Scheduling Jobs with Releases Dates and Delivery Times on M Identical Non-idling Machines

Fatma Hermès[1] and Khaled Ghédira[2]
[1]High Institute of Computer Sciences (ISI), Tunis El Manar University, Ariana, Tunisia
[2]High Institute of Management (ISG), Tunis University, Tunis, Tunisia

Keywords: Scheduling, Identical Parallel Machines, Non-idling Constraint, Release Dates, Delivery Times, Makespan.

Abstract: This paper considers the problem of scheduling jobs with release dates and delivery times on m identical machines where the machines must work under the non-idling constraint. Indeed, each machine must process all the jobs affected to it continuously without any intermediate delays. The objective is to minimize the makespan. This problem is strongly NP-hard since its particular case on only one machine has been proved to be strongly NP-hard (Chrétienne, 2008). Furthermore, the complexity of the considered problem where the jobs are unit-time remains an open question (Chrétienne, 2014). Recently, the particular case on only one non-idling machine has been studied and some efficient classical algorithms proposed to solve the classic one machine scheduling problem (i.e without adding the non-idling constraint) have been easily extended to solve its non-idling version (see (Chrétienne, 2008), (Carlier et al., 2010) and (Kacem and kellerer, 2014)). In this paper, we propose some heuristics to solve the considered m machines problem under the non-idling constraint. We first suggest a generalization of the well known rule of Jackson (Jackson, 1955) in order to construct feasible schedules. This rule gives priority to the ready jobs with the greatest delivery time. Then, we extend Potts algorithm (Potts, 1980) which has been proposed to solve the one machine problem. Finally, we present the results of a computational study which shows that the proposed heuristics are fast and yields in most tests schedules with relative deviation which is on average equal to 0,4%.

1 INTRODUCTION

Most scheduling problems have neglected the cost incurred by machines idle times. Indeed, such waiting delays are often necessary to get optimality and making a machine wait for a more urgent job is a key feature to solve great number of problems (see for example (Simons, 1983)). However, in various scheduling environments such as those described in (Landis, 1983), the machine set up is relatively high and the cost incurred by machine idle times is often considerable. For example, if the machine is an oven that must heat different pieces of work at a given high temperature, clearly, keeping the required temperature of the oven while the machine is empty may be too costly. In this paper, we consider the problem of scheduling a set I of n jobs on m identical non-idling machines $(n > m \geq 2)$. Each job i $(1 \leq i \leq n)$ has to be processed for p_i units of time by one machine out of the set of machines and has a release date (or head) r_i before which it cannot be started. The job i has also a delivery time (or tail) q_i that must elapse between its completion on the machine and its exit from the system. The job i is completed after spending p_i time on one machine and then q_i time in the system (i.e. not on machine). Giving, a feasible schedule σ, let $C_i(\sigma)$ denotes the completion time of the job i. Thus, we have $C_i(\sigma) = t_i(\sigma) + p_i + q_i$ where $t_i(\sigma)$ is the starting time of the job i in the scheduling order of σ. All data are assumed to be deterministic and integer and all machines are ready from time zero onwards. The machines must work under the non-idling constraint which means that each machine k $(1 \leq k \leq m)$ must process all the jobs affected to it continuously without any idle time. The makespan of the schedule σ is then calculated as follows:

$$C_{max}(\sigma) = \max_{1 \leq i \leq n}(t_i(\sigma) + p_i + q_i) \qquad (1)$$

The schedule σ is said to be feasible if the following conditions are satisfied:

- We have $t_i(\sigma) \geq r_i$ for all $i = 1, \dots, n$.

Hermès, F. and Ghédira, K.
Scheduling Jobs with Releases Dates and Delivery Times on M Identical Non-idling Machines.
DOI: 10.5220/0006428100820091
In *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2017) - Volume 1*, pages 82-91
ISBN: 978-989-758-263-9

- Each machine must process at most one job at one time and no job is processed by more than one machine.
- There is no idle time between two consecutive jobs on the same machine. If the job i_1 precedes immediately the job i_2 on machine k, then, we must have, $t_{i_1}(\sigma) + p_{i_1} = t_{i_2}(\sigma)$.

The rest of the paper is organized as follows: In section 2, we survey the state of the art. In section 3, we discuss some dominant sets of solutions (i.e. a set which must contain at least one optimal solution). We also expose some conditions for obtaining non-idling dominant schedules. In section 4, we present the main results obtained in the literature for the particular case of one non-idling machine scheduling problem and then the case of identical non-idling machines. We finally propose a first heuristic which constructs a good feasible schedule for the studied problem using Jackson's rule. Then, we propose a second one in order to improve the obtained feasible schedules. In section 5, we present the lower bounds used to evaluate the proposed heuristics. In section 6, we present an evaluation of computational tests and we conclude in section 7.

2 THE STATE OF THE ART

In the 3-field notation $\alpha|\beta|\gamma$, the non-idling constraint is represented in (Chrétienne, 2008) by the notation NI associated with the machine field α. Thus, the considered problem is denoted $P, NI|r_i, q_i|C_{max}$. As mentioned in (Carlier,1987), the problems $P, NI|r_i, q_i|C_{max}$ and $P, NI|r_i|L_{max}$ (i.e. minimizing the maximum lateness on identical parallel machines) are equivalent. It is enough to set $q_i = D - d_i$ for all $i \in I$, where $D = \max_{i \in I} d_i$. In the equivalent form $P, NI|r_i|L_{max}$, Jackson's rule schedules the available job with the smallest due date instead of scheduling the job with the largest delivery time.

The problem $P, NI|r_i, q_i|C_{max}$ is NP-hard in the strong sense since it is a generalization of the one machine scheduling problem $1, NI|r_i, q_i|C_{max}$ which has been proved to be strongly NP-hard in (Chrétienne, 2008). It is also an extension of the problem $P|r_i, q_i|C_{max}$ which is also strongly NP-hard. We note that $P||C_{max}$ and $P, NI||C_{max}$ are equivalent since the set of dominant schedules (i.e. the earliest ones) for the problem $P||C_{max}$ are non-idling and therefore, the problem $P, NI||C_{max}$ is strongly NP-hard and then $P, NI|r_i, q_i|C_{max}$ is also NP-hard. However, Carlier deduced in (Carlier, 1987) that when all data are integers and the processing times are unit (or equal), the classic problem $P|p_i = 1, r_i, q_i|C_{\max}$ is solved in polynomial time using Jackson's rule (Jackson, 1955). Otherwise, the deviation of Jackson's schedule from the optimum is smaller than twice the largest processing time. Also, the preemptive version $P|r_i, q_i, pmtn|C_{\max}$ is solvable in polynomial time using a network flow formulation (Horn, 1974) and gives a tight lower bound for the classic problem $P|r_i, q_i|C_{\max}$. With adding the non-idling constraint, the complexity of the problem $P, NI|p_i = 1, r_i, q_i|C_{\max}$ remains unknown (see Chrétienne, 2014). Also, the preemptive problem $P, NI|r_i, q_i, pmtn|C_{\max}$ is not yet studied and its complexity is thus unknown.

The non-idling machine constraint has just begun to receive research attention in the literature and there are few papers dealing with such problems. To the best of our knowledge, the first works on such problems concern the earliness-tardiness one machine scheduling problem with no unforced idle time, where a Branch and Bound approach has been developed in (Valente and Alves, 2005). Recently, some aspects of the impact of the non-idling constraint on the complexity of the one machine scheduling problems as well as the important role played by the earliest starting time of a non-idling schedule has been studied in (Chrétienne, 2008). Moreover, a branch and bound method has been designed to solve the problem $1, NI|r_i, q_i|C_{max}$ in (Carlier *et al.,* 2010). In a recent paper (Kacem and Kellerer, 2014), the authors developed approximation algorithms for the same problem $1, NI|r_i, q_i|C_{max}$ with extending some classic results. Another exact method has been presented in (Jouglet, 2012) where the author defined some necessary and/or sufficient conditions for obtaining non-idling dominant sets of schedules (i.e. a dominant set is a set containing at least one optimal schedule). He also described a constraint programming approach for solving exactly the one non-idling machine scheduling problem with release dates and optimizing a regular criterion (i.e. an objective function which is nondecreasing with respect to all completion times of jobs). We note that the makespan is a regular criterion.

In the case of parallel non-idling machines, the first work considering the non-idling constraint are, to the best of our knowledge, those of (Quilliot and Chrétienne, 2013) where the authors introduced the Homogeneously Non-Idling (HNI in short) constraint. A schedule satisfies the HNI constraint if, for any subset M' of machines, the time slots at

which at least one machine of this subset is active make an interval. They studied the problem where weakly dependent unit-time jobs have to be scheduled within the time windows between their release dates and due dates. They also introduced the notion of pyramidal structure and provided a structural necessary and sufficient condition for an instance of the problem to be feasible. Later, Chrétienne gave in (Chrétienne, 2014) an overview of the main results obtained on the complexity of scheduling under the non-idling constraint for non-idling one machine scheduling problems and some cases of non-idling parallel machines scheduling problems.

3 DOMINANCE RULES

In parallel scheduling, a schedule σ is represented by a permutation $\sigma = (\bar{t}_1, \bar{t}_2, \dots, \bar{t}_n)$ where $\bar{t}_1 \leq \bar{t}_2 \leq \cdots \leq \bar{t}_n$. In this permutation $\bar{t}_i$ denotes the starting time of the i^{th} job which is scheduled on the first available machine. A schedule σ can also be seen as a set of sub-schedules $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$ where σ_k is the sequence of jobs scheduled on machine k and I_k the subset of jobs affected to machine k. Thus, we have:

$$C_{max}(\sigma) = \max_{1 \leq k \leq m}\big(C_{max}(\sigma_k)\big) \quad (2)$$

where

$$C_{max}(\sigma_k) = \max_{i \in I_k}(t_i + p_i + q_i) \quad (3)$$

and

$$\bigcup_{k=1}^{m} I_k = I \quad (4)$$

A dominant set of solutions (i.e. schedules) is a set in which there is at least one optimal solution. In this section we discuss the set of dominant solution with adding the non-idling constraint.

3.1 The Non-idling Semi-active Schedule

A non-idling semi-active schedule for the one machine scheduling problem is defined in (Jouglet, 2012) as a feasible schedule where no job can be scheduled earlier without either changing the sequence of execution of jobs or violating a model constraint including the non-idling constraint. The set of semi-active is dominant for a regular criterion which means that there exists at least one optimal schedule which is semi-active.

In our context, (i.e. identical parallel machines), the definition of a non-idling semi-active schedule can be extended as follows.

Definition:
A non-idling semi-active schedule for the problem $P, NI|r_i, q_i|C_{max}$ is a feasible schedule where, on each machine, no job can be scheduled earlier without either changing the sequence of execution of jobs or violating a model constraint including the non-idling constraint. In other words, each sub-schedule σ_k on machine k must be a non-idling semi-active sub-schedule of the correspondent sub-problem on the set I_k.

The following theorem gives a necessary and sufficient condition for a non-idling schedule to be semi-active for a non-idling identical parallel machines scheduling problem optimizing a regular criterion.

Theorem 1:
A non-idling schedule σ for the problem $P, NI|r_i, q_i|C_{max}$ is semi-active if, and only if, on each machine k, there is at least one job which starts at its release date, i.e.

$$\min_{i \in I_k}(t_i(\sigma) - r_i) = 0 \quad (5)$$

where I_k is the set of jobs scheduled on machine k.

Proof:
If $\min_{i \in I_k}(t_i(\sigma) - r_i) > 0$ then on machine k we can start earlier with $\delta = \min_{i \in I_k}(t_i(\sigma) - r_i)$. In this case the considered schedule will be not semi active. In other word, if no job starts at its release date in a non-idling schedule, then on each machine, the jobs can be scheduled earlier without changing the sequence of execution of jobs.

Without loss of generality, we suppose that the release dates are arranged as follows $\bar{r}_1 \leq \bar{r}_2 \leq \cdots \leq \bar{r}_n$ where $\bar{r}_i$ present the $i^{t\square}$ greatest release time. This date isn't necessarily the release time of job i. An upper bound of the earliest starting time on machines in a non-idling semi-active schedule is provided in the following corollary.

Corollary 1:
The latest starts times for a non-idling semi-active schedule on machines $1, \dots, m$ are respectively $\bar{r}_{n-m+1}, , \dots \bar{r}_{n-1}, \bar{r}_n$ and these bounds are tight.

Proof:
Given the scheduling technique used on parallel machines, the earliest machine on the set of machines is the machine 1 and latest one is the

machine m. The latest machine m cannot start strictly after $\bar{r}_n$. In fact, if a schedule σ starts on machine m strictly after $\bar{r}_n = \max_{i \in N} r_i$, then no job can start at its release date on this machine and then σ is not semi-active. So by induction, the machine $m-1$ cannot start strictly after $\bar{r}_{n-1}$, the machine $m-2$ cannot start strictly after $\bar{r}_{n-2}$ and so on until the machine 1 which cannot start strictly after $\bar{r}_{n-m+1}$. Moreover, any semi-active schedule whose first job on machine m has the latest release date starts at $\bar{r}_n = \max_{i \in I} r_i$ and the first job on machine $m-1$ has the second latest release date starts at $\bar{r}_{n-1}$ and so on until machine 1 whose first job starts at $\bar{r}_{n-m+1}$. If all other jobs are scheduled as soon as possible after these first jobs, then whatever the sequence the obtained schedule is non-idling semi-active.

In the same way, a lower bound of the earliest starting time on machines in a non-idling semi-active schedule is provided in the following corollary.

Corollary 2:
The earliest starts times for a non-idling semi-active schedule on machines $1, \ldots, m$ are respectively $r_1, r_2, \ldots, r_m$ and these bounds are tight.

Proof:
If the machine starts at the time $\bar{r}_1$ which is the smallest start time then the second machine cannot start before $\bar{r}_2$ and so on until the machine m which cannot start before $\bar{r}_m$.

It is well known that the subset of semi-active schedules is dominant (i.e. there exists at least one optimal non-idling schedule which is semi-active) for problems with a regular criterion. In the same way, the following proposition can therefore be obviously derived.

Proposition 1:
The set of non-idling semi-active schedules is dominant for non-idling problems where a regular criterion is to be minimized.

Proof:
In a giving non-idling semi-active schedule, the jobs are scheduled as early as possible on each machine.

3.2 The Non-idling Active Schedule

A non-idling active for the problem $1, NI|ri, qi|Cmax$ is defined in (Jouglet, 2012) as a feasible schedule where no job can be completed earlier without either delaying another job or violating a model constraint (including the non-idling constraint). There is an obvious relation between non-idling semi-active schedules and non-idling active schedules.

Proposition 2:
A non-idling active schedule is a non-idling semi-active schedule.

Proof:
Consider a schedule σ which is not semi-active. A job may therefore be scheduled earlier without changing the sequence of$$ jobs and without violating the non-idling constraint. Consequently, σ is not active.

It is well known, that the subset of active schedules is dominant (i.e. there is at least one optimal schedule which is non-idling active) for problems with a regular criterion and the following propositions may therefore be derived:

Proposition 3:
The set of non-idling active schedules is dominant for the problem $P, NI|r_i, q_i|C_{max}$.

Proof:
Consider a non-idling schedule σ which minimizes a regular objective function. If σ is not active, then there exists a job i which can be scheduled earlier without violating a model constraint and without delaying another job. This yields a non-idling schedule σ with a cost not greater than that of σ. The process is repeated until no such a job i exists. Thus an optimal non-idling active schedule has been obtained.

The following theorem provides a necessary condition for a schedule σ to be a non-idling active schedule.

Theorem 2:
If σ is a non-idling active schedule then $\forall i \in I_{k_1}$, at least one of the two following equations is true:

$$\min_{\{i_1 \in I_{k_1} : i_1 \neq i / t_{i_1}(\sigma) > t_i(\sigma)\}} t_{i_1}(\sigma) - r_{i_1} < p_i \qquad (6)$$

$$r_i = \max(r_i, \underline{C} - \sum_{l=1}^{n} p_l) \qquad (7)$$

Proof:
Consider a non-idling active schedule σ where Conditions (6) and (7) are not satisfied for a job *i*. Since the schedule is active, it is also semi-active, and a job cannot be scheduled earlier simply by scheduling the entire schedule earlier. We suppose that the job i is scheduled on machine k_1. In the first

case, we have $k_1 = k_2$. In this case, the proof is similar to that in Jouglet (2012).

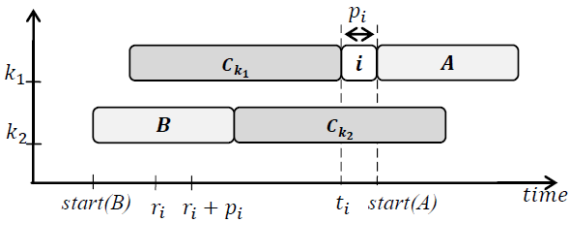

Figure 1: A non-idling non-active schedule.

In the second case, we have $k_1 \neq k_2$. Let $A = \{j \in I_{k_1}/t_j(\sigma) > t_i(\sigma)\}$ be the set of jobs which are scheduled after the job i on the same machine k_1. Suppose it exist a machine k_2 such that $B = \{j \in I_{k_2}/t_j(S) < r_i + p_i\}$ be the set of jobs starting before $r_i + p_i$, and let $C_{k_2} = \{j \in I_{k_2}/r_i + p_i \leq t_j(\sigma) < t_i(\sigma)\}$ be the set of jobs which are scheduled between jobs belonging to B and job i (see Figure 1). If Condition 4 is not satisfied, this means that $\min_{j \in A}(t_j(\sigma) - r_j) \geq p_i$. Thus, job i can be removed from the schedule and the jobs belonging to A can be scheduled p_i units of time earlier since they are scheduled at least p_i units of time from their release dates. The non-idling constraint is thus restored (see figure 2). Indeed, the jobs belonging to the set B can be also scheduled p_i units of time earlier since $min_{\{j \in B\}}(t_j(\sigma) - r_j) \geq p_i$. Job i can then be inserted just after B without delaying subsequent jobs, giving us a non-idling schedule σ in which job i has been scheduled earlier without delaying any other job. This contradicts the fact that σ is active. Note that the starting times of jobs belonging to C_{k_1} and C_{k_2} do not change during the move of i. Note also that if C_{k_1} or C_{k_2} is empty, it means that the sub-schedules σ_{k_1} or σ_{k_2} can be simply brought forward from at least p_i units of time contradicting the fact that σ is semi-active and then active.

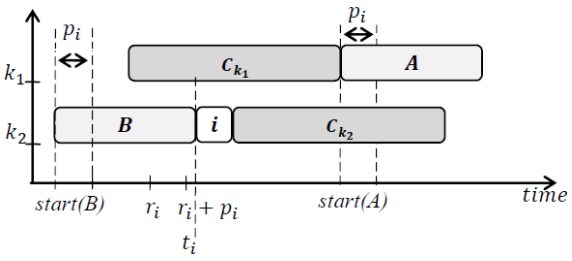

Figure 2: The schedule obtained after inserting the job i earlier on machine k_2 without delaying the other jobs.

The theorem 2 relies on the fact that if the condition is not satisfied for a schedule σ, a job i may be inserted earlier without delaying the other jobs.

4 THE PROPOSED HEURISTICS

The problem $1, NI|r_i, q_i|C_{max}$ is a particular case of the problem $P, NI|r_i, q_i|C_{max}$. In this section, we present the main results obtained in the literature for the problem $1, NI|r_i, q_i|C_{max}$. Then, we show in a first heuristic how we can obtain efficient feasible schedules for the problem $P, NI|r_i, q_i|C_{max}$. Finally, we construct a second heuristic by extending an efficient classic algorithm proposed by Potts (Potts, 1980) to solve the problem $1|r_i, q_i|C_{max}$.

4.1 General Points

Some classic algorithms, proposed to solve the problem $1|r_i, q_i|C_{max}$, has been easily extended to solve its non-idling version $1, NI|r_i, q_i|C_{max}$ with keeping the same efficiency (see (Chrétienne, 2008) and (Kacem and kellerer, 2014)). According to (Carlier *et al.*, 2010), to solve the problem $1, NI|r_i, q_i|C_{max}$, we must first adjust the release dates of all jobs by increasing some ones, then we apply Jackson's rule for the modified instance. More accurately, we first apply Jackson's rule to the problem instance. Let $\underline{C}$ denotes the machine ending time of the schedule obtained after the first application of Jackson's rule. A new instance is generated after transforming the release dates of each job $i (i = 1, \ldots, n)$ as follows:

$$r_i = \max(r_i, \underline{C} - \sum_{l=1}^{n} p_l) \quad (8)$$

If we apply again Jackson's rule for the new instance, the obtained schedule is non-idling and thus it is a feasible schedule for the problem $1, NI|r_i, q_i|C_{max}$ (Carlier et al., 2010). This solution is far from the optimal schedule with a distance smaller than $\max_{1 \leq i \leq n} p_i$ (Chrétienne, 2008) and has a tight worst-case performance ratio of 2 (Kacem and kellerer, 2014). To improve the performance of Jackson's algorithm for the relaxed problem $1|r_i, q_i|C_{max}$, Potts proposed in (Potts, 1980) to run Jackson's algorithm at most n times to some modified instances. Potts's algorithm starts with Jackson's sequence. Let $\sigma = (1,2,\ldots,n)$ be the Jackson's schedule where we suppose that the jobs are reindexed according to this order and let B be the

critical block (i.e. the set of jobs in the critical path). The job c which attains the maximum completion time in Jackson's schedule is called the critical job. It is the last job in the sequence of B. The maximum completion time of σ is defined as follows:

$$C_{max}(\sigma) = \min_{i\in B} r_i + \sum_{i\in B} p_i + q_c \qquad (9)$$

We suppose that a is the first job in the block B. thus, there is no-idle time between the processing of a to c. The sequence of jobs, $a, a+1, \dots, c$ in the critical block B, represents the critical path. An interference job b is a job in the critical path having a delivery time smaller than c ($q_b < q_c$). If there is an interference job b, then Pot's algorithm forced it to be scheduled after the critical job c in the next iteration by setting $r_b = r_c$. Kacem and kellerer (Kacem and kellerer, 2013) extended Potts's algorithm (Potts, 1980) and proved that it has a tight worst-case performance ratio of $\frac{3}{2}$. The non-idling version of Potts's algorithm solving the problem $1, NI|r_i, q_i|C_{max}$ is described as follows. It can be executed in $O(n^2 \log n)$ operations.

We Note also that Chrétienne proved in (Chrétienne, 2008) that when the preemption of jobs is allowed or when the jobs are unit-time, the obtained schedules are respectively optimal for the problems $1, NI|pmtn, r_i, q_i|C_{max}$ and $1, NI|p_i = 1, r_i, q_i|C_{max}$.

NIPotts algorithm for the one machine problem $1, NI|r_i, q_i|C_{max}$
Begin
  Step 1.
  Initialization: $h = 0$; $I = (r_i, p_i, q_i)_{1\le i\le n}$
  Step 2.
    2.1 Apply Jackson's rule to the instance I ;
    2.2 Update the current instance I by applying for each job i $(i = 1, \dots, n)$: $r_i = \max(r_i, \underline{C} - \sum_{i=1}^{n} p_i)$.
    2.3 Apply Jackson's rule to I and store the obtained schedule σ^h
    2.4 Set $h = h + 1$;
  Step 3.
    3.1 If $h = n$ or if there is no interference job in σ^{h-1}, then stop and return the best generated schedule among $\sigma^0, \sigma^1, \dots, \sigma^{h-1}$.
    Otherwise, identify the interference job b and the critical job c in σ^{h-1}.
    3.2 Set $r_b = r_c$ and go to step 2.
End NIPotts

4.2 The Non-idling M Machines Problem

Jackson's rule is also used to solve the problem $P|r_i, q_i|C_{max}$ but there is no extension for its non-idling version $P, NI|r_i, q_i|C_{max}$. Carlier proved in (Carlier, 1987) that a schedule constructed with applying Jackson's rule for the problem $P|r_i, q_i|C_{max}$ is far from the optimal schedule with a distance smaller than $2(\max_{1\le i\le n} p_i - 1)$. Gusfield (Gusfield, 1984) proved that this solution is far from the optimal schedule with a distance smaller than $\frac{2m-1}{m}\max_{1\le i\le n} p_i$. Thus Gharbi and Haouari deduced in (Gharbi and Haouari, 2007) that the constructed solution must be far from the optimal schedule with a distance smaller than $\min\left\{2(\max_{1\le i\le n} p_i - 1); \left\lceil\frac{2m-1}{m}\max_{1\le i\le n} p_i\right\rceil - 1\right\}$.

4.2.1 Construction of a Feasible Schedule

In order to construct a good feasible schedule for the problem $P, NI|r_i, q_i|C_{max}$, we propose the following procedure. First, we apply Jackson's rule for the relaxed problem $P|r_i, q_i|C_{max}$. Then, we consider the obtained solution σ as a set of sub-schedules $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$ where σ_k is the sequence of jobs scheduled on machine k and I_k the subset of jobs affected to machine k ($\bigcup_{k=1}^{m} I_k = I$). Thus, we have m sub-schedule σ_k where σ_k is related to the one machine scheduling problem of the subset of jobs I_k on machine k. So, we can apply Jackson's rule to each sub-set I_k as a non-idling one machine sub-problem. Finally, we can see that as a result, we have constructed a feasible non-idling schedule for the identical parallel non-idling machines problem. The corresponding algorithm is described below:

NIJSPARA Algorithm for the identical machines problem $P, NI|r_i, q_i|C_{max}$
Begin
  Step 1. Initialization: $I = (r_i, p_i, q_i)_{1\le i\le n}$;
  Step 2. Apply Jackson's rule to the instance I on m machine;
  Step 3.
    3.1 For k from 1 to m
    Begin
      Update the instance I_k by applying for each job $i \in I_k$: $r_i = \max(r_i, \underline{C}_k - \sum_{i\in I_k} p_i)$;
      Apply Jackson's rule to I_k and store the obtained subschedule σ'_k. $\sigma_k = \sigma'_k$;

End for
3.2 $\sigma = (\sigma_1, \dots, \sigma_k, \dots, \sigma_m)$;
End NIJSPARA

The algorithm NIJSPARA can be computed in $O(n \log n)$ operations. Indeed, step 1 needs n operation, Step 2 needs $n \log n$ operations and Step3 needs $n_1 \log n_1 + n_2 \log n_2 + \cdots + n_m \log n_m$ operations where n_k is the number of jobs attribuated to the machine m_k.

Since, we have $n_1 \log n_1 + n_2 \log n_2 + \cdots + n_m \log n_m \leq (n_1 + n_2 + .. n_m) \log n = n \log n$, we deduce that the algorithm NIJSPARA can be computed in $O(n \log n)$ operations.

4.2.2 Extension of Potts's Heuristic

We propose below an extension of Pot's algorithm for the case of identical parallel machines under the non-idling constraint. We first construct a feasible schedule by applying NIJSPAA. Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$ be the obtained schedule. Then, we find the indice k of the critical machine, that is, the machine having $C_{max}(\sigma_k) = C_{max}(\sigma)$. So, if there is an interference job on this machine, we update the instance I as in (Potts, 1984) and we apply again NIJSPARA.

NIPottsPARA algorithm for the problem $P, NI|pmtn, r_i, q_i|C_{max}$
Begin
Step 1. Initialization: $h = 0$; $I = (r_i, p_i, q_i)_{1 \leq i \leq n}$
Step 2. /*Apply NIJSPAR*/
2.1 Apply Jackson's rule to the instance I on m machines;
2.2 For k from 1 to m **do**
Update the instance I_k by applying for each job $i \in I_k$: $r_i = \max(r_i, \underline{C}_k - \sum_{i \in I_k} p_i)$.
Apply Jackson's rule to I_k and store the obtained sub-schedule σ_k
End for
2.3 Store the obtained schedule σ^h
2.4 Set $h = h + 1$;
Step 3.
3.1 Find the indice k_0 of the critical machine
3.2 If $h = n$ or there is no interference job in σ^{h-1} on the critical machine k_0
Then stop and return the best generated schedule among $\sigma^0, \sigma^1, \dots, \sigma^{h-1}$.
Else, identify the interference job b and the critical job c in σ_{k_0} of σ^{h-1}.
Set $r_b = r_c$ and go to step 2.
End NIPottsPARA

NIPottsPARA can be computed in $O(n^2 \log n)$ operations since we apply at most n times NIJSPARA which has a complexity of $O(n \log n)$.

In conclusion, it is easy to see that the algorithms NIJSPARA and NIPottsPARA construct semi-active schedules.

5 LOWER BOUNDS

To evaluate the quality of the constructed solutions, we need lower bound. There is no lower bound in the literature for the problem $P, NI|r_i, q_i|C_{max}$. However, we deduce the following proposition.

Proposition 4:
The optimal $\boldsymbol{C_{max}}$ for the problem $\boldsymbol{P|r_i, q_i|C_{max}}$ is a tight lower bound for the optimal $\boldsymbol{C_{max}}$ of the problem $\boldsymbol{P, NI|r_i, q_i|C_{max}}$.

Proof:
The non idling constraint is a strong constraint which needs to delay some jobs to avoid idle time intervals. Without adding this constraint the set of earliest schedule is dominant for all regular criteria. Indeed in this schedule all the jobs are scheduled as early as possible.

From this proposition we deduce that a lower bound for the classic problem $P|r_i, q_i|C_{max}$ is also a lower bound for its non-idling version $P, NI|r_i, q_i|C_{max}$. The first lower bound for the classic problem $P|r_i, q_i|C_{max}$ is

$$LB_0 = \max(r_i + p_i + q_i) \quad (10)$$

According to (Carlier, 1987), if we associate the data (mr_i, p_i, mq_i) of a one machine problem with the data (r_i, p_i, q_i) of an m-machines problem, then the optimal value of a preemptive solution of the one machine problem is a lower bound for the problem $P|r_i, q_i|C_{max}$. Let LB_1 be this value which can be computed in $O(n \log n)$ operations.

5.1 The Preemptive Lower Bound

The problem $P|r_i, q_i|C_{max}$ is strongly NP-hard. However its preemptive relaxation denoted $P|pmtn, r_i, q_i|C_{max}$ can be computed in polynomial time using a max-flow formulation as showed in (Horn, 1974). Given a $P|pmtn, r_i, q_i|C_{max}$ instance,

the optimal C_{max} is obtained after repeatedly checking the existence of a preemptive schedule with C_{max} equal to an integer trial value C^{trial}. If LB and UB denote a lower bound and upper bound on the trial value CC , then the optimal C_{max} of the problem a $P|pmtn, r_i, q_i|C_{max}$ is computed using a bisection search on the trial interval $[LB, UB]$. We can attribue to LB the value of $\max(LB_0, LB_1)$ as an initial value and to UB the value obtained with NIJSPARA. A deadline d_i is associated with each job $i \in I$ where $d_i = C^{trial} - q_i$. Let $\{e_1, .., e_H\}$ be the set of containing all release dates and deadlines of all jobs ranked in increasing order. A time interval $E_{\square} = [e_{\square}, e_{\square} + 1]$ is defined for each $\square = 1, \dots, H-1$. Consider the flow network composed of job nodes $\{J_1, \dots, J_n\}$, interval nodes $\{E_1, \dots, E_{H-1}\}$, a source node s and a sink node t. For each job node J_i $(j = 1, \dots, n)$ there is an arc (s, J_i) with capacity p_i. For each interval node $E_{\square}$ $(\square = 1, \dots, H-1)$, there is an arc $(E_{\square}, t)$ with capacity $m(e_{\square+1} - e_{\square})$. There is an arc $(J_i, E_{\square})$ with capacity $e_{\square+1} - e_{\square}$ if and only if $r_i \le e_{\square}$ and $e_{\square+1} \le d_i$. A preemptive schedule with $C_{max} = C^{trial}$ is defined as an assignment of portions of processing times of each job $i(i = 1, \dots, n)$ to different time interval $E_{\square}$ $(\square = 1, \dots, H-1)$. The obtained preemptive schedule is feasible if and only if the maximum flow value obtained is equal to $\sum_{i=1}^{n} p_i$. Let LB_3 be the value obtained for the preemptive solution.

5.2 The Semi-preemptive Lower Bound

The preemptive lower bound provides a strong lower bound for the problem $P|r_i, q_i|C_{max}$. To the best of our knowledge, the only lower bound which has been proved to dominate this bound is the semi preemptive lower bound introduced in (Haouari and Gharbi, 2003). This concept was used to derive a max-flow-based lower bound for the $P|r_i, q_i|C_{max}$ in order to improve the classic preemptive lower bound. A semi-preemptive schedule is defined as a schedule where the fixed parts of jobs are constrained to start and to finish at fixed times with no preemption, whereas the free parts can be preempted.

The semi preemptive lower bound is similar in spirit to the preemptive lower bound. It consists in checking the feasibility of a schedule with C_{max} equal to a trial value C^{trial} for the corresponding semi preemptive problem. In a semi preemptive problem, we first associate, to each job i $(i = 1, \dots, n)$ a deadline d_i where $d_i = C^{trial} - q_i$. Then, each job i satisfying $d_i - r_i < 2p_i$ is composed of a fixed part and a free part. Its fixed part is the amount of time $2p_i - (d_i - r_i)$ which must be processed in $[d_i - r_i, r_i + p_i]$ and its free part is the amount of time $p_i' = d_i - (r_i + p_i)$ which has to be processed in $[r_i, d_i - p_i] \cup [r_i + p_i, d_i]$. The other jobs are composed only of a free processing part $p_i' = p_i$ which has to be processed in $[r_i, d_i]$. That is, a free part of any job is $i \in I$ is $p_i' = \min\{p_i, d_i - (r_i + p_i)\}$. The feasibility of a semi-preemptive schedule with C_{max} equal to a trial value CC can be checked as follows. Let $S = \{i \in I, d_i - r_i < 2p_i\}$ denotes the set of jobs having a fixed processing part. Let $e_1, e_2, \dots, e_W$ be the different values of r_i $(i \in I)$, d_i $(i \in I)$, $d_i - p_i$ $(i \in S)$ and $r_i + p_i$ $(i \in S)$ ranked in increasing order. We denote by m_w the number of machines which are idle during the time interval $= [e_w, e_{w+1}]$ $(1 \le w \le W)$. In (Haouari and Gharbi, 2003), the authors proposed an extension of Horn's approach to solve the feasibility problem. They consider the flow network composed of job nodes $\{J_1, J_2, \dots, J_n\}$, interval nodes $\{E_1, E_2, \dots, E_{W-1}\}$, a source node s, and a sink node t. For each job node J_i $(i = 1, .. n)$ such that $p_i' > 0$, there is an arc (s, J_i) with capacity p_i' representing the free part of job i. For each $w = 1, \dots, W-1$, there is an arc (E_w, t) with capacity $m_w(e_{w+1} - e_w)$. There is an arc (J_i, E_w) with capacity $e_{w+1} - e_w$ if and only if one of the three following conditions holds: $d_i - r_i < 2p_i$, $r_i \le e_w$ and $e_{w+1} \le d_i - p_i$; $d_i - r_i < 2p_i$, $r_i + p_i \le e_w$ and $e_{w+1} \le d_i$, $d_i - r_i \ge 2p_i$, $r_i \le e_w$ and $e_{w+1} \le d_i$.

To evaluate the proposed heuristics for the problem $P, NI|r_i, q_i|C_{max}$, we use as lower bound the value of the semi-preemptive schedule. Let LB_4 be this value.

6 COMPUTATIONAL RESULTS

We have implemented NIJSPARA and NIPottsPARA in the programming language C and we have used an experimental analysis. The experiments were run on a packard bell intel R core i5-3230M with 4GB DDR3 Memory. We have also implemented the different lower bounds described in section 5. The data set for experiments was generated in the same way as the data set in [Carlier, 1982]. The tests were randomly generated according to a uniform distribution for a number of jobs $n \in \{100, 200, 300, 400, 500\}$ and $m \in \{5, 10, 20\}$. The jobs processing times were random integers from a uniform distribution in $[1, p_{max}]$, the jobs release dates were random integers from a uniform

distribution in $[1, r_{max}]$ and the jobs delivery times were random integers from a uniform distribution in $[1, q_{max}]$. We fix the value of p_{max} at 50 ($p_{max} = 50$) and we fix the values of r_{max} and q_{max} such that $r_{max} = q_{max} = \left(\frac{1}{50}\right) np_{max}k$ where $k = 1, 2, \ldots 30$ and $k = 35, 40, \ldots, 60$ (36 tests for each value of n and m).

Table 1 shows the performance of NIJSPARA and Table 2 shows the performance of NIPottsPARA. We provide the average relative gap produced by each algorithm where the gap is equal to $100(C_{max} - LB_4)/LB_4$. Table 3 shows the average CPU time of NIPottsPARA. We have omitted to report the CPU time required by NIJSPARA because for all of the instances this time was negligible ((≈ 0). NBzero denotes the number of tests solved optimally in table 1 and 2 and denotes the number of tests solved with a negligible CPU time (≈ 0) in table 3 for NIPottsPARA.. In fact NIJSPARA the CPU time is negligible for all tests.Table 1 provides evidence that NIJSPARA is fast and effective. Indeed its average relative deviation from the semi-preemptive lower bound is equal to 0,4% and 56,85% of tests are solved optimally on average.

Table 1: Performance of NIJSPARA.

n	*m*	%NBzero	Average	Maximum
100	5	47,22	0,71	5,56
	10	69,44	1,02	10,46
	20	58,33	***1,54***	***11,43***
200	5	52,78	0,11	0,75
	10	69,44	0,29	3,04
	20	66,67	0,66	9,17
300	5	41,67	0,1	0,67
	10	58,33	0,12	0,98
	20	72,22	0,58	3,95
400	5	**38,89**	0,07	0,29
	10	41,67	0,2	1,67
	20	58,33	0,43	2,86
500	5	41,67	***0,05***	***0,18***
	10	58,33	0,06	1,23
	20	**77,78**	0,11	1,95
Minimum		38,89	0,05	0,18
Average		56,85	0,4	3,61
Maximum		77,78	1,54	11,43

Table 2: Performance of NIPottsPARA.

n	*m*	%NBzero	Average	Maximum
100	5	52,78	0,57	5,56
	10	69,44	1,02	10,46
	20	58,33	***1,54***	***11,43***
200	5	58,33	0,06	0,75
	10	75	0,25	3,04
	20	66,67	0,66	9,17
300	5	47,22	0,04	0,31
	10	61,11	0,08	0,98
	20	72,22	0,58	3,95
400	5	41,67	0,05	0,29
	10	44,44	0,16	1,67
	20	61,11	0,4	2,86
500	***5***	47,22	***0,03***	***0,12***
	10	63,89	0,06	1,23
	20	80,56	0,1	1,95
Minimum		41,67	***0,03***	***0,12***
Average		60	0,37	3,59
Maximum		80,56	***1,54***	***11,43***

Table 2 shows that NIPottsPARA improves slightly the quality of solutions obtained by NIJSPARA. However, it needs more CPU time than NIJSPARA. Indeed, 60% of tests are solved optimally, on average. Also, the average relative deviation from the semi-preemptive lower bound is equal to 0,36% . Table 3 shows that the CPU time is negligeable (≈ 0) for this heuristic for 35,74% tested problems, on average.

We also note that the performance of the heuristics is improved when the number of jobs is greater for the two heuristics.

Table 3: The average CPU time of NIPottsPARA in seconds.

n	*m*	%NBzero	Average	Maximum
100	5	55,56	3,04	10,44
	10	88,89	0,26	9,51
	20	**91,67**	**0**	**0**
200	5	33,33	16,35	41,57
	10	41,67	5,24	39,52
	20	63,89	0	0
300	5	16,67	49	102,09
	10	22,22	31,9	100,23
	20	47,22	0	0,02
400	5	19,44	91,7	179,68
	10	8,33	65,95	162,92
	20	13,89	8,7	156,75
500	**5**	**5,56**	**137,27**	**277,43**
	10	11,11	83,49	274,46
	20	16,67	7,22	245,03
Minimum		5,56	0	0
Average		35,74	33,34	106,64
Maximum		91,67	137,27	277,43

7 CONCLUSIONS

In this paper, we have investigated heuristic approaches for minimizing makespan subject to release dates and delivery times under the non-idling constraint. We have first proposed an heuristic NIJSPARA in order to construct a feasible non-idling schedule using Jackson's rule. Then we have proved experimentally that the proposed heuristic is efficient. We have also proposed a second heuristic NIPottsPARA in order to improve the feasible non-idling schedule obained by the first heuristic NIJSPARA. The computational tests proved that there is a slightly improvement with NIPottsPARA. This paper presents a first attempt and proposed a good upper bound and a way to construct feasible schedules for this type of problem. The computational results show that the semi preemptive lower bound is tight. In future research we intend to use these heuristics as starting solutions either to propose more efficient heuristics or to develop a branch and bound in order to built optimal solutions.

REFERENCES

Moore, R., Lopes, J., 1999. Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. SCITEPRESS.

Smith, J., 1998. *The book*, The publishing company. London, 2nd edition.

Carlier J., 1982. The one-machine sequencing problem. *European Journal of Operational Research,* 11: 42-47.

Carlier J., 1987. Scheduling jobs with release dates and tails on identical parallel machines to minimize the makespan. *European Journal of operational research*, 29: 298–306.

Jacques J., Hermès F., Moukrim A., and Ghédira K. 2010. An exact resolution of one machine problem with no machine idle time, *Comput. Ind. Eng*, 59 (2): 193–199.

Chrétienne P. On single-machine scheduling without intermediate delay. *Discrete Applied Mathematics*, 13: 2543–2550, 2008.

Chrétienne P., 2014. On scheduling with the non-idling constraint. 4OR*: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 12: 101–121.

Frederickson G.N., 1983. Scheduling unit-time tasks with integer release times and deadlines. *Information Processing Letters*, 16: 171–173,

Garey M.R. and Johnson D.S., 1978. Strong NP-completeness results: Motivation, examples and implications. *Journal of the Association of Computer Machinery*, 25: 499 –508.

Gusfield D., 1984. Bounds for naïve multiple machine scheding with release times ahd deadlines. *Journal of algorithms*, 5: 1-6, 1984.

Haouari M. and Gharbi A., 2003. An improved max-flow lower bound for minimizing maximum Lateness on identical parallel machines. *Operations Research Letters*, 31: 49-52.

Horn W.A. , 1974. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 177–185.

Jackson J.R., 1955. Scheduling a production line to minimize maximum tardiness, *Research report 43, Management Science Research Project Univ. of California, Los Angeles*.

Jouglet A., 2012. Single-machine scheduling with no-idle time and release dates to minimize a regular criterion. *Journal of Scheduling,* 15: 217–238,.

Kacem I., and Kellerer H., 2014. Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times. *Discrete Applied Mathematics*, 164: 154–160, 2014.

Labetoulle J., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., 1984. Preemptive scheduling of uniform machines subject to release dates. In: Pulleyblank, W.R. (ED.), *Progress in Combinatorial Optimization, Academic Press, New York*, 245–261.

Lageweg B.J., Lenstra J.K., and A.H.G. Rinnooy Kan, 1976. Minimizing maximum lateness on one machine: computational experience and some applications. *Statistica Neerlandica,* 30: 25–41, 1976.

Landis K. , 1983. Group Technology and Cellular Manufacturing in the Westvaco Los Angeles VH department. *Project Report in IOM 581, School of Business, University of Southern California*.

Potts CN., 1980. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28: 1436-1441, 1980.

Quilliot A. and Chrétienne P., 2013. Homogenously non-idling schedules of unit-time jobs on identical parallel machines. *Discret Appl Math,* 161(10-11): 1586-1597.

Simons B., 1983. Multiprocessor scheduling of unit time jobs with arbitrary release times and deadlines. *Siam J. Comput.*, volume (12), 294-299.

Valente J.M.S., and Alves R.A.F.S.. 2005. An exact approach to early/tardy scheduling with release dates. Comput. Oper. Res., 32: 2905-2917.