Deterministic Monotone Algorithms for Scheduling on Related Machines*

Pasquale Ambrosio and Vincenzo Auletta

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Universitá di Salerno
84081, Baronissi (SA), Italy
{ambrosio, auletta}@dia.unisa.it

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. For 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 is monotone for any $c \geq 2$ while it is not monotone for $c \leq 1.78$; algorithm List Scheduling, instead, is monotone only for $c > 2$.

For the case of m machines we restrict our attention to the class of "greedy-like" monotone algorithms defined in [AP04]. We propose the greedy–like algorithm Uniform_RR and we prove that it is monotone when speeds are powers of a given integer constant $c > 0$ and it obtains an approximation ratio that is not worse than algorithm Uniform, proposed in [AP04]. We also experimentally compare performances 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, recently rediscovered by [AT01], states that monotonicity is a necessary condition to design truthful (dominant strategies) mechanisms for this scheduling problem. Mechanisms are a classical concept of the theory of non-cooperative games [OR94]. 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 its own private valuation function, maybe different from the global objective function, and may lie if this can improve its valuation of the game

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

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 heterogenous entities have to cooperate in computing some global function but they compete for "resources" (e.g. the autonomous systems that regulate the routing of traffic in Internet) [Pa01].

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 the sequel 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 [NR99, Ro00, Pa01].

State of the Art. The celebrated *VCG mechanism* [Cl71, Gr73, V61] 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 [NR99, NR00] it is pointed out that VCG mechanisms do not completely fit in a context where computational issues play a crucial role since they assume that it is possible to compute an optimal solution of the corresponding optimization problem (maybe a NP-hard problem). Scheduling, is a classical optimization problem that is not utilitarian, since we aim at minimizing 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 not VCG based techniques have been introduced.

Nisan and Ronen [NR99, Ro00] give an m-approximation truthful mechanism for the problem of scheduling tasks on m *unrelated* machines, when each machine is owned by a different agent that declares the processing times of the tasks assigned to his/her machine and the algorithm has to compute the scheduling based on the values declared by the agents. In [AT01] it is considered the simpler variant of the task scheduling on *related* machines (in short $Q||C_{max}$), where 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. They show that a mechanism $M = (A, P)$ for the $Q||C_{max}$ problem is truthful if and only if algorithm A 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 result of [AT01] reduces the problem of designing a truthful mechanism for $Q||C_{max}$ to the algorithmic problem of designing a good algorithm which also satisfies the additional *monotonicity* requirement.

Several algorithms are known in 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 algorithm LPT, which considers the tasks in non-increasing order by weight, is $(4/3 - 1/(3m))$-approximated [Gr69], while algorithm List Scheduling, which considers tasks in the same order as given in input, is $(2 - 1/m)$–approximated [Gr66]. Moreover, a PTAS can be constructed using LPT as a subroutine [Gr69]. The same algorithms can be used for $Q||C_{max}$. In particular, LPT is ϕ-approximated for two machines, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio, and $\frac{2m}{m+1}$–approximated for m machines [GI80]. LS, instead, is $O(\log m)$–approximated [AA+93, CS80]. However, both these algorithms are not monotone [AT01]. Not greedy techniques have been used to provide a PTAS for $Q||C_{max}$ [HS88] and constant approximations for the online version of the problem [AA+93, BC97]. However, all these algorithms are intrinsically not monotone.

The first non-trivial monotone algorithm for $Q||C_{max}$ is given in [AT01], where a randomized 3-approximated mechanism for $Q||C_{max}$ is presented that is truthful in expectation. In [AP04] a technique is provided to construct a family of $(2 + \varepsilon)$-approximated monotone algorithms 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; however, in order to guarantee monotonicity, the scheduling of the large tasks and of the small tasks are computed independently. In [AP04] it is proposed the algorithm Uniform, which is greedy–like and it is monotone in the particular case where machine speeds are divisible (see Section 2 for a formal definition). Thus, they obtain a family of deterministic truthful $(2+\varepsilon)$-approximated mechanisms for the case of divisible speeds. This result, combined with payment functions of [AT01], implies the existence, for any fixed number of machines and any $\varepsilon > 0$, of deterministic truthful $(4 + \varepsilon)$-approximated mechanisms for the case of *arbitrary* speeds.

Our Results. In [AP04] 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 to both the questions. With respect to the first question we give answers only for the case of 2 machines (see Section 3). We 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 $c \leq 1.78$. 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 to obtain monotonicity but this implies a much weaker approximation factor. We also describe a new algorithm Uniform_RR, based on Uniform, and prove that it is more efficient than Uniform and it is monotone for divisible speeds (see Section 4).

We experimentally evaluate the performances of algorithms Uniform and Uniform_RR, comparing it to LPT and to several other "Uniform–like" heuristics. 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.

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 $s_1 \leq s_2 \leq \ldots \leq s_m$, and n tasks of weights $\sigma = (t_1, t_2, \ldots, t_n)$. In the sequel we simply denote the i-th task with its weight t_i. 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 $Q||C_{max}$, $A(\sigma, s) = (A_1(\sigma, s), A_2(\sigma, s), \cdots, A_m(\sigma, s))$ 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 $\text{Cost}(A, \sigma, s)$. In the sequel we omit σ and s every time it is clear from the context. Following the standard notation of game theory, we denote by $s_{-i} = (s_1, s_2, \cdots, s_{i-1}, s_{i+1}, \cdots, s_m)$ 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, 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 $w_i(s_{-i}, s'_i) \geq w_i(s_{-i}, s''_i)$.*

An *optimal algorithm* computes a solution of minimal cost $\text{opt}(\sigma, s)$. Throughout the paper we assume that the optimal algorithm always produces the *lexicographically minimal* optimal assignment. As shown in [AT01], this algorithm is monotone.

An algorithm A is a c-approximation 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.

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 small 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 [Gr69].

We say that a scheduling algorithm A is greedy-close if for any speed vector s and for any task sequence σ we have that $\textsc{Cost}(A, \sigma, s) \leq \textsc{Cost}(\text{LPT}, \sigma, s) + O\left(\frac{t_{max}}{s_1}\right)$, where t_{max} is the largest task of σ and s_1 is the smallest speed in s. In [AP04] 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 two parts: the h largest tasks are scheduled optimally; the remaining tasks are scheduled by Gc, independently from the schedule obtained in the first two phases. They prove that for any sequence of tasks and for any $\varepsilon > 0$ there exists an integer $h > 0$ such that PTAS-Gc gives a solution that is within a factor of $(2 + \varepsilon)$ from the optimum.

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 proving two interesting properties of the schedules computed by LPT. The first lemma proves that the scheduling computed by LPT is such that 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, known as a Nash Equilibrium, 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.

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

Lemma 2. *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.*

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 on speed of machine i and $s_i \leq s'_i$. For each sequence of tasks $\sigma = \langle t_1, t_2, \cdots, t_n \rangle$, 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

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

Theorem 3. *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

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

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

$$T_2(\sigma') < \frac{s_2}{s_1}\left(T_1(\sigma') + L(\sigma') + t\right) - R(\sigma') - t. \tag{3}$$

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

$$T_2(\sigma') + L(\sigma') + t \geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) \tag{4}$$

Substituting Eq. 3 in Eq. 4 and making some algebraic manipulations we obtain that

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

where the last inequality holds since by hypothesis $\frac{s_2}{s_1} \geq 1$ and, since t is the smallest task, $R(\sigma') \geq L(\sigma')/2$. However, this contradicts Eq. 2 and therefore there is no instance σ for which the schedule computed by LPT is not monotone. The other two cases can be solved by reduction to the previous case.

A similar argument can be used to prove the following theorem.

Theorem 4. *For any $c > 2$, the algorithm* LS *is monotone when restricted to the case of two machines with c–divisible speeds.*

Intuitively, Theorem 3 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 $c(\text{LPT})$. Next Lemma gives a constructive lower bound on this value.

Lemma 5. *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 $s' = \langle 1, c^2 \rangle$. Assume that x, y and ε are chosen in such a way that LPT, on input s, assign all tasks except for x to machine 2, while, on input s' it assigns the first two tasks to machine 2 and the other tasks to machine 1. 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. We observe that LPT produces the previous schedules when

$$\frac{y+x}{c} > x, \qquad \frac{y+x+\varepsilon}{c} < \frac{3}{2}x - \varepsilon \tag{5}$$

and

$$\frac{y+x}{c^2} < x, \qquad \frac{y+x+x/2-\varepsilon}{c^2} > x + \varepsilon. \tag{6}$$

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 5 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 Eq. 5–6.

4 Algorithms Uniform–Like

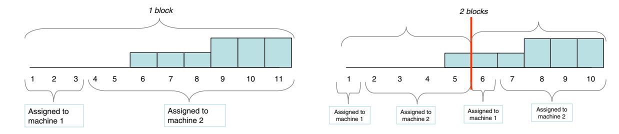
In this section we prove that algorithm Uniform, proposed in [AP04], is not monotone with respect to not divisible speeds. In the sequel we assume that machine speeds are positive integers.

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 $i = 1, 2, \cdots, m$, it assigns s_i/g consecutive virtual machines to the real machine i, starting from the virtual machine with less work.

In [AP04] it is proved that Uniform is greedy-close and it is monotone in the particular case of divisible speeds. It is left open the question whether Uniform is monotone also when speeds are not divisible. Next Theorem gives a negative answer to this question.

Theorem 6. *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. 1(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. 1(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.



(a) (b)

Fig. 1. 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 $S' = (2, 8)$

The proof of Theorem 6 shows that any algorithm based on the partition of virtual machines in blocks will be not 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 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. 2(a)). We describe now a variation of Uniform that computes $g = GCD(s_1, s_2, \cdots, s_m)$ blocks but it makes a more clever assignment of the virtual machines of each block to the real machines.

Algorithm Uniform_RR, described in Alg. 1, 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. Fig. 2 shows the assignments computed by Uniform and Uniform_RR on an instance of 7 virtual machines and gives evidence of the more balanced scheduling computed by Uniform_RR. We state that algorithm Uniform_RR is monotone with respect to divisible speeds and it is $(2+\varepsilon)$-approximated. The proof of the monotonicity of Uniform_RR is a technical extension of the proof of monotonicity of Uniform given in [AP04] and we omit it from this extended abstract (a complete version of the paper can be found in [AA04]). In order to prove the bound on the approximation factor we show that for any speed vector s and any task sequence σ it holds that $\text{Cost}(\text{Uniform\_RR}, \sigma, s) \leq \text{Cost}(\text{Uniform}, \sigma, s)$.

Algorithm 1 Uniform_RR

```
Input: a task sequence $\sigma$, 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 $\sigma$ 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 $k_x = k_x - 1$.
```

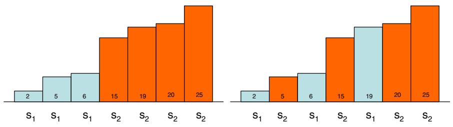


Fig. 2. Assignments computed by Uniform (left) and Uniform_RR (right) on an instance with two machines with speeds $s = \langle 3, 4 \rangle$. Uniform produces an assignment with makespan equal to 19.75; Uniform_RR produces an assignment with a makespan equal to 16.25

In [AP04] 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 7. *For any speed vector $s = \langle s_1, s_2, \cdots, s_m \rangle$ and any task sequence σ it holds that the makespan of the solution computed by* Uniform_RR *is equal to the completion time of the fastest 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,

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

Lemma 8. *For any speed vector $s = \langle s_1, s_2, \cdots, s_m \rangle$ and any task sequence σ it holds that the cost of the solution computed by* Uniform_RR *is not greater than the cost of the solution computed by* Uniform.

Proof. By Lemma 7 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 9. *For any speed vector $s = \langle s_1, s_2, \cdots, s_m \rangle$ 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 8 and Theorem 16 of [AP04].

5 Experimental Results

In this section we describe the results of an experimental analysis on the performances of several monotone scheduling algorithms. We have performed two 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, instead, we have measured the approximation factors of the algorithms obtained by plugging different monotone greedy–like

Table 1. Algorithms considered in our testing and their theoretical approximation factors

Upper bounds to approximation factors of monotone scheduling algorithms	
LPT (1)	$(\frac{2n}{n+1})$
LPT RESTRICTED(2)	$(\frac{2n}{n+1})$
UNIFORM G=1 (3)	$(4+\varepsilon)$
UNIFORM G=1 RESTRICTED (4)	$(4+\varepsilon)$
UNIFORM RESTRICTED (5)	$(4+\varepsilon)$
UNIFORM_RR G=1 (6)	$(4-\varepsilon)$
UNIFORM_RR G=1 RESTRICTED (7)	$(4-\varepsilon)$
UNIFORM_RR RESTRICTED (8)	$(4-\varepsilon)$

Table 2. 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	11070	14760	22140	29520
25	5	3690	11070	14760	19680	29520	39360
100	10	5538	17694	33232	54310	66466	88620

algorithms in the scheme described in [AP04]; 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 testing 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 two algorithms take in input the machine speeds, round up the speeds to a power of 2 and then compute the scheduling. Table 1 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 2 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 substantially equivalent and we omit them. Figure 3 shows the worst case approximation factors obtained in the three runs. Table 3, 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.

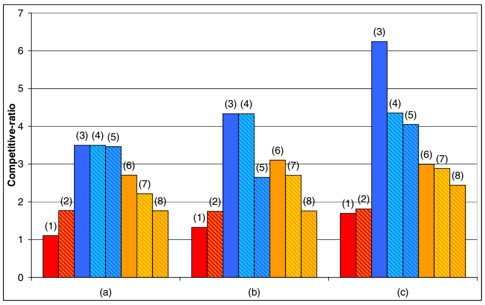


Fig. 3. Worst case approximation factors obtained in our testing. Algorithms are labeled as shown in Table 5. (a) 10 tasks and 4 machines (b) 15 tasks and 5 machines (c) 100 tasks and 15 machines

Table 3. Average Competitive ratio computed on all the instances

Average approximation factors of monotone scheduling algorithms	
Largest Processing Time	1.377031
Largest Processing Time 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

Its approximation factor is very close to LPT, both in the worst case and in the average case. 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 performances improve when the number of tasks increase. Our interpretation is that the restriction 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 performances of the PTAS-Gc algorithm defined in [AP04]. 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 $h \in \{0, 3, 5, 8\}$ Our experiments point out two interesting aspects: the first one is that, since the largest tasks are

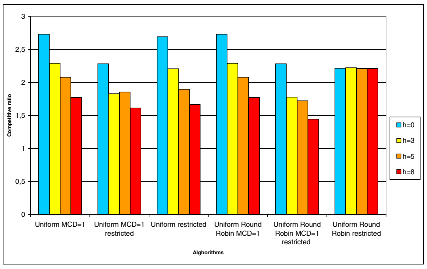


Fig. 4. Average approximation factor of PTAS-Gc, computed on all the instances of our testing, for different selections of Gc and h

assigned optimally, the difference in the performances between Uniform_RR and the other heuristics is significantly less; the second one is that all the considered heuristics, except for Uniform_RR, improve their performances when h increases. In particular, for h sufficiently large, algorithm Uniform outperforms Uniform_RR. However, this relatively small improvement in the approximatio factor is counterbalanced by a dramatic growing in the computatio time.

6 Conclusion

The contribution of this paper is twofold. From a theoretical point of view, we have proved that greedy algorithms like LPT and LS are monotone if we restrict to the case of 2 machines with c–divisible speeds, for c large enough. We think that this technique can be generalized to prove that greedy algorithms can be made monotone with a loss in the approximation factor even for 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 a more clever assignments of virtual machines to real machines can significantly improve the performances of the algorithm. In particular, we have shown that in several cases rounding machine speeds 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

[AA04] P. Ambrosio and V. Auletta. Deterministic monotone algorithms for scheduling on related machines. Technical Report Università di Salerno, 2004.

[AT01] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of FOCS*, pages 482 - 491, 2001.

[AA+93] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin and O. Warts. On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling. In *Journal of ACM*, 44: 486 - 504.

[AD+04] V. Auletta, R. De Prisco, P. Penna and G. Persiano. The Power of Verification for One-Parameter Agents. In *Proc. of ICALP '04*, 2004.

[AP04] V. Auletta, R. De Prisco, P. Penna and G. Persiano. Deterministic Truthful Mechanisms for Scheduling on Selfish Machines. In *Proc. of STACS '04*, LNCS 2996, pages 608 - 619, 2004.

[BC97] P. Berman, M. Charikar and M. Karpinski. On-Line load Balancing for Related Machines. In *Proc. of WADS '97*, 1997.

[CS80] Y. Cho and S. Sahni. Bounds for list scheduling on uniform processors. *SIAM Journal of Computing*, vol.9 n.1, 1980.

[Cl71] E. H. Clarke. Multipart Pricing of Public Goods. *Public Choice*, pages 17 - 33, 1971.

[GI80] T. Gonzalez, O. Ibarra and S. Sahni. Bounds for LPT schedules on uniform processors. *SIAM Journal of Computing*, vol.6, 1977.

[Gr73] T. Groves. Incentive in Teams. *Econometrica*, 41 : 617 - 631, 1973.

[Gr66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. Journal*, 45 : 1563 - 1581, 1966.

[Gr69] R. L. Graham. Bound on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416-429, 1969.

[HS88] D. S. Hochbaum and D. B. Shmoys. A Polynomial Approximation Scheme for Scheduling on uniform processors: Using the Dual Approximation Approach. In *SIAM Journal of Computing*, vol. 17(3) pages 539 - 551, 1988.

[NR00] N. Nisan and A. Ronen. Computationally Feasible VCG Mechanisms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC)*, pages 242 - 252, 2000.

[NR99] N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st STOC*, pages 129 - 140, 1999.

[OR94] M. J. Osborne and A. Rubinstein. *A course in Game Theory*. MIT Press, 1994.

[Pa01] C. H. Papadimitriou. Algorithms, Games and the Internet. In *Proc. of the 33rd STOC*, 2001.

[Ro00] A. Ronen. *Solving Optimization Problems Among Selfish Agents*. PhD thesis, Hebrew University in Jerusalem, 2000.

[V61] W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, 8 - 37, 1961.