A new effective heuristic for the intelligent management of the preventive maintenance tasks of the distributed systems

Kondo Hloindo Adjallah, Kossi Pélopé Adzakpa, J. R. Lee

A new effective heuristic for the intelligent management of the preventive maintenance tasks of the distributed systems

K. H. Adjallah[a]*, K. P. Adzakpa[a] and J. Lee[b]

[a] *Institute of Computer Science and Engineering of Troyes,*
University of Technology of Troyes, 12 rue Marie Curie, 10010 Troyes Cedex, France
Phone: (+33) 325 71 56 31 - Fax: (+33) 325 71 56 49
{adjallah, adzakpa}@utt.fr

[b] *Center for Intelligent Maintenance Systems,*
University of Wisconsin-Milwaukee, 9100 Swan Road, Milwaukee, WI53224, USA
jaylee@uwm.edu

Abstract

We consider a class of scheduling problems of n weighted tasks on M identical, and parallel processors with an objective of minimizing the sum of the tasks weighted flow-times. A priority rule for total weighted flow-time (PRTWF), is then proposed for locally optimal scheduling of tasks with unequal release dates and processing times. Then, an algorithm based on a heuristic and the PRTWF, is worked out to minimize the total weighted flow-time of the given set of tasks on a single server. This algorithm is designed for implementation in a dynamic process of real-time decision-making. It is next extended to tasks scheduling (with unequal release dates and processing times) on parallel servers, while minimizing their total weighted flow-time. A lower bound of solutions is also proposed to evaluate the algorithm, with a complexity in $O(n^3)$ in the off-line scheduling process. The rule is then used in an algorithm of on-line planning and scheduling of maintenance tasks in a large size distributed system with weighted Equipments.

Keywords: Tasks scheduling, weight, flow-time, release date, real-time, maintenance, decision making

1. Introduction

In several areas of activity as in maintenance, delays in starting tasks engender, for the user, extra costs due to the degradations of Equipments, raw materials and products' quality, and disorganization of the production process. Such cost may be very high in some cases, particularly in maintenance tasks management for several reasons: the considered tasks are not always available from the initial time t_0 before the decision-making times t; in some cases, tasks cannot be started before a given date which marks the beginning of inescapable degradations, etc. As a result, the processing of many tasks may be delayed in relation to their release dates. Under these conditions, the costs include additional costs due to delayed completion of tasks. It is then necessary, in this case, to set up a real-time decision making strategy to manage tasks.

Obviously, these costs increase with the flow-times of worsening operation conditions. A way to reduce these costs is to schedule tasks so as to minimize their flow-times in penalizing situations. When the additive cost increases uniformly and in an identical way for all the tasks, the problem consists in minimizing, through optimal scheduling, the sum of the flow-times of all the tasks in penalizing situations.

*Corresponding author

Works addressing the typical problem of maintaining optimal task scheduling and processors allocation are rare in the existing literature. We mention in this respect Weinstein and Chung [27] who proposed a mixed-integer linear model which incorporates preventive maintenance into the production policies. This method aims at minimizing the cost and the deviation of production tasks in contrast to maintenance free production policies. In regards to preventive maintenance, we also mention the heuristic method of Qi *et al.* [17] based on the Shortest Processing Time (SPT) priority rule to minimize the makespan in a production policy. They studied the parallel-machines scheduling problem where preventive maintenance tasks are performed on each machine, just as in Lee and Chen [15] and Graves and Lee [13]. Lee and Chen [15], Qi *et al.* [17] proposed a branch and bound method to minimize weighted completion time of tasks based on the partition formulation of the problem.

In [1], we proposed an on-line algorithm for real-time maintenance planning and scheduling methods on a given horizon. In the given algorithm, tasks are scheduled in real-time, and the precedence constraints linking the different tasks on the same Equipment have been dealt with. But the existing methods in production are quite limited where the maintenance assets' sensitivity (importance) and tasks' set-up time are concerned. In this paper, we propose a priority rule for the total weighted flow-time. This rule, which considers task pairs, has very good properties. It can be used as well in production as in maintenance activities planning and scheduling. Basing

on it, we propose an algorithm for maintenance tasks real-time scheduling and resources (processors) allocation.

In the continuation of this paper, we state in section 2 the real-time maintenance decision making problem and the link to a scheduling problem of weighted tasks with unequal release dates. In order to solve it, section 3 is devoted to a static (off-line) approach to the scheduling problem. A local optimality priority rule (PRTWF) is proposed, proved and compared to existing rules in the literature in this section. In section 4, we adapted the approach to the real-time maintenance decision making process. Experimentations are provided in section 5 and finally section 6 concludes and proposes future extensions to this work.

2. From maintenance activities management to scheduling problem

This paper deals with the problem of maintenance process modelling with cost minimization through tasks scheduling on the one hand and repairmen (processors) assignment to tasks on the other hand. In the remainder of this paper, we will use the term "processor" to signify the principal maintenance resource or repairman assigned to the tasks. Let us consider a distributed system composed of N sites working independently and in parallel, and sharing M repairmen for the preventive maintenance activities on the different Equipments of the system. On a given site, Equipments operate in series. Obviously, the number of repairmen is less than the total number of Equipments in the system $\left(M << \sum_{k=1}^{N} N_k\right)$ and the repairmen are shared by the overall system, N_k being the number of Equipments on a site k. An Equipment may be a production machine or a simple equipment. It generally requires important logistic times to move a repairman from a site to another, but in the approach proposed in this paper, we assume that the logistic time are small enough to be included in the maintenance tasks' processing times. The main problem is then to dynamically evaluate the processor needs at sites levels so as to establish priorities. The available processors must be assigned to the Equipments so as to ensure a minimum required availability to the sites while minimizing the involved cost on a working horizon.

To model this problem, we consider a planning horizon H on which each site's unavailability and maintenance cost must be minimal. We assume that the maintenance tasks are processed without pre-emption. So, once begun, each maintenance task is processed to its completion without interruption. We also consider them to be perfect maintenance tasks, which mean that an Equipment is supposed to be renewed after a preventive maintenance. The different Equipments in the system are weighted according to their sensitivity and their importance in the system. The maintenance tasks processors are supposed to have equivalent performance and can be used in parallel. In the approach the Equipments' availabilities are assumed to follow the exponential law. But this assumption is not restrictive. The model can still be used when the Equipment behave according to other laws such as Weibull's law. The failure and repair rates (λ_{ik} and μ_{ik}) of an Equipment E_{ik} (equipment i of site k) are then assumed to be constant on the planning horizon H. The exponential availability A_{ik} of an Equipment E_{ik} is renewed at the end of a maintenance task at time T (considered also to be a start up date of the Equipment) as

$$A_{ik}(t) = \frac{\mu_{ik}}{\lambda_{ik} + \mu_{ik}} + \exp[-(\lambda_{ik} + \mu_{ik})(t - T)] \quad (2.1)$$

In order to guarantee the minimum required availability of each site, a threshold α_{ik} given in]0, 1[is imposed on the availability A_{ik} of each Equipment E_{ik}. An Equipment E_{ik} which works for a time without failure reaches that threshold and should be submitted to a preventive maintenance task. An Equipment working under its threshold is supposed to be in a critical state with a high probability of failure. The time spent from the occurrence of such a critical state on the Equipment E_{ik} to the completion date of the maintenance task on that Equipment (which is considered to be also the start-up of the Equipment), involves a time-unit cost w_{ik} which is the weight of E_{ik} in the system. From the expression of the availability function of the Equipment E_{ik} in equation (2.1) and the threshold α_{ik}, the duration from a start-up of the Equipment to the critical date is derived from the inequality $A_{ik} \geq \alpha_{ik}$. By taking the start up date T to be the initial date 0, this duration for the Equipment E_{ik} is then τ_{ik} expressed as below:

$$\tau_{ik} = \frac{-1}{\lambda_{ik} + \mu_{ik}} \ln\left[\alpha_{ik}\left(1 + \frac{\mu_{ik}}{\lambda_{ik}}\right) - \frac{\mu_{ik}}{\lambda_{ik}}\right] \quad (2.2)$$

Each Equipment has as many start ups on the planning horizon as there are completions of maintenance tasks following a critical state. Let us note $r_{m,ik}$ the m^{th} occurrence date of a critical event on the Equipment E_{ik} and $c_{m,ik}$ the completion date of the corresponding maintenance intervention. Then the critical states cost on the system on the planning horizon H is expressed as

$$C_{tot}(H) = \sum_{k=1}^{N} \sum_{i=1}^{N_k} \left(\sum_{m / r_{m,ik} \in H} w_{ik}(c_{m,ik} - r_{m,ik}) \right) \quad (2.3)$$

Each maintenance task on the Equipment E_{ik} is assumed to have a processing time equal to the Equipment's mean time to repair (*Mtp*) which is $1/\mu_{ik}$. The problem is then formulated as

(P_1): *Minimize* $C_{tot}(H)$ *with the M processors under the conditions* $A_{ik} \geq \alpha_{ik}$.

In scheduling, this formulation will be:

(P_2): *Minimize the total weighted flow-time of tasks released on unequal dates on M parallel machines (processors), under the availability conditions.*

However, the problem considered herein in not a static scheduling problem as it is generally considered in scheduling problems. The maintenance tasks arrive in real-time until the end of the horizon. When all the Equipments have equal weights and the logistic times are still very small, we considered the problem in [1]. To deal with this problem with unequally weighted Equipments, we developed an approach for local optimality which will be used in a real-time decision-making algorithm. Before the section on the local optimality, let us notice the following two properties of the problem (P_1). The proof of the first one is straightforward and the second one can be proved in the same way as in [1] where the Equipments are equally weighted. The flow-time of a maintenance task corresponds to the time that the corresponding Equipment spends in a critical state.

Proposition 2.2.1. *Tasks relative to the same Equipment E_{ik} are linked by a precedence constraint defined as hereafter. If $c_{m,ik}$ is the completion date of the m^{th} preventive maintenance task on E_{ik}, then the $(m+1)^{th}$ task's release date is determined by $r_{m+1,ik} = c_{m,ik} + \tau_{ik}$, where τ_{ik} is defined in the expression* (2.2)

Proposition 2.2.2. *Problem **(P_1)** is NP-hard.*

See the proof in [1].

As we said above, the maintenance problem considered herein is real-time decision making problem. In the following section, we develop some tools to solve the scheduling aspect of this problem. To this end, we consider the static problem consisting in scheduling a given set of *n* weighted tasks with unequal release dates. As mentioned earlier in this section, in maintenance activities, the flow-time corresponds to the time an Equipment spends a critical state.

3. Solving the scheduling problem for minimizing the weighted flow-time

3.1. Overview of weighted flow-time scheduling problems

Authors concerned by the flow-time minimization problem, have often assumed that all tasks are of equal importance. This assumption does not necessarily hold, since in real life, tasks may, for example, be of different unit costs, different holding costs, or may carry different contractual penalties for overdue deliveries. For this reason, when it comes to minimizing the overall cost associated with the total flow-time, tasks cannot be treated as being equivalent. Now, as the evolution of the costs is not the same for all tasks, the problem then comes to minimizing the sum of the weighted flow-times of all penalized tasks through optimal scheduling. In this respect, tasks of the same release dates have received considerable attention in the literature.

Both "unweighted" and weighted flow-time minimization problems can arise for tasks on single machine, or parallel machines in flow-shop, be they uniform or not. We are interested in the minimization of the sum of weighted flow-times of *n* tasks released at different dates on *M* parallel processors (machines) ($0<M<<n$), inside a given time horizon *H*. We consider that each released task is processed by only one processor among available *M* processors. This problem requires an optimum scheduling of the *n* tasks, while minimizing the total weighted flow-time of tasks on the time horizon *H*.

Lenstra [16] proved the NP-hardness of the flow-time minimization problem for tasks with different release dates. Over the last three decades, the flow-time minimization problem has indeed retained considerable research attention both on single and parallel machines, and different solution approaches have been developed. Among the algorithms proposed to minimize the "unweighted" flow-time of tasks with different release dates, we cite, in particular, the single machine scheduling heuristics developed by Chu [7] and [9]. These heuristics are based on an efficient priority rule that enables real-time decision by local optimal tasks scheduling, as stated in our problem, and they take into account tasks' release dates and processing times. Chu uses the same rule in the branch and bound algorithm [8] proposed later for solving this problem. The weighted flow-time was considered by Bianco and Ricciardelli [5], who developed a Branch and Bound algorithm based on a set of rules for tasks scheduling on single machine. Their work is, to our knowledge, the one to which our formulation comes closest.

Works on the NP-hard problem of weighted flow-time minimization fall roughly into two categories: those dedicated to single machine and flow-shops, and those focussing on parallel machines and on uniform machines. Among the investigations dedicated to flow-shop model, we mention four heuristics developed by Gelders and Sambandam [12] to minimize the sum of weighted flow-time and weighted tardiness of tasks. Rajendran and Ziegler also proposed two heuristics; the first one minimizes the weighted flow-time [19], while the second one [20] minimizes the weighted flow-time in the presence of sequence-dependent set-up times. Both heuristics are based on the WSPT rule (Weighted Shortest Processing Time), adapted in some recursive relations. Rajendran [18] also developed a heuristic based on a priority rule and some recursive relations for minimizing an objective-function corresponding to the sum of weighted flow-time, weighted earliness and weighted tardiness. This priority rule was used in a scheduling improvement strategy based on sequential task insertions.

Sharadapriyadarshini and Rajendran [24] studied the same problem and used a convex combination of the makespan and the flow-time as objective function in the optimization. Framinan *et al.* [11] recently proposed efficient heuristics for this problem, where the objective-function is the sum of the weighted waiting-time and the weighted flow-time.

Several works have also focussed on the weighted flow-time minimization on parallel machines with tasks released on the same date. Bruno *et al.* [6] showed that this problem is NP-hard even in the case of two machines. Sarin *et al.* [23] developed a branch and bound method for solving this problem. Compared to the branch and bound methods of Elmaghraby and Park in [10] and the one of Barnes and Brennan in [3], the algorithm of Sarin *et al.* is better because it is based on a better lower bound. Furthermore, Azizoglu and Kirca [2], discussed some properties of an optimal solution and proposed a branch and bound which is better than that of Sarin *et al.* Webster established in [26] a sufficient condition for the weighted flow-time optimality in a class of parallel machines problems. This condition is an adaptation of the weighted shortest processing time (WSPT) rule. He also proposed in [25] two lower bounds to the problem and compared them to those in [23], [10] and [3]. Belouadeh and Potts [4] proposed a method based on the dominance properties in [10]. All the above mentioned works assume that tasks are available and released at the initial time of the optimization time window.

Now, in several real-life problems, different tasks become available at different times. This is, for example, the case for maintenance tasks scheduling. Reports on works aiming at finding methods for solving total weighted flow-time minimization problems in the case of tasks with unequal release dates, are very scarce in the existing literature. In [21], Rajendran and Ziegler dealt with a flow-shop problem in which the objective is to minimize the sum of the weighted flow-time and weighted tardiness of tasks. They considered one case where tasks were all released at time 0 and another case where tasks were released at different dates. They proposed then a heuristic based on some recursive relations and a priority setting rule that takes into account the task's weight relative to the flow-time and the tardiness, and the lower bound of tasks completion times on the machines. The algorithm consists in generating *2m* sequences (*m* is the number of machines) and selecting the best case among simultaneously released tasks. They proposed some modifications to deal with the case of different release dates. As for Rinaldi and Sassano [22] and Bianco and Rincciardelli [5], they proved some dominance properties for the total weighted flow-time minimization on a single machine in the presence of release dates. These dominance properties were used in an implicit enumerative algorithm based on a branch and bound concept that Bianco and Ricciardelli proposed in [5] allowing the solution for up to 10 tasks. The dominance properties of Rinaldi and Sassano have also been used by Hariri and Potts in [14] who proposed a lagrangian relaxation method for lower bound determination.

In this section, we propose a local optimality priority rule for total weighted flow-time minimization with unequal release dates of tasks.

The algorithm utilizes the priority rule and the dominance concept defined in a heuristic that minimizes the total weighted flow-time on single processor and on parallel processors. We use the priority rule with relating concepts in an initial scheduling algorithm before adapting it for the on-line maintenance problem. The proposed algorithm allows the solution of large size problems in a small computational time.

3.2. Weighted flow-time problem formulation

Here is the list of notations used in the formulation of the weighted flow-time problem.

M is the number of processors
n is the number of tasks to be scheduled
p_i is the processing time of task i
r_i is the release date of task i
w_i is the weight of task i
c_i is the completion time of task i
$[\pi]$ is a partial sequence of tasks
$[\pi,i]$ is the sequence obtained after task i is appended to the end of $[\pi]$.

In this problem, each processor can process only one task at a time. The release dates of tasks are different, and each task can be processed from its release date on any of the M processors. Once begun, a task ought to be processed completely without preemption. The aim is to minimize the total weighted flow-time of the tasks expressed as $F_w = \sum_{i=1}^{n} w_i(c_i - r_i)$ or equivalently the mean weighted flow-time
$\bar{F}_w = \frac{1}{n}\sum_{i=1}^{n} w_i(c_i - r_i)$.
This problem is stated as:
(P): *Minimize the total weighted flow-time of n tasks with unequal release dates, on M identical parallel processor*s.

Following the usual notation in scheduling, the problem **(**P) can be formulated as $Pm/r_i/\Sigma\, w_iC_i$. Lenstra [16] proved that the problem of "unweighted" flow-time minimization on parallel processors of tasks with unequal release dates is NP-hard. Bruno *et al.* [6] also showed that even in the case where the release dates are the same, the problem of the weighted flow-time minimization is NP-hard. Consequently, the problem **(**P) is NP-hard. In our approach, we shall first focus on the case of single processor to design a locally optimal priority rule which will, in turn, be used on parallel processors as well. The priority criterion, the dominance concept and the

associated properties used in the algorithm will be defined in the following subsection.

3.3. The priority rule and its properties

Turning to the problem of total weighted flow-time minimization on a single processor in the context of different release dates, we denote the release date for task i by r_i and its weight and processing time by w_i and p_i respectively.

3.3.1. Concepts and definitions.

Definition 3.3.1. Consider a task-pair $\{i, j\}$ to be scheduled at the end of a sequence $[\pi]$ and at time t. We say that task i dominates task j at time t and symbolize this by writing $i \prec j$, if the scheduling of i before j, yields a smaller value for the total cost (in term of total weighted flow-time) than the scheduling of task j before task i.

Remark 3.3.2. In classical scheduling terminology, the term dominance is used for partial schedules. But in the above definition, it can be noticed that if at time t, $[\pi]$ is a partial schedule, saying "$i \prec j$" is equivalent to saying that "the sequence $[\pi, i]$ dominates $[\pi, j]$".
Let us consider the task-pair $\{i, j\}$ to be scheduled at the end of the partial schedule $[\pi]$, at time t. We formally define the associated function PRTWF (Priority Rule for Total Weighted Flow-time) as follows.

Definition 3.3.3. For any task-pair $\{i, j\}$, and at any time t, the function PRTWF (Priority Rule for Total Weighted Flow-time) is given by

$$PRTWF(i, j, t) = (w_i + w_j).\max(r_i, t) + w_j.p_i \quad (3.1)$$

We next prove the following theorem.
Theorem 3.3.4. *At any time t, $i \prec j$ if and only if PRTW*F$(i, j, t) \le$ *PRTW*F(j, i, t).

Proof. The parameters r_i, p_i, w_i are affected to the task i, and the parameters r_j, p_j, w_j are affected to the task j.
Assume that $r_i \le r_j$.

One can easily verify that the following are satisfied, by scheduling i and j at time t.
i) If $r_i \le r_j \le t$ then $i \prec j$ iff $w_j.p_i - w_i.p_j \le 0$.
ii) a) If $r_i \le t \le r_j$ and $t + p_i \le r_j$ then $i \prec j$ is obvious.
ii) b) If $r_i \le t \le r_j$ and $t + p_i > r_j$ then $i \prec j$ iff
$[(w_i + w_j)t + w_j.p_i] - [(w_i + w_j)r_j + w_i.p_j] \le 0$.
iii) a) If $t \le r_i \le r_j$ and $r_i + p_i \le r_j$ then $i \prec j$ is obvious.
iii) b) If $t \le r_i \le r_j$ and $r_i + p_i > r_j$ then $i \prec j$ iff
$[(w_i + w_j)r_i + w_j p_i] - [(w_i + w_j)r_j + w_i p_j] \le 0$.

All the cases i), ii) b) and iii) b) can be summarized in $i \prec j$ if and only if
$(w_i + w_j).\max(t, r_i) + w_j.p_i \le (w_i + w_j).\max(t, r_j) + w_i p_j$.

In the case ii) a), we have $r_i \le t \le r_j$ and $t + p_i \le r_j$. From the second inequality, one can derive
$(w_i + w_j)(t + p_i) \le (w_i + w_j)r_j$
which leads to
$(w_i + w_j)t + w_j p_i \le (w_i + w_j)(t + p_i) \le$
$(w_i + w_j)r_j \le (w_i + w_j)r_j + w_i.p_j$.
The first member and the last member of this last inequality string in conjunction with the first inequality in the case ii) a) leads to
$(w_i + w_j).\max(t, r_i) + w_j.p_i \le$
$(w_i + w_j).\max(t, r_j) + w_i.p_j$.

In the case iii) a), we also have $t \le r_i \le r_j$ and $r_i + p_i \le r_j$. From the second inequality, one derives
$(w_i + w_j)(r_i + p_i) \le (w_i + w_j)r_j$,
which implies
$(w_i + w_j)r_i + w_j.p_i \le (w_i + w_j)(r_i + p_i) \le$
$(w_i + w_j)r_j \le (w_i + w_j)r_j + w_i p_j$.

The first and the last members of this latter inequality string and the first inequality in the case iii) a) allow us to write
$(w_i + w_j).\max(t, r_i) + w_j p_i \le$
$(w_i + w_j)\max(t, r_j) + w_i p_j$.

This proves that in all the cases where $r_i \le r_j$ we have $i \prec j$ if and only if
$(w_i + w_j).\max(t, r_i) + w_j p_i \le$
$(w_i + w_j).\max(t, r_j) + w_i p_j$.

Now, assume that $r_i > r_j$.
By inverting the roles of i and j in the previous expression, we have
$j \prec i$ if and only if $(w_i + w_j).\max(t, r_j) + w_i p_j \le (w_i + w_j).\max(t, r_i) + w_j p_i$
which is the same as $i \prec j$ if and only if
$(w_i + w_j).\max(t, r_i) + w_j p_i \le (w_i + w_j).\max(t, r_j) + w_i p_j$.

So, in all the cases, we have
$i \prec j$ if and only if $(w_i + w_j).\max(t, r_i) + w_j p_i \le (w_i + w_j).\max(t, r_j) + w_i p_j$. □

Remark 3.3.5. The function PRTWF can be considered as a local optimality priority rule for the total weighted flow-time in presence of different release dates.

Remark 3.3.6. In the particular case where all the tasks have the same weight $w_i = 1$ (i=1,...,n), the function PRTWF becomes

$$PRTW(i,j,t) = 2.\max(r_i, t) + p_i \quad (3.2)$$

This particular function depends only on task i, not on j and is the value of the PRTF function defined in [7] for the priority rule for total flow-time in the case of "unweighted" (or equally weighted) tasks and unequal release dates, and which we use in [1] when Equipments in the system are equally weighted.

Definition 3.3.7. The dominance matrix Ω helps to identify the subset of tasks to be scheduled among the set of tasks and is defined by:

$$\Omega(i,j) = \begin{cases} 1 \text{ if } i \neq j \text{ and } i \prec j \\ 0 \text{ otherwise} \end{cases} \quad (3.3)$$

With the matrix Ω as defined above, we have the following property.

Proposition 3.3.8. With Ω as defined above, the following two statements are equivalent

$\Omega(i,j) + \Omega(j,i) = 1$ for $i \neq j$, or

$$\Omega(j,i) = (1 - \Omega(i,j)).\mathbf{1}_{i\neq j} \quad (3.4)$$

The calculus of Ω can then be reduced to the upper triangular matrix.

Definition 3.3.9. At any time t, the strength f_i of task i is the number of tasks that i dominates.

$$f_i = \sum_j \Omega(i,j) \quad (3.5)$$

The strength vector verifies the following property.

Proposition 3.3.10. *If n is the number of task to be scheduled at time t, then the strength vector satisfies the following relation.*

$$\sum_i f_i = \frac{n(n-1)}{2} \quad (3.1.6)$$

Proof

$$\begin{aligned} \sum_i f_i &= \sum_i \sum_j \Omega(i,j) \\ &= \sum_{i,j,i\neq j} \Omega(i,j) + \Omega(j,i), \text{ because } \Omega(i,j) = 0 \\ &= \sum_{\{i,j\},j\geq i} (\Omega(i,j) + \Omega(j,i)) \\ &= \sum_{\{i,j\}} (\Omega(i,j) + \Omega(j,i)), \text{ the symmetry} \end{aligned}$$

Now, $\Omega(i,j) + \Omega(j,i) = 1$ if $i \neq j$, and the number of such sums where $i \neq j$, is C_n^2, which is equal to $n(n-1)/2$. □

With the properties of the PRTWF function, and those of the matrix Ω and the strength vector relating to a task, the ground is now set for stating the heuristic algorithm. The principle of the algorithm is stated in section 5. But before that, let us carry some comparative studies on the dominance rules in the literature for this kind of problem.

3.4. Comparative studies on the dominance rules in [5] and [22]

The analysis presented in this section enables us to highlight the power of the function PRTWF compared to the dominance criteria suggested in [5] and [22], for the weighted flow-time minimization problem. To this end, we shall first recall the dominance properties in [5] and [22], and compare them to the PRTWF criterion. The theorems concerning the dominance properties in these references are enumerated using the notations of the present paper. The reader can refer to [5] and [22] for the proofs of the theorems referred to here.

Theorem 3.4.1. *Given a set of n tasks and a partial sequence s_k of tasks ($k<n$), where t is the completion time for s_k, and a task i not yet scheduled. If all tasks $j\neq i$ which are not yet scheduled verify the two conditions with respect to the task i*

(a) $\frac{p_i}{w_i} \leq \frac{p_j}{w_j}$

(b) $\max(r_i,t) \leq \max(r_j,t)$

then i dominates j.

Theorem 3.4.2. *Given a set of n tasks and a partial sequence s_k ($k < n$) where t is the completion time for s_k, and two tasks i, j not yet scheduled,*
if $r_j \geq \max(r_i, t) + p_i$, *then i dominates j.*

Theorem 3.4.3. *Given a set of n tasks and a partial sequence s_k ($k < n$) of completion time t, and two tasks i, j not yet scheduled, if*
(a) $w_i \geq w_j$
(b) $\max(r_j ; t) + p_j \leq \max(r_i, t) + p_i$
(c) $w_i.[\max(r_j, t) + p_j - \max(r_i, t) - p_i] + w_i p_i - w_j p_j \geq (p_j - p_i) \sum_{l\in Q} w_l$
then i dominates j, where Q is the set of unscheduled tasks not containing i and j.

Theorem 3.4.4. *Given a set of n tasks and a partial sequence s_k ($k < n$) with the completion time t, and two tasks i, j not yet scheduled, if*
(a) $w_i \geq w_j$
(b) $\max(r_j, t) + p_j \leq \max(r_i, t) + p_i$
(c) $w_i p_i - w_j p_j \geq [\max(r_i, t) + p_i - \max(r_j, t) - p_j] \times \sum_l w_l + \delta_{-1}(p_j - p_i)(\sum_l w_l - w_j - w_i)$
(*where δ is the Kronecker delta symbol and the sum is on the set of tasks not yet scheduled noted* $\overline{K}$*), then i dominates j.*

Theorem 3.4.5. *Consider a set of n tasks and a partial sequence s_k ($k < n$) of completion time t and two tasks i, j not yet scheduled. Let K (the set of tasks not yet scheduled) be the "dense" set, i.e., the set in which no sequence has an inserted idle time. If*
(a) $w_i \geq w_j$
(b) $\max(r_j, t) \leq \max(r_i, t)$
(c) $\max(r_j, t) + p_j \leq \max(r_i, t) + p_i$
then i dominates j.

Theorems 3.4.1 to 3.4.5 describe the dominance criteria used to schedule a task at a time t at the end of a partial

schedule s_k. From these theorems, we formulate the following remark in the form of a proposition followed by the proof. Note also that this proposition can be thought of intuitively, the PRTWF function being a local optimality condition.

Proposition 3.4.6. *Theorem 3.3.4 is a necessary condition for the assumptions in theorems 3.4.1 to 3.4.5 which check if task i dominates task j.*

In other words, if one can show that task i does not dominate another task j by using theorem 3.3.4, then there is no need to try to use theorems 3.4.1 to 3.4.5, since their assumptions will not be satisfied.

For the proof of proposition 3.4.6, consider the assumptions in each of the theorems 3.4.1 to 3.4.5 on the one hand, and the function PRTWF on the other hand. We do not detail the proof in this paper, but the reader can verify that if a pair of tasks i and j satisfy at a time t these assumptions, then PRTWF(i,j,t) $\leq$ PRTWF(j,i,t) which means that task i dominates j.

Finally, we observe that the function PRTWF provides a more powerful tool than all the dominance criteria of 3.4.1 to 3.4.5 put together. These dominance criteria are, to our knowledge, the only ones in the literature for this kind of problem. Furthermore, as we will see later, the use function PRTWF takes very little CPU time for large size problems. That is why we base the weighted flow-time minimization on the function PRTWF and its properties. This will unable its use for real-time maintenance planning on large size distributed systems.

3.5. The off-line algorithm and the lower bound

3.5.1. The scheduling algorithm HPRTWF. It is worth recalling that our aim is to minimize the sum of weighted flow-times of n tasks released at different dates on a single processor or M parallel processors ($0<M<<n$). We assume that each released task is processed by only one among M available processors. Each time t a processor is available, the following steps determine the task to be scheduled.

(1) Θ = *Set of all tasks not yet scheduled.*
(2) *Calculate PRTWF(i,j,t) for all tasks i, j in* Θ.
(3) *Calculate the matrix* Ω *associated with the set* Θ.
(4) *While card(*Θ*)>1, calculate the tasks' strength in* Θ.
(5) *Redefine* Θ = *Subset of tasks in* Θ *with the greatest strength. Reduce the research to this subset. Go to step 4.* (The task with the greatest priority is selected in step 4 and step 5.)
(6) *Affect the selected task to the processor having the smallest index among the earliest available processors.*

All the above steps are repeated until the last task is scheduled.

Proposition 3.5.1. *The algorithm based on this principle has a complexity in* $O(n^3)$ *where n is the number of tasks waiting to be scheduled.*

Proof. Step 1 and step 2 are done in $O(n^2)$ and they are the most complex for the decision-making task. So each time a task is scheduled, it is done in $O(n^2)$. As this must be done till all the n tasks awaiting are scheduled, the total complexity of the algorithm is $nO(n^2)$ which comes to $O(n^3)$, thereby terminating the proof. □

3.5.2. Lower bound to the weighted flow-time.

A lower bound for the solution based on PRTWF algorithm is obtained as follow. Assume that preemption is permitted, and let us note by $\varepsilon(t, p_i)$ the remaining processing time for task i at time t. Then, the following proposition holds for the case of a single machine.

Proposition 3.5.2. *The scheduling in which at each time t, the task i scheduled is the one having the least ratio* $\varepsilon(t, p_i)/w_i$, *is a lower bound of the optimal solution.*

In other words, each time, the task to be scheduled is the one with the least index among those available tasks having this least ratio. Each time t, the task being processed is preempted when there arrives another task with a less ratio. This rule is the Shortest Weighted Remaining Processing Time rule (SWRPT).

Before coming to the experimental analysis on the static scheduling problem, we present hereafter how we applied this rule for on-line maintenance planning on large size distributed systems.

4. Solving the on-line problem of maintenance

We recall that the problem under consideration consists in assigning maintenance tasks to M maintenance processors on the Equipments of a system distributed on N sites, each one containing a number of repairable Equipments. We aim to minimize the total maintenance cost on the planning horizon H. We also aim to keep the system in good states (in terms of availability). This problem is NP-hard and there are precedence relationships linking tasks

on the same Equipment. The algorithm to solve it is based on the concepts defined in the previous section. Some additional concepts are defined in this section. The following proposition allows the use of the PRTWF function on the Equipments and solves the problem of the precedence conditions relating to a given Equipment. We skip the proof because it is in the same way as a similar theorem we used in [1] when the Equipments in the system were equally weighted.

Proposition 4.0.1. *Taking into account theorem 3.3.4 in a scheduling implies that we also respect the order of the interventions on an Equipment E_{ik}, whichever it is. In other words, for $m_1 < m_2$ and for any Equipment E_{ik}, theorem 3.3.4 allows to schedule the m_1^{th} intervention on E_{ik} before the m_2^{th} intervention.*

So, the remaining problem is to establish priorities between the different Equipments in the system. In addition to the PRTWF function and the matrix Ω, we also use the concept of tasks' urgency as defined below. The reader can also refer to [1] for the initial use of this definition.

Definition 4.0.2. Let $\Delta_{ik}(t)$ denote at time t, the duration between t and last start up date of an Equipment E_{ik} (confused to the last completion date of preventive maintenance task on the Equipment) before that time. We assume that it is the $(m\text{-}1)^{th}$ work on E_{ik}. We define *Urgent*(t) (U_t) at time t to be the set of tasks satisfying the condition $\Delta_{ik}(t) \geq \tau_{ik}$ where τ_{ik} is defined in expression 2.2. $\Delta_{ik}(t)$ is as on Figure 1.

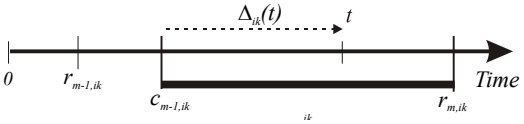


Figure 1. Definition of $\Delta_{ik}(t)$

Now, basing on the PRTWF function, the definition of the dominance matrix, the tasks' strength, their urgency and theorem 3.1.4 and proposition 4.0.1, we develop the real-time algorithm for the processors assignment to the maintenance tasks. This algorithm is called OL-MTSA-W and summarized as follow.

4.1. The on-line algorithm.

Algorithm 4.1.1. **OL-MTSA-W procedure for real-time decision making**

Any time t when a processor (repairman) is available, the tasks' subset to consider is the subset U_t. If this subset is empty, the whole set of tasks is considered. In the algorithm, each Equipment E_{ik} is characterized by its failure rate μ_{ik}, repair rate λ_{ik} and the duration τ_{ik} as in expression (2.2) basing on the availability threshold α_{ik}.

(1) *Consider the time t when a processor is available the earliest.*
If two or more processor are available at time t, select the one with the least index.
If $t < H$ then go to step 2.
Else, go to step 5.

(2) *Compute the subset U_t .*

(3) *If $card(U_t) \geq 1$ then the search set $S_t = U_t$.*
Else S_t = {all the Equipments of all the sites}.

(4) *Apply to the set S_t the steps described in the algorithm 3.5.1 basing on the PRTWF function and additional concepts for the tasks scheduling, to select the task that should be scheduled at time t.*
Determine its completion time (the next start up date of the Equipment).
Determine its next release date by adding the quantity τ_{ik} to its completion time.
Go to step 1.

(5) *End.*

The following section is devoted to numerical experimentations. It provides an evaluation of the PRTWF function relating to the lower bound in a first time, experimentation of the algorithm on several parallel processors in a second time, and finally experimental results on the on-line algorithm for maintenance planning and scheduling.

5. Numerical experimentations

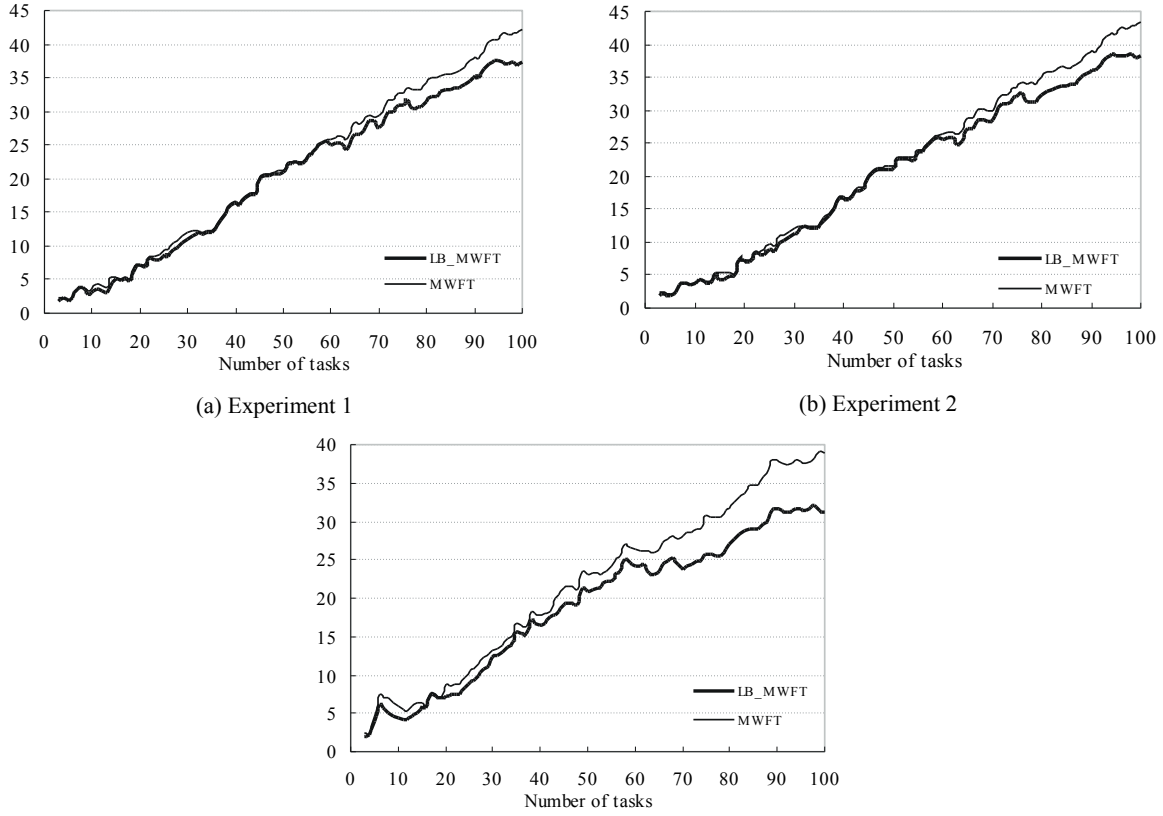
This section contains two subsections. The first one gives experimentations on static instances to minimize the weighted flow-time in presence of unequal release date. The second subsection shows experimental results on instances for real-time decision making for maintenance planning and scheduling on a given horizon. All the programs are coded in C language. They were tested on a Compaq AlphaServer ES40 DEC6600 station, operating under UNIX with 2048 MB RAM memory

5.1. Experimentations on off-line instances

Let's recall that in the static scheduling process, the lower bound and the algorithm HPRTWF were designed to schedule a given set of n tasks so as to minimize the total

weighted flow-time $\sum_{i=1}^{N} w_i (c_i - r_i)$ of tasks, where w_i, r_i and c_i are respectively the weight, the release date and the completion time of task i. In this section, we first apply the HPRTWF algorithm to schedule a set of tasks on a single processor, and compare the results to the lower bound obtained with the SWRPT-based scheduling algorithm. In this experimentation, we scheduled sets of 4 to 100 tasks on a single processor. We applied the HPRTWF algorithm to schedule sets of 100, 200, 300, 400 and 500 tasks on groups of parallel processors. Respective release dates and processing times were obtained by absolute value of normal random data with the standard deviation equal to 1, while weights are uniform random data inside [0.1,1].

The results on a single processor for the HPRTWF algorithm and SWRPT-based algorithm plotted on the figure 2 hereafter. The curves on figure 2 present three different results compared to the lower bound obtained with the SWRPT-based algorithm.



(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

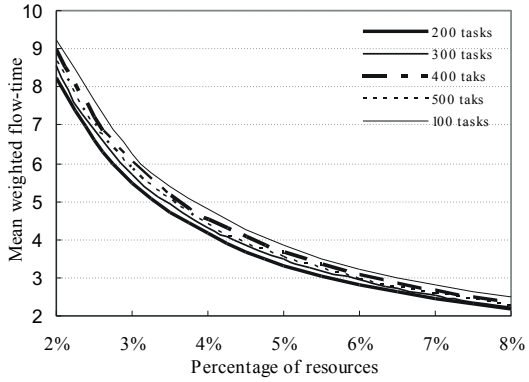
Figure 2. Experiment results and their lower bounds for three data sets

On figure 2, the mean weighted flow-time solutions obtained, respectively with the HPRTWF and the SWRPT-based algorithms correspond respectively to the MWFT (mean weighted flow-time) curves and the LB_MWTF (lower bound of the mean weighted flow-time) curves. The three numerical experiments clearly show that the results of the HPRTWF algorithm are very close to the lower bound. In these results, for more than 80% of the experiments the difference between the mean weighted flow-time and the lower bound is less than 20% of the mean weighted flow-time, and less than 8% for more than the half of the experiments. The computing times for all the results summarized in Table 1 did not exceeded 0.033 CPUs. These results confirm that the PRTWF function is a good local priority rule for weighted flow-time minimization.

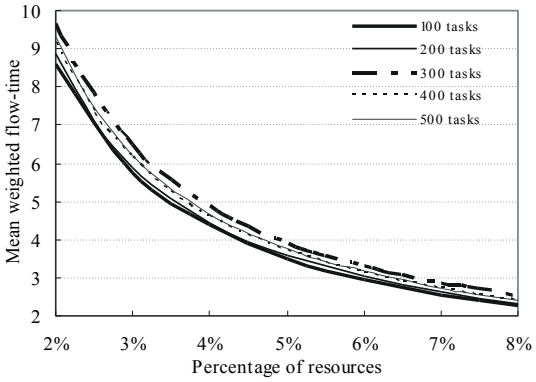
The curves on figure 3 show the results when applying the HPRTWF algorithm to several data instances containing from 100 to 500 tasks on parallel processors. The number of processors varies from 2% to 8% of the total number of tasks to be scheduled. This number is 100, 200, 300, 400 or 500 in the considered instances. This means that for 100 tasks (resp. for 500 tasks), the number of processors varies from 2 to 8 (resp. from 10 to 40). The numerical results plotted on figure 3 are obtained on the basis of three different groups of data. For each one of the three groups of experiments, a total of 500 data is generated. The program is first run on a first subset of 100 data, and then

another subset of 100 tasks is added to the first 100, and so on until the whole of the 500 data set is treated. The diagram of the three groups of experimental results highlights the robustness of the HPRTWF algorithm. It turns out that the flow-time has a very little variability when the number of processors varies in the same proportion as to the total number of tasks to be scheduled.

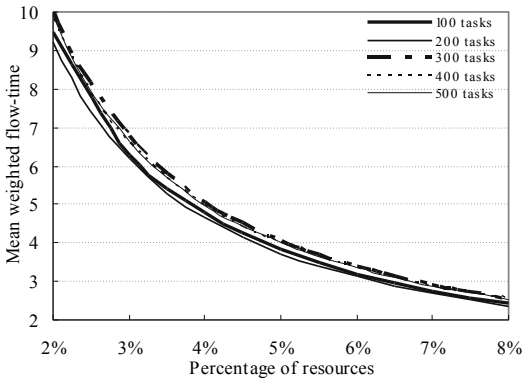
Furthermore, the CPU times for six sets of randomly generated data, including those used in the previous experiments, are summarized in table 1 below. The CPU times are expressed in seconds.



(a) MWFT on set 1



(b) MWFT on set 2



(c) MWFT on set 3

Figure 3. Mean weighted flow-time (100 to 500 tasks)

the sets	100 tasks	200 tasks	300 tasks	400 tasks	500 tasks
Set 1	0.035	0.28	1.05	2.61	5.54
Set 2	0.038	0.29	1.06	2.62	5.29
Set 3	0.031	0.29	1.06	2.59	5.22
Set 4	0.031	0.29	1.06	2.79	5.26
Set 5	0.033	0.29	1.07	2.64	5.29
Set 6	0.035	0.29	1.07	2.64	5.33

Table 1. Table of CPU processing times (in CPU seconds)

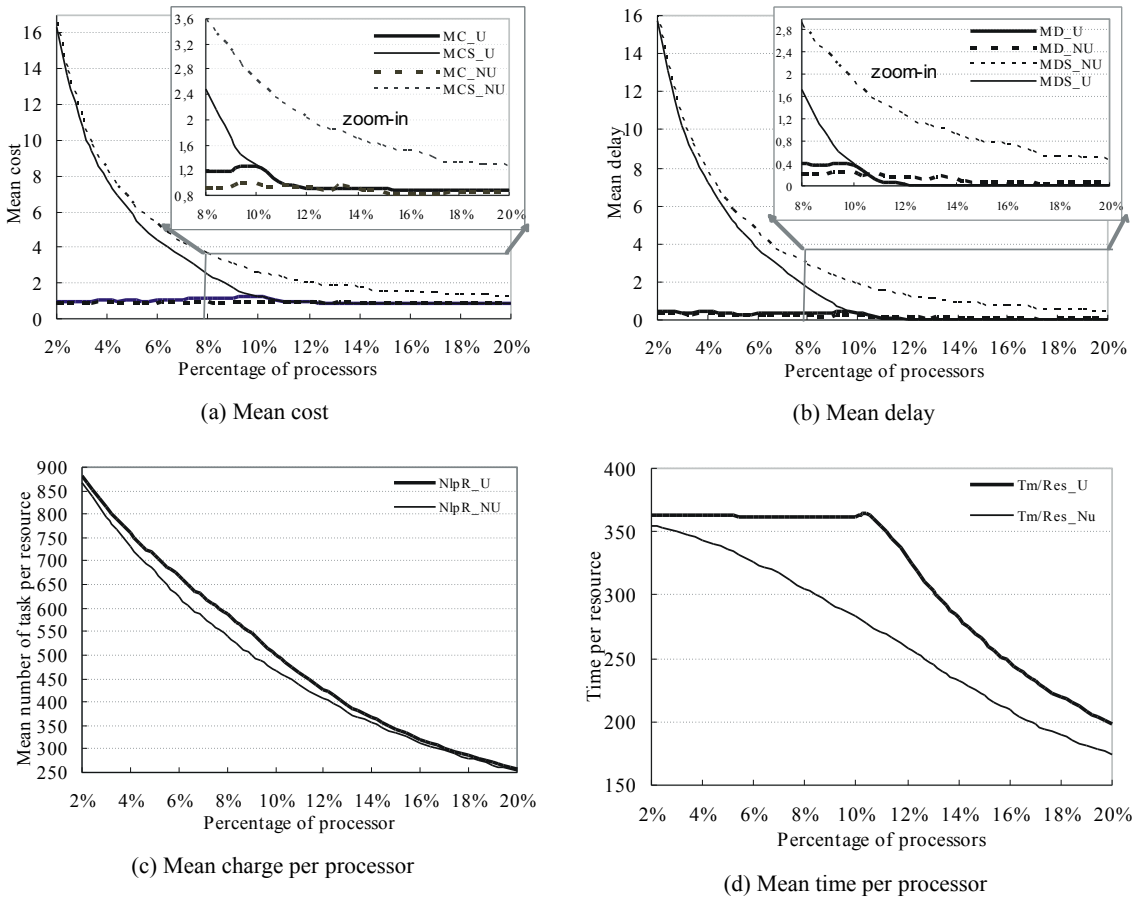
As it can be seen, the CPU processing times for the different data sets remain stable for the various numbers of tasks considered and evolve according to a polynomial curve in relation with the polynomial complexity of the algorithm. This contrasts with results obtained in [5] where, to solve the same kind of problem for 10 tasks, the algorithm proposed by the author took 7 to 26 CPU seconds.

Finally, the HPRTWF algorithm, based on the PRTWF function, is also very efficient in solving the total weighted flow-time minimization problem on identical parallel processors, in cases where the tasks are of unequal release dates.

In Section 5.2, we present some results on the on-line maintenance planning.

5.2. Experimentations on for on-line maintenance planning and scheduling

The algorithm OL-MTSA-W for on-line maintenance planning using the PRTWF rule and the additional concepts is programmed in two versions on a 365-day planning horizon. In the first version, the urgency concept is used in the program whereas in the second version, it is not used. The experimentations are carried on a system with an overall number of 500 Equipments. As the efficiency of the maintenance process depends on the number of repairmen, this number is taken according to the number of Equipments, varying from 2% to 20% of the total number of Equipments in the system (from 10 to 100 repairmen). The different data characterizing the system are randomly generated. The Equipments' failure rates (λ_{ik}) and repair rates (μ_{ik}) are generated as absolute values of normal laws. Their weights (w_{ik}) are generated uniformly in]0.1, 1[and the availability thresholds (α_{ik}) are uniformly generated in the interval]0.5, 1[in respect to the asymptotic availability $\mu_{ik}/(\lambda_{ik}+\mu_{ik})$. Some portions of the curves (a) and (b) (between 8% and 20% of the number of Equipments) on figure 4 are zoomed in order to better show their behaviour.



(a) Mean cost

(b) Mean delay

(c) Mean charge per processor

(d) Mean time per processor

Figure 4. Experimental results on 500 Equipments

Because the efficiency of the maintenance process depends on the available number of repairmen, some of the maintenance tasks are postponed until the end of the planning horizon without being able to be processed. For this reason, we always represent each parameter on the whole system on the one hand and on the effectively processed maintenance tasks on the planning horizon on the other hand. On the figures 4(a) to 4(d), the different parameters used are respectively MC (Mean Cost on the maintenance tasks effectively processed), MD (Mean Delay on the maintenance tasks effectively processed relatively to the critical dates), NIpR (the mean Number of Interventions per Repairman), Tm/Res (the mean Time a repairman is used on the planning horizon), MCS (Mean Cost on the whole System), MDS (Mean Delay on the whole System relatively to the critical dates). Those parameters have the extensions U or NU. The extension U means that the result corresponds to the program in which the urgency concept is used and the extension is NU otherwise.

Figure 4(a) presents the mean cost of maintenance (equivalent to the mean weighted flow-time of the maintenance tasks). On these curves it can be seen that the costs MCS_U and MCS_NU (mean costs on the whole system) are important with little number of repairmen, and

decreases when the number increases. The mean cost on the whole system is definitely more important when the urgency concept is not used than otherwise. This fact can be well observed on the zoom-in of figure 4(a). Moreover the mean cost on the whole system and the mean cost on the maintenance tasks effectively processed when the urgency concept is used (MCS_U and MC_U) converge from a number of repairmen corresponding to 10% of the total number of Equipments in the system (see in the zoo-in). In fact, with insufficient number of repairmen, all the maintenance tasks on all the Equipments in the system can not be processed on the planning horizon. They are then postponed several times until the end of the planning horizon. This is the reason why the mean cost on the whole system is considerably greater than the mean cost on the tasks effectively processed (for little number of repairmen). The convergence of the mean costs MC_U and MCS_U corresponds to the number of repairmen from which all the maintenance activities on the system can be completed. The convergence of the two costs is rather slow to arrive when the urgency concept is not used in the decision-making process. It can also be noticed on the zoom-in of figure 4(b) that from 10%, after the convergence of the mean cost on the system and the mean cost on the tasks effectively processed, the mean delay of the maintenance tasks from their critical date tend to 0 and is definitely null from 12% when the urgency concept is used.

Figure 4(c) represent the mean number of time each repairman is required on the planning horizon and figure 4(d) shows the mean time each processor is used on the planning horizon. It can be seen that, as the number of repairmen increases, the mean time each repairman is used decreased considerably. The difference between the mean time per processor when the urgency concept is used and not used, is important. When the urgency concept is used, the processors are used at their maximum capacity up to 10% corresponding to the sufficient number to cover the whole system. All these observations show the need to use, in addition to the local optimality rule PRTWF, the urgency concept in the decision making.

In addition, for the CPU time, we have the following observations. The smartness of the decision-making process observed in the static scheduling is replicated in the real-time decision-making process. For example for 200 Equipments with 4 repairmen, the CPU time when running the programs on a one year-horizon was 8 CPU second for 3443 tasks processed and for 16 repairmen, this CPU time was 19 CPU seconds for 9366 tasks. These CPU times are the ones when the urgency concept is used to reduce the subset of the system in which the local optimality priority rule is used. Othewise, the previous CPU time become respectively 12 CPU seconds for 4 repairmen (and 3424 interventions) and 31 CPU seconds for 16 repairmen (and 8831 interventions).

6. Conclusion

We presented in this paper an efficient algorithm for on-line tasks scheduling in a distributed system. The tasks are weighted relatively to their sensitivity or importance. We proposed a method based on a priority rule (PRTWF - Priority Rule for Total Weighted Flow-time). We proved that this rule is locally optimal for the total weighted flow-time minimization, and defined the different related concepts and spelled out the underlying properties. We also identified a lower bound which allows the rule's performance evaluation. The problem under consideration is NP-hard and existing literature proposes just a limited number of methods for solving the weighted flow-time minimization problem. The PRTWF function is of polynomial computational complexity, and this has been confirmed by the CPU computation times. This approach is adapted, with additional concepts, to real-time maintenance decision making in large size distributed systems. In extension to this work, we will propose approaches to solve the problem, first in the case of tasks with unequal release dates and set-up times, and next in the case of tasks with unequal release dates, set-up times and weights while adapting them to the maintenance decision making.

References

[1] Adzakpa, KP, Adjallah, KH, Yalaoui, F. On-line Maintenance Job Scheduling and assignment to Resources in Distributed Systems by Heuristic-Based Optimization. Journal of Intelligent Manufacturing 2004; 15(2):131-40.

[2] Azizoglu M, Kirca O. On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 1999; 133:91–100.

[3] Barnes JW, Brennan JJ. An improved algorithm for scheduling jobs on identical machines. AIIE Transactions 1977; 9:25-31

[4] Belouadeh H, Potts CN. Scheduling identical parallel machines to minimize the total weighted completion time. Discrete Applied Mathematics 1994; 48:201–18

[5] Bianco L, Ricciardelli S. Scheduling of a single machine to minimize total weighted completion time subject to release dates. Naval Research Logistics Quaterly 1982; 29:151–67.

[6] Bruno J, Coffman EG, Sethi R. Scheduling independent tasks to reduce mean finishing time. AIIE Transactions 1974; 17(7):382–7.

[7] Chu C. One-machine scheduling for minimizing total flow time with release dates. In Proceedings of Rensselaer's Second Conference on C.I.M May 1990; 570–6.

[8] Chu C. A branch-and-bound algorithm to minimize total flow time with unequal release dates. Naval Research Logistics 1992; 39:859–75.

[9] Chu C. Efficient heuristics to minimize total flow time with release dates. Operations Research Letters 1992; 12:321–30.
[10] Elmaghraby SE, Park SH. Scheduling jobs on a number of identical machines. AIIE Transactions 1974; 6(1):1–13.
[11] Framinan JM, Leisten R, Ruiz-Usano R. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimization. European Journal of Operational Research 2002; 141:559–69.
[12] Gelders LF, Sambandam N. Four simple heuristics for scheduling a flow-shop. International Journal of Production Re-search 1978; 16:221–31.
[13] Graves GH, Lee CY. Scheduling maintenance and semi resumable jobs on a single machine. Naval Research Logistics 1999; 46:845–63.
[14] Hariri AMA, Potts CN. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. Discrete Applied Mathematics 1981; 5:99–109.
[15] Lee CY, Chen ZL. Scheduling jobs and maintenance activities on parallel machines. Naval Research Logistics 2000; 47:145–65.
[15] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977; 1:343–62.
[17] Qi X, Chen T, Tu F. Scheduling the maintenance on a single machine. Journal of the Operation Research Society 1999; 50(10):1071–8.
[18] Rajendran C. Formulation and heuristics for scheduling in a kanban flow-shop to minimize the sum of weighted flowtime, weighted tardiness and weighted earliness of containers. International Journal of Production Research 1999; 37(7):1137–58.
[19] Rajendran C, Ziegler H. An efficient heuristic in a flow-shop to minimize the total weighted flowtime of jobs. European Journal of Operational Research 1997; 103:129–38.
[20] Rajendran C, Ziegler H. A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. Computers and Industrial Engineering 1997; 33(2):281–4.
[21] Rajendran C, Ziegler H. Heuristics for scheduling in flowshops and flowline-based manufacturing cells to minimize the sum of weighted flowtime and weighted tardiness of jobs. Computer and Industrial Engineering 1999; 37:671–90.
[22] Rinaldi G, Sassano A. On a job scheduling problem with different ready times: Some properties and a new algorithm to determine the optimal solution. Rapporto dell'Ist. di Automatica dell'Universita di Roma e del C.S.S.C.C.A.-C.N.R.R. 77-24. 1977.
[23] Sarin SC, Ahn S, Bishop AB. An improved branching scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime. International Journal of Production Research 1988; 26(7):1183–91.
[24] Sharadapriyadarshini B, Rajendran C. Heuristics for scheduling in kanban system with dual blocking mechanisms. European Journal of Operational Research 1997; 103:439–52.
[25] Webster S. Weighted flow time bounds for scheduling identical processors. European Journal of Operational Research 1995; 80:103–11.
[26] Webster ST. A priority rule for minimizing weighted flow time in a class of parallel machine scheduling problems. European Journal of Operational Research 1993; 70:327–34.
[27] Weinstein L, Chung CH. Integrating maintenance and production decision in a hierarchical production planning. Computer and Operations Research 1999; 26:1059–074.