# Single machine scheduling problems with release dates

Chengbin Chu, Marie-Claude Portmann

▶ **To cite this version:**

Chengbin Chu, Marie-Claude Portmann. Single machine scheduling problems with release dates. [Research Report] RR-1232, INRIA. 1990, pp.22. inria-00075326

**HAL Id: inria-00075326**
**https://inria.hal.science/inria-00075326**

Submitted on 24 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1) 39 63 55 11

Rapports de Recherche

N° 1232

*Programme 5*
*Automatique, Productique,*
*Traitement du Signal et des Données*

# SINGLE MACHINE SCHEDULING PROBLEMS WITH RELEASE DATES

**Chengbin CHU**
**Marie-Claude PORTMANN**

**Juin 1990**

# Single Machine Scheduling Problems with Release Dates

# Problèmes d'ordonnancement à une machine avec des dates d'arrivées des tâches différentes

**CHU Chengbin (*) & PORTMANN Marie-Claude (* & **)**

** Projet SAGEP, INRIA-Lorraine, CESCOM Technopôle Metz 2000*
*4, rue Marconi, 57070 Metz*
*** Ecole des Mines de NANCY, Parc de Saurupt, 54042 NANCY*

Abstract:
The single machine scheduling problems have been extensively studied with various criteria to be optimized and under various assumptions. In this work, we review some results obtained recently in the case of different release dates. Most problems with different release dates are NP-hard.
Some researchers have proved some dominance properties or sufficient conditions for local optimality which lead to an optimal schedule in some specific cases. We present some properties or conditions for two regular criteria, total tardiness and total flow time.

Keywords:
Scheduling, One-machine, Flow Time, Total Tardiness, Release Dates, Priority Rules, Dominant Subset, Branch and Bound.

Résumé:
Les problèmes d'ordonnancement à une machine ont été intensément étudiés avec différents critères à optimiser et sous des hypothèses très variées. Dans ce travail, nous passons en revue quelques résultats nouveaux pour le cas où les tâches ont différentes dates d'arrivée au plus tôt. La plupart de ces problèmes sont NP-difficiles.
Des chercheurs ont démontré des propriétés de dominance et des conditions suffisantes d'optimalité locale, qui conduisent à un ordonnancement optimal dans des cas particuliers. Nous présentons quelques propriétés et conditions pour deux critères réguliers: la somme des retards et la somme des temps de présence.

Mots clés:
Ordonnancement, une machine, somme des temps de présence, somme des retards, dates d'arrivée, règles de priorité, sous-ensembles dominants, procédure par séparation et évaluation.

# 1. Introduction

The single machine scheduling problem has been extensively studied under various assumptions and various objective functions. In [17], the authors reviewed various results related to this problem. In this paper, we consider single machine scheduling problems with different release dates for two regular criteria: total tardiness and total flow time. In both cases, we assume that no preemption is allowed. Both problems are NP-hard. For each of them, we present dominance properties, some found by previous authors and others proved by ourselves, we show how to use these dominance properties as rules to build heuristics, we build or improve branch and bound exact methods, and we end by presenting some numerical experiments on both heuristic and exact methods. Section 2 will be devoted to total tardiness and Section 3 to total flow time.

# 2. Minimizing total tardiness with release dates

We considered the $n/1/r_i/\sum T_i$ scheduling problem, where $n$ is the number of jobs, $r_i$ is the release date of job i, $T_i$ is its tardiness in a schedule. A beginning time $t_i$ or a completion time $C_i$ for each job i must be found to minimize:

$$\sum_{i=1}^{n} T_i = \sum_{i=1}^{n} \max(C_i - d_i, 0) = \sum_{i=1}^{n} \max(t_i + p_i - d_i, 0)$$

subject to:

$$\begin{cases} \forall i,\ t_i \geq r_i \\ \forall i,j,\ i \neq j,\ [t_i, C_i) \cap [t_j, C_j) = \emptyset \end{cases}$$

where $p_i$ is the processing time of job i and $d_i$ is its due date.

This problem is known to be NP-hard if the job release dates are not identical ([29]). When the jobs have *simultaneous arrivals*, but do not have identical processing times, it was recently proved to be NP-hard too ([14]).

In the latter particular case, some exact or heuristic algorithms have been proposed in the literature:

— Wilkerson & Irwin ([33]) constructed a polynomial heuristic algorithm using a priority rule for local optimality. Baker & Bertrand ([2]) investigated the MDD (Modified Due Date) rule proposed by Smith ([31]). Lawler ([20]) modified his earlier dynamic programming algorithm proposed in [19] to obtain a fully polynomial approximation scheme.

— The elimination criteria proposed by Emmons ([15]) lead to a reduction of the problem size. These theorems are widely used in optimal enumerative algorithms. Srinivasan ([32]) developed a three-phase hybrid algorithm able to solve problems with up to 50 jobs. Using Lagrangean relaxation to obtain a tight lower bound, Fisher ([16]) proposed a branch and bound method. Sen et al. ([30]) established some precedence relationships amongst jobs and used them for an algorithm tested on problems with up to 100 jobs. Lawler ([19]) proposed a "pseudopolynomial" dynamic programming algorithm. This algorithm was improved by Potts & Van Wassenhove ([26]) by using a decomposition technique. Problems with up to 100 jobs were solved. In [28], Potts & Van Wassenhove presented different algorithms for this problem.

—The very particular case when the jobs have all the same processing time is polynomial and the EDD rule (Earliest Due Date) gives an optimal schedule.

—For the problems involving non equal weights but with always identical release dates, Mahon & Rachamadugu ([22]) proved a dominance rule. Using Lagrangean relaxation with subproblems that are total weighted completion time problems, Potts & Van Wassenhove ([27]) obtained a lower bound which is used in a branch and bound algorithm.

Unfortunately, *with non identical release dates,* these results are no more applicable and we found no result in international journals. We proved a sufficient condition for local optimality for $n/1/r_i/\sum T_i$ scheduling problem. We defined a new dominant subset of schedules based on this condition and proposed efficient heuristic algorithms to produce a schedule belonging to this subset ([9]). We also proposed a lower bound in order to build an exact branch and bound method ([6]).

Section 2.1. is devoted to the presentation of our recent results. Section 2.2. is devoted to heuristic and exact algorithms and section 2.3. presents some numerical results.

## 2.1. Recent results

We do not find any results in the literature about the $n/1/r_i/\sum T_i$ scheduling problem except for complexity results. However, an heuristic algorithm proposed by Baker & Bertrand ([2]) for the $n/1/r_i=0/\sum T_i$ scheduling problem and called MDD (Modified Due Dates) can easily be extended to the $n/1/r_i/\sum T_i$ scheduling problem, using the non delay generator defined by Baker in ([1]). In ([2]), Baker & Bertrand used the dynamic priority rule proposed by Smith ([31]) to choose a job i to add to a partial schedule: one which minimizes $\max(\Delta+p_i,d_i)$, where $\Delta$ is the completion time of the

last scheduled job. If we only consider at time $\Delta$ the jobs such that $r_i \leq \Delta$, then we obtain an heuristic algorithm which we will call NDPRTT (for Non Delay algorithm using Priority Rule for Total Tardiness as we will define later).

In this way, with non identical release dates, Smith's rule could be applied to build non delay schedules, however it becomes inefficient to build active schedules. Our contribution in [9] was to propose a generalization of Smith's rule efficient for active schedules.

This rule is a function called PRTT and defined as follows.

***Priority Rule for Total Tardiness***

We define the function PRTT(i,$\Delta$) of job i and time $\Delta$ such as:

$PRTT(i,\Delta)=\max(r_i,\Delta)+\max[\max(r_i,\Delta)+p_i,d_i]$

PRTT(i,$\Delta$) is the sum of a "dynamic" *earliest beginning time* ($\max(r_i,\Delta)$) and a "dynamic" *modified due date* ($\max[\max(r_i,\Delta)+p_i,d_i]$). This is a non decreasing piecewise linear function.

Let (u,v) be the partial schedule obtained by processing job u immediately before job v on a machine available at time $\Delta$, and $T_{uv}$ be the corresponding sum of $T_u$ and $T_v$. Based on the function defined above, a sufficient condition for local optimality is given by the following theorem.

***Sufficient condition for Total Tardiness local optimality:***

Consider two jobs i and j to be processed on a machine available from the moment $\Delta$. If $PRTT(i,\Delta) \leq PRTT(j,\Delta)$ then $T_{ij} \leq T_{ji}$.

The proof can be found in ([9]).

In order to present a converse form of the above theorem, we introduce the notion of "corrected due date" (1) by the following definition.

***Corrected due date:***

Let $d_i^c$ be the "corrected due date " of job i defined by:

$d_i^c=\max(r_i+p_i,d_i)$

The following assertion is quite obvious:

---

1 "Corrected due date" is a particular case of "Modified Due Date" ($\Delta$=0). It is a static parameter, independent of time.

«The minimization of total tardiness with original due dates is equivalent to minimization with corrected due dates.»

Therefore only corrected due dates should be used for the $n/1/r_i/\sum T_i$ scheduling problem.

Using the corrected due dates, the following theorem holds:

***Restrictive reverse form***
If for some pair (i, j), $i \neq j$, i and j being adjacent jobs,
we have PRTT (i, $\Delta$) > PRTT (j, $\Delta$), then $T_{ij} > T_{ji}$ or $T_{ij} = T_{ji} = 0$.

Remark:
If the jobs arrive simultaneously, the sufficient condition for local optimality to process i before j from the moment $\Delta$ becomes:
$\max(\Delta+p_i,d_i) \leq \max(\Delta+p_j,d_j)$
This is the same condition as the one derived by Baker & Bertrand ([2]) from ([31]).

In order to define a dominant subset (i.e. a subset which contains at least one optimal schedule), we use the notion of active schedule proposed by Baker ([1]) and Conway et al. ([11]). No global left-shift can be made in an active schedule, a global left-shift being the possibility of beginning some operation earlier without delaying any other operation.

***T-active schedules:***
For any pair of adjacent jobs i and j (i followed by j) in an active schedule S, if i and j verify at least one of the two following conditions

1. $\max(r_i,\Delta) < \max(r_j,\Delta)$
2. $PRTT(i,\Delta) \leq PRTT(j,\Delta)$

then we say that S is T-active,

where $\Delta = \begin{cases} C_k \text{ if k precedes immediately i} \\ -\infty \text{ if i is the first job in the sequence} \end{cases}$

The sufficient condition for local optimality enabled us to obtain the following result:

***Dominance of T-active schedule subset:***
The subset of T-active schedules is dominant for total tardiness criterion.

The proof can be found in ([9]).

## 2.2. Heuristic and exact algorithms

### 2.2.1. Heuristic algorithms

We used three types of algorithms. The first one was based on general generators proposed by Baker ([1]) and Conway et al. ([11]). We proposed a second scheme with insertion phases, and a third scheme with alternative choice.

Baker proposed in [1] a generator of non delay schedules (denoted by ND) and a generator of active schedules (denoted by ACT). Non delay schedules are included in active schedules. The active schedule subset is dominant for any regular criterion (see [1]), and therefore for the total tardiness criterion (it contains the T-active schedules which also composes a dominant subset). But this is false for non delay schedules.

We constructed a lot of heuristic algorithms by combining ND and ACT schedule generators with PRTT and other classical rules such as SPT or SLACK: see ([10]) for comparative numerical experiments. We select here only one of the best heuristic algorithm which builds active schedules using an insertion step and one non delay algorithm NDPRTT.

*Non delay (ND) generator*

In a non delay schedule no machine is kept idle at a time when it could begin processing some operation. Using PRTT rule in a non delay generator, we obtain the NDPRTT heuristic mentioned above. In order to improve the heuristic, we use SPT as a secondary rule to break ties.

*Insertion algorithm*

In this algorithm called IPRTT, at each step, we definitively schedule amongst unscheduled jobs a job l with the smallest PRTT function value, the ties being broken by ECT (Earliest Completion Time). We then try to see whether there are some jobs that can be scheduled in the idle time before l, if there are, we always schedule from amongst unscheduled jobs the job with the smallest release date ties broken by PRTT, until there is no job left that could be scheduled in the idle time.

We proved in [9] that, after scheduling the job with the smallest PRTT value, the jobs left for insertion would not be late.

## 2.2.2. An exact algorithm

We proposed a detailed paper on the construction and the efficiency of a specific branch and bound algorithm ([6]). In this paper, we sum up the main results. To build a branch and bound algorithm for this problem, we need some more dominance properties and a tight lower bound.

In this section, we use the following notations:

N: the set of all the jobs to be scheduled;

P: a given partial schedule;

$f(P)$: the completion time of the last job of a given partial schedule P;

P | i: a partial schedule obtained by scheduling job i immediately after the given partial schedule P;

$\Sigma(P,i)$: the complete schedule obtained by optimally scheduling the jobs belonging to $N-(P\cup\{i\})$ behind P followed by i;

C[s,S]: the completion time of the job finished at the s-th position in the schedule S;

$F(i,\Delta)$: the earliest completion time of job i at time $\Delta$ ($F(i,\Delta)=\max(\Delta,r_i)+p_i$);

$T^*(\Pi)$: the optimal total tardiness of a problem $\Pi$.

In the branch and bound method, at each step, we split a set of schedules beginning with a partial sequence P into a series of schedule sets beginning with a partial sequence P | i for each unscheduled job i. It is very important to eliminate all P | i which can not lead to an optimal schedule.

***Dominance property for Total Tardiness when adding a job to a partial sequence:***
Given a partial schedule P and two jobs $i,j\in N-P$ (i and j are both unscheduled jobs), if i and j verify the following conditions
$p_i\geq p_j$ and $F(i,f(P))\leq F(j,f(P))$ and $d_i\leq d_j$
then we have: $T(\Sigma(P,i))\leq T(\Sigma(P,j))$.

The proof is quite obvious.

If we look for only one optimal solution, we can eliminate the schedule set which begins with P | j.

We proposed in ([6]) a lower bound for the $n/1/r_i/\Sigma T_i$ which will be used to compute a lower bound for the end of the sequence in the branch and bound method.

***Lower bound for Total Tardiness***

Given a problem $\Pi$, we construct a schedule S in which the jobs are scheduled according to SRPT (Shortest Remaining Processing Time), by relaxing the problem under the assumption that the jobs are preemptive. If $(d'_1, d'_2, ..., d'_n)$ is the series obtained by sorting the series $(d_1, d_2, ..., d_n)$ in non decreasing order, the following relation holds

$$\sum_{i=1}^{n} \max(C[i,s]-d'_i,0) \leq T^*(\Pi)$$

The complexity of this lower bound computation is polynomial (at most $O(n^2)$).

## *The branch and bound algorithm*

The algorithm consists of three phases: initialization, branching and termination.

Initialization involves defining the initial value of the variables, constructing an initial sequence and calculating an initial upper bound by applying some of the heuristic algorithms presented above.

Branching is a step by step procedure. It develops a search tree, in which a node represents a partial sequence of scheduled jobs. A descendent of a node is thus the partial sequence obtained by scheduling an unscheduled job after the partial sequence corresponding to the node.

At each step, the procedure (1) identifies the current node (the one with the least lower bound, breaking ties by choosing the one having the maximum number of jobs scheduled in the partial schedule); (2) generates the descendents of the current node; (3) eliminates dominated nodes; (4) computes lower bounds of non eliminated nodes and updates the file of active nodes. (A non current node is dominated if its lower bound is not strictly smaller than the current lowest upper bound that was the best solution reached so far).

The termination phase consists in identifying the optimal sequence and computing the criterion value.

We summarize the elimination tests when creating P | i:
Test 1 ≡ "elimination of P | i when it can not lead to an active schedule"
Test 2 ≡ "elimination of P | i when it can not lead to a T-active schedule"
Test 3 ≡ "elimination of P | i when it is dominated by some P | j"

The other eliminations are carried out when a better feasible schedule is found, as is usually done in all branch and bound methods.

## 2.3. Numerical results

### 2.3.1. Instance generation

We present here two series of experiments. In the first one, we took $n$=20 in order to be able to always obtain an optimal schedule in all cases. In the second one, we took $n$=200, it is impossible in this case to obtain an optimal schedule.

Considering that the relative performances of the heuristic algorithms may be related to the instance nature (great or small scattering of the release dates and of the slacks), each series consisted of 12 samples of 20 instances. A sample was defined by a range of release dates BR and a range of slack BSLK. Release dates were uniformally generated between 0 and BR and the absolute slacks (for each job i, the absolute slack is $d_i$-$p_i$-$r_i$) were uniformally generated between 0 and BSLK.

In the first series ($n$=20) we generated the processing times uniformally between 1 and 100. In the second one ($n$=200), the processing times were uniformally generated between 1 and 10. In this way, the average sum of the processing times was almost identical for the two series. So we could compare the series from the point of view of identical parameters BR and BSLK; in the first one, the individual processing times of each instance were more scattered.

### 2.3.2. Results

The results are given by tables.

Table 2.1 contains the average mean tardiness (total tardiness divided by $n$) corresponding to each of the 12 samples for the heuristic and exact algorithms. The first column called «G» contains the two generator parameters BR:BSLK. The following columns give the average mean tardiness for algorithms IPRTT, NDPRTT, UPRTT (=best result of IPRTT and NDPRTT) and the optimal value obtained with a branch and bound method (for $n$=20). We can see that NDPRTT is on the average generally better than IPRTT, but as IPRTT is better for some examples of each sample, using both NDPRTT and IPRTT is an improvement which costs little as compared to the exact method and which becomes more interesting when BR increases, i.e. when the release dates are more scattered. For n=20, the average of the optimal values is 87.52, NDPRTT gives 88.58 (relative average error: 1.21%) , IPRTT gives 90.61 (relative average error: 3.53%), but UPRTT gives 88.07 (relative

gives 88.07 (relative average error: 0,63%). For $n$=200, the improvement of UPRTT upon NDPRTT is less significant, probably because the processing times are less scattered (upper value 10 instead of 100) and because of the great number of jobs (some explanations could be derived from the formulas given in the conclusion for a particular case). Other experiments show that all algorithms which build T-active schedules give performances similar with those of IPRTT and that algorithms using other rules (as for example SPT and SLACK) are significantly less efficient.

Table 2.1 ($n$=20)

| «G» | IPRTT | NDPRTT | UPRTT | OPT |
|---|---|---|---|---|
| 0:50 | 299.405 | | | 299.358 |
| 0:250 | 235.265 | | | 234.598 |
| 0:500 | 123.620 | | | 122.630 |
| 500:50 | 153.915 | 149.872 | 149.372 | 148.980 |
| 500:250 | 105.265 | 97.990 | 97.980 | 96.207 |
| 500:500 | 56.225 | 45.510 | 45.470 | 44.123 |
| 1000:50 | 56.287 | 54.327 | 53.682 | 53.030 |
| 1000:250 | 27.265 | 25.350 | 24.340 | 24.078 |
| 1000:500 | 4.230 | 3.423 | 2.898 | 2.652 |
| 1500:50 | 20.103 | 21.172 | 19.983 | 19.880 |
| 1500:250 | 4.340 | 5.560 | 3.930 | 3.880 |
| 1500:500 | 1.405 | 1.510 | 0.900 | 0.795 |
| | 90.6104 | 88.5837 | 88.0704 | 87.5176 |

Table 2.1 ($n$=200)

| «G» | IPRTT | NDPRTT | UPRTT |
|---|---|---|---|
| 0:50 | 358.210 | | |
| 0:250 | 257.150 | | |
| 0:500 | 186.301 | | |
| 500:50 | 169.938 | 168.089 | 168.088 |
| 500:250 | 113.610 | 111.766 | 111.766 |
| 500:500 | 52.024 | 50.995 | 50.994 |
| 1000:50 | 27.019 | 24.954 | 24.954 |
| 1000:250 | 1.762 | 1.016 | 1.009 |
| 1000:500 | 0.001 | 0.024 | 0.001 |
| 1500:50 | 1.485 | 1.161 | 1.146 |
| 1500:250 | 0.002 | 0.032 | 0.002 |
| 1500:500 | 0.000 | 0.000 | 0.000 |
| | 97.2918 | 96.6415 | 96.6351 |

Table 2.2 shows the average computation times for the PRTT algorithms and for the branch and bound method. An optimal solution is considerably much more time-consuming and provides a criterion value that is not significantly better.

Table 2.2 Computation time

| | PRTT | OPT |
|---|---|---|
| $n$=20 | 41.35 | 49609.4 |
| $n$=200 | 100.62 | ∞ |

## 2.4. Conclusion

We have defined a new dominant subset of schedules. This allows us to search for an optimal solution in a smaller solution space. The proposed heuristic algorithms efficiently construct "good" schedules belonging to this subset.

The NDPRTT algorithm is one of the best. This would not be surprising if we could generalize the two results presented below and proved in ([24]) which concern the particular case where all the processing time are equal to an unique value D, but where the $r_i$ are not all multiples of D. (The complexity of this particular case remains open while it is polynomial when D divides all of the $r_i$.)

***Upper bound of the difference between the Total Tardiness of two active schedules***
For the $n/1/r_i;p_i=D/\sum T_i$ scheduling problem,
If T(O) is the total tardiness of an active schedule O for a problem $\Pi$ and if T* is the optimal value of the total tardiness for the problem $\Pi$, then we have:

$$T(O) - T^* \leq \frac{n^2}{4} \cdot (2 \cdot D - 1)$$

and there exist problem instances for which this upper bound is reached.

***Upper bound of the difference between the Total Tardiness of two active schedules when one is a non delay U-active ([1]) schedule***
For the $n/1/r_i;p_i=D/\sum T_i$ scheduling problem,
If T(O) is the total tardiness of a non delay U-active schedule O for a given problem and if T* is the optimal value of the total tardiness for the same problem, then we have:

$$T(O) - T^* \leq (n - 1) \cdot (2 \cdot D - 1)$$

and there exist problem instances for which this upper bound is reached.

This result remains true for non delay T-active schedules (when $\forall i$, $p_i=D$), the problem instance for which this upper bound is reached is the same as for non delay U-active schedules.

# 3. Minimizing total flow time with release dates

In this section, we consider the $n/1/r_i/\sum F_i$ scheduling problem, where $n$ is the number of jobs, $r_i$ is the release date of job i, $F_i$ is its flow time in a schedule. A beginning time $t_i$ or a completion time $C_i$ for each job i must be found to minimize:

$$\text{Min} \sum_{i=1}^{n} F_i = \text{Min} \sum_{i=1}^{n} (C_i - r_i)$$

subject to

[1] U-active schedules are another family of active schedules defined in ([24]): a U-active schedule is an active schedule in which for each couple of jobs (i,j), if i precedes j in the schedule then we have ($d_i \leq d_j$) or ($C_i - p_i < r_j$).

$\begin{cases} \forall i,\ C_i - p_i \geq r_i \\ \forall i, \forall j, \text{ and } i \neq j,\ [C_i - p_i,\ C_i) \cap [C_j - p_j,\ C_j) = \emptyset \end{cases}$

where $p_i$ is the processing time of job i.

This problem is equivalent to the problems of minimizing the total job lateness, total completion time, total waiting time, mean number of jobs in the shop ([11]).

In the case of identical release dates, this problem can easily be solved by applying SPT (Shortest Processing Time) priority rule ([31]). However, it is NP-hard with different release dates ([21], [29]). Several researchers have proved some dominance properties in different cases and presented some dominant subsets. Chandra ([5]) discussed two models. The first one is the model discussed in this section, the second one is with preemptive-repeat (an operation can be preempted, but the work done before the preemption is lost. If it is processed later, the processing requires a time equal to its initial processing time). Dessouky & Deogun ([13]) have proved several dominance properties for the problem discussed in this section. Deogun ([12]) has proposed a partitioning method using these dominance theorems. For the total weighted completion time criterion, Bianco & Ricciardelli ([3]) have proved several dominance properties, Hariri & Potts ([18]) proposed an approach using a relaxation method to obtain a lower bound in a branch and bound enumerative algorithm. Posner ([25]) considered the problem with clustered jobs. Chand & Schneeberger ([4]) considered the problem with maximum allowable tardiness constraints.

In ([7]), we proved a sufficient and necessary condition for local optimality in the same way as for the Total Tardiness criterion, but it is not only a sufficient but also a necessary condition. This condition can also be considered as a priority rule. Based on this condition, we defined a subset, called F-active subset, containing all the optimal schedules. We proved that any schedule in this subset also verifies some dominance properties proven by Dessouky & Deogun ([13]). We also proposed some heuristic algorithms to build a F-active schedule, and we compared their performance with some other previous heuristic algorithms.

In Section 3.1., we summarize the theoretical results. In section 3.2., we present previous and new heuristics and some considerations on branch and bound methods. The numerical results are given in Section 3.3.

## 3.1. Previous and recent results

We follow the same approach as for Total Tardiness: a priority rule which can be used, not only for non delay schedules, but also for active schedules. This rule enabled us to build a sufficient and necessary condition of local optimality and furthermore a dominant subset called F-active. This new notion can be used to improve both heuristic and exact algorithms.

The new rule is a function called PRTF (Priority Rule for Total Flow time) and defined as follows.

***Priority Rule for Total Flow Time***
We define the function PRTF(i,Δ) of a job i at the time Δ as:
$PRTF(i,\Delta)=2\max(\Delta,r_i)+p_i$

PRTF(i,Δ) is the sum of a "dynamic" *earliest beginning time* ($\max(r_i,\Delta)=R(i,\Delta)$) and a "dynamic" *earliest completion time* ($\max(r_i,\Delta)+p_i=C(i,\Delta)$). This is a non decreasing piecewise linear function.

Let (u,v) be the partial schedule obtained by processing job u immediately before job v on a machine available at time Δ, and $F_{uv}$ be the corresponding sum of $F_u$ and $F_v$. Based on the function defined above, a sufficient and necessary condition for local optimality is given by the following theorem.

***Sufficient and necessary condition for Total Flow Time local optimality:***
Consider two jobs i and j to be processed on a machine available from the moment Δ, we have:
$PRTF(i,\Delta)\leq PRTF(j,\Delta) \Leftrightarrow F_{ij}\leq F_{ji}$.

The proof can be found in ([7]).

This sufficient and necessary condition enabled us to build a dominant subset to minimize $\Sigma F_i$ (i.e. the total flow time).

***F-active schedules:***
For any pair of adjacent jobs i and j (i followed by j) in an active schedule S, if i and j verify at least one of the two following conditions

1. $\max(r_i,\Delta)<\max(r_j,\Delta)$
2. $PRTF(i,\Delta)\leq PRTF(j,\Delta)$

then we say that S is F-active,

where $\Delta=\begin{cases} C_k \text{ if k precedes immediately i} \\ -\infty \text{ if i is the first job in the sequence} \end{cases}$

The sufficient and necessary condition for local optimality gives immediately the following result:

***Dominance of F-active schedule subset:***
The subset of F-active schedules is dominant for total flow time criterion and all the optimal schedules of a $n/1/r_i\geq 0/\sum F_i$ scheduling problem are F-active.

Remarks:

1° The function PRTF(i,Δ) is in fact a combination of rules FIFO (First In First Out) and SPT (Shortest Processing Time).
2° If $\forall i$, $r_i=0$, the sufficient and necessary condition becomes $p_i\leq p_j$ (SPT rule)
3° If $\forall i$, $p_i=D$, D being a constant, PRTF(i,Δ)≤PRTF(j,Δ) becomes
$\max(\Delta,r_i)\leq\max(\Delta,r_j)$ (a dynamic FIFO rule which leads in this particular case to an optimal solution).
4° If $\forall i$, $d_i=r_i$, then $d_i<r_i+p_i\leq C_i$ and the total tardiness is

$$\sum_{i=1}^{n}\max(C_i-d_i,0)=\sum_{i=1}^{n}(C_i-d_i)=\sum_{i=1}^{n}(C_i-r_i)=\sum_{i=1}^{n}F_i$$

In this case, minimizing total tardiness is equivalent to minimizing total flow time. We have $d_i<\max(\Delta,r_i)+p_i$ which leads to PRTT(i,Δ)=2max(Δ,$r_i$)+$p_i$=PRTF(i,Δ), it is a coherent result that the two rules are equivalent when the problems are equivalent.
We also established in ([7]) some properties of F-active schedules. First, any F-active schedule verifies some theorems proven by Dessouky & Deogun ([13]).

***"Dessouky" dominance property 1:***
Given a partial sequence P (which ends at time Δ) and two jobs i, j ∈ N-P,
if $p_i\leq p_k$ $\forall k\in$ N-P and $\max(\Delta,r_i)\leq\max(\Delta,r_j)$ then Σ(P,i) dominates Σ(P,j).

***"Dessouky" dominance property 1 is always roughly ([1])verified in the set of F-active schedules***
For any job x, if A is the set of jobs following x in a F-active schedule S and $\Delta_x$ is the end of the sequence which precedes x, then there is no job y∈A of S such that:
$\forall i\in A\cup\{x\}-\{y\}$, $p_y<p_i$ and $\max(\Delta_x,r_x)\geq\max(\Delta_x,r_y)$

[1] We have here $p_y<p_i$ instead of $p_y\leq p_i$, but for the case where $p_y=p_i$, jobs y and i are equivalent and one of them must be put first.

The proof can be found in ([7]).

***"Dessouky" dominance property 2:***
Given a partial sequence P (which ends at time $\Delta$) and two jobs i, j $\in$ N-P,
if $C(i,\Delta) \leq C(k,\Delta)$ $\forall k \in$ N-P and $r_j \geq C(i,\Delta)$ then $\Sigma(P,j)$ is strictly dominated.

***"Dessouky" dominance property 2 is always verified in the set of F-active schedules***
For any job x, if A is the set of jobs following x in an optimal schedule S, then there is no job y$\in$A of S such that:
$r_x \geq \max(\Delta_x, r_y) + p_y$

The proof can be found in ([7]).

Then, any F-active schedule verifies both dominance properties proposed by Dessouky & Deogun ([13]), furthermore, we have a rule to decide which job to schedule first when only two jobs are to be scheduled.

## 3.2. Heuristic and exact algorithms

### 3.2.1. Heuristic algorithms

According to the result of the previous section, we must only consider F-active schedules in order to reach an optimal solution. We present here some heuristic algorithms proposed by previous researchers and some proposed by ourselves in ([7]) using F-active schedules.

In [13], authors have examined the behaviors of the ECT (Earliest Completion Time) priority rule. In [5], Chandra compared EST rule [1] with an optimal solution. He concluded that EST gives a solution close to the optimum.

We built some new heuristic algorithms using PRTF called PRTF and APRTF . To evaluate their performance, we compared them with the ECT and EST rules.

All the algorithms are "greedy" algorithms, they definitively schedule an unscheduled job behind a partial schedule. At each step, the machine is available at time $\Delta$, and the set of unscheduled jobs is denoted by A. $\Delta$ and A are respectively initialized to $-\infty$ and {1, 2, ..., n}.

[1] In ([5]), Chandra called this SPT rule. However, we prefer to use the abbreviation EST used in ([13]).

*ECT algorithm (Dessouky & Deogun[13])*
At each step, select a job i with $\min_{i\in A}(C(i,\Delta))$. Break ties by choosing i with $\min_{i\in A}(R(i,\Delta))$, and end if necessary by breaking ties choosing i with $\min_{i\in A}(i)$. Schedule i. Update $\Delta$ and A, such that $\Delta$:=C(i,$\Delta$), A:=A-{i}.

*EST algorithm (CHANDRA [5])*
At each step, select a job i with $\min_{i\in A}(R(i,\Delta))$. Break ties by choosing i with $\min_{i\in A}(p_i)$, and end if necessary by breaking ties choosing i with $\min_{i\in A}(i)$. Schedule i. Update $\Delta$ and A, such that $\Delta$:=C(i,$\Delta$), A:=A-{i}.

*Algorithm PRTF ([7])*
At each step, select job i with $\min_{i\in A}(PRTF(i,\Delta))$. Break ties by choosing i with $\min_{i\in A}(R(i,\Delta))$, and end if necessary by breaking ties choosing i with $\min_{i\in A}(i)$. Schedule i. Update $\Delta$ and A, such that $\Delta$:=C(i,$\Delta$), A:=A-{i}.

*Algorithm APRTF ([7])*
At each step:
1: Select job $\alpha$ with $\min_{\alpha\in A}(PRTF(\alpha,\Delta))$. Break ties by choosing $\alpha$ with $\min_{\alpha\in A}(R(\alpha,\Delta))$, and end if necessary by breaking ties choosing $\alpha$ with $\min_{\alpha\in A}(\alpha)$.
2: Select job $\beta$ with $\min_{\beta\in A}(R(\beta,\Delta))$. Break ties by choosing $\beta$ with $\min_{\beta\in A}(p_\beta)$, and end if necessary by breaking ties choosing $\beta$ with $\min_{\beta\in A}(\beta)$.
3: **If** $r_\alpha \le R(\beta,\Delta)$ **then** x=$\alpha$. **Goto** 5.
4: Denoting card(A-{$\alpha$,$\beta$}) by $\mu$, **if** there exists a job $\gamma\in$ A-{$\alpha$,$\beta$} such that $\forall i\in$ A-{$\alpha$,$\beta$}, $r_\gamma \le r_i$, **and if** $F_{\beta\alpha}-F_{\alpha\beta}<\mu\cdot\min[R(\alpha,\Delta)-R(\beta,\Delta),C(\beta,C(\alpha,\Delta))-r_\gamma]$, **then** x=$\beta$, **otherwise**, x=$\alpha$.
5: Schedule x. $\Delta$=C(x,$\Delta$), A=A-{x}.

Comment:

In fact, at step 4, the algorithm chooses a job to be scheduled by comparing the two values which can be considered as gain and loss. (If we scheduled $\beta$ at time $\Delta$, there would not be less idle time on the machine and it could be better for the end of the sequence, but with $\alpha$ scheduled at that time, we respect the priority rule PRTF.)

### 3.2.2. Exact algorithm

There are some efficient branch and bound methods proposed to solve this problem up to 50 jobs ([5], [12], [13]) and even for the more general scheduling problem $n/1/r_i/\sum w_iF_i$ ([3], [18]). In ([8]) we proposed some specific improvements for the $n/1/r_i/\sum F_i$ scheduling problem and we obtained an optimal solution for problems with up to 70 jobs.

## 3.3. Numerical results

### 3.3.1. Generation of instances

We randomly generated 6 samples of examples with respectively 20, 30, 40, 50, 60, 70 jobs. The examples were generated in the same way as in [18], The processing times were generated between 1 and 100. The release dates were generated between 0 and $50.5 \cdot n \cdot \alpha$. 10 values of $\alpha$ (0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.50, 1.75, 2.0, 3.0) permitted us to generate 10 examples for each sample.

### 3.3.2. Results

In this section, we give some numerical results in order to compare performance of the algorithms. For this purpose, we also construct a branch and bound algorithm as proposed by Dessouky & Deogun ([13]) and improved by using only F-active schedules. The initial solution is given by the best solution obtained by applying all the heuristic algorithms.

Table 3.1 Mean criterion value for each sample

| n | 20 | 30 | 40 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|
| APRTF | 3995.2 | 6529.0 | 10131.8 | 17048.1 | 21394.2 | 25505.7 |
| PRTF | 4042.8 | 6613.2 | 10229.8 | 17439.7 | 22152.2 | 26366.2 |
| EST | 4006.3 | 6543.0 | 10141.3 | 17056.7 | 21412.1 | 25535.0 |
| ECT | 4419.5 | 7317.6 | 11586.6 | 19293.8 | 25252.5 | 29749.4 |
| UPRTF | 3995.2 | 6521.4 | 10094.9 | 17042.7 | 21383.8 | 25495.1 |
| UET | 4006.3 | 6536.1 | 10141.3 | 17056.7 | 21412.1 | 25535.0 |
| OPT | 3992.2 | 6475.4 | 10047.7 | 16966.6 | 21346.1 | 25385.7 |

UPRTF: = best result given by APRTF and PRTF (like UPRTT)

UET: = best result given by EST and ECT

Table 3.1 shows the mean criterion value given by the heuristics presented above and by the branch and bound method for each sample. The last line of the table corresponds to to the optimal value. From this table, we see that, on the average, the best solution found by algorithms ARPTF and PRTF is close to the optimum. The average deviation is never greater than 0.71%. We also see that for each sample, the best solution found by algorithms APRTF and PRTF (namely, the solution found by UPRTF) is better than the best one found by EST and ECT (namely the one found by UET). The difference between the solution given by UPRTF and an optimal one is smaller than one half of that between the solution given by UET and an optimal one.

Table 3.2 mean computation time (in CPU seconds)

| n | 20 | 30 | 40 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|
| APRTF | 0.0244 | 0.0561 | 0.1261 | 0.1692 | 0.2476 | 0.3325 |
| PRTF | 0.0144 | 0.0361 | 0.0643 | 0.0894 | 0.1211 | 0.1727 |
| EST | 0.0112 | 0.0327 | 0.0509 | 0.0728 | 0.1093 | 0.1627 |
| ECT | 0.0194 | 0.0378 | 0.0676 | 0.0925 | 0.1360 | 0.1844 |
| UPRTF | 0.0388 | 0.0922 | 0.1904 | 0.2586 | 0.3687 | 0.5052 |
| UET | 0.0306 | 0.0705 | 0.1185 | 0.1653 | 0.2453 | 0.3471 |
| OPT | 0.326 | 2.001 | 14.73 | 39.08 | 290.9 | 363.9 |

Table 3.2 shows the mean computation time on a SUN 3/110 consumed by the heuristics and by the branch and bound method for each sample. We see that the algorithm APRTF is slightly more time-consuming. However, for each problem instance, APRTF is always better than UET. PRTF is sometimes worse than UET, but it is interesting to keep it as it sometimes provides a better solution than the one given by APRTF. Consequently, using UPRTF or simply APRTF seems to be the best way if one wants to obtain a solution close to an optimal one.

### 3.4. Conclusion

We defined a new dominant subset of schedules for the total flow time criterion. This allowed us to look for an optimal solution in a smaller solution space. This improved the branch and bound approach. The heuristic algorithms using PRTF construct "good" schedules belonging to this subset, better than previously published heuristic algorithms.

# 4. General conclusion

Studying the dynamic local optimality of a couple of adjacent jobs permitted us — for two criteria, total tardiness and total flow time — to find new dominant subsets in which finding the optimum is possible and to propose new efficient heuristic algorithms using two new rules linked to the dominant subset. We already extended these rules to build heuristic algorithms for flow shops and job shops. We now examine if some results can be extended to m identical parallel machines.

## REFERENCES


[1] **K.R. BAKER**
Introduction to Sequencing and Scheduling
John & Wiley, 1974

[2] **K.R. BAKER, J.W.M. BERTRAND**
*"A Dynamic Priority Rule for Scheduling Against Due-Dates"*
Journal of Operations Management, Vol. 3, n° 1, pp. 37-42, November 1982

[3] **L. BIANCO, S. RICCIARDELLI**
*"Scheduling of a Single Machine to Minimize Total Weighted Completion Time subject to Release Dates"*
Nav. Res. Logist. Quart. Vol. 29, n° 1, pp. 151-167, 1982

[4] **S. CHAND, H. SCHNEEBERGER**
*"A Note on the Single Machine Scheduling Problem with Minimum Weighted Completion Time and Maximum Allowable Tardiness"*
Naval. Res. Logist. Quart. vol. 33, pp. 551-557, 1986

[5] **R. CHANDRA**
*"On $n/1/\bar{F}$ Dynamic Deterministic Systems"*
Nav. Res. Logist. Quart. vol. 26, pp. 537-544, 1979

[6] **C. CHU**
*"A Branch and Bound Algorithm for Minimizing Total Tardiness with Different Release Dates"*
Proposed for publication in Nav. Res. Logist. Quart.

[7] **C. CHU**
*"One-machine Scheduling Problem for Minimizing Total Flow Time with Release Dates"*
Second International Conference on C.I.M., Troy, NY, 21-23 may 1990

[8] **C. CHU**
*"A Branch and Bound Algorithm for Minimizing Total Flow Time with Different Release Dates"*
Proposed for publication in Nav. Res. Logist. Quart.

[9] **C. CHU, M.C. PORTMANN**
*"Minimisation de la somme des retards pour les problèmes d'ordonnancement à une machine"*
Rapport de recherche n° 1023, INRIA, Rocquencourt, France, avril 1989

[10] **C. CHU, M.C. PORTMANN**
*"New Approximate Methods for Solving $n/1/r_i/\Sigma T_i$ Scheduling Problem"*
Proposed for publication in Europ. J. Oper. Res.

[11] **R. W. CONWAY, W. L. MAXWELL, L.W. MILLER**
Theory of scheduling
Addison-Wesley, 1967

[12] **J.S. DEOGUN**
*"On Scheduling with Ready Times to Minimize Mean Flow Time"*
The Computer Journal, Vol. 26, n° 4, pp. 320-328, 1983

[13] **M.I. DESSOUKY, J.S. DEOGUN**
*"Sequencing Jobs with Unequal Ready Times to Minimize Mean Flow Time"*
SIAM. J. Comput. Vol. 10, n° 1, pp. 192-202, February 1981

[14] **J. DU, J. Y.-T. LEUNG**
*"Minimizing total tardiness on one machine is NP-hard"*
Math. Oper. Res., to appear

[15] **H. EMMONS**
*"One-machine Sequencing to Minimize Certain Functions of Job Tardiness"*
Opns. Res., 17 (1969)w 701-715

[16] **M.L. FISHER**
"A dual Algorithm for the One-Machine Scheduling Problem"
Math. Prog. 11 (1976) 229-251

[17] **S.K. GUPTA, J. KYPARISIS**
*"Single Machine Scheduling Research"*
OMEGA Int. J. Mgmt. Sci. vol. 15, n° 3, pp. 207-227, 1987

[18] **A.M.A. HARIRI, C.N. POTTS**
*"An Algorithm for Single Machine Sequencing with Release Dates to Minimize Total Weighted Completion Time"*
Disc. Appl. Math. n° 5, pp. 99-109, 1983

[19] **E.L. LAWLER**
*"A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness"*
Ann. Discr. Math. 1 (1977) 331-342

[20] **E.L. LAWLER**
*"A fully polynomial approximation scheme for the total tardiness problem"*
Operations Research Letters 1 (1982) 207-208

[21] **J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER**
*"Complexity of Machine Scheduling Problems"*
Ann. Discrete. Math. n° 4, pp. 281-300, 1977

[22] **R. MOHAN, V. RACHAMADUGU**
*"A Note on the Weighted Tardiness Problem"*
Opns. Res. 35 (1987) 450-452

[23] **S. S. PANWALKER, W. ISKANDER**
*"A Survey of Scheduling Rules"*
Opns. Res. vol. 25, n° 1, pp. 45-60, January-February 1977

[24] **M.C. PORTMANN**
*"Méthodes de décomposition spatiale et temporelle en ordonnancement de la production"*
Thèse d'état, Université de Nancy 1, 19 sept. 1987

[25] **M.E. POSNER**
*"A Sequencing Problem with Release Dates and Clustered Jobs"*
Man. Sci. Vol. 32, n° 6, pp. 731-738 June 1986

[26] **C.N. POTTS, L.N. VAN WASSENHOVE**
*"A Decomposition Algorithm for the Single Machine Total Tardiness Problem"*
Operations Research Letters, Vol. 1, n° 5, pp. 177-181, 1982

[27] **C.N. POTTS, L.N. VAN WASSENHOVE**
*"A Branch and Bound Algorithm for the Total Weighted Tardiness Problem"*
Opns. Res. 33 (1985) 363-377

[28] **C.N. POTTS, L.N. VAN WASSENHOVE**
*"Dynamic Programming and Decomposition Approaches for the Single Machine Total Tardiness Problem"*

Europ. J. Opns. Res., 32 (1987) 405-414

[29] **A.H.G. RINNOOY KAN**
*"Machine Sequencing Problems: Classification, Complexity and Computation"*
Nijhoff, The Hague, 1976

[30] **T. T. SEN, L. M. AUSTIN, P. GHANDFOROUSH**
*"An Algorithm for the Single-Machine Sequencing Problem to Minimize Total Tardiness"*
IIE Transactions, Vol. 15, n° 4, pp. 363-366, 1983

[31] **W.E. SMITH**
*"Various Optimizers for Single-stage Production"*
Nav. Res. Log. Quart., Vol. 3, pp. 59-66, 1956

[32] **V. SRINIVASAN**
*"A Hybrid Algorithm for the One-machine Sequencing Problem to Minimize Total Tardiness"*
Nav. Res. Log. Quart., Vol. 18, n° 3, pp. 317-327, 1971

[33] **L. J. WILKERSON, J. D. IRWIN**
*"An Improved Algorithm for Scheduling Independent Tasks"*
AIIE Transactions 3 (1971), 239-245

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399