorithm UNIFORM splits the virtual machines in 2 blocks producing the schedule given in Fig. 2(b), where machine 1 obtains a work of 2. Thus, the algorithm is not monotone because machine 1 increases its load while reducing its speed. □

The proof of Theorem 10 shows that any algorithm based on the partition of virtual machines in blocks will not be monotone if the number of blocks depends on the speeds of the machines. We can modify UNIFORM, so that it sets $g = 1$ and it

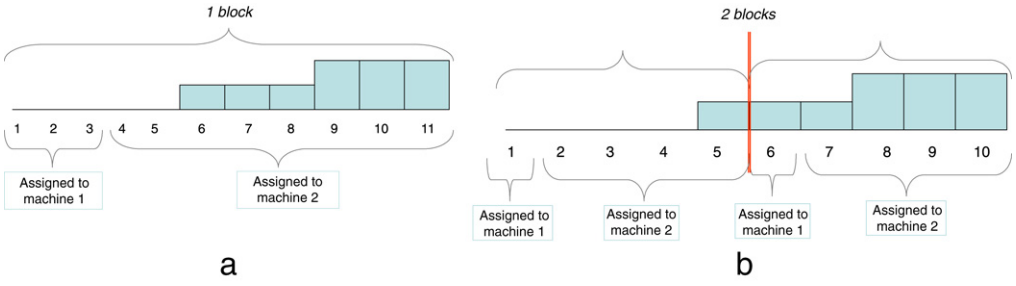


Fig. 2. An example of non monotone scheduling computed by UNIFORM. In (a) it is given the scheduling computed for $s = (3, 8)$; in (b) it is given the scheduling computed for $S' = (2, 8)$.

Algorithm UNIFORM_RR
Input: a task sequence σ, speed vector $s = \langle s_1, s_2, \ldots, s_m \rangle$, with $s_1 \leq s_2 \leq \ldots \leq s_m$.

(1) Run algorithm LPT to allocate tasks of σ on $S = \sum_{i=1}^{m} s_i$ identical virtual machines.
(2) Order the virtual machines by nondecreasing load $l_1, \ldots, l_S$.
(3) Set $g := GCD(s_1, s_2, \ldots, s_m)$ and partition the virtual machines into g blocks $B_1, \ldots, B_g$, each consisting of S/g consecutive virtual machines. For $1 \leq i \leq g$ and $1 \leq k \leq S/g$, denote by B_{ik} the k-th virtual machine of the i-th block. Thus the virtual machine B_{ik} has load $l_{(i-1)\frac{S}{g}+k}$.
(4) For each block j
  (a) set $k_i = s_i/g$, for $1 \leq i \leq m$, and $x = 1$.
  (b) for $1 \leq k \leq S/g$
    - while $k_x = 0$ set $x = (x+1) \bmod m$. Then, allocate the total load of the virtual machine B_{jk} to the real machine x and set $k_x = k_x - 1$.

Fig. 3. Algorithm UNIFORM_RR.

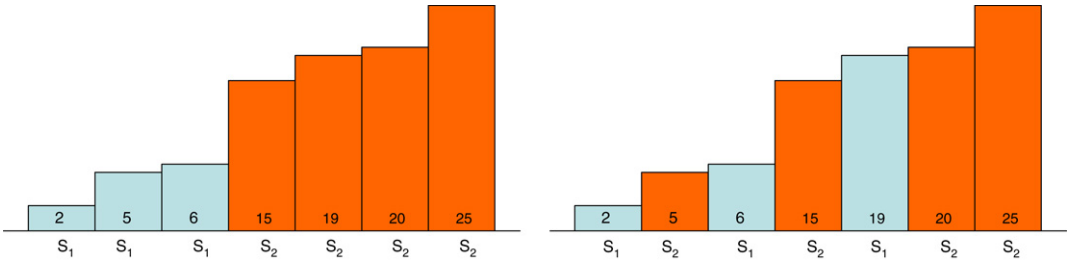


Fig. 4. Assignments computed by UNIFORM (left) and UNIFORM_RR (right) on an instance with two machines with speeds $s = \langle 3, 4 \rangle$. UNIFORM produces an assignment with makespan equal to 19.75; UNIFORM_RR produces an assignment with a makespan equal to 16.25.

considers all the virtual machines as in the same block. This new algorithm is monotone but it obtains a weak approximation since the assignment of the virtual machines to real machines is completely unbalanced (see Fig. 4(a)). We describe now a variation of UNIFORM that computes $g = GCD(s_1, s_2, \ldots, s_m)$ blocks but it makes a more clever assignment of the virtual machines of each block to the real machines.

4.1. *Algorithm* UNIFORM_RR

Several variations of UNIFORM can be designed, differing for the strategy used to assign virtual machines of a block to the real machines. In this paper we restrict our attention to algorithms that allocate virtual machines to real machines while preserving the property that for each block B and for each pair of machines i, j, with $s_i < s_j$, the total load assigned to machine i in block B must be not greater than the total load assigned to machine j. We require this property since it is a sufficient condition to guarantee the monotonicity of the allocation.

We now describe algorithm UNIFORM_RR, that uses a round-robin strategy to assign virtual machines of a block to real machines, starting from the virtual machine with lowest work that is assigned to the real machine with lowest speed. Algorithm UNIFORM_RR is described in Fig. 3. In Fig. 4 it is given evidence of the more balanced scheduling computed by UNIFORM_RR by showing the assignments computed by UNIFORM and UNIFORM_RR on an instance of 7 virtual machines.

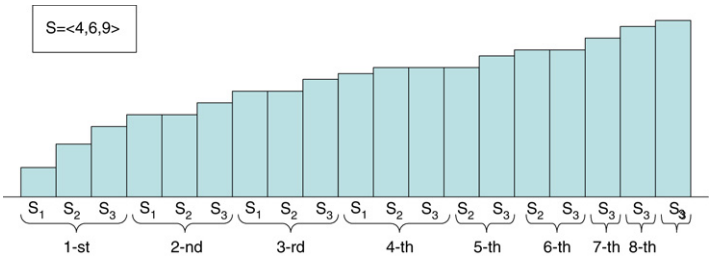


Fig. 5. An example of partitioning the virtual machines of a block in rounds.

In the rest of the section we prove that algorithm UNIFORM_RR is monotone when restricted to c-divisible speeds, for any positive integer c, and it is $(2+\varepsilon)$-approximate. The proof of the monotonicity of UNIFORM_RR is a technical extension of the proof of monotonicity of UNIFORM, given in [4].

Consider the allocations computed by algorithm LPT on N and $N' = N + d$ identical machines. We denote by

- L_i the load of the i-th least loaded machine when LPT allocates tasks on N identical machines;
- l_i the load of the i-th least loaded machine when LPT allocates tasks on $N + d$ identical machines.

The following technical result is fundamental for our proof.

Lemma 11 ([4]). *Consider the allocations computed by the algorithm* LPT *on N and $N' = N + d$ identical machines, respectively. Then, for $1 \le i \le N$ and $0 \le k \le d$ it holds that*

$$l_{i+k} \le L_i$$
$$l_i + l_{i+1} \le L_i$$

Consider a block B. We partition the virtual machines of B in rounds, where a round is a set of consecutive virtual machines such that each virtual machine is assigned to a distinct real machine (see Fig. 5 for an example). It can be easily seen that the machines of B are partitioned in s_m/g rounds. Moreover, if round i contains a virtual machine assigned to the real machine j then it contains also virtual machines assigned to all the real machines k, with $k > j$. To prove the monotonicity of the algorithm we have to show that increasing the speed of one machine the work assigned to such a machine does not decrease. In the following we denote by $w_i(s)$ the work assigned to the machine i when the algorithm receives in input the vector speed s.

Theorem 12. *For any integer $c > 0$, algorithm* UNIFORM_RR *is monotone when restricted to c-divisible speeds.*

Proof. Let $s = (s_1, s_2, \ldots, s_m)$ be a vector of machine speeds that is c-divisible and denote by s' the vector obtained from s by substituting s_j with $s'_j \ge c \cdot s_j$. Thus, the algorithm UNIFORM_RR uses $S = \sum_j s_j$ virtual machines when it receives in input s while it uses $S' = S + (s'_j - s_j)$ virtual machines when it receives in input s'. We will show that the algorithm, on input s, allocates to machine j some work that is not greater than the work that the algorithm allocates to the same machine on input s'. For the sake of clarity we prove the theorem only for the case where $c = 2$, $s'_j = 2s_j$ and thus $S' = S + s_j$. The general case is a trivial generalization of this case.

We distinguish two cases depending on the value of j. In fact, when $j \ne 1$ the number of blocks constructed by the algorithm on input s is the same as the number of blocks constructed on input s'; when $j = 1$, instead, the number of blocks constructed on input s' is twice the number of blocks constructed on input s.

- [$j \ne 1$] Denote by $B_i(t)$ the i-th block of virtual machines constructed by the algorithm on input the speed vector t. Observe that $|B_i(s)| = S/s_1$, $|B_i(s')| = (S + s_j)/s_1$ and if the index of the first virtual machine of $B_i(s)$ is a then the first index of $B_i(s')$ is $a' = a + d$, where $d \le s_j$. Suppose first that $s'_j \le s_{j+1}$ and thus machine j does not change its position in the speed vector (note that the speed vector is sorted). In this case the first s_j rounds of the algorithm allocate virtual machines to real machines in the same order. Thus if, on input s, machine $h \ne j$ in round r receives the work of the virtual machine x then the same machine, on input s' receives in round r the work of the virtual machine $x' \le x + d$. By Lemma 11 for each machine $h \le j$ it holds that $l_{x'} \le L_x$. Summing over all the blocks we obtain that for each machine $h \ne j$ we have that $w_h(s) \ge w_h(s')$, and this implies that $w_j(s) \le w_j(s')$.

  Suppose now that $s'_j > s_{j+1}$ and machine j moves to position j' in s' (and thus $s_j = s_{j+1} = \cdots = s_{j'}$). We have to prove that $w_j(s) \le w_{j'}(s')$. Observe that in this case all machines with indices from j to j' change their position in the round robin allocation computed by the algorithm. However, since, for each $1 \le h < m$, algorithm UNIFORM_RR assigns a load to the h-th machine not greater than the load assigned to the $(h+1)$-th machine, we know that $w_j(s) \le w_{j'}(s)$. Thus, it is sufficient to prove that $w_{j'}(s) \le w_{j'}(s')$. Without loss of generality assume that $j' = j + 1$. The general case can be proved by repeatedly applying the same argument to pairs of machines $(j+1, j+2), (j+2, j+3), \ldots, (j'-1, j')$. Consider now a new vector $\bar{s}$, obtained from s by exchanging machines in position j and $j+1$. Notice that the algorithm computes the same allocation on input s and $\bar{s}$ and thus $w_{j+1}(s) = w_{j+1}(\bar{s})$. Consider now the vector $\bar{s}'$ obtained by $\bar{s}$ substituting

$\bar{s}_{j+1}$ with $2\bar{s}_{j+1}$. Observe that $s' = \bar{s}'$ and by the previous case $w_{j+1}(\bar{s}) \leq w_{j+1}(\bar{s}') = w_{j+1}(s')$. Putting things together, we have that

$$\begin{aligned} w_j(s) &\leq w_{j+1}(s) \\ &= w_{j+1}(\bar{s}) \\ &\leq w_{j+1}(\bar{s}') \\ &= w_{j+1}(s'). \end{aligned}$$

- $[j = 1]$ If $s'_1 \leq s_2$ then machine 1 does not change its position in the round robin allocation and it receives the works of the first virtual machine of each block. However, the number of blocks doubles. Observe that the first machine of $B_i(s)$, for $1 \leq i \leq s_1$, has index $a(i) = (i-1)\frac{S}{s_1} + 1$, while the first index of $B_i(s')$, for $a \leq i \leq 2s_1$, is $a'(i) = (i-1)\frac{S+s_1}{2s_1} + 1$. Since $a'(2i-1) - a(i) = i - 1 < s_1$, then by Lemma 11 we have that $L_{a(i)} \leq l_{a(i)} + l_{a(i)+1} \leq l_{a'(2i-1)} + l_{a'(2i)}$. Finally, we can conclude that

$$w_1(s) = \sum_{i=1}^{s_1} L_{a(i)} \leq \sum_{i=1}^{s_1} l_{a'(2i-1)} + l_{a'(2i)} = \sum_{i=1}^{2s_1} l_{a'(i)} = w_1(s').$$

The case where machine 1 changes its position in s' can be solved as the similar case for $j \neq 1$. In fact, in this case $s_1 = s_2$ and the number of blocks constructed by the algorithm is the same on input s and s'. □

In order to give an upper bound on the approximation factor of Uniform_RR, we show that for any speed vector s and any task sequence σ it holds that

$$\textsc{Cost}(\textsc{Uniform\_RR}, \sigma, s) \leq \textsc{Cost}(\textsc{Uniform}, \sigma, s).$$

In [4] it is proved that the makespan of Uniform is obtained by machine m. We prove that a similar property holds also for Uniform_RR.

Lemma 13. *For any speed vector $s = \langle s_1, s_2, \ldots, s_m \rangle$ and any task sequence σ it holds that*

$$\textsc{Cost}(\textsc{Uniform\_RR}, \sigma, s) = w_m/s_m,$$

where m is the fastest machine and w_m is the work assigned to this machine.

Proof. We prove the lemma by showing that for each block B the load assigned to machine m in block B is greater than or equal to the load assigned in the same block to any other machine.

Let $x_1 \leq x_2 \leq \cdots \leq x_{s_m/g}$ and $y_1 \leq y_2 \leq \cdots \leq y_{s_j/g}$ be the loads of the virtual machines assigned to machine m and j, for any $j < m$, respectively. We observe that $x_h \geq y_h$ for $1 \leq h \leq s_j/g$ and $x_h \geq y_{s_j/g}$ for $s_j/g < h \leq s_m/g$. Then,

$$\frac{1}{s_m} \sum_{h=1}^{s_m/g} x_h \geq \frac{1}{s_j} \sum_{h=1}^{s_j/g} y_h. \quad \square$$

Lemma 14. *For any speed vector $s = \langle s_1, s_2, \ldots, s_m \rangle$ and any task sequence σ it holds that*

$$\textsc{Cost}(\textsc{Uniform\_RR}, \sigma, s) \leq \textsc{Cost}(\textsc{Uniform}, \sigma, s).$$

Proof. By Lemma 13 it is sufficient to prove that Uniform_RR assigns to machine m a total load not greater than the load assigned by Uniform to the same machine.

Observe that the two algorithms compute the same assignment of tasks to the virtual machines and the same partition of virtual machines in blocks. Thus, it is sufficient to prove that the load assigned by algorithm Uniform_RR to machine m for each block B is not greater than the load assigned by Uniform to the same machine. Let $x_1 \leq x_2 \leq \cdots \leq x_{s_m/g}$ and $y_1 \leq y_2 \leq \cdots \leq y_{s_m/g}$ be the works of the virtual machines of block B assigned to machine m algorithms by Uniform and Uniform_RR, respectively. It can be easily seen that $x_h \geq y_h$ for $1 \leq h \leq s_m/g$ and the lemma follows. □

Theorem 15. *For any speed vector $s = \langle s_1, s_2, \ldots, s_m \rangle$ and any task sequence σ it holds that*

$$\textsc{Cost}(\textsc{Uniform\_RR}, \sigma, s) \leq (2 + \varepsilon)\textsc{opt}(\sigma, s).$$

Proof. The theorem follows by Lemma 14 and Theorem 16 of [4]. □

Table 2
Algorithms considered in our testing and their theoretical approximation factors

Upper bounds to approximation factors of monotone scheduling algorithms	
LARGEST PROCESSING TIME	$(\frac{2n}{n+1})$
LARGEST PROCESSING TIME RESTRICTED	$(\frac{2n}{n+1})$
UNIFORM G = 1	$(4+\varepsilon)$
UNIFORM G = 1 RESTRICTED	$(4+\varepsilon)$
UNIFORM RESTRICTED	$(4+\varepsilon)$
UNIFORM_RR G = 1	$(4+\varepsilon)$
UNIFORM_RR G = 1 RESTRICTED	$(4+\varepsilon)$
UNIFORM_RR RESTRICTED	$(4+\varepsilon)$

Table 3
Summary of the number of instances performed in each run

Jobs	**Machines**	**Instances**					
		$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=5$	$\beta=6$
10	4	3690	7380	11 070	14 760	22 140	29 520
25	5	3690	11 070	14 760	19 680	29 520	39 360
100	10	5538	17 694	33 232	54 310	66 466	88 620

Table 4
Average Competitive ratio computed on all the instances

Average approximation factors of monotone scheduling algorithms	
LPT	1.377031
LPT RESTRICTED	1.777902
UNIFORM MCD = 1	4.692374
UNIFORM MCD = 1 RESTRICTED	4.062987
UNIFORM RESTRICTED	3.387385
UNIFORM ROUND-ROBIN MCD = 1	2.935213
UNIFORM ROUND-ROBIN MCD = 1 RESTRICTED	2.600026
UNIFORM ROUND-ROBIN RESTRICTED	1.988051

5. Experimental results

In this section we describe the results of an experimental analysis on the performance of several monotone scheduling algorithms. We have performed three different experiments:

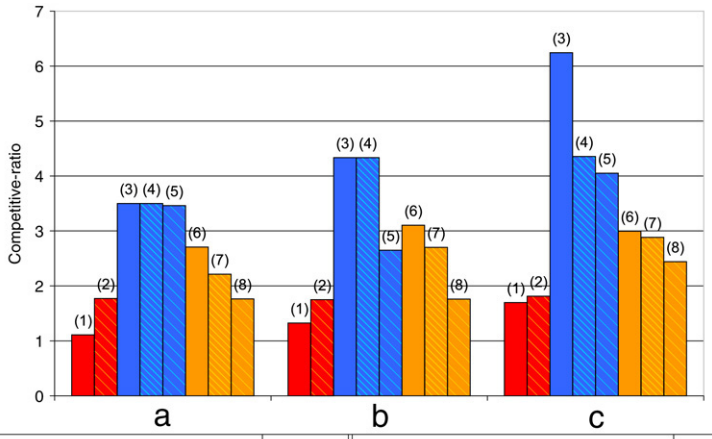
- in the first experiment we have measured the approximation factors of several monotone heuristics, comparing them to the approximation of LPT;
- in the second experiment we have measured the approximation factors of the algorithms obtained by plugging different monotone greedy-like algorithms in the scheme described in [4];
- in the third experiment we have measured the total quantity of money paid by the mechanisms induced from the algorithms UNIFORM and UNIFORM_RR.

In our test we have considered three basic algorithms: LPT, UNIFORM and UNIFORM_RR. We have also considered several variations of these three algorithms, obtained by changing the number of blocks, if used, or rounding the speeds of the machines. In particular, the restricted versions of the three algorithms take in input the machine speeds, round up the speeds to a power of 2 and then compute the scheduling. Table 2 summarizes the algorithms we have considered in our testing.

We have performed experiments with respect to arbitrary speeds. We executed our measures on three different runs: in each run we fix the number of machines and the number of tasks and select speeds uniformly in a range $[1, 2^{\beta}]$ with $1 \leq \beta \leq 6$ and task weights uniformly in a range $[1, 2^{\alpha}]$ with $0 \leq \alpha \leq 8$. Table 3 gives a summary of the instances performed in each run. For each instance we have measured the makespan and the approximation factor. Then we have computed the average makespan, the average approximation factor and the worst case approximation factor in each run. We have also performed similar experiments for speeds and weights selected according to a normal distribution and for 2-divisible speeds. The results obtained are similar and we omit them.

Fig. 6 shows the worst case approximation factors obtained in the three runs. Table 4, instead, shows the average approximation factors, where the average is computed on the set of all the instances.

Experiments give evidence that UNIFORM_RR obtains the best results among the monotone algorithms considered in our testing. Its approximation factor is very close to LPT, both in the worst case and in the average case.



Algorithm	Label	Algorithm	Label
LPT	(1)	UNIFORM RESTR.	(5)
LPT RESTR.	(2)	UNIFORM RR MCD=1	(6)
UNIFORM MCD=1	(3)	UNIFORM RR MCD=1 RESTR.	(7)
UNIFORM MCD=1 RESTR.	(4)	UNIFORM RR RESTR.	(8)

Fig. 6. Worst case approximation factors obtained in our testing. (a) 10 tasks and 4 machines (b) 15 tasks and 5 machines (c) 100 tasks and 15 machines.

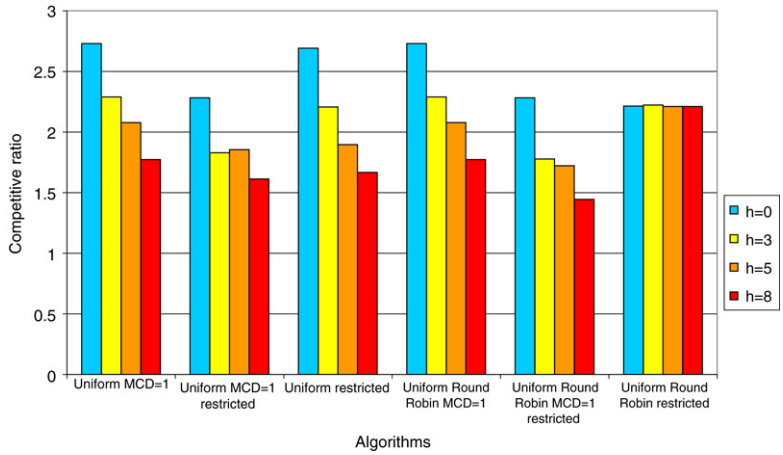


Fig. 7. Average approximation factor of PTAS-Gc, computed on all the instances of our testing, for different selections of Gc and h.

UNIFORM, instead, obtains an approximation factor that is very close to the theoretical bound. An unexpected result is that the restricted version of UNIFORM_RR obtains better results than the unrestricted version of the same algorithm and its performance improves when the number of tasks increase. Our interpretation is that the restricted version of the problem uses more blocks and thus obtains a more balanced assignment of virtual machines to real machines, counterbalancing the approximation induced by the rounding of the machine speeds.

We have also experimentally measured the impact of the proposed greedy-close monotone algorithms on the performance of the PTAS-Gc algorithm defined in [4]. Notice that PTAS-Gc takes three inputs: the task sequence, the speed vector and a parameter h, that is the number of tasks that are allocated optimally in the first phase of the algorithm. Our testing is organized in two runs: the first run is performed on instances with 15 tasks and 4 machines; the second run is performed on instances with 25 tasks and 5 machines. For each instance of σ and s we run the algorithm with $h \in \{0, 3, 5, 8\}$ (see Fig. 7 for a summary of the approximation factors and the computation times).

Our experiments point out two interesting aspects: the first one is that, since the largest tasks are assigned optimally, the difference in the performance between UNIFORM_RR and the other heuristics is significantly smaller; the second one is that all the considered heuristics, except for UNIFORM_RR, improve their performance when h increases. In particular, for h sufficiently large, algorithm UNIFORM outperforms UNIFORM_RR. Moreover, we notice that the computation time grows dramatically when h increases (see Fig. 8), while the approximation factor is improved only by a small factor.

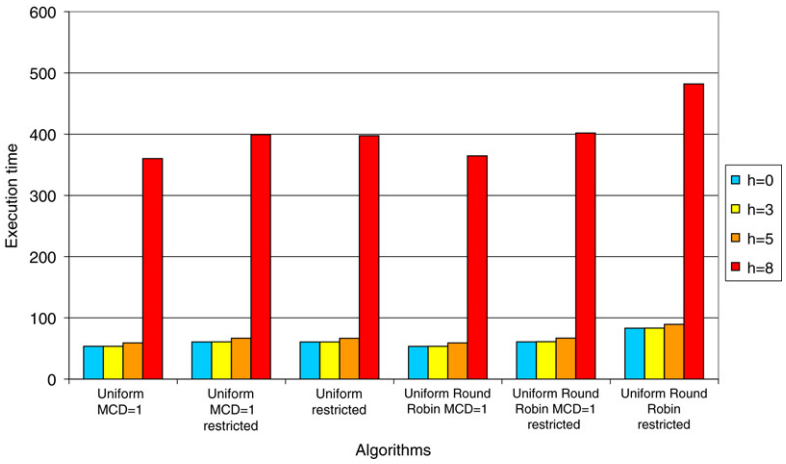


Fig. 8. Average computation times of PTAS-Gc, computed on all the instances of our testing, for different selections of Gc and h.

Table 5
Summary of the number of instances computed in each run and the average total payment

	Instances	E[P(UNIFORM)]	E[P(UNIFORM_RR)]
100 tasks and 10 machines	100.000	7.979	7.935
25 tasks and 5 machines	100.000	1.857	1.851
15 tasks and 4 machines	88.560	1.149	1.147

Table 6
Maximum difference between payments computed by mechanisms induced on algorithms UNIFORM and UNIFORM_RR

	100 tasks 10 machines	**25 tasks 5 machines**	**15 tasks 4 machines**
$\max(P(\text{UNIFORM}) - P(\text{UNIFORM\_RR}))$	8.660, 8	3.490, 7	1.423, 8
$\max(P(\text{UNIFORM\_RR}) - P(\text{UNIFORM}))$	743, 3	541, 5	375

Finally, we have considered the truthful mechanisms obtained from algorithms UNIFORM and UNIFORM_RR using the payment functions described in [2]. We have organized the experiment in three different runs: for each run we have fixed the number of machines and tasks and we have computed several instances by selecting the weights of the tasks and the speeds of the machines as in the previous experiments. Table 5 summarizes the number of instances computed in each run and the average payment computed in each run. It can be easily seen that the the average payments are approximately equal. In more than half of the instances the two mechanisms computed exactly the same payments and in the remaining instances the difference between the payments is always small (see Table 6).

6. Conclusion

The contribution of this paper is twofold. From a theoretical point of view, we have proven that greedy algorithms like LPT and LS are monotone if we restrict our study to the case of 2 machines with c-divisible speeds, for large enough c. This implies that greedy algorithms can be made monotone with a loss in the approximation factor. We think that this technique can be extended to the case of $m > 2$ machines.

From an experimental point of view we have analyzed several heuristics, based on the algorithm UNIFORM, and proved that making cleverer assignments of virtual machines to real machines can significantly improve the performance of the algorithm. In particular, we have shown that in several cases rounding machine speeds and using a "UNIFORM-like" scheduling algorithm (i.e. algorithm UNIFORM_RR) can yield better results than solving the problem with respect to the original speeds. However, if we could prove that LPT is monotone for c-divisible speeds, for a small c, we could obtain even better approximations.

References

[1] A. Archer, C. Papadimitriou, K. Talwar, E. Tardos, An approximate truthful mechanism for combinatorial auctions with single parameter agent, in: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '03, 2003, pp. 205–214.

[2] A. Archer, E. Tardos, Truthful mechanisms for one-parameter agents, in: Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science, FOCS '01, IEEE Press, 2001, pp. 482–491.
[3] J. Aspnes, Y. Azar, A. Fiat, S.A. Plotkin, O. Waarts, On-line routing of virtual circuits with applications to load balancing and machine scheduling, J. ACM 44 (3) (1997) 486–504.
[4] V. Auletta, R. De Prisco, P. Penna, G. Persiano, Deterministic truthful approximation mechanisms for scheduling related machines, in: Proceedings of 21st Annual Symposium on Theoretical Aspects of Computer Science, STACS '04, in: Lecture Notes in Computer Science, vol. 2996, Springer, 2004, pp. 608–619.
[5] V. Auletta, R. De Prisco, P. Penna, G. Persiano, The power of verification for one-parameter agents, in: Proceedings of 31st International Colloquium on Automata, Languages and Programming, ICALP '04, in: Lecture Notes in Computer Science, vol. 3142, Springer, 2004, pp. 171–182.
[6] P. Berman, M. Charikar, M. Karpinski, On-line load balancing for related machines, Electronic Colloquium on Computational Complexity (ECCC) 7 (1) (2000).
[7] Y. Cho, S. Sahni, Bounds for list schedules on uniform processors, SIAM J. Comput. 9 (1) (1980) 91–103.
[8] E.H. Clarke, Multipart pricing of public goods, Public Choice (1971) 17–33.
[9] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, P. Spirakis, The structure and complexity of Nash equilibria for a selfish routing game, in: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP '02, in: Lecture Notes in Computer Science, vol. 2380, Springer, 2002, pp. 123–134.
[10] T.F. Gonzalez, O.H. Ibarra, S. Sahni, Bounds for lpt schedules on uniform processors, SIAM J. Comput. 6 (1) (1977) 155–166.
[11] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Tech. J. 45 (1966) 1563–1581.
[12] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (2) (1969) 416–429.
[13] T. Groves, Incentive in teams, Econometrica 41 (1973) 617–631.
[14] D.S. Hochbaum, D.B. Shmoys, A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach, SIAM J. Comput. 17 (3) (1988) 539–551.
[15] R. Myerson, Optimal auction design, Mathematics 6 (1981) 58–73.
[16] N. Nisan, A. Ronen, Algorithmic mechanism design (extended abstract), in: Proceedings of 31st Annual ACM Symposium on Theory of Computing, STOC '99, 1999, pp. 129–140.
[17] N. Nisan, A. Ronen, Computationally feasible VCG mechanisms, in: Proceedings of 2nd ACM Conference on Electronic Commerce, EC '00, 2000, pp. 242–252.
[18] M.J. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, 1994.
[19] C.H. Papadimitriou, Game theory and mathematical economics: A theoretical computer scientist's introduction, in: Proceedings of 42nd IEEE Annual Symposium on Foundations of Computer Science, FOCS '01, IEEE Press, 2001, pp. 4–8.
[20] A. Ronen, Solving optimization problems among selfish agents, Ph.D. Thesis, Hebrew University of Jerusalem, 2000.
[21] W. Vickrey, Counterspeculation, auctions and competitive sealed tenders, J. Finance (1961) 8–37.