# Multi-Users Scheduling in Parallel Systems

Érik Saule, Denis Trystram

# Multi-Users Scheduling in Parallel Systems

Erik Saule and Denis Trystram
LIG, Grenoble University
51, avenue J. Kuntzmann
38330 Montbonnot St. Martin, France
{erik.saule,denis.trystram}@imag.fr

## Abstract

*We are interested in this paper to study scheduling problems in systems where many users compete to perform their respective jobs on shared parallel resources. Each user has specific needs or wishes for computing his/her jobs expressed as a function to optimize (among maximum completion time, sum of completion times and sum of weighted completion times). Such problems have been mainly studied through Game Theory. In this work, we focus on solving the problem by optimizing simultaneously each user's objective function independently using classical combinatorial optimization techniques. Some results have already been proposed for two users on a single computing resource. However, no generic combinatorial method is known for many objectives.*

*The analysis proposed in this paper concerns an arbitrarily fixed number of users and is not restricted to a single resource. We first derive inapproximability bounds; then we analyze several greedy heuristics whose approximation ratios are close to these bounds. However, they remain high since they are linear in the number of users. We provide a deeper analysis which shows that a slightly modified version of the algorithm is a constant approximation of a Pareto-optimal solution.*

**Keywords**: Multi-objective optimization, Multi-users scheduling, approximation algorithms

## 1 Introduction

### 1.1 Context and Motivation

High performance computing systems such as clusters and computational grids are generally not dedicated to a single user but they are shared among many users that each execute many jobs. These users compete to perform their respective jobs on the common resources. In such systems, the resource allocation problem is handled by a scheduler whose mechanism is rather simple: the jobs are submitted by users into queues and they are scheduled by batches. To our knowledge, the most elaborated cluster scheduler is MAUI [10]. It uses queue parameters to take into account various parameters related to the jobs' characteristics. MAUI can be used in such a way that each user has a dedicated queue. Tuning its queue's parameters enables the user to express his/her needs. However, theoretical properties of such a scheduler have not been studied from the users' point of view. Notice that most existing schedulers are centralized, which means that a single program is responsible for scheduling the jobs.

In this work, we are interested in studying the problem of scheduling jobs belonging to different users optimizing simultaneously the needs (or wishes) of each user. The users have no decision to make, they only submit their jobs to the scheduler and choose an objective function among some existing classical objectives. Our goal is to propose and analyze policies for sharing the resources taking into account the objectives of the users.

### 1.2 Related approaches

Several attempts have been proposed for optimizing simultaneously several objectives. We mainly distinguish between two approaches for solving the multi-users scheduling problem, namely, Game Theory and classical Multi-objective Combinatorial Optimization.

Game Theory is a very useful framework for studying systems where many agents (users in our context) interact [16]. In computing systems, we distinguish two classes of problems. On the first hand, a centralized arbiter constructs a solution depending on users' decisions. In such cases, the problem is often considered under an economic perspective where prices are associated to time slots, and can be solved using auctions or division protocols. On the second hand, the users construct the solution in a distributed way. For instance, Pascual *et al.* address the load bal-

ancing problem involving multiple organizations and show that cooperation between organizations can always been achieved for improving the global behavior [17]. Legrand and Touati study a simple master-slave system where many non-cooperating users submit their own divisible jobs [13].

Multi-objective Combinatorial Optimization received recently a great interest [18]. Most investigations have been conducted as bi-objective optimization problems and sometimes involved three simultaneous objectives. Such results are based on sophisticated algorithms, however, they are difficult to extend to a larger number of objectives. And as far as we know, no generic combinatorial method is known for many objectives. Two papers that can serve as a basis for optimizing many objectives in the multi-users scheduling problem [4, 1]. Both of them address the restricted problem of two users that submit jobs to a single machine. We analyze in detail the content of both papers in Section 2.

Multi-users problems can also be considered by optimizing non-classical objective functions. User objective functions can be aggregated within a fairness function. An infinite number of fairness functions can be defined, all of them have their advantages and none can be easily discarded [15]. However, the aggregation technique only makes sense when all the users' objectives are the same.

### 1.3 Contributions

We address in this work the problem of scheduling independent sequential jobs belonging to $k$ different users on $m$ processors where each user selects an objective function among a list of classical well-studied objectives, namely, makespan, sum of (weighted or not) completion times. Our main contribution is to provide algorithms that schedule the jobs of $k$ users with different and mixed objectives with the best achievable theoretical guarantee. More precisely:

First, we show that if all users are interested in the makespan, the problem can not be approximated with a ratio better than $(1, 2, \ldots, k)$. We use here the natural extension of the approximation ratios notation where the $i$-th number corresponds to the approximation ratio on the $i$-th user objective. If all users are interested in the sum of completion times, the problem can not be approximated with a ratio better than $(k, \ldots, k)$.

Then, we propose an algorithm (called MULTICMAX) for the case where all the users are interested in the makespan and prove that it is a $(\rho, 2\rho, \ldots, k\rho)$ approximation where $\rho$ is the approximation ratio of an algorithm for the single-user case (notice that a PTAS is known for this problem). Then, we propose an algorithm (called MULTISUM) for the case where all users are interested in the sum of completion times and prove that it is a $(k, \ldots, k)$-approximation. Both algorithms are optimal in the sense that no algorithm achieves better approximation ratios.

These ratios may appear disappointing since they depend on the number of users. However, the ratios are high because they take as a reference a non valid solution, namely the one that achieves the best objective value for each user as if he/she was the only one using the system. We show for the makespan case that an adaptation of MULTICMAX is a constant approximation of a Pareto-optimal solution.

Finally, we tackle the mixed case where $k'$ users are interested in the sum of completion times and $k''$ users are interested in the makespan. We propose for this problem an algorithm (called MULTIMIXED) that is a $(k, \ldots, k)$-approximation for the sum of completion times and a $(\frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$-approximation for the makespan.

### 1.4 Organization

The rest of the paper is organized as follows. Section 2 is devoted to the formal description of the problem and notations. It contains a brief description of two related papers and a discussion on how to extend their results to a more general framework. Section 3 states the complexity and approximability of the general problem and of all sub-problems. Section 4 deals with the sub-problem where all the users are interested in the makespan. The MULTICMAX algorithm is given and analyzed. Then, a modification of this algorithm is provided and its distance to the Pareto set is computed. Section 5 describes and analyzes MULTISUM, an approximation algorithm for the sub-problem where all users are interested in the sum of completions times or the sum of weighted sum completion times. Finally, in Section 6, previous results are used to derive MULTIMIXED, an approximation algorithm for the mixed case where users may be interested in both the makespan and the sum of completion times.

## 2 Model and Preliminaries

### 2.1 Problem Definition and Notations

Let us consider $k$ independent users (indexed by $u$) competing for the resources on a parallel platform composed of $m$ identical resources (processors). Each user $u$ owns $n^{(u)}$ jobs (denoted by $J_i^{(u)}, 1 \leq i \leq n^{(u)}$). The jobs are submitted to a centralized scheduler.

Let $n = \sum_{u=1}^{k} n^{(u)}$ be the total number of jobs. Job $J_i^{(u)}$ belongs to user $u$ and has a processing time of $p_i^{(u)}$. A schedule is defined by the starting times of all the jobs: $\sigma(J_i^{(u)})$. The completion time of $J_i^{(u)}$ is $C_i^{(u)} = \sigma(J_i^{(u)}) + p_i^{(u)}$. The usual constraints of scheduling must hold: each job is scheduled without interruption and no more than $m$ jobs can run at a time.

The makespan of jobs belonging to $u$ is $C_{max}^{(u)} = \max_{J_i^{(u)}} C_i^{(u)}$. The average completion times of jobs belonging to $u$ is $\sum C_i^{(u)}$ (the weighted variant is $\sum w_i^{(u)} C_i^{(u)}$). In general $f^*$ denotes the optimal value of objective $f$. The value of $f$ achieved in a given schedule $S$ is denoted by $f(S)$.

The multi-objective problem where each objective is the one chosen by each user is called $MUSP$ (for Multi-Users Scheduling Problem). However, we will consider sub-problems where the users' choice are fixed in the problem. For example, we will denote by $MUSP(k' : \sum C_i \; ; \; k'' : C_{max})$ the sub-problem where $k'$ users are interested in the sum of completion times and $k'' = k - k'$ users are interested in the makespan.

## 2.2 Analysis of basic related works

We recall in this section the results of two recent papers dealing with the same problem with restricted hypotheses. They can serve as a basis for solving the general multi-users scheduling (allocation) problem. Both papers (Baker and Smith [4] and Agnetis *et al.* [1]) consider two users competing for the execution of their jobs on a common single resource.

In the first paper [4], the problem is to optimize a linear combination of two of the three following objectives : makespan, sum of completion times (weighted or not) and maximum lateness. They proved that the combination of any pair of objectives can be solved in polynomial time except for the combination of maximum lateness and sum of weighted completion times. Moreover, on a single machine, if a user wants to optimize the makespan of his-her own set of jobs, then they can be merged into a single job.

These results can be extended to several users (more than 2) as long as the objective function remains a linear combination of users' objective functions: two users optimizing the makespan of their jobs can be seen as a single user that wants to optimize the sum of weighted completion times of two jobs, each one being the merge of the original jobs. Moreover, a user interested in the makespan is seen has a single task of a sum of completion time interested user. However, optimizing a linear combination of users' objectives does not seem to be a good idea. Let us consider the instance involving two users where the first one is interested in minimizing the makespan of several jobs which are merged into one big job with processing time $p^{(1)}$, while the second one wants to minimize the sum of completion times of $Kp^{(1)}$ unitary jobs (where $K$ is an integer). By optimizing the sum of both objectives, the system will schedule all the jobs of the second user before scheduling the job of the first one. This behavior is unfair because the jobs of the first user can be arbitrarily delayed by jobs of the second user when $K$ goes to infinity. Moreover, the first user has incentive to lie: he/she would have obtained a better performance if he/she declared to be interested by the sum of completion times instead of the makespan... In the terminology of Game Theory, this method does not guarantee the *truthfulness* [3].

A second argument against using similar approaches based on linear (weighted) combinations of objectives is that in the objective space, the Pareto set[1] of such a problem is not always convex or all Pareto optimal solutions may be aligned. Thus, it is possible to achieve only a few interesting solutions by such a technique.

The second paper of Agnetis et al. [1] also concerns the optimization of the objectives of two users (among the makespan, the sum of completion times and the sum of weighted completion times) on a single common resource. They distinguish between two questions:

First, they are interested in Constrained Optimization Problems where one objective is fixed as a constraint and the second objective is optimized. The authors show that when both users are interested in the makespan, the problem can be solved in polynomial time. If they are both interested in the sum of completion times, the problem becomes binary NP-hard and they provide a pseudo-polynomial dynamic program to solve it. With mixed objectives, if one user is interested in the weighted sum of completion times, the problem is binary NP-hard. Other cases are polynomial.

Secondly, they are interested in Pareto Optimization Problems in which the output of the algorithm is all Pareto optimal schedules. The main issue is the cardinality of the Pareto set. When one user is interested in the makespan, if the other one is interested in the makespan or in the sum of completion times, the cardinality of the Pareto set is polynomial. If the second one is interested in the weighted sum of completion times, the problem is open. If both users are interested in the sum of completion times, the cardinality of the Pareto set can be exponential.

All those results are interesting from a theoretical point of view. For instance, the constrained version of a multi-objective problem helps to construct an approximation of the Pareto set for this problem [2]. They help to determine the frontier between easy and hard problems. However, they can not be directly used to deal with the multi-machine/multi-users case. They do not provide a generic way to optimize different objective functions.

[1] The set of Pareto optimal solutions, *i.e.* solutions where no objective value can be improved without degrading another objective value.

### 2.3 Discussion on Objective Functions

In this section, we give another point of view about this multi-user optimization problem by showing how to reduce the optimization of a fairness function to the problem we are addressing.

In the literature, when scheduling individually important jobs of different sizes, the problem of fairness arises. It can be tackled using the *stretch* objective function which is defined as the time the task spent in the system normalized by its processing time. Stretch optimization was studied on non-preemptible independent tasks [5]. In a sense, the stretch of a job is a degradation factor it obtains by not being the only job in the system. The stretch is also used in bag-of-task applications [12] which is comparable to our setting. In bag-of-tasks, the application is composed of small jobs (possibly infinitely small jobs), whereas in MUSP the user submits a set of jobs.

Thus, the idea is to optimize a stretch-like function derived from the user objective function. If $f^{(u)}$ is the objective function of user $u$, the degradation of $u$ in schedule $S$ is $d^{(u)}(S) = \frac{f^{(u)}(S)}{f^{(u)*}}$ where $f^{(u)*}$ is the minimum value of $f^{(u)}$.

Stretch of applications needs an aggregation function to ensure fairness. Three of them are usually considered which corespond to the standard norms $L_\infty, L_1, L_2$: minimizing the maximum of stretches, the sum of stretches and the product of stretches. Thus, optimizing the sum (or maximum, ...) of degradations seems the logical extension, in the following let $L$ be the considered norm.

Such an objective function would be NP-hard to optimize and approximation algorithms would be needed. However, no non-trivial lower bound on such functions is known which is a pre-request to design approximation algorithms.

The point is that $L$ is decreasing and monotone function of the degradations according to the Pareto order and the degradation of user $u$ is an increasing function of $f^{(u)}$. Thus, in the multi-objective problem of minimizing the tuple $(f^{(1)}, \ldots, f^{(k)})$, if $S$ Pareto-dominates $S'$, then $L(S) \leq L(S')$. In other words, solving the multi-objective optimization problem will solve the problem of optimizing $L$ since the minimum of $L$ is reached by a Pareto optimal solution.

## 3 Complexity and Inapproximability

We first study the complexity of all sub-problems (all possible combinations of objectives). Then, we focus on bounding the performance of the best achievable approximation algorithms.

Optimizing the makespan of a single set of jobs (*i.e.* one objective) on an arbitrary number of processors is strongly NP-hard [6]. Thus, as soon as one user wants to optimize the makespan, the problem becomes strongly NP-hard. Let us now concentrate on the case where all the users are interested in the sum of completion times which is a sub-case of the weighted sum of completion times problem. Recall that the decision problem restricted to two users interested in the sum of completion times on a single machine is binary NP-complete [1]. Thus, on an arbitrary number of processors, the problem is harder. In conclusion, all optimization sub-problems we are interested in are NP-hard.

As all sub-problems are NP-hard, it is likely that no optimal polynomial-time algorithms can be designed. Thus, we are interested in polynomial approximation algorithms. In the reminder of this section, we are looking for lower bounds on best achievable approximation vector-ratios of a single solution. Formally, a scheduling algorithm $algo$ has a performance vector-ratio $(\rho_1, \ldots, \rho_k)$ if for all valid schedules $S$ and for all users $u$, $f^{(u)}(algo) \leq \rho_u f^{(u)}(S)$ where $f^{(u)}$ is the objective of user $u$. First, let us remark that the optimal value of one particular objective function is obtained by scheduling optimally the jobs of one user like if he/she was the only user of the machine. Thus, all known bounds on the best approximation ratios for the single user case are still valid.

We now examine two sub-problems successively, each one is for an arbitrary fixed number of machines that serve for establishing inapproximability bounds. These results are summarized in Propositions 3.1 and 3.2. They are expressed as "The problem can not be approximated with a performance vector ratio better than $(\rho_1, \ldots, \rho_k)$" where 'better than' stands for the component wise relation. This means that there is no $(\rho_1 - \epsilon, \rho_2 \ldots, \rho_k)$ or even $(\rho_1, \ldots, \rho_{i-1}, \rho_i - \epsilon, \rho_{i+1} \ldots, \rho_k)$-approximation algorithm ($\epsilon > 0$). However, this formulation does not prevent a $(\rho_1, \ldots, \rho_{i-1}, \rho_i - \epsilon, \rho_{i+1}, \ldots, \rho_{j-1}, \rho_j + \epsilon, \rho_{j+1}, \ldots, \rho_k)$-approximation algorithm from existing.

**Proposition 3.1** *$MUSP(k : C_{max})$ can not be approximated with a performance vector-ratio better than $(1, 2, \ldots, k)$.*

**Proof.** In this case, all the users are interested in the makespan. Let us consider the following instance. Each user has only $m$ jobs. All the jobs have the same unit length. Obviously, for all the users independently, the best makespan that can be achieved is 1. In any efficient schedule, one user will have a makespan of 1, another one will have a makespan of 2, and so on. Thus, it is impossible to obtain an algorithm that guarantees a vector-ratio better than $(1, 2, \ldots, k)$. Moreover, remark that any permutation of this vector-ratio can be obtained.

However, the vector-ratio $(1, 2, \ldots, k)$ is not achievable in all instances. For example, consider the instance where the $k - 1$ last users have $m$ jobs of unit length and the first

one has $m$ jobs of processing time $k$. In such an instance, if the first user gets a performance ratio of 1, all other users will get a performance ratio worse than $k$.

In conclusion, the best achievable vector-ratio is valid for a permutation depending on the instance. Therefore, if the scheduler aims at obtaining a performance vector-ratio of $(1, 2, \ldots, k)$, then it is impossible to guarantee that a given user will get a performance ratio better than $k$ without having a look at other users' jobs. □

**Proposition 3.2** *$MUSP(k : \sum C_i)$ can not be approximated with a vector-ratio better than $(k, \ldots, k)$.*

**Proof.** Let us consider the instance with $k$ users who are interested in the sum of completion times. Each user owns $n^{(u)} = mx$ unitary jobs ($\forall u, \forall i, p_i^{(u)} = 1$).

The optimal schedule for one user is straightforward since all tasks have the same length *i.e*, $x$ tasks per processor. The optimal value for each user $u$ is $\sum C_i^{(u)*} = m\sum_{i=1}^{x} i = m\frac{x(x+1)}{2}$. Whatever the order of the tasks, the optimal sum of completion times of all the tasks is given by any well balanced schedule : $\sum C_i^* = m\sum_{i=1}^{kx} i = m\frac{kx(kx+1)}{2}$.

Remark that the sum of completion times of all the tasks is the sum of the objective values of all the users : $\sum C_i = \sum_u \sum C_i^{(u)}$. Thus, all the values of the $\sum C_i^{(u)}$ that lead to $\sum C_i^* = \sum_u \sum C_i^{(u)}$ are Pareto-optimal (supposing the solution is feasible). In particular the solution that distributes $\sum C_i^*$ equitably between the $k$ users is Pareto-optimal. In such a solution, the objective values are $\forall u, \sum C_i^{(u)} = \frac{\sum C_i^*}{k} = m\frac{kx^2+x}{2}$ and the associated performance vector-ratio is $\frac{\sum C_i^{(u)}}{\sum C_i^{(u)*}} = \frac{m(kx^2+x)/2}{m(x^2+x)/2} = k - \frac{k-1}{x+1}$. This ratio tends to $k$ when $x$ goes to infinity.

The existence of an approximation algorithm with a ratio better than $(k, \ldots, k)$ would produce a solution which is better than a Pareto-optimal solution on the instance described in this proof for a well chosen value of $x$. □

Notice that the previous inapproximability results have been obtained from the non existence of a solution reaching a given performance vector. Thus, they are not complexity results *i.e*, approximation algorithms with better bounds would not exist even if $P = NP$.

# 4 $MUSP(k : C_{max})$

## 4.1 Absolute Approximation

We proved in the last section that $MUSP(k : C_{max})$ is unary NP-hard. It is even true for a single user. Moreover, no algorithm can guarantee a constant performance vector-ratio better than $(1, 2, \ldots, k)$ over all permutations of objective functions. If the number of processors is fixed, the single user case can be approximated by a PTAS proposed by Hochbaum and Shmoys using dual approximation [9]. Other approximation algorithms exist for the case of arbitrary number of processors. For instance, List Scheduling where the jobs are ordered by decreasing processing times (also called LPT order) is a $\frac{4}{3}$-approximation algorithm with low cost complexity [14]. A better $\frac{5}{4}$-approximation algorithm has been proposed in [11].

Consider the following algorithm called MULTICMAX. For each user $u$, compute a schedule $S^{(u)}$ with a $\rho$-approximation algorithm. Then, sort the users by increasing values of $C_{max}^{(u)}(S^{(u)})$. Finally, schedule jobs of user $u$ according to $S^{(u)}$ between $\sum_{u'<u} C_{max}^{(u')}(S^{(u')})$ and $\sum_{u'\leq u} C_{max}^{(u')}(S^{(u')})$.

The following theorem states the theoretical guarantee of this algorithm.

**Theorem 4.1** MULTICMAX *is a $(\rho, 2\rho, \ldots, k\rho)$-approximation algorithm of $MUSP(k : C_{max})$.*

**Proof.** There are two properties to check. First, the schedule is valid. Indeed, the jobs of each user $u$ are scheduled in disjoint intervals of length $C_{max}^{(u)}(S^{(u)})$ according to $S^{(u)}$. Secondly, the performance vector-ratio is $(\rho, 2\rho, \ldots, k\rho)$. Indeed, $C_{max}^{(u)} = \sum_{u'\leq u} C_{max}^{(u')}(S^{(u')})$. The users are ordered by increasing values of $C_{max}^{(u)}(S^{(u)})$. Thus, $C_{max}^{(u)} \leq uC_{max}^{(u)}(S^{(u)})$. Moreover, $S^{(u)}$ was generated by a $\rho$-approximation algorithm. Thus, $C_{max}^{(u)} \leq u\rho C_{max}^{(u)*}$. □

A straightforward corollary is that if an exact algorithm is used ($\rho = 1$), the vector-ratio obtained is $(1, 2, \ldots, k)$ which ensures the tightness of the bound given in Proposition 3.1. Using the PTAS proposed by Hochbaum and Shmoys, the algorithm is a $(1 + \epsilon, 2 + 2\epsilon, \ldots, k + k\epsilon)$-approximation algorithm.

The permutation of objectives problem pointed out in Section 3 also holds in Theorem 4.1: the theorem is valid for a given unknown permutation of users. Remark that one user can not know in advance his/her rank in the algorithm.

## 4.2 Distance to the Pareto Set

The performance vector-ratio of MULTICMAX depends on $k$ and it has been shown to be tight (using an exact algorithm for computing $S^{(u)}$). One could argue that such a solution is inefficient and thus it is not interesting. However, the performance vector-ratio represents the distance between the proposed schedule and the solution that reaches the absolute best makespan for every users which is not feasible (in the general case). In this section, we present a class

of solutions $List$, which are a MULTICMAX with $\rho = 2$, and which contains efficient schedules that are close to the Pareto set. We will prove this result for a restricted class of instances where each user submits a reasonable number of jobs, namely $\forall u, \sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$ (Remark that such instance are not trivial : the instance that provided the approximability lower bound respects this property and the instance used to proove the NP-completeness of the makespan minimization problem also respects it). Our purpose is to show that a constant approximation of a Pareto optimal solution can be constructed while keeping the absolute bound.

List Scheduling is a mechanism that greedily schedules jobs as soon as possible in any given order [7]. It is well-known that List Scheduling is a 2-approximation algorithm. More precisely, if the jobs are sorted according to the given order, we have the following property: $\forall i, C_i \leq \frac{\sum_{i' \leq i} p_{i'}}{m} + (1 - \frac{1}{m})p_i$. We denote by $List$ the schedules obtained by List Scheduling using any order such that all the jobs of user $u$ are scheduled after each job of user $u' < u$ where the users are sorted in non-decreasing order of $C_{max}^{(u)*}$. One can easily see that each solution of $List$ can be worsened to be a solution of MULTICMAX with $\rho = 2$. Thus, any solution of $List$ is a $(\rho, 2\rho, \ldots, k\rho)$-approximation algorithm of $MUSP(k : C_{max})$.

We need a reference solution on the Pareto set for computing the distance between the solutions of $List$ and the Pareto set. Let us define a set of schedules $Lex$ that are optimal for the lexicographical order of the users. Formally, $Lex$ is defined as: $\forall S \in Lex, C_{max}^{(1)}(S) = C_{max}^{(1)*}; \forall u > 1, C_{max}^{(u)}(S) = \min_{S' | \forall u' < u, C_{max}^{(u')}(S') = C_{max}^{(u')}(S)} C_{max}^{(u)}(S')$. In the following, $Lex(u)$ denotes the set of $Lex$ solutions of the problem restricted to the first $u$ users *i.e.*, the jobs of other users are not scheduled. Remark that all solutions in $Lex$ have the same objective values. Thus, it makes sense to define $C_{max}^{(u)}(Lex)$. Let us also define $Idle(Lex(u)) = mC_{max}(Lex(u)) - \sum_{u' \leq u} \sum_i p_i^{(u')}$, the idle area in schedules of $Lex(u)$.

**Property 4.2** $Idle(Lex(u)) < mC_{max}^{(u)*}$

**Proof.** Recall first that in an efficient schedule, all the idle time are at the end of the schedule since all the jobs are released simultaneously.

Let $J_i^{(u')}$ be the latest job to complete in any $S \in Lex(u)$. $p_i^{(u')} \leq C_{max}^{(u)*}$ since $p_i^{(u')} \leq C_{max}^{(u')*} \leq C_{max}^{(u)*}$ (for $u' \leq u$) .

By contradiction, if $Idle(Lex(u)) \geq mC_{max}^{(u)*}$, there exists a processor that is idle for more than $C_{max}^{(u)*}$ units of time. This processor could have schedule $J_i^{(u')}$ sooner. Thus, schedule $S$ does not belong to $Lex(u)$. □

If the previous hypothesis $\forall u, \sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$ does not hold, one can construct an instance where a user with a single job should be scheduled at time 0. Any solution of $List$ does not schedule it that soon. We can derive the following lemma from the hypothesis.

**Lemma 4.3** $\forall u > 2$, *if* $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ *then* $C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$

**Proof.** If $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ then $Idle(Lex(u-1)) = Idle(Lex(u-2)) - \sum p_i^{(u-1)}$.

Recalling that $Idle(Lex(u-2)) < mC_{max}^{(u-2)*}$ (from Property 4.2) leads to $Idle(Lex(u-1)) < mC_{max}^{(u-2)*} - \sum p_i^{(u-1)}$. Since $\sum p_i^{(u-1)} > \frac{mC_{max}^{(u-1)*}}{2}$ (from the hypothesis) and $C_{max}^{(u)*} \geq C_{max}^{(u-1)*} \geq C_{max}^{(u-2)*}$ (from the order of the users), we obtain $Idle(Lex(u-1)) < \frac{mC_{max}^{(u)*}}{2}$. Thus, all the jobs of user $u$ can not fit in the idle time before $C_{max}^{(u-2)}(Lex)$. At least one job of user $u$ finishes after this value. □

**Corollary 4.4** $\forall u > 2, C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$

The proof of the corollary comes directly from the previous lemma.

The next theorem proves that any solution of $List$ is a 3-approximation of $Lex$.

**Theorem 4.5** $\forall u, C_{max}^{(u)}(List) \leq (3 - \frac{1}{m})C_{max}^{(u)}(Lex)$

**Proof.** Let $p_{max}^{(u)} = \max_i p_i^{(u)}$. From the standard analysis of List Scheduling [7], we derive: $C_{max}^{(u)}(List) \leq \frac{\sum_{u' \leq u} \sum_i p_i^{(u')}}{m} + (1 - \frac{1}{m})p_{max}^{(u)}$ which can be rewritten as:

$$C_{max}^{(u)}(List) \leq \left( \frac{\sum_{u' \leq u-2} \sum_i p_i^{(u')}}{m} + \frac{\sum_i p_i^{(u)}}{m} \right) + \left( \frac{\sum_i p_i^{(u-1)}}{m} \right) + (1 - \frac{1}{m})p_{max}^{(u)}$$

The first term of this expression is less than $C_{max}^{(u)}(Lex)$ (from Corollary 4.4). The second one is less than $C_{max}^{(u)*}$ since it is a lower bound of $C_{max}^{(u-1)*}$ and since the users are sorted in non-decreasing values of their optimal makespan. $p_{max}^{(u)}$ is a lower bound of $C_{max}^{(u)*}$.

Finally, we obtain the following expression: $C_{max}^{(u)}(List) \leq C_{max}^{(u)}(Lex) + (2 - \frac{1}{m})C_{max}^{(u)*}$. As $C_{max}^{(u)*} \leq C_{max}^{(u)}(Lex)$, we obtain the bound. □

In the previous analysis, it is required to know the optimal makespan of each user for constructing a solution in $List$. However, if only an $\rho$-approximation of the makespan of each user is known, the approximation of a solution in $List$ remains constant.

## 5 $MUSP(k : \sum C_i)$

We are now interested in the case where all the users want to optimize the sum of completion times of their jobs. Recall that for a single user, this problem can be solved in polynomial time by scheduling the tasks in non decreasing order of their processing time. This algorithm is called SPT [14]. However, for more than one user, this problem is NP-hard. Moreover, no algorithm can guarantee a constant vector-ratio better than $(k, \ldots, k)$ over all permutations of objective functions.

To solve this problem, we first study the single processor case and we analyze its performance vector-ratio. Finally, we extend the analysis to an arbitrary number of processors.

### 5.1 Preliminary analysis for $m = 1$

Similarly as in Section 4, we compute the final schedule of all users' jobs from schedules of each user's jobs. Let us consider $S^{(1)}, \ldots, S^{(k)}$ independent schedules for the jobs of each user. We construct schedule $S$ using the following greedy algorithm called AGGREG: schedule jobs in non-decreasing order of $C_i^{(u)}(S^{(u)})$.

This simple algorithm ensures that the completion times are not degraded by a factor greater than $k$. This is stated by the following proposition.

**Proposition 5.1** *Schedule $S$ is such that $\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq kC_i^{(u)}(S^{(u)})$.*

**Proof.** Let us consider job $J_i^{(u)}$. Notice first that $C_i^{(u)}(S) \leq C_{i'}^{(u')}(S)$ implies $C_i^{(u)}(S^{(u)}) \leq C_{i'}^{(u')}(S^{(u')})$. Thus, for each user $u'$, the set of job $J^{(u')}$ scheduled before $C_i^{(u)}(S)$ in $S$ is such that $\sum_{J_{i'}^{(u')} \in J^{(u')}} p_{i'}^{(u')} \leq C_i^{(u)}(S^{(u)})$. There is one such set per user. $J_i^{(u)}$ completes when all the jobs of $\cup_{u'} J^{(u')}$ are completed. Thus, $C_i^{(u)}(S) = \sum_{u'} \sum_{J_{i'}^{(u')} \in J^{(u')}} p_{i'}^{(u')} \leq kC_i^{(u)}(S^{(u)})$ which proves the proposition. □

The previous algorithm considers each user with the same priority. We can derive another algorithm that does not give the same priority to the users. Let $\lambda_1, \ldots, \lambda_k$ be real positive values such that $\sum_u \lambda_u = 1$. We construct schedule $S^\lambda$ using the following greedy algorithm called AGGREG$^\lambda$: schedule the job in non-decreasing order of $\frac{C_i^{(u)}(S^{(u)})}{\lambda_u}$. Remark that if $\forall u, \lambda_u = \frac{1}{k}$ then $S = S^\lambda$.

The completion times in $S^\lambda$ of jobs belonging to user $u$ are not degraded by a factor greater than $\frac{1}{\lambda_u}$. This is stated by the following proposition. The proof is similar to the proof of Proposition 5.1 and thus it is omitted.

**Proposition 5.2** *Schedule $S^\lambda$ is such that $\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq \frac{C_i^{(u)}(S^{(u)})}{\lambda_u}$.*

Using AGGREG, it is possible to mix $k$ schedules and ensure that the completion time of each job in each schedule is not degraded by a factor greater than $k$. In particular, if a schedule is a $\rho$-approximation on the sum of (weighted or unweighted) completion times then by mixing it with the $k-1$ other schedules, the final schedule is a $(k\rho)$-approximation for this user.

**Lemma 5.3** AGGREG $(SPT(1), \ldots, SPT(k))$ *is a $(k, \ldots, k)$-approximation algorithm of $MUSP(k : \sum C_i)$ on one processor.*

The proof of this lemma comes directly from the optimality of SPT for one user and from Proposition 5.1.

### 5.2 Extension to $m$ processors

The previous lemma states an interesting result for the single machine case. In this section, we show that the general framework can be easily extended to $m$ processors.

We now describe the MULTISUM algorithm. Let $S^{(1)}, \ldots, S^{(k)}$ be $k$ SPT schedules on $m$ processors. Each schedule $S^{(u)}$ can be seen as $m$ independent schedules denoted $S_1^{(u)}, \ldots, S_m^{(u)}$. For each processor $j$, schedules $S_j^{(u)}$ of each user $u$ are mixed by AGGREG into $S_j$. Then, a global schedule on $m$ processors is constructed by executing $S_j$ on processor $j$ $(1 \leq j \leq m)$.

**Theorem 5.4** MULTISUM *is a $(k, \ldots, k)$-approximation algorithm for $MUSP(k : \sum C_i)$.*

The proof of Theorem 5.4 comes directly from Lemma 5.3 and from the independence of processors.

Optimizing the sum of weighted completion times case is NP-hard even for a single user. However, there exist approximation algorithms such as the one proposed by Hall *et al.* that reaches a performance ratio of $(4 - \frac{1}{m})$ [8]. Using a $\rho$-approximation algorithm instead of SPT leads to the following MULTIWEIGHTEDSUM algorithm.

**Corollary 5.5** MULTIWEIGHTEDSUM *is a $(\rho k, \ldots, \rho k)$-approximation algorithm for $MUSP(k : \sum w_i C_i)$.*

Using Hall *et al.*'s algorithm leads to a $((4 - \frac{1}{m})k, \ldots, (4 - \frac{1}{m})k)$-approximation algorithm.

# 6 Mixed objectives

## 6.1 Multi-objective optimization

In this section we are interested in the mixed objective case where $k'$ users are interested in the sum of completion times and $k''$ users are interested in the makespan : $MUSP(k' : \sum C_i \ ; \ k'' : C_{max})$ with $k = k' + k''$.

Remark that the technique presented in the last sectionfor the sum of completion times also works for the makespan objective. Every user will get a performance of $k$. However, it is possible to improve this result. Indeed, if no users are interested in the sum of completion times, we can obtain a performance ratio of $(1, \ldots, k)$. While if one user is interested in the sum of completion times, the ratio of each user interested in the makespan is worsened to $k$.

The idea in this section is to incorporate techniques presented in Section 4 and 5 into a single scheduling algorithm called MULTIMIXED.

Consider the sub-instance of the $k''$ users interested in the makespan. Let $S^{(cmax)}$ be the schedule generated by MULTICMAX (using any $\rho$-approximation algorithm) on this sub-instance. For each user $u$ interested in the sum of completion times, let $S^{(u)}$ be a SPT schedule of jobs belonging to $u$ on $m$ processors. Let $S_j^{(u)}$ be the sub-schedule of $S^{(u)}$ on the $j$-th processor ($S_j^{(cmax)}$ is similarly defined). For each processor $j$, mix all the schedules $S_j^{(u)}$ ($\forall u \leq k'$) and $S_j^{(cmax)}$ into $S_j$ using AGGREG$^\lambda$ with $\lambda_u = \frac{1}{k}, \forall u \leq k'$ and $\lambda_{cmax} = \frac{k''}{k}$. Construct the final global schedule $S$ by executing $S_j$ on processor $j$.

The theoretical guarantee of MULTIMIXED is stated by the following theorem.

**Theorem 6.1** MULTIMIXED *is a* $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$*-approximation algorithm for* $MUSP(k' : \sum C_i \ ; \ k'' : C_{max})$.

**Proof.** All the users interested in the sum of completion times get a performance ratio of $k$. The corresponding proof is similar to the one of Theorem 5.4 and it is omitted.

Consider now user $u$ interested in the makespan. Without loss of generality, we have $C_{max}^{(u)}(S^{(cmax)}) \leq u\rho C_{max}^{(u)*}$ (from Theorem 4.1) which means that $\forall i \leq n^{(u)}, C_i^{(u)}(S^{(cmax)}) \leq u\rho C_{max}^{(u)*}$.

Due to Proposition 5.2, all the jobs of the users interested in $C_{max}$ are not degraded by a factor greater than $\frac{k}{k''}$. Thus, $\forall i \leq n^{(u)}, C_i^{(u)}(S^{(cmax)}) \leq \frac{k}{k''} u\rho C_{max}^{(u)*}$ which concludes the proof. □

## 6.2 Optimizing the fairness on degradations

In Section 2.3, we introduced a related problem, that is the optimization of a fairness (in fact a norm) objective function of degradations. By using the monotony of norms according to the Pareto order, we show that solving the multi-optimization problem solves the norm optimization problem. The minimum of the objective function is reached by a Pareto optimal solution. The algorithms we proposed do not generate all the Pareto set. However, several interesting properties can be derived.

First, the vector-approximation ratio of an algorithm implies an approximation for the norm optimization problem. For instance, consider the maximum of degradation ($L_\infty$ norm), a $(\rho, \rho, \ldots, \rho)$-approximation algorithm is a $\rho$-approximation of the norm.

Second, all the algorithms we proposed have parameters that can be tuned in order to generate different guaranteed trade-offs. Thus, it is possible to scan the solution space by changing those parameters and keep the solution that optimizes the norm function. For instance, on the $L_\infty$ norm case, the following iterative process can be used. Generate a solution using the algorithm. Sort users in decreasing order of their degradation. Increase the priority of the first user by decreasing the priority of the last ones. This will probably not generate an approximation algorithm (in terms of worst case bounds) but should give good results in practice.

# 7 Conclusion

In this paper, we analyzed the Multi-Users Scheduling Problem with many users interested in optimizing objectives among the most popular existing objectives on $m$ processors. For the cases where all the objectives are the same, we derived some inapproximability results and we proposed algorithms that are very close or equal to the inapproximability bounds. In the case of mixed objectives, we lack an inapproximability bound but we proposed an algorithm with not-straightforward vector-ratio. Indeed, its restrictions to the cases where all the users share the same objective function have the same performances than the dedicated algorithms.

This analysis leads us to the following discussion. The approximation vector-ratios are linear in the number of users which may appear disappointing. Indeed, the vector-ratios are computed relatively to a absolute best solution which is usually unfeasible. However, it ensures the performance degradation of each user is always reasonable. Using a deeper analysis for the makespan case, we showed that a modification of the algorithm provides a solution close to the Pareto set. This ensures that the solution is efficient.

## References


[1] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

[2] E. Angel, E. Bampis, and L. Gourvès. Approximation results for a bicriteria job scheduling problem on a single machine without preemption. *Information Processing Letters*, 94(1):19–27, Apr. 2005.

[3] A. Archer, C. Papadimitriou, K. Talwar, and E. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agent. In *Proc. of the 14th Annual ACM Symposium on Discrete Algorithms (SODA)*, 2003.

[4] K. Baker and J. Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6:7–16, 2003.

[5] M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 762–771, 2002.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[7] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[8] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[9] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Pratical and theoretical results. *Journal of ACM*, 34:144–162, 1987.

[10] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In D. G. Feitelson and L. Rudolph, editors, *Proc. of the 7th International workshop JSSPP*, number 2221 in LNCS. Springer, 2001.

[11] H. Kellerer. Algorithms for multiprocessor scheduling with machine release times. *IIE Transactions*, 30(11):991–999, Nov. 1998.

[12] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of biological requests. In *Symposium on Parallelism in Algorithms and Architectures SPAA'2006*. ACM Press, 2006.

[13] A. Legrand and C. Touati. Non-cooperative scheduling of multiple bag-of-task applications. In *Proc of InfoCom 2007*, pages 427–435, 2007.

[14] J. Y.-T. Leung. Some basic scheduling algorithms. In J. Y.-T. Leung, editor, *Handbook of Scheduling*, chapter 3. CRC Press, 2004.

[15] J. Mo and J. Warland. Fair end-to-end window-based congestion control. In *Proc. of SPIE '98: International Symposium on Voice, Video and Data Communications*, 1998.

[16] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[17] F. Pascual, K. Rzadca, and D. Trystram. Cooperation in multi-organization scheduling. In *Proc. of EuroPar 2007*, number 4641 in LNCS, pages 224–233, Aug. 2007.

[18] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling*. Springer, 2007.