New objective functions based on jobs positions for single machine scheduling with deadlines

Thanh Thuy Tien Ta, Kivin Ringard, Jean-Charles Billaut

New objective functions based on jobs positions for single machine scheduling with deadlines

(presented at the 7^{th} IESM Conference, October 11–13, 2017, Saarbrücken, Germany)

Thanh Thuy Tien TA, Kivin RINGARD, Jean-Charles BILLAUT
Laboratoire d'Informatique, ERL CNRS 6305,
64 avenue Jean Portalis
F-37200, Tours
Email: thanhthuytien.ta@etu.univ-tours.fr, jean-charles.billaut@univ-tours.fr

***Abstract*—In this paper, we study new single machine scheduling problems where the objective functions are based on jobs positions. The original idea is to characterize easily a huge set of optimal solutions, without enumerating them. The (polynomially solvable) scheduling problems that can be addressed by this approach are such that a simple rule exists, giving a priority between two consecutive jobs. Assuming that the solution obtained by applying the rule is sequence $Id = (J_1, J_2, ..., J_n)$ (correct after a renumbering of jobs), we search for a feasible solution (i.e. also optimal), at a maximum distance from sequence Id. Such considerations are also usefull for some robustness considerations. We identify some new objective functions and present first theoretical results and first computational experiments. Some new problems remain open.**

Key words: Scheduling, Single machine, Complexity, MILP, branch-and-bound

I. Introduction

Many scheduling problems can be solved in polynomial time by applying a simple sequencing rule, also called "priority rule" [10]. The application of such a rule indicates whether it is preferable to place a job J_i before a job J_j, or the contrary. For example, it is well known that problem $1||\sum C_j$ is solved optimally by sorting the jobs according to their nondecreasing processing time order (SPT rule [10]), the problem $1||L_{max}$ is solved optimally by sorting the jobs according to their non decreasing due dates order (EDD rule, [7]), the problem $F2||C_{max}$ is solved by the Johnson's rule ([8]), etc. (see [5] for a generalization to other scheduling problems). It is also well known that each scheduling problem generally has a lot of optimal solutions (potentially an exponential number).

The original goal is to characterize a set of optimal solutions without enumerating them. Such an approach is useful in a dynamic environment, to react in real time to unexpected events, or to data uncertainty. Some preliminary studies concerning the search of the characteristics of the optimal solutions (characteristics but not the list) have been conducted in this direction, using the lattice of permutations as support, for problems $1||L_{max}$ and $F2||C_{max}$ in [2] and for problem $F2||C_{max}$ only in [3].

In this paper, we only consider problem $1||L_{max}$. For any instance, we apply the following pre-treatment in polynomial time: (1) we apply EDD rule and we obtain L^*_{max}, the optimal maximum lateness value. Then, (2) we modify the due dates in order to obtain deadlines: $\tilde{d}_j = d_j + L^*_{max}$, for any $j \in \{1, 2, ..., n\}$, limiting the deadlines to $\sum p_j$. Finally, (3) we renumber the jobs in EDD order. At the end we know that sequence $Id = (J_1, J_2, ..., J_n)$ is feasible and that jobs are numbered in EDD order. Suppose for example that we have $n = 4$ jobs and that – after the pre-treatment – it is found a sequence $\sigma = (J_3, J_1, J_4, J_2)$ which is also a feasible sequence. Then, because of the EDD rule, it is possible by simple pairwise exchanges, to modify this sequence iteratively and to reach sequence Id. All the sequences obtained during these changes are also feasible sequences: sequence (J_1, J_3, J_4, J_2) is feasible, sequence (J_3, J_1, J_2, J_4) is feasible, etc. So, all the "predecessors" of sequence σ are feasible. These sequences are exactly those such that $J_1 \prec J_4$ and $J_1 \prec J_2$ and $J_3 \prec J_4$, i.e. are characterized by the conjunction of three simple precedence relations. Therefore, finding a feasible sequence, *as far as possible* from sequence Id, allows immediately to characterize a *big* set of feasible solutions.

Such an approach – described in details in [3] is valid for any scheduling problem for which such a rule exists such as problems $1|r_j|C_{max}$, $F2|s_{nsd}|C_{max}$, etc.

The paper is organized as follows. We describe the problem more formally in Section II, where we define some new objective functions and we give first results. In Section III we focuse on the minimization of the sum of weighted positions. We show that this problem is strongly NP-hard and we propose some exact and approximated resolution methods. These methods are evaluated by randomly generated instances in Section IV. Finally, in the section V, we present some ideas for the future research directions.

II. Problems definitions

We consider a set of n jobs $\{J_1, J_2, ..., J_n\}$ to schedule on a single machine. Preemption is not allowed and the machine can perform only one job at a time. To each job J_j is associated a processing time p_j and a deadline $\tilde{d}_j$. It is assumed that the sequence is feasible (maximum lateness equal to 0) and that the jobs are numbered in EDD order (notice that it can be done in $O(n \log n)$ time for any instance).

Let consider a sequence σ. We denote by N_j the number of jobs after J_j in σ with an index greater than j. For example, consider $\sigma = (J_3, J_1, J_4, J_2)$, we have $N_1 = 2$, $N_2 = 0$, $N_3 = 1$ and $N_4 = 0$. The sum of N_j, $\sum_{j=1}^{n} N_j$ is exactly equal to the number of simple precedence constraints in the

conjunction (here $\sum_{j=1}^{n} N_j = 3$). It also corresponds to the level of sequence σ in the lattice of permutations (see [3]).

Finding a sequence as deep as possible in this lattice, or finding a sequence as far as possible from sequence Id is the same, and is equivalent to minimise the objective function $\sum N_j$. The problem can be denoted by $1||Lex(L_{\max}, \sum N_j)$ with a notation of a bi-objective problem [11], which is equivalent to $1|\tilde{d}_j, \tilde{d}_1 \leq ... \leq \tilde{d}_n, \tilde{d}_n = \sum p_j| \sum N_j$ for the instance after pre-treatment.

A. Expression of N_j

Let us define the boolean variables $x_{j,k} = 1$ if job J_j is in position k and 0 otherwise. The expression of $\sum N_j$ is the following:

$$\sum N_j = \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h} x_{j,k}$$

This expression is quadratic, it can be linearized for a linear programming formulation [1], [3].

Let us define by $y_{j,k} = \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h}$. We have:

$$\sum N_j = \sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k} x_{j,k}$$

Proposition 1: $\sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k}$ is a function related to the positions of the jobs in the sequence.

Proof. We have:

$$\sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k} = \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h}$$

(notice that this expression is related to, but different than the expression of $\sum N_j$).

We can show that

$$\begin{aligned} \sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k} &= -n^2 + \sum_{j=1}^{n} \sum_{k=1}^{n} jk x_{j,k} \\ &= -n^2 + \sum_{j=1}^{n} j \sum_{k=1}^{n} k x_{j,k} \end{aligned}$$

In this expression, $P_j = \sum_{k=1}^{n} k x_{j,k}$ is exactly the position of J_j in the sequence. So, minimizing $\sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k}$ is equivalent to minimize $\sum_{j=1}^{n} j P_j$.

This new indicator, P_j, and this new objective function $\sum jP_j$ never appeared in the scheduling literature before, to the best of our knowledge. Notice that this objective function is not related to the jobs completion time, which is also not frequent in the literature.

B. New objective functions and first results

Even if $\sum N_j$ and $\sum jP_j$ are not the same objective functions, we can notice that these functions are related in some sense.

We introduce the following new objective functions:

- $P_{\max} = \max_{1 \leq j \leq n} P_j$, the maximum position of a job in the sequence,
- $\sum P_j = \sum_{j=1}^{n} P_j$, the sum of positions of the jobs in the sequence,
- $\sum w_j P_j = \sum_{j=1}^{n} w_j P_j$, the weighted sum of positions of jobs in the sequence

Let us consider only single machine problems, without deadlines. We can exhibit the first following results:

- Problem $1||P_{\max}$ is trivial (similar to $1||C_{\max}$) since for any solution, we have $P_{\max} = n$. It is the same result for any sequencing problem.
- Problem $1||\sum P_j$ is trivial since in any solution we have $\sum P_j = n(n+1)/2$. Again, it is the same result for any sequencing problem.
- Problem $1||\sum w_j P_j$ is equivalent to problem $1|p_j = 1|\sum w_j C_j$, and therefore it can be solved in $O(n \log n)$ by sorting the jobs in w_j nonincreasing order.

Let us consider now the case with deadlines. We can exhibit the first following results:

- Problem $1|\tilde{d}_j|P_{max}$ is trivial because if EDD is a feasible solution, this solution has $P_{\max} = n$, which is optimal. Otherwise, there is no solution.
- Problem $1|\tilde{d}_j|\sum P_j$ is trivial for the same reasons, and EDD is such that $\sum P_j = n(n+1)/2$, which is optimal.
- Problem $1|\tilde{d}_j, p_j = 1|\sum w_j P_j$ is equivalent to problem $1|\tilde{d}_j, p_j = 1|\sum w_j C_j$, which can be solved in polynomial time [6].

We focuse in the rest of the paper on problem $1|\tilde{d}_j|\sum w_j P_j$.

III. Problem $1|\tilde{d}_j|\sum w_j P_j$

In this section, we assume that jobs are numbered in EDD order and that sequence $Id = (J_1, J_2, ..., J_n)$ is feasible and has a maximum lateness of 0. We search to minimise the weighted sum of jobs positions.

A. Complexity

Proposition 2: Problem $1|\tilde{d}_j|\sum w_j P_j$ is strongly NP-hard.

Proof. It is clear that the problem is in NP. Then, we proceed by reduction from 3-PARTITION problem.

3-PARTITION problem:
Instance: $A = \{a_i\}_{1 \leq i \leq 3m}$ a set of $3m$ elements such that $B/4 < a_i < B/2$ and $\sum_A a_i = mB$, with a bound $B \in N$.

Question: Can A be partitioned into m disjont sets $A_1, A_2, ..., A_m$ such that $\sum_{A_k} a_i = B, \forall k \in \{1, 2, ..., m\}$ (note that each A_k must contain exactly 3 elements of A)

Clearly, problem $1|\tilde{d}_j|\sum w_j P_j$ is in NP. We define $\overline{p} = |A| \times \max(A)$. We build an instance to our problem with m dummy jobs $J_i, 1 \leq i \leq m$ such that $p_i = 1, w_i = 0, \widetilde{d}_i = i(B + 3\overline{p} + 1)$. We add $3m$ regular jobs $J_{m+j}, 1 \leq j \leq 3m$ with $p_{m+j} = w_{m+j} = a_j + \overline{p}$ and $\widetilde{d}_{m+j} = m(B + 3\overline{p} + 1)$.

We first show that in an optimal solution, each job J_i is precisely at position $4i$, $1 \leq i \leq m$ and then we show that it completes exactly at its deadline if and only if the answer to 3-PARTITION is yes.

(a) We consider an optimal sequence S^*. We prove by induction that in any optimal sequence, each job $J_i, 1 \leq i \leq m$, has a position equal to $4i$.

First, we have to check it for $i = 1$.
Suppose that J_1 is in position k, and suppose that it completes at time $C_1 < \tilde{d}_1$. We denote by D the set of the first $k-1$ jobs (before J_1). We denote by J_l the job in position $k+1$, and we know that this job completes after $\tilde{d}_1$ (since otherwise swapping J_1 and J_l would lead to a better solution). We denote by Δ the difference between the starting time of J_1 and $\widetilde{d}_1$.

We have:

$$\begin{aligned}\Delta \geq 1 &\Rightarrow \widetilde{d}_1 - \sum_{J_j \in D} p_j \geq 1 \\ &\Rightarrow \widetilde{d}_1 - \sum_{J_j \in D} a_j - (k-1)\overline{p} \geq 1 \\ &\Rightarrow B + 3\overline{p} + 1 - \sum_{J_j \in D} a_j - k\overline{p} + \overline{p} \geq 1 \\ &\Rightarrow B + 4\overline{p} - \sum_{J_j \in D} a_j - k\overline{p} \geq 0\end{aligned}$$

So we have:

$$k \leq 4 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j) \quad (1)$$

Because $C_l > \widetilde{d}_1$, we have:

$$\sum_{J_j \in D} a_j + (k-1)\overline{p} + 1 + a_l + \overline{p} > B + 3\overline{p} + 1$$

$$\sum_{J_j \in D} a_j + k\overline{p} + a_l > B + 3\overline{p}$$

So we have:

$$k > 3 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j - a_l) \quad (2)$$

We deduce from (1), (2) that:

$$3 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j - a_l) < k \leq 4 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j)$$
$$\Leftrightarrow 3 + \varepsilon_1 < k \leq 4 + \varepsilon_2$$

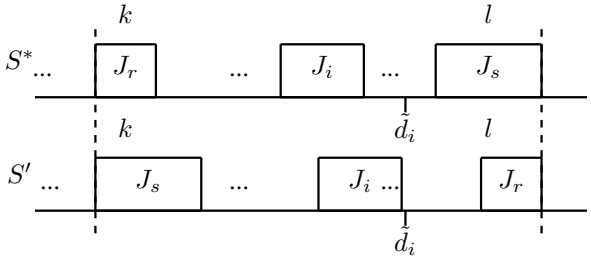

Fig. 1. Case where J_i does not complete at time $\tilde{d}_i$ $(1 \leq i \leq m)$

It shows that J_1 is necessarilly in position 4.

Suppose now that job J_i is in position $4i, \forall i, 1 \leq i \leq m-1$. We want to show that job J_{i+1} is in position $4(i+1)$. Suppose that J_{i+1} is in position k. We denote by D the jobs preceding J_{i+1}. There are $(k-1-i)$ jobs in D that are regular jobs. We have: $C_{i+1} \leq \widetilde{d}_{i+1}$

$$\Rightarrow (i+1)(B + 3\overline{p} + 1) - (\sum_{J_j \in D} a_j + (k-1-i)\overline{p}) - i - 1 \geq 0$$

$$\Rightarrow k \leq 4(i+1) + \varepsilon \quad (3)$$

Let us denote by J_l the job just after J_{i+1}. We have:

$$C_l > \tilde{d}_{i+1} \Rightarrow \sum_{J_j \in D} a_j + |D|\overline{p} + i + 1 + \overline{p} + a_l > (i+1)(B + 3\overline{p} + 1)$$

If J_{i+1} is in position k, the number of jobs before J_{i+1} is $k-1$, and before this job, there are $|D|$ regular jobs and i dummy jobs. So we have $k - 1 = |D| + i$. Hence:

$$\sum_{J_j \in D} a_j + (k-1-i)\overline{p} + i + 1 + \overline{p} + a_l > (i+1)(B + 3\overline{p} + 1) \quad (4)$$

$$\Rightarrow k > (4i+3) + \varepsilon \quad (5)$$

We deduce from (3) and (5) that $k = 4i + 4$.

(b) We must show now that there is an answer YES to 3-PARTITION problem, if and only if in an optimal solution to our problem, each dummy job completes at its deadline.

Suppose there is an answer YES to 3-PARTITION problem. In this case, we can take the partition of the jobs and put a dummy job between each partition. Because of the deadlines definition, each dummy jobs completes exactly at its deadline. Breaking the partition would either violate a deadline, or generate a weaker solution.

Suppose that we do not follow the 3-PARTITION solution and that in S^*, we have $C_i < \widetilde{d}_i$ for one job J_i with $1 \leq i \leq m$ (i.e. we have an optimal solution that is not a 3-partition). Suppose there are two jobs: J_r before J_i in position k and J_s after J_i in position l such that if the jobs are swapped, then $C_i = \widetilde{d}_i$. In this case, illustrated in Fig. 1 we must have $a_r < a_s$ and $k < l$.

We denote by S' the solution S^* except that J_r and J_s are swapped in S'. We have:

$$\begin{aligned}Z(S') - Z(S^*) &= k(a_s + \overline{p}) + l(a_r + \overline{p}) - k(a_r + \overline{p}) - l(a_s + \overline{p}) \\ &= k(a_s - a_r) + l(a_r - a_s) \\ &= (l-k)(a_r - a_s) < 0\end{aligned}$$

This quantity is negative, which means that S^* is not optimal, so this case is not possible.

Suppose now that there are two jobs J_{r_1} and J_{r_2} before J_i and two jobs J_{s_1} and J_{s_2} after J_i so that if the jobs are swapped, then $C_i = \widetilde{d}_i$. In this case, we have $a_{s_1} + a_{s_2} > a_{r_1} + a_{r_2}$ and :
$Z(S') - Z(S^*) = k_1 a_{s_1} + k_2 a_{s_2} + l_1 a_{r_1} + l_2 a_{r_2} - k_1 a_{r_1} + k_2 a_{r_2} + l_1 a_{s_1} + l_2 a_{s_2} = (k_1 - l_1)(a_{s_1} - a_{r_1}) + (k_2 - l_2)(a_{s_2} - a_{r_2})$

We know that $k_1 - l_1 < 0, k_2 - l_2 < 0$
$\Rightarrow \max(k_1 - l_1, k_2 - l_2) < 0$ and we have
$Z(S') - Z(S^*) = (k_1 - l_1)(a_{s_1} - a_{r_1}) + (k_2 - l_2)(a_{s_2} - a_{r_2})$
$< (a_{s_1} - a_{r_1}) \max(k_1 - l_1, k_2 - l_2) + (a_{s_2} - a_{r_2}) \max(k_1 - l_1, k_2 - l_2) < (a_{s_1} - a_{r_1} - a_{s_2} - a_{r_2})) \max(k_1 - l_1, k_2 - l_2) < 0$

For the same reasons, this case is not possible because S^* is optimal. By extending this reasoning to any number of jobs to exchange (same number before and after J_1 due to the position of this job), we prove that if the answer to 3-PARTITION problem is YES, we must follow the 3-PARTITION solution and therefore we have a feasible solution that is optimal.

B. Two exact resolution methods

We propose two ways to solve the problem to optimality: a linear programming formulation and a branch-and-bound algorithm.

1) MILP formulation: An MILP formulation of the problem with variables $x_{j,k}$ is the following.

$$\text{Minimize} \sum_{j=1}^{n} \sum_{k=1}^{n} w_j k x_{j,k}$$

$$\sum_{j=1}^{n} x_{j,k} = 1, \forall k \in \{1, 2, ...n\}$$

$$\sum_{k=1}^{n} x_{j,k} = 1, \forall j \in \{1, 2, ...n\}$$

$$\sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{j,l} \leq \sum_{j=1}^{n} \tilde{d}_j x_{j,k}, \forall k \in \{1, 2, ...n\}$$

This model contains n^2 binary variables and $3n$ constraints. This model will be solved by CPLEX solver.

2) Branch-and-bound: A branch-and-bound procedure is an implicite enumeration method, characterized by two elements: the branching and the bounds. Here, a node is characterized by a partial ending sequence σ composed by k jobs (from position $n - k + 1$ to n), a set S of unscheduled jobs and a lower bound. The branching consists in adding a job of S at position $n - k$ in σ.

The initial upper bound is given by the (simple) Backward Algorithm presented in Alg. 1.

The lower bound is evaluated as follows. The partial sequence σ is evaluated, and the jobs of S are sequenced in w_j non-increasing order. If the partial sequence is not feasible, the evaluation is put to HV (high value).

Let consider the following instance with $n = 5$ jobs.

Algorithm 1 BW algorithm

$\mathcal{J} \leftarrow \{J_1, J_2, ..., J_n\}$
$t \leftarrow \sum_{j=1}^{n} p_j$; $k \leftarrow n$
while $\mathcal{J} \neq \emptyset$ **do**
  Let $J_r \in \mathcal{J}$ such that $w_r = \min_{j \in \mathcal{J}/\tilde{d}_j \geq t} w_j$
  Put J_r in position k
  $t \leftarrow t - p_r$; $k \leftarrow k - 1$
  $\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_r\}$
end while

j	1	2	3	4	5
p_j	1	2	5	2	4
w_j	1	2	4	2	3
$\tilde{d}_j$	7	7	12	14	14

The BW algorithm gives sequence $(J_2, J_1, J_3, J_5, J_4)$. The evaluation is equal to 38.

Let consider node $\sigma = (J_3, J_4)$ and $S = \{J_1, J_2, J_5\}$. The partial sequence σ has an evaluation equal to 16+10=26. The unscheduled jobs are sequenced in an 'optimal' (but not necesserally feasible) way, leading to sequence (J_5, J_2, J_1) evaluated by 3+4+3=10. The evaluation of this node is equal to 36.

C. Heuristic algorithms

We propose two heuristic algorithms. The first one is the previous Backward algorithm. This algorithm is important because it is very intuitive (remember that such an algorithm is optimal for problems $1|prec|f_{\max}$ [9]). We also propose a Forward algorithm described in Alg. 2 ($J_{[si]}$ is the job in position si).

Algorithm 2 FW algorithm

$\sigma = (J_1, J_2, ..., J_n)$
$si = 1$ // si stands for "smallest index"
while $P_{si} < n$ **do**
  $S' \leftarrow \{J_j / P_j > si\}$
  while $S' \neq \emptyset$ **do**
    Let $J_k \in S'$ such that $w_k = \max_{J_j \in S'} w_j$
    // σ is a sequence of type $\sigma_1 // J_{[si]} // \sigma_2 // J_k // \sigma_3$
    $\sigma' \leftarrow \sigma_1 // J_k // J_{[si]} // \sigma_2 // \sigma_3$
    if σ' is not feasible **then**
      $S' \leftarrow S' \setminus \{J_k\}$
    else
      $\sigma \leftarrow \sigma'$
      $si \leftarrow si + 1$
    end if
  end while
  $S' \leftarrow \{J_j / P_j > si\}$
  $si \leftarrow$ smallest index in S'
end while

Let us consider the previous example. We start with $\sigma = (J_1, J_2, J_3, J_4, J_5)$. We have $si = 1$ Job J_3 cannot be put before J_1 (otherwise J_2 is late). Job J_5 can be put before J_1. We obtain sequence $\sigma = (J_5, J_1, J_2, J_3, J_4)$. Then, J_2 can be put before J_1. We obtain sequence $\sigma = (J_5, J_2, J_1, J_3, J_4)$. Then, we have $si = 3$. It is not possible to put J_4 before J_3. Then, we have $si = 4$ and the procedure stops because $P_4 = 5$. The evaluation of this solution is equal to 36.

A global lower bound LB can also be easily computed by relaxing the MILP.

IV. Computational experiments

A. Data generation

The computational experiments have been run on a MacBook Air Intel Core i5 1,6 GHz, using IBM ILOG CPLEX 12.6. Two types of instances have been generated. Instances of type 1 are classical random data sets, instances of type 2 are difficult random data sets. For each type, 30 instances have been generated for each value of n, with $n \in \{10, 20, ..., 100\}$. For instances of type 1, random data sets have been generated as follows:

- $p_j \in [1, 100]$, $w_j \in [1, 100]$
- $d_j \in [(\alpha - \beta/2)P, (\alpha + \beta/2)P]$ with $P = \sum p_j$, $\alpha = 0.75$ and $\beta = 0.25$

Then, these instances receive the pre-treatment.

For instances of type 2, random data sets have been generated as follows:

- for $n' = \lfloor n/4 \rfloor$ jobs:
  - $p_j = 1$, $w_j = 0$
  - $d_j = 4jP/n$
- for the $(n - n')$ remaining jobs:
  - $p_j \in [1, 100]$, $w_j = w_{0j} + P$, with $w_{0j} \in [1, 100]$ and $P = \sum p_j$
  - $\tilde{d_j} = P + \lfloor n/4 \rfloor$

These instances do not need the pre-treatment.

The CPU time to solve each instance has been limited to 180 seconds.

B. Results

The computational results are reported in Tables I to IV. Tables I and II are related to instances of type I, Tables III and IV are related to instances of type II. Tables I and III give the results of the exact methods. Columns cpu and opt indicate respectively the CPU time in seconds and the number of times the method found the optimal solution in less than 180 seconds. Tables II and IV give the results of the heuristic methods. Columns best, DeltaBest and DeltaLB indicate respectively the number of times the method returns the best solution among all the methods (including the exact methods), the relative deviation to the best solution, and the relative deviation to the lower bound. Column Δ indicates the relative deviation between BW and FW:

$$\Delta = \frac{BW - FW}{BW}$$

About the exact methods (Tables I and III), one can see that the B&B algorithm is competitive with CPLEX for type I instances, both methods solving 29 problems over 30 with up to 70 jobs, and less with more jobs. However, for instances of type II, CPLEX is much more better than the B&B, solving instances with up to 40 jobs, where the B&B cannot solve instances with more than 10 jobs. It is clear that it is possible to improving the B&B algorithm, by pruning some nodes by using some dominance conditions.

	B&B		CPLEX	
n	cpu	opt	cpu	opt
10	0,01	30	0,11	30
20	0,03	30	0,15	30
30	0,09	30	0,42	30
40	0,18	30	0,99	30
50	0,54	30	3,22	30
60	0,81	30	7,22	30
70	11,87	29	26,45	29
80	53,11	23	52,17	26
90	87,15	18	86,94	23
100	139,17	11	145,93	10

TABLE I. Results of the exact methods for Type I instances

	BW			FW			Δ
n	best	DeltaBest	DeltaLB	best	DeltaBest	DeltaLB	BW/FW
10	28	0,08%	2,28 %	28	0,04%	2,24%	0,04%
20	23	0,08%	1,37 %	28	0,02%	1,31%	0,06%
30	18	0,19%	1,51 %	24	0,03%	1,35%	0,16%
40	6	0,44%	1,52 %	16	0,04%	1,09%	0,41%
50	3	0,24%	1,72 %	11	0,11%	1,59%	0,13%
60	6	0,15%	1,46 %	8	0,08%	1,38%	0,07%
70	3	0,24%	1,22 %	4	0,1%	1,08%	0,14%
80	3	0,28%	1,2 %	2	0,17%	1,08%	0,12%
90	3	0,35%	1,22 %	3	0,15%	1,01%	0,20%
100	0	0,29%	1,08 %	6	0,09%	0,87%	0,20%

TABLE II. Results of the heuristic methods for Type I instances

About the heuristic methods (Tables II and IV), FW algorithm returns better solution than BW algorithm. For instances of type I, BW returns solution that are not so bad, with a relative deviation to the best solution under 0.44%, whereas FW is under 0.17%. Note that the relative deviation between BW and FW is in favor of FW, but with a percentage less than 0.41%. However, for instances of type II, BW is dominated by FW. BW returns solutions that are far from the optimal solution, and the relative deviation between BW and FW is around 10%.

V. Conclusion and perspectives

We identify in this paper a new category of scheduling problems, with the definition of new objective functions, based on jobs positions. Some trivial problems are identified, and a problem with weights is proved strongly NP-hard. Some problems remain open, such as the $1|\tilde{d_j}|\sum jP_j$ problem and the original $1|\widetilde{d_j}|\sum N_j$ problem.

For the difficult problem, heuristic algorithms and exact methods are proposed and evaluated on randomly generated instances. The results of the exact methods and of the heuristic algorithms are relatively good for the first type instances, where data are generated 'as usual' in the scheduling literature. However, for more difficult instances, the limits of the exact methods are reached, and the difference between the heuristic algorithms is more clear.

The perspectives consist in continuing the investigations for the open problems, and in the improvment of the exact methods – introducing cuts and dominance conditions – and of the heuristic methods, by using metaheuristic algorithms.

	B&B		CPLEX	
n	cpu	opt	cpu	opt
10	0,12	30	0,18	30
20	178,18	1	1,57	30
30	180	0	9,85	30
40	180	0	50,87	28
50	180	0	174,78	5
60	180	0	180,06	0
70	180	0	180,06	0
80	180	0	180,08	0
90	180	0	180,11	0

TABLE III. RESULTS OF THE EXACT METHODS FOR TYPE II INSTANCES

	BW			FW			Δ
n	best	DeltaBest	DeltaLB	best	DeltaBest	DeltaLB	BW/FW
10	0	8,44%	21,5 %	1	1,26%	12,62%	7,26%
20	0	7,84%	20,8 %	0	0,31%	11,66%	7,56%
30	0	10,5%	23,43 %	0	0,34%	10,85%	10,19%
40	0	10,55%	23,55 %	0	0,39%	10,95%	10,20%
50	0	9,74%	22,39 %	0	0,48%	11%	9,31%
60	0	10,66%	23,35 %	0	0,35%	10,59%	10,34%
70	0	10,49%	23,47 %	0	0,46%	11,04%	10,07%
80	0	13,6%	27,22 %	0	0,52%	10,5%	13,14%
90	0	8,85%	21,88 %	0	0,25%	11,38%	8,62%

TABLE IV. RESULTS OF THE HEURISTIC METHODS FOR TYPE II INSTANCES

REFERENCES

[1] Billaut J-C., Lopez P. and Takouti L, *A method for characterizing the set of optimal sequences for some scheduling problems*. 12th Congress of the French OR Society (ROADEF'2011), Saint-Etienne, March 2011.

[2] Billaut J-C., Lopez P., *Characterization of all $\rho-$approximated sequences for some scheduling problems*. IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, art. no. 6059026, 2011.

[3] Billaut J-C., Hebrard E. and Lopez P., *Complete Characterization of Near-Optimal Sequences for the Two-Machine Flow Shop Scheduling Problem*, Ninth International Conference on Integration of Articial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'2012), Nantes, June 2012.

[4] Blazewicz J, Ecker K.H., Pesch E., Schmidt G., Weglarz J., *Scheduling Computer And Manufacturing Processes*. Springer, 2nd edition, 2007.

[5] Bouquard J-L., Lente C, Billaut J-C., *Application of an optimization problem in Max-Plus algebra to scheduling problems*, Discrete Applied Mathematics, 154(15), pp. 2041-2238, 2006.

[6] Chen C-L., Bulfin R.L., *Scheduling unit processing time jobs on a single machine with multiple criteria*, Computers & Operations Research, 17(1), pp. 1-7, 1990.

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

[8] Johnson S.M., *Optimal two- and three-stage production schedules with setup times included*. Naval Research Logistics Quarterly, 1, pp. 61-68, 1954.

[9] Lawler E., emphOptimal sequencing of a single machine subject to precedence constraints. Management Science, 19(8), 544–546, 1973.

[10] Smith W.E, *Various optimizers for single stage production*. Naval Research Logistics Quarterly, 3(1-2):59-66, 1956.

[11] T'Kindt V., Billaut J-C., *Multicriteria scheduling. Theory, models and algorithms*, Springer, 2nd edition, 2006.