FLOWSHOP SCHEDULING WITH DELIVERY DATES AND CUMULATIVE PAYOFFS

Safia Kedad-Sidhoum
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6
F-75005, Paris, France
safia.kedad-sidhoum@lip6.fr

Luciana Pessoa
Department of Industrial Engineering, Pontifícia Universidade
Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225,
Gávea - 22453-900 Rio de Janeiro, RJ, Brasil
lucianapessoa@esp.puc-rio.br

Yasmina Seddik
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6
F-75005, Paris, France
yasmina.seddik@lip6.fr

ABSTRACT

In this paper, we address a new permutation flowshop problem with job release dates and a stepwise job cost function. A mathematical model is proposed. We tackle this strongly NP-hard problem with several heuristic methods: four constructive heuristics, three of them found in the literature, and a new one; as well as some local search methods. Besides, we present two heuristics based on metaheuristics - a GRASP and an ILS. As we are dealing with a new problem, we developed a benchmark with 150 instances. The experimental results show the merit of the developed greedy constructive heuristic, which is competitive to the classical NEH heuristic for flowshop scheduling problems, by finding similar solution quality while presenting greater complexity and running times. Furthermore, we found out that ILS approach outperforms GRASP.

KEYWORDS. Flowshop Scheduling. Heuristics. Metaheuristics.

Main Area: Scheduling, Metaheuristics, Mathematical Programming

1. Introduction

In the *flowshop scheduling problem with delivery dates and cumulative payoffs* a set of n jobs $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ must be sequentially processed without preemption on m machines. Each job J_i must first be processed on machine 1, then on machine 2 and so on until machine m, $i = 1, \dots, n$. In this work, we deal with a permutation flowshop, i.e. a flowshop where jobs are processed in the same order on every machine. Each job $J_j \in \mathcal{J}$ has a release date $r_j \geq 0$ and a processing time $p_{j,l}$ on machine l, $l = 1, \dots, m$. Moreover, a set of K delivery dates $\mathcal{D} = \{D_1, D_2, \dots, D_K\}$ is given, where $0 < D_1 < \dots < D_K$. All parameters are integer.

A schedule S is defined as a vector of starting times of the jobs on each machine: $S = (s_{1,1}, \dots, s_{1,m}, s_{2,1}, \dots, s_{2,m}, \dots, s_{n,1}, \dots, s_{n,m})$, where $s_{j,l}$ is the starting time of job j on machine l in schedule S. Given a schedule S, we denote by $C_{j,l}(S)$ ($C_{j,l}$ for short) the completion time of job j on machine l: $C_{j,l} = s_{j,l} + p_{j,l}$. For shortness, let us denote $C_{j,m}$ by C_j.

The objective value of a given schedule S is equal to $\sum_{j=1}^{n} \mathcal{F}(C_j(S))$, where $\mathcal{F}(t)$ is the number of delivery dates that are greater than or equal to t, for any time instant t. Extending the three-field notation of Graham *et al.* (1979), the problem addressed in this paper can be defined as $F|r_j, perm| \sum_{j=1}^{n} \mathcal{F}(C_j)$. The *flowshop scheduling problem with delivery dates and cumulative payoffs* consists in finding a maximum payoff scheduling S^*.

Figure 1 illustrates a schedule for a permutation flowshop whose instance is defined as follows:

- 3 machines, 4 jobs, 3 delivery dates
- $r_1 = 2, r_2 = 7, r_3 = 9, r_4 = 12$
- $p_1 = (1, 2, 2); p_2 = (3, 2, 2); p_3 = (3, 6, 2); p_4 = (5, 1, 4)$
- $D_1 = 16, D_2 = 21, D_3 = 26$

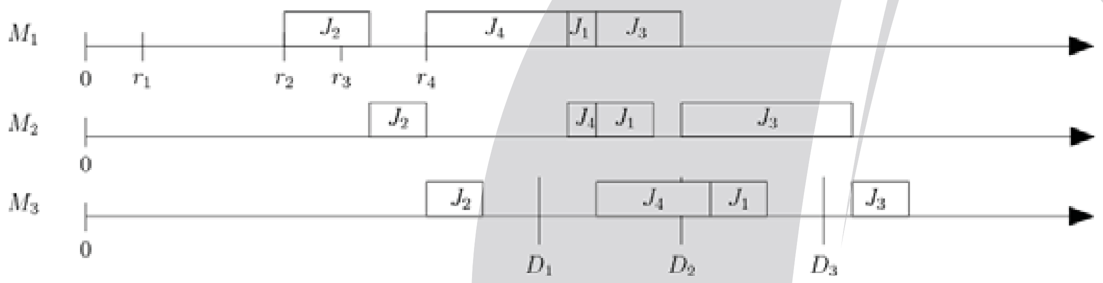

Figure 1: A schedule for a permutation flowshop.

Note that $\sum_{j=1}^{n} \mathcal{F}(C_j)$ is a regular criterion, i.e. left-shifted schedules are dominant. Hence, a schedule can be represented simply by a sequence of jobs. For instance, the schedule of Figure 1 is a left-shifted schedule that can be represented by the sequence (J2, J4, J1, J3), of payoff 4.

This problem is a generalization of the related single machine problem, whose application is in book digitization (Seddik *et al.* (2013)). The book digitization process is linear and includes several steps (although two main ones: digitization and segmentation). While the single machine problem can model a bottleneck machine, a flowshop problem models more accurately the real machine configuration.

Complexity and mathematical formulation $F|r_j, perm|\sum_{j=1}^{n}\mathcal{F}(C_j)$ is strongly NP-hard, since the single-machine problem $1|r_j|\sum_{j=1}^{n}\mathcal{F}(C_j)$, defined by Seddik *et al.* (2013), is strongly NP-hard. The special case with two machines, no release dates and a common delivery date $F2|D_i = D, perm|\sum_{j=1}^{n}\mathcal{F}(C_j)$, equivalent to problem $F2|perm|\sum C_j$, is already NP-hard, as shown by Della Croce *et al.* (2011).

We propose a Mixed Integer Linear Program (MILP) for this problem, formulated with positional variables. According to Della Croce *et al.* (2011), this formulation is superior to other models based on disjunctive variables and constraints, for $F2|perm|\sum C_j$. The mathematical formulation of $F|r_j, perm|\sum_{j=1}^{n}\mathcal{F}(C_j)$ is the following, where the binary decision variable $X_{i,j}$ is equal to 1 if job J_i is the j-th job of the sequence and $X_{i,j} = 0$ otherwise. Variable $C_{j,l}^{pos}$ denotes the completion time of the job on the j-th position on machine l, $j = 1, \ldots, n, l = 1, \ldots, m$. C_j^{pos} denotes $C_{j,m}^{pos}$.

$$\max \sum_{j=1}^{n} \mathcal{F}(C_j^{pos}) \tag{1}$$

s.t.

$$\sum_{i=1}^{n} X_{ij} = 1, \;\; j = 1, \ldots, n, \tag{2}$$

$$\sum_{j=1}^{n} X_{ij} = 1, \;\; i = 1, \ldots, n, \tag{3}$$

$$C_{j,1}^{pos} \geq \sum_{i=1}^{n} (p_{i,1} + r_i) X_{i,j}, \;\; j = 1, \ldots, n, \tag{4}$$

$$C_{j,l}^{pos} \geq C_{j-1,l}^{pos} + \sum_{i=1}^{n} p_{i,l} X_{i,j}, \;\; j = 2, \ldots, n, \;\; l = 1, \ldots, m, \tag{5}$$

$$C_{j,l+1}^{pos} \geq C_{j,l}^{pos} + \sum_{i=1}^{n} p_{i,l+1} X_{i,j}, \;\; j = 1, \ldots, n, \;\; l = 1, \ldots, m-1, \tag{6}$$

$$C_{j,l}^{pos} \geq 0, \;\; j = 1, \ldots, n, \;\; l = 1, \ldots, m. \tag{7}$$

$$X_{i,j} \in \{0, 1\}, \;\; i = 1, \ldots, n, \;\; j = 1, \ldots, n. \tag{8}$$

The objective function is stated in (1) where:

$$\mathcal{F}(C_j^{pos}) = \begin{cases} 0 & \text{if} \;\; D_K < C_j^{pos} \\ 1 & \text{if} \;\; D_{K-1} < C_j^{pos} \leq D_K \\ \vdots & \\ K & \text{if} \;\; 0 < C_j^{pos} \leq D_1 \end{cases}$$

Constraints (2) and (3) ensure that each position is attributed to one job and each job is processed exactly once. Constraints (4) ensure that on the first machine job j cannot start before its release date. Constraints (5) forbid the overlapping of a job and its predecessor on a machine. Constraints (6) establish that a job cannot be processed on a machine before its completion on the preceding machine. Constraints (7) and (8) give the domain definition of the variables.

Related work To the best of our knowledge, no flowshop problem with stepwise job cost functions has been studied yet. Indeed, such objective function was only studied on single machine (Seddik *et al.* (2013), Detienne *et al.* (2012)) and parallel machines (Detienne *et al.* (2011), Curry and Peters (2005)). However, the algorithms for single and parallel machines can not be easily adapted to flowshop because they often rely on structural properties that do not apply any more.

The most closely related problems are flowshops with release dates and regular sum objective functions $\sum_{i=1}^{n} f_i(C_i)$. Rakrouki and Ladhari (2009) consider $F2|r_i|\sum C_i$, for which they propose some Branch and Bound methods, solving instances with up to 100 jobs for some classes of instances and 30 jobs for the hardest class of instances. Ladhari and Rakrouki (2009) consider the problem $F2|r_i, perm|\sum C_i$, for which they develop lower bounds, heuristics and a genetic algorithm. Other related problems are flowshop problems with regular sum objective functions $\sum_{i=1}^{n} f_i(C_i)$, but without release dates. Della Croce *et al.* (2011) propose a matheuristic for solving the two machine flowshop problem $F2||\sum C_i$; Della Croce *et al.* (2000) solve $F2|d_i = d|\sum U_i$ with a Branch and Bound method; and Vallada *et al.* (2008) compare several heuristics for solving $F||\sum T_i$.

In the remaining of this paper, we propose heuristic methods for the flowshop scheduling problem with delivery dates and cumulative payoffs. Section 2 is devoted to constructive methods. Local search operators are presented in Section 3. Section 4 shows computational experiments on constructive and local search methods. Results are used to customize GRASP and ILS heuristics, described in Sections 5 and 6 respectively, for the problem tackled in this paper. Then, a comparative evaluation of GRASP, ILS and a solving method based on the commercial CPLEX solver is presented in Section7, and concluding remarks are provided in Section 8.

2. Constructive Heuristics

In this section, we describe four constructive heuristics for the flowshop scheduling with delivery dates and cumulative payoffs: three of them found in the literature, and a new one.

R (Release dates) This heuristic produces a left-shifted schedule where jobs are ordered by increasing order of their release dates. This $O(n \log n)$ method was first described by Potts (1985) for the two-machine permutation flow-shop problem of minimizing the maximum completion time.

ECT (Earliest Completion Time) This heuristic produces a left-shifted schedule where jobs are ordered w.r.t. their earliest possible completion time $E_i = r_i + \sum_{l=1}^{m} p_{il}$. This $O(n \log n)$ algorithm was proposed by Ladhari and Rakrouki (2009) for the minimization of the total completion time on a two-machine flowshop problem with release dates.

NEH (Nawaz, Enscore and Ham) The steps of the NEH-based algorithm (Nawaz *et al.* (1983)) are the following:

1. Sort the jobs in the increasing order of $E_i = r_i + \sum_{l=1}^{m} p_{il}$
2. Consider the two jobs J_{i_1}, J_{i_2} with the smallest values of E_i. Among the two left-shifted partial schedules corresponding to sequences (J_{i_1}, J_{i_2}) and (J_{i_2}, J_{i_1}), choose the one with the greatest payoff as the current schedule.
3. Repeat until all jobs are scheduled: Let J_e be the unscheduled job with the smallest value of E_i. Update the current schedule, by inserting J_e, in the position that maximizes the payoff, while keeping a left-shifted partial schedule.

Step 3 concentrates the largest computational effort of the algorithm. For each tentative position, the method evaluates the solution cost in $O(nm)$ steps. As each job can be inserted in $O(n)$ positions, the complexity of NEH heuristic for the problem under consideration is $O(n^3m)$.

The NEH heuristic, originally proposed by Nawaz *et al.* (1983) for a flowshop problem without release dates, was modified by Ladhari and Rakrouki (2009) by considering the sorting criterion described above. The method described in Ladhari and Rakrouki (2009) is easily adapted for the addressed problem by simply changing the payoff evaluation.

IECT (Iterated Earliest Completion Time) Following the ideas of NEH and ECT methods, we developed a new iterative construction method. A jobs sequence is built from scratch and the first added job is that with the smallest value of E_j, as defined for the ECT method. After that, at each iteration, the earliest possible completion time of all jobs that are not in the partial schedule are reevaluated by considering the completion time, on each machine, of the last job added in the schedule. Algorithm 1 describes this $O(n^2m)$ method.

The schedule (solution) S and the list L of jobs indices are initialized at lines 2 and 3, respectively. The earliest possible completion time is computed at lines 4 to 6 for all jobs. The index j^* of the job with the earliest possible completion time is identified at line 7. Then, the corresponding job J_{j^*} is inserted in the sequence at line 8, where $S.J_{j^*}$ denotes the concatenation of job J_{j^*} at the end of S. Index j^* is, then, removed from the job list at line 9. The loop at lines 10 to 20 adds one job at a time to the sequence, until all jobs are scheduled. The loop at lines 11 to 16 calculates the earliest possible completion time of all jobs indexed in L. Line 12 considers the first machine and calculates the earliest possible completion time E_{j1} of job J_j by adding its processing time to the maximum value between its release date and the completion time on machine 1 of the last job in the schedule. Remaining machines are treated by the loop at lines 13 to 15. Line 14 calculates the earliest possible completion time of job J_j on machine l by adding its processing time to the maximum value between its predecessor on the machine under consideration and its own completion time on the preceding machine. At lines 17 to 20, respectively, the job with the smallest possible completion time in the last machine is identified, concatenated to the partial schedule, and its index is removed from L.

```
1  ConstructiveHeuristicIECT
2  $S \leftarrow \emptyset$;
3  $L \leftarrow \{1, \cdots, n\}$;
4  forall $j \in L$ do
5      $E_j = r_j + \sum_{l=1}^{m} p_{jl}$;
6  end
7  $j^* \leftarrow \text{argmin}\{E_j : j \in L\}$;
8  $S \leftarrow S.J_{j^*}$;
9  $L \leftarrow L \setminus \{j^*\}$;
10 while $L \neq \emptyset$ do
11     forall $j \in L$ do
12         $E_{j1} \leftarrow max\{r_j, C_{j^*,1}\} + p_{j1}$;
13         for $l = 2, \ldots, m$ do
14             $E_{jl} \leftarrow max\{C_{j^*,l}, E_{j,l-1}\} + p_{jl}$;
15         end
16     end
17     $j^* \leftarrow \text{argmin}\{E_{jm} : j \in L\}$;
18     $S \leftarrow S.J_{j^*}$;
19     $L \leftarrow L \setminus \{j^*\}$;
20 end
```

Algorithm 1: Constructive Heuristic IECT

3. Local Search Methods

Local search methods attempt to find cost-improving solutions by exploring the neighborhood of an initial solution. If none is found, then the search returns the initial solution as a local minimum. Otherwise, if an improving solution is found, it is made the new initial solution, and the procedure repeats itself (Talbi (2009)). We use here the *best improvement strategy*: the new initial solution is the best one in the neighborhood of the previous solution.

The neighborhoods explored in this work are based on those standard ones studied by Den Besten and Stützle (2011) for scheduling problems. In the *interchange* neighborhood, a neighbor solution is obtained by swapping two jobs J_i and J_j at the ith and jth positions, resulting in a neighborhood size of $n(n-1)/2$. The *insert* neighborhood attempts to move a job J_i to a new position that improves the solution cost. For each job, $n-1$ positions can be verified, therefore the whole neighborhood contains $n(n-1)$ solutions. For both methods, the cost of each solution is verified in nm steps which make the local search a $O(n^3m)$ method.

Neighborhoods can be combined in order to better explore the solution space. Variable Neighborhood Descent (VND) establishes an order in which the local search neighborhoods will be explored. Starting from a solution generated by the constructive algorithm, a VND exploits the first neighborhood until a local optimum is found. This solution becomes the initial solution to the next neighborhood, and the sequence continues until the last neighborhood is reached. If, during the investigation in the local search algorithm, an improving solution is found, then the procedure returns to the first neighborhood.

4. Computational experiments on constructive and local search methods

In this section, we present an experimental evaluation of the described heuristics, and we compare them to the results obtained by solving the ILP formulation.

The ILP formulation was solved with IBM ILOG CPLEX solver and the function STEP embedded in the OPL language was used to implement the objective function. CPLEX was executed on a 3.33 GHz Intel Core2-Duo processor running Ubuntu and stopped when an optimal integer solution is found, or when the CPU time limit of 6 hours is reached. Heuristics were implemented in C++ and the experiments were performed on a 3.40 GHz Intel Core i7 processor running Ubuntu.

4.1. Instances generation

We randomly generated 150 test instances for the problem addressed in this work. Each instance contains 100 jobs which must be processed on 2 machines. Processing times are random integers from a uniform distribution in the interval $[1, 100]$. The number of delivery dates varies in $\{1, 2, 3, 5, 7, 10\}$. Delivery dates for each instance are defined as follows. Let C be the makespan of a schedule obtained by using Johnson's rule (Brucker (2007)), while ignoring release dates. The first delivery date $D_1 = \lfloor(\alpha \times C)/K\rfloor$, where α is a parameter, $\alpha \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The other delivery dates are computed as: $D_k = k \times D_1, 1 < k \leq K$. Following Seddik *et al.* (2013), the release date of each job is picked randomly in one of the intervals $r_k = [D_k, D_k + R \times D_k)$, with $D_0 = 0$ and $k = 1, \ldots, K$. R is a parameter taking values in $\{0.1, 0.3, 0.5, 0.7, 1.0\}$. The choice of the interval is also random (each with probability $1/K$).

4.2. Comparative metrics

We used the following metrics to compare the heuristics:

- BestValue: for each instance, BestValue is the best solution value obtained over all executions of the methods considered.
- Dev: for each method, Dev is the relative deviation in percentage between CPLEX solution (V_c) and BestValue (V_b). $Dev = 100 * (V_b - V_c)/V_c$. Therefore, Dev can be negative if BestValue is worse (lower) than the CPLEX solution.
- AvgDev: average value of Dev over all instances of a method.

- #Imp: for each method, this metric gives the number of instances whose solution value is better than CPLEX solutions. In our experiments, CPLEX was able to find 8 optimal solutions out of 150 instances.
- Time: for each method, this metric gives the average computation time (in seconds) over all instances.

4.3. Numerical results

Table 1 displays a summary of the results obtained by the constructive methods defined in Section 2. The results show that methods R and ECT, which builds a solution by just following some sorting criterion, yield the worst average relative deviation (-26.15% and -19.30%, respectively) although their average processing times are less than 1 ms. The quality of the solutions obtained by NEH and IECT methods are similar. Their AvgDev difference is less than 1, however the NEH complexity makes its average processing time to reach 0.34 seconds while IECT average processing time is less than 1 ms. Note that negative AvgDev values mean that, in average, the method finds worse solutions in comparison with CPLEX solutions. Regarding the total number of improved solutions, NEH gets the best result by improving 42 solutions, while the other methods improve about 25 solutions.

Table 1: Summary of the constructive methods

Method	AvgDev(%)	# Imp	Time
R	-26.15	24	0.00
ECT	-19.30	25	0.00
NEH	-6.25	42	0.34
IECT	-7.05	27	0.00

Table 2 displays a summary of the results obtained by the local search methods defined in Section 3. Since heuristics NEH and IECT present the best and similar quality results, we use these two heuristics to compute initial solutions for local search methods. AvgDev results for Interchange local search (Itc) application are similar, around -3.7%, for both constructive methods. Insertion local search (Ist) combined with NEH was the least effective method. Its quality results are almost one percentage point worse than those obtained by the use of the Interchange method. Note that these three schemes obtained negative AvgDev values, which means that, in general, they found worse solutions than CPLEX solutions as confirmed by #Imp metric. The most effective results were obtained by applying the Insertion local search on the IECT solutions, despite consuming longer running times. This scheme was the only one to reach a positive AvgDev metric and it was able to find better solutions than CPLEX for 87 instances. Besides, by comparing these results with those displayed in Table 1, we note that Insertion local search was able to improve by more than 12 percentage points the solutions obtained by the IECT constructive methods.

Table 2: Summary of the local search methods

Method	AvgDev(%)	# Imp	Time
NEH Itc	-3.70	50	0.59
NEH Ist	-4.99	43	4.71
IECT Itc	-3.76	36	0.49
IECT Ist	4.93	87	11.13

In the next experiment, we combine Interchange and Insertion neighborhoods in a VND method, so that two VND variants are considered by changing the order of the Interchange and Insertion neighborhoods. The initial solution is computed with IECT, since this method shows the best results in the previous experiments. Table 3 shows that the order in which local search operators are applied does not affect the solution quality. Besides, by comparing these results with those showed in Table 2, we note only a small improvement on the solutions quality.

Table 3: Summary of the VND methods applied to IECT solutions

Method	AvgDev(%)	# Imp	Time
IECT Itc+Ist	5.06	85	12.06
IECT Ist+Itc	5.18	87	13.02

5. GRASP

The *Greedy Randomized Adaptive Search Procedures* (GRASP) is a multi-start metaheuristic which consists of applying local search to feasible starting solutions generated with a greedy randomized construction heuristic. A recent review of GRASP can be found in Resende and Ribeiro (2010).

In this section, we specialize GRASP into a heuristic for the flowshop scheduling with delivery dates and cumulative payoffs.

The construction phase of GRASP is a randomized variant of the IECT greedy algorithm. At each iteration, the jobs not in the solution are still evaluated by the greedy function E_j, as defined above. However, instead of choosing the job with the earliest possible completion time, the randomized algorithm first identifies the minimum (E^-) and maximum (E^+) greedy function values of the candidate elements. Then, a restricted candidate list (RCL), formed by all candidate elements whose greedy function value is less than or equal to $E^- + \alpha(E^+ - E^-)$, is built. A job index j^r is chosen at random from the RCL and the corresponding job J_{j^r} is inserted in the scheduling. Parameter α used in the GRASP construction phase was set to 0.1 since we observed in preliminary experiments that this value led to the best results.

Solutions built with the randomized greedy algorithm are not guaranteed to be locally optimal, even with respect to simple neighborhood structures. Therefore, the application of local search to such a solution usually results in an improved locally optimal solution.

In the GRASP local search, the method Insertion is followed by the Interchange in a VND strategy. As described in the previous section, this approach led to the best average solution.

6. Iterated Local Search

Iterated Local Search (ILS) is a metaheuristic which builds a sequence of solutions by iteratively applying perturbations and local search methods from an initial solution. Lourenço *et al.* (2003) review some applications of ILS and describe the main parts of this method. Applications of ILS for flowshop problems can be found in Den Besten *et al.*(2001), Dong *et al.*(2009) and Stützle (1998).

According to Den Besten *et al.* (2001), a basic ILS algorithm is built with four components: (1) a method to generate an initial solution, (2) a perturbation method, to allow the ILS to escape local optima, (3) a local search procedure, and (4) an acceptance criterion, to decide if a solution will replace the current one to the next iteration.

The method to generate an initial solution can be either a randomized method or a greedy constructive heuristic. Both can be combined with a local search. Once an initial solution is created, the method execute steps (2), (3) and (4) in a loop until a stopping condition is reached. Perturbations are applied to the current local minimum and modify a number of solution elements. The higher the number of modified elements, the greater the strength of the perturbation, which must be carefully studied in order to avoid random restarts or falling back into previous solutions just visited. The perturbation method together with the local search brings a new solution that replaces the current one to the next loop iteration if an acceptance criterion is met. This component may range between a strong intensification criterion, by only accepting improving solutions, and a strong diversification, by accepting any new solution, not taken its cost into consideration.

An ILS heuristic for the flowshop scheduling with delivery dates and cumulative payoffs has the following components:

(1) the IECT constructive heuristic is used to generate the initial solution,

(2) the perturbation method consists of applying eight swap moves. We alternate a move that swaps two neighbor jobs (J_i, J_{i+1}) and another move that swaps a no neighbor pair of jobs (J_i, J_j), at the ith and jth random positions.

(3) it uses the method Insertion followed by Interchange in a VND strategy, for the local search.

(4) the acceptance criterion allow only better solutions to replace the current one.

7. Comparative evaluation of GRASP, ILS and CPLEX

In this section, we compare the solution quality obtained by CPLEX with the results of the ILS and GRASP algorithms. The experimental framework is identical to the one described in Section 4. To evaluate GRASP and ILS, eight runs were carried out for each instance, varying the initial seed given to the random number generator. The algorithms stop when it reaches 10 minutes of CPU time.

Table 4 brings the results of GRASP heuristic, which also can be compared to the ILS heuristic. Comparing the results of AvgDev($\%$) and #Imp metrics with that ones presented in Table 3, we note that GRASP performed similarly to the algorithm based on a greedy construction although is more time consuming. Metrics MinDev($\%$) and MaxDev($\%$) in Table 4 give the maximum and minimum value of Dev over all instances of a method. In the worst case, ILS found a solution 5.13$\%$ inferior than the CPLEX solution while the GRASP worst solution is 6.94$\%$ worse than CPLEX. MaxDev($\%$) show that the best improvement over all instances was obtained by ILS that was able to find a solution 47,90$\%$ better than the CPLEX solution.

Table 4: Summary of the GRASP and ILS results in comparison with CPLEX solutions

Method	AvgDev($\%$)	MinDev($\%$)	MaxDev($\%$)	# Imp
GRASP	5.37	-6.94	44.75	87
ILS	7.21	-5.13	47.90	95

As mentioned before, CPLEX ran 6 hours and it was able to find 8 optimal solutions out of 150 instances. For those solutions whose optimality was not proven, primal-dual gaps vary up to 527$\%$. Table 5 shows the top 10 and bottom 10 instances, regarding their CPLEX percentage gaps. In the first column, the instance name brings the details about its generation as explained in Section 4. For example, instance *k2n100a0.1r0.7* has $k = 2$ delivery dates, $n = 100$ jobs, parameter α, used to give the first delivery date, is $a = 0.1$ and $r = 0.7$, which is the same parameter R used to define the release date of each job. Columns 2 and 3 give, respectively, the best value and the percentual primal-dual gap found by CPLEX for each instance. Finally, columns 4 and 5 show the percentual relative deviation from the solutions found, respectively, by GRASP and ILS to the solutions found by CPLEX.

We noted that ILS was more effective than GRASP in finding better solutions than CPLEX, for those instances whose optimal solution is not known. Besides, ILS was able to find solutions known as optimal, but GRASP did not find any optimal solution.

It is worth to mention that when the deviation is positive, CPLEX could not provide the optimal solution within the CPU time limit. Some experiments are currently driven to assess the quality of the solution provided by the heuristics by comparing the solution values to upper bounds.

Table 5: Detailed results of GRASP and ILS comparison with CPLEX for 20 instances

Instance	CPLEX Best value	CPLEX % gap	GRASP % Dev	ILS% Dev
k2n100a0.1r0.5	30	0.00	-6.67	-3.33
k2n100a0.1r0.7	30	0.00	-3.33	0.00
k3n100a0.1r0.1	40	0.00	-2.50	0.00
k3n100a0.1r0.3	35	0.00	-5.71	0.00
k3n100a0.1r0.5	39	0.00	-5.13	-5.13
k3n100a0.1r0.7	40	0.00	-2.50	-2.50
k7n100a0.1r0.5	84	0.00	-1.19	0.00
k7n100a0.1r0.7	77	0.00	-2.60	0.00
k5n100a0.1r0.1	55	0.01	-3.64	0.00
k10n100a0.1r0.1	111	0.01	-1.80	0.00
k5n100a0.1r0.5	69	265.21	0.00	1.45
k1n100a0.1r0.3	22	268.18	0.00	0.00
k10n100a0.1r0.5	118	289.22	0.85	3.39
k7n100a0.3r0.1	161	290.02	4.97	7.45
k5n100a0.3r1.0	112	296.43	15.18	17.86
k1n100a0.1r0.7	18	311.11	0.00	0.00
k5n100a0.3r0.7	108	312.04	11.11	12.96
k10n100a0.3r0.7	194	326.21	15.46	15.98
k10n100a0.3r0.3	209	355.86	12.44	12.44
k7n100a0.1r0.3	87	527.42	8.05	9.20

8. Concluding remarks

In this paper, we introduced a new permutation flowshop problem with a stepwise job cost function as well as its mathematical model. A new greedy constructive heuristic for flowshop was described. Experiments showed that the proposed method is competitive to the classical NEH heuristic for flowshop, by finding similar solution quality while presenting greater complexity and running times, in particular when combined to local search operators.

Two algorithms based on GRASP and ILS metaheuristics were developed. We would like to point out that these methods deserve further investigation, and thus that the results presented here are intended as preliminary results. However, computational experiments showed ILS is a promising approach since it was able to find better solutions than GRASP.

Another research avenue to enhance the current work on the permutation flowshop is to attempt to identify structural properties, and especially designing good upper bounds (for more than two machines) in order to better evaluate the heuristic methods.

9. Acknowledgements

This study was partially supported by PNPD/Capes (PNPD20131046-31005012014P4) and FUI project Dem@tFactory, financed by DGCIS, Conseil général de Seine-et-Marne, Conseil général de Seine-Saint-Denis, Ville de Paris, Pôle de compétitivité Cap Digital.

References

Brucker, P. *Scheduling algorithms*, Berlin: Springer, 2007.
Curry, J., and Peters, B. (2005). Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. *International Journal of Production Research*, 43(15), 3231-3246.
Della Croce, F., Grosso, A. and Salassa, F. (2011). A matheuristic approach for the total completion time two-machines permutation flow shop problem. In Evolu- tionary Computation in Combinatorial Optimization, *LNCS*, 38-47.
Della Croce, F., Gupta, J.N.D., and Tadei, R. (2000). Minimizing tardy jobs in a flowshop with common due date. *European Journal of Operational Research*, 120(2), 375-381.
Den Besten, M. and Stützle, T. (2011), Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling, *Proceedings of The Fourth Metaheuristics International Conference (MIC2001)*, 545-549.
Den Besten, M., Stützle, T., and Dorigo, M. (2001), Design of iterated local search algorithms, *Applications of Evolutionary Computing*, 441-451.
Detienne, B., Dauzère-Pérès, S., and Yugma, C. (2011), Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling*, 14(6), 523-538.
Detienne, B., Dauzère-Pérès, S., and Yugma, C. (2012), An exact approach for scheduling jobs with regular step cost functions on a single machine. *Computers & Operations Research*, 39(5), 1033-1043.
Dong, X., Huang, H., and Chen, P. (2009), An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, *Computers & Operations Research*, 36(5), 1664-1669.
Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G., Optimization and approximation in deterministic sequencing and scheduling: a survey, in Hammer, P.L., Johnson, E.L., and Korte, B.H. (Eds.), *Annals of Discrete Mathematics*, 5, Elsevier, 287-326, 1979.
Ladhari, T. and Rakrouki, M. A. (2009), Heuristics and lower bounds for minimizing the total completion time in a two-machine flowshop, *International Journal of Production Economics*, 122(2), 678-691.
Lourenço, H. R., Martin, O. C., and Stützle, T., *Iterated local search*, Springer, USA, 2003.
Nawaz, M., Enscore Jr, E.E. and Ham, I. (1983), A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega*, 11(1), 91-95.
Potts, C. N. (1985), Analysis of heuristics for two-machine ow-shop sequencing subject to release dates, *Mathematics of Operations Research*, 10(4), 576-584.
Rakrouki, M. A., and Ladhari, T. (2009), A branch-and-bound algorithm for minimizing the total completion time in two-machine flowshop problem subject to release dates. *International Conference on Computers & Industrial Engineering, IEEE*, 80-85.
Resende, M. G. C. and Ribeiro, C. C.(2010). Greedy randomized adaptive search procedures: advances and applications. In Gendreau, M., Potvin, J.-Y.,(Eds.) *Handbook of Metaheuristics (2nd edn).* Springer Science and Business Media, New York, 281-317.
Seddik, Y., Gonzales, C. and Kedad-Sidhoum, S. (2013). Single machine scheduling with delivery dates and cumulative payoffs. *Journal of Scheduling*, 16(3), 313-329.
Stützle, T. Applying iterated local search to the permutation flow shop problem, *FG Intellektik*, TU Darmstadt, 1998.
Talbi, E.-G., *Metaheuristics - from design to implementation*, Wiley, 2009.
Vallada, E., Ruiz, R. and Minella, G. (2008), Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics, *Computers & Operations Research*, 35(4), 769-780.