International Journal of Sciences: Basic and Applied Research (IJSBAR)

ISSN 2307-4531
(Print & Online)

http://gssrr.org/index.php?journal=JournalOfBasicAndApplied

Minimizing Total Tardiness in the m-Machine Flow-Shop Problem by Heuristic Algorithms

Quang Chieu Ta*

Faculty of Information Technology, Hanoi University of Mining and Geology, Duc Thang, Bac Tu Liem, 100000 Ha Noi, Viet Nam

Email: quangchieu.ta@gmail.com

Abstract

In this work the m-machine permutation flow-shop problem has been considered. The permutation flow-shop scheduling problem where a set of jobs have to be scheduled on a set of machines in the same order. We propose heuristic algorithms for the flow-shop problem to minimizing the total tardiness. A new genetic and Tabu search algorithm which initialized by the solution of EDD, NEH and EN algorithm. Computational experiments are performed on benchmark instances and the results show the good performances of these methods. Finally, some future research directions are given.

Keywords: Flowshop; Tardiness; Tabu search; Scheduling; Heuristic; Genetic algorithm.

1. Introduction

We consider in this paper the permutation flow-shop scheduling problem, one of the most famous scheduling problems. We consider that there is a set J = $\{J_1, ..., J_n\}$ of n jobs to schedule on a set M = $\{M_1, ..., M_m\}$ of m machines. A machine can process only one job at a time and we assume that the machines are immediately available. All the jobs have the same routing, they are processed in the same order, i.e. on machine M_1 first and then on machine M_2, M_3, etc. Also we assume that the sequence of jobs on each machine is the same. We denote by $p_{i,j}$ the processing time of J_j on machine M_i and d_j is the due date of J_j. Variable C_j denotes the completion time of job J_j and variable T_j its tardiness, dened by $T_j = \max(C_j - d_j, 0)$, $\forall j$, $1 \leq j \leq n$.

---

* Corresponding author.

The objective is to minimize the total tardiness denoted by $\sum T_j = \sum_{j=1}^{n} T_j$. The problem is classically denoted by $F|prmu|\sum T_j$ [1], where *prmu* indicates a "permutation flow-shop" (same sequence on each machine). This problem is known to be *NP-hard* in the ordinary sense when there is a single machine and *NP-hard* in the strong sense for m ≥ 2 [2, 3].

The literature contains a lot of papers dealing with this problem, some of them dealing with the particular case of two machines. In the case of two machines, some exact methods have been proposed such as branch-and-bound algorithms [4–6]. In [7], instances with up to 24 jobs can be solved to optimality, which shows the diculty to solve this problem with only two machines. Some heuristic approaches have been proposed, such as greedy heuristics using priority rules or inspired by NEH algorithm [8], and a shifting bottleneck procedure [9]. Some metaheuristics have also been proposed in the literature, such as simulated annealing [10], tabu search algorithms [11–15], genetic algorithms [16], particle swarm optimization [17, 18]. Della Croce and his colleagues [19] and Ta and his colleagues [20], the authors propose a matheuristic method for this problem, etc.

For the *m*-machine flow-shop scheduling problem, Onwubolu and Mutingi propose in [16] a genetic algorithm minimizing a combination of the total tardiness and the number of tardy jobs. In the survey of Vallada, Ruiz and Minella [21], a lot of algorithms are implemented and compared. A neighborhood search algorithm based on the permutation of blocks of consecutive jobs seems to be one of the most ecient methods. In [22], the authors propose three genetic algorithms including advanced techniques such as path relinking, local search, and a procedure to control the diversity of the population. Victor Ferrnadez-Viagas and his colleagues [23] propose several tie-breaking mahanisms for the NEH to solve the problem. We do not mention the wide literature concerning flow-shop problems with total completion time minimization (equivalent to the total tardiness if due dates are all equal to 0), but a lot of exact and approximate methods have also been proposed. The interested reader can find a more complete state-of-the art survey on the *m*-machine flow-shop problem with total tardiness and makespan minimization in [21, 24].

In this paper, we propose several genetic and Tabu search algorithms which initialized by the solution of a EDD, NEH algorithm. The solutions of the Tabu search algorithm are compared to the solutions of the genetic algorithm. For the evaluation, 108 benchmark instances proposed in [21] have been used. The rest of the paper is organized as follows. In Section 2, the resolution methods are described. In Section 3 reports the settings of the methods and the computational results. A conclusion and some future research directions are proposed in Section 4.

2. Resolution methods

In this section, we propose several heuristics and metaheuristic algorithms. Two basic heuristic algorithms, EDD and NEH, that run in $O(n \log n)$ time. We present two metaheuristics developed for solving our problem. The first is a genetic algorithm, the second is a Tabu search. We give some basic notions on algorithms and then we describe our implementation.

2.1. EDD algorithm

EDD *(Earliest Due Date)*: jobs are sorted in the due date non decreasing order, i.e. $d_{[1]} \leq d_{[2]} \leq ... \leq d_{[n]}$ where $d_{[k]}$ is the due date of the job in position *k*. The algorithm is described in **Table 1**

Table 1: EDD algorithm

Algorithm 1: EDD algorithm
1: Input: S = a set of jobs,
2: Sorted: the jobs by non decreasing order of d_j ,
3: Output: A set of jobs sorted in non decreasing order of d_j

2.2. NEH algorithm

In [8], the authors develop NEH heuristic for *m*-machine flow-shop scheduling problem with makespan minimization. We propose and apply the method for minimizing the total tardiness for the m-machine permutation flow shop scheduling problem. NEH algorithm is described in details for the problem below (see **Table 2**).

Table 2: NEH algorithm

Algorithm 2: NEH algorithm
1: Input: S = jobs sorted in the decreasing order of P_j,
2: where $P_j = \sum_{i=1}^{m} P_{i,j}, \forall j = 1, \dots, n$
3: Consider the partial sequence with minimum total tardiness and minimum makespan in case of ties among $\{(S_{[1]}, S_{[2]}), (S_{[2]}, S_{[1]})\}$
4: for $k = 3$ to n do
5: Test the insertion of $S_{[k]}$ at any possible position in S' from 1 to $k + 1$
6: Keep the best insertion, i.e. the insertion with minimum ΣT_j , and the insertion with minimum makespan in case of ties.
7: end for

2.3. Genetic algorithm

- Principles of a genetic algorithm

Genetic algorithms (GA) have been originally proposed by Holland [25]. This is a general search technique where a population composed by individuals evolves following nature inspired mechanisms called “genetic operators”. The population is composed by individuals that are valuated by a fitness, which is often related to the objective function.

Starting from an initial population, new solutions are generated by selecting some “parents” randomly, but with

a probability growing with fitness, and by applying genetic operators such as *selection*, *crossover* and *mutation*, which introduces random modifications. Some existing solutions are randomly selected for crossover, some solutions are selected for mutation, and a new population of the same size is obtained. The process is repeated until a given stopping criterion is reached, e.g. a time limit or when a sufficiently satisfactory solution has been found.

Genetic algorithms have been largely used for solving scheduling problems. According to [26], the main steps of a genetic algorithm are:

1. Generation of the *initial population* P_0,

2. Evaluation of the *fitness* of each individual,

3. *Selection* of the individual couples in population P_{k-1},

4. Application of the *crossover* operator: with a probability ρ_c, two individuals of P_{k-1} will be crossed to create two new individuals in a set C_k,

5. Application of the *mutation* operator: with a probability ρ_m, each individual is modied by a mutation and inserted in a set M_k,

6. Replace population P_{k-1} by population P_k: P_k contains the *PopSize* best individuals of $P_{k-1} \cup M_k \cup C_k$.

7. Repeat the process at step 2 until a stopping condition is satisfied.

A genetic algorithm is designed by several parameters of high importance. First of all, there are several ways for coding a solution. In our scheduling problem, solving the problem is equivalent to finding a sequence of jobs, and it is generally convenient to consider that an individual is exactly this sequence. This is called in the litterature “direct encoding” because an individual corresponds to a solution without ambiguity. For more complicated scheduling problems such as job-shop or parallel machine problems, an individual may represent a list of jobs, but an algorithm has to be used to determine the corresponding solution. This is called in the literature “undirect encoding” because an individual does not correspond “immediately” to a solution.

The other key points in a genetic algorithm are the crossover and the mutation operators. The literature contains a lot of definitions, strongly related to the coding definition. For classical scheduling problems, the most famous crossover operators are *1*-point crossover up to *k*-point crossover. Mutation generally consists in changing arbitrarily an element of an individual. Fixing the probabilities ρ_c of crossover and ρ_m of mutation is not an easy task. It is generally done after some preliminary computational experiments on a subset of the data set. A survey of the applications of genetic algorithms to scheduling problems can be found in [27].

- Genetic operators

Coding: The crucial step in designing a Genetic algorithm is to de ne an encoding, i.e. a way to represent a solution. In the case of the *m*-machine permutation flow shop scheduling problem with n jobs indexed from *1* to *n*, an individual is represented by a *permutation*.

Initial population: The initial population P_0 contains *PopSize* individuals. One individual is obtained by sequencing the jobs according to a given rule. The other individuals are randomly generated. The way that the first individual is generated leads to different versions of the algorithm. If the first individual is given by *EDD* rule (see **Table 1**), the method is called in the following GA_{EDD}. If the initial sequence is given by applying an adaptation of *NEH* algorithm [8] (described in **Table 2**), the method is called in the following GA_{NEH}. Finally, if one initial sequence is given by the best among *EDD* and *NEH*, the method is called GA_{EN}.

Fitness: The fitness of an individual S is the value of the objective function ΣT_j *(S)* of the corresponding sequence.

Crossover: Severla crossover operators are used: the *one-point crossover* (X1) [28] and the *linear order crossover* (LOX) [28], the *Similar Job Order Crossover* or (SJOX), the *Similar Block Order Crossover* or (SBOX) and the *Similar Block 2-Point Order Crossover* or (SB2OX) [29]. The operators are described the follow:

-X1: One crossover point is randomly generated. Let *A = A1 // A2* and *B = B1 // B2* be the two parents. Two offsprings are calculated. Offspring 1 denoted by *O1* contains the jobs of *A1* in the order of *A* and the jobs of *A2* in the order of *B*. Offspring 2 denoted by *O2* contains the jobs of *B1*in the order of *B* and the jobs of *B2* in the order of *A*.

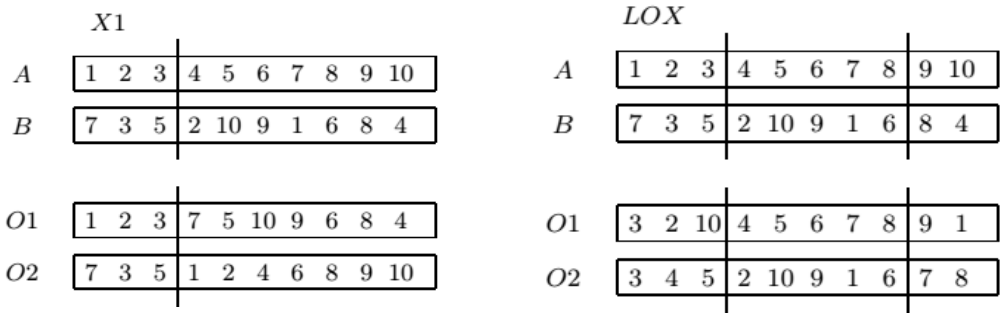

Figure 1: Illustration of two crossover operators

-LOX: Two different crossover points are randomly generated. Let A = A1 // A2 // A3 and B = B1 // B2 // B3 be the two parents. Two offsprings are calculated. Offspring 1 denoted by *O1* contains in the middle the jobs of *A2* in the order of *A*. The jobs of $A1 \cup A3$ in the order of B fill the first and the last part of *A*. Offspring 2 denoted by *O2* contains in the middle the jobs of *B2* in the order of *B*. The jobs of $B1 \cup B3$ in the order of *A* fill the first and the last part of *B*. The two crossover operators are illustrated in **Figure 1**

In our genetic algorithm, the crossover operator is chosen randomly, with equal probability.

Mutation: We denote by S the current sequence. The *mutation operators* applied to $S = S_1 / S_{[i]} / S_2 / S_{[j]} / S_3$ with S_1, S_2 and S_3 three subsequences of S and $S_{[i]}$ and $S_{[j]}$ the jobs in positions i and j in S (i < j), two positions i and j (j > i) are randomly choosen. The mutation operators [30], [31] be created as follows (see **Figure 2**):

- SWAP: A neighbor of S is created by interchanging the jobs in position i and j, leading to sequence $S' = S_1 / S_{[j]} / S_2 / S_{[i]} / S_3$.
- EBSR (*Extraction and Backward Shifted Re-insertion*): A neighbor of S is created by extracting $S_{[j]}$ and re-inserting $S_{[j]}$ backward just before $S_{[i]}$, leading to sequence $S' = S_1 / S_{[j]} / S_{[i]} / S_2 / S_3$.
- EFSR (*Extraction and Forward Shifted Re-insertion*): A neighbor of S is created by extracting $S_{[i]}$ and re-inserting it forward immediately after $S_{[j]}$, leading to a sequence $S' = S_1 / S_2 / S_{[j]} / S_{[i]} / S_3$.
- Inversion: A neighbor of S is created by inserting $S_{[j]} / S_2' / S_{[i]}$ between S_1 and S_3, where S_2' is the inverse of sequence S_2.

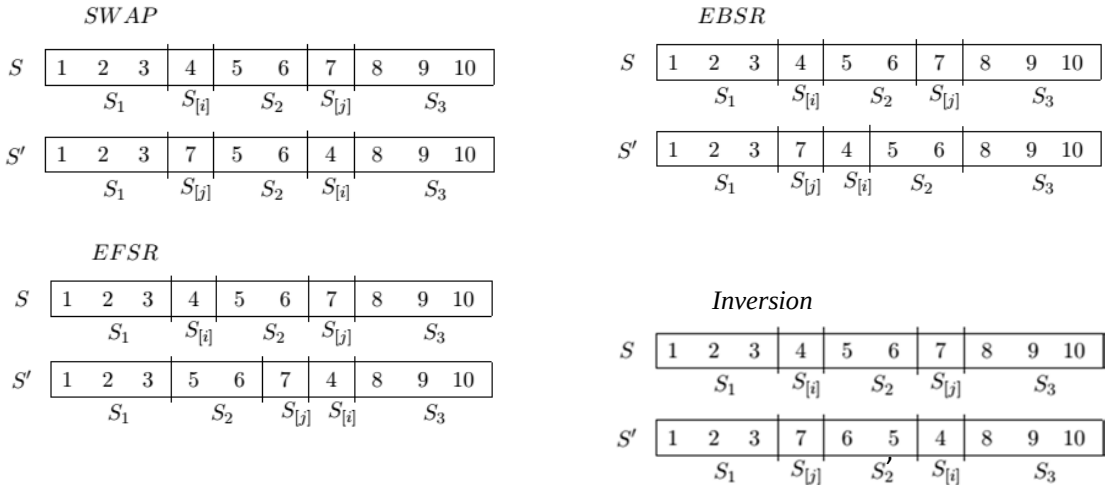

Figure 2: Illustration of mutation operators

Selection and generational scheme: At iteration k, two parents are randomly selected in population P_{k-1}. The two crossover operators are applied on the two parents, generating four offsprings, inserted into population set C_k. The process is repeated until *CrossSize* offsprings have been generated. The mutation operator is applied on randomly selected individuals of population P_{k-1}. The new individuals constitute a population M_k of size *MutSize*. The *PopSize* best individuals of $P_{k-1} \cup C_k \cup M_k$ constitute population P_k.

Stopping criterion: The process iterates until a given time limit has been reached. This time limit is denoted by *TimeLimGA*.

A lot of parameters and operators have been tested for the genetic algorithms, it concerns: the generation on the initial population, the crossover operators, the mutation operators.

2.4. Tabu search algorithm

Tabu search (TS) has been initially proposed by Glover [32,33]. TS is a metaheuristic local search algorithm that begins with an initial solution and successively moves to the best solution in the neighborhood of the

current solution. The algorithm maintains a list of forbidden solutions, to prevent the algorithm from visiting solutions already examined (these solutions are called *tabu*). The elements of our TS algorithm are described below.

- Initial solution

The proposed tabu search algorithm starts from an initial solution. This initial solution is classically generated by using simple heuristic methods, such as *EDD*, *SPT*, *NEH*, etc. In this paper, we propose different initial solutions are considered. If the initial solution is given by *EDD* rule in **Table 1**, the method is denoted by TS_{EDD}. If the initial sequence is given by applying an adaptation of *NEH* algorithm (see **Table 2)**, the method is denoted by TS_{NEH}. Finally, if the initial sequence is given by the best solution beetween *EDD* and *NEH*, the method is denoted by TS_{EN}.

- Neighborhood definition

We denote by *S* the current sequence. We denote by *N(S)* the set of all neighbors of *S* which can be created by *SWAP*, *EBSR*, *EFSR*, *Inversion* operators (see **Figure 2**).

- Moves and selecting the best neighbor

The objective function is the total tardiness. The best neighbor in the candidate list is the non-tabu sequence which generates the smallest total tardiness. The move strategy which is applied to the list is the first-in-first-out (FIFO) strategy. Old attributes are deleted as new attributes are inserted.

- Tabu list

The size of the tabu list is a very important parameter, which can be either fixed or variable. In [32,33], the author provided some general methods of tabu list implementions.

In [12] the authors generate a tabu list by storing attributes of the visited permutations, defined by certain pair of adjacent jobs. Our tabu list contains pairs of positions (i, j), corresponding to the neighborhood definition and the size of the tabu list is fixed.

- Stopping condition

The algorithm is stopped when the time limit has been reached. This time limit is denoted by *TimeLimitTS*.

- Detailed algorithm

The detailed TS algorithm is given in **Table 3**. *FlagSwap*, *FlagNB* with $NB \in \{EBSR, EFSR, Inversion\}$ allow to make a selection of the neighbors. *LimitSwap*, *LimitNB* allow to limit the size of the neighborhood. Del(T) deletes the upper element of the Tabu list and Add(T, (k, j)) adds element (k, j) to the Tabu list.

Table 3: Tabu Search algorithm

Algorithm 3: Tabu Search algorithm

Initialization
S_0 = initial solution, S = current solution
S' = S_0 // best solution of $N(S)$
S^* = S_0 // best solution of $N(S)$ and non-tabu
f^* = $f(S_0)$ // f^* value of S^* and $f(S_0)$ value of S_0
T = ∅; // T is the tabu list
while (CPU ≤ *TimeLimitTS*) **do**
    $f(S')$ = ∞,
    for k = 0 to n - 1 **do**
        for j = k + 1 to n **do**
            if (FlagSwap = 1) and (j - k ≤ LimitSwap) **then**
                S = S', $f(S)$ = $f(S')$, SWAP(S, (k, j)),
                if ((k, j) ∉ T) **then**
                    Calculate($f(S)$),
                    if ($f(S)$ < $f(S')$) then S' = S, $f(S')$ = $f(S)$, move = (k, j), **end if**
                end if
            end if
            if ($j \neq k + 1$) and (FlagNB = 1) and (j - k ≤ LimitNB) **then**
                S = S', $f(S)$ = $f(S')$, NB(S, (k, j)),
                if ((k, j) ∉ T) **then**
                    Calculate($f(S)$),
                    if ($f(S)$ < $f(S')$) **then** S' = S, $f(S')$ = $f(S)$, move = (k, j), **end if**
                end if
            end if
        end for
    end for
    if ($f(S')$ < f^*) then S^* = S', f^* = $f(S)$, **end if**
    if (SizeTabu ≥ TabuMax) then Del(**T**) **end if**
    Add(T, (k, j))
end while

3. Computational experiments

The algorithms have been tested on a PC Intel *core*^{TM}*i*5 CPU 2.4GHz. 108 benchmark instances proposed in [34] have been used for the evaluation. Nine instances of these benchmark instances are used for each combination of n and m, with $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$. In these instances, the processing times are uniformly distributed between 1 and 99. The due dates are generated with an uniform distribution between $P(1 - \tau - \rho/2)$ and $P(1 - \tau - \rho/2)$ following the method of Potts and VanWassenhove [35] with P a lower bound of the makespan and τ and ρ two parameters called tardiness factor and due date range, which take the following values: $\tau \in \{0.2, 0.4, 0.6\}$, $\rho \in \{0.2, 0.6, 1\}$. The first instance (among five) of [34] for each tuple (n, m, τ, ρ) has been used for the tests, which gives the108 instances. In all tables, each line summarizes the results for 9 instances and of course, the methods may return solutions with the same quality, so the total per line of '*Best*' may exceed 9.

3.1. Comparison of the genetic algorithms

The time limit of the GA is fixed to *TimeLimGA* = (*n*(*m/2*) x 90)/1000 seconds (as dened in [21]). For the genetic algorithms, a lot of preliminary experiments have been conducted for the parameters settings. At the end, two parameters sets seem to lead to the best results, denoted case 1 and case 2.

Table 4: The two case of parametres

		case1	*case2*
PopSize	$= \|P_k\|$	150	150
CrossSize	$= \|C_k\|$	200	600
MutSize	$= \|M_k\|$	100	360

For the same instance, the genetic algorithm has been executed ten times and it returns quite always solutions with the same quality. The average relative deviation between ten runs is less than 3%.

$$\Delta_G = \frac{GA_x(G) - min\left(GA_x(EDD), GA_x(NEH), GA_x(EN)\right)}{GA_x(G)} \quad (1)$$

The several *GA* methods are compared in terms of quality. In **Table 5** and **Table 6**, column 'Best' for '$GA_x(G)$' with ($x \in \{1, 2\}$), ($G \in EDD, NEH, EN$) indicates the number of times the method $GA_x(G)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $GA_x(G)$ per nine instances, column `Δ_G' indicates the average deviation between $GA_x(G)$ and the best method between $GA_x(EDD)$, GA_x(NEH) and $GA_x(EN)$.

Table 5: Comparison of genetic algorithms of case 1

n × m	GA_1(EDD)			GA_1(NEH)			GA_1(EN)		
	Best	Cpu(s)	$\Delta GA1_{EDD}$	Best	Cpu(s)	$\Delta GA1_{NEH}$	Best	Cpu(s)	$\Delta GA1_{EH}$
50 × 10	5	22.00	0.79%	5	22.00	2.78%	3	22.00	3.83%
50 × 30	3	67.01	2.76%	3	67.00	2.70%	3	67.00	3.92%
50 × 50	5	112.01	1.22%	0	112.01	2.62%	4	112.01	1.61%
150 × 10	6	67.02	0.49%	5	67.02	4.35%	2	67.02	6.58%
150 × 30	7	202.04	1.53%	2	202.02	3.46%	2	202.05	4.70%
150 × 50	3	337.05	1.39%	2	337.03	5.81%	3	337.05	1.93%
250 × 10	8	112.03	0.08%	3	112.04	4.60%	4	112.07	5.76%
250 × 30	7	337.05	0.23%	2	337.05	7.09%	4	597.56	5.40%
250 × 50	5	562.08	1.07%	3	562.06	5.21%	3	562.06	2.15%
350 × 10	8	157.09	0.58%	2	157.03	17.13%	4	157.07	4.47%
350 × 30	8	472.13	0.06%	1	472.06	15.83%	3	472.10	4.20%
350 × 50	6	787.09	0.23%	1	787.11	6.22%	4	787.14	3.00%
Sum/avg	**71**	**269.55**	**0.87%**	**29**	**269.54**	**6.48%**	**39**	**291.26**	**3.96%**

Table 6: Comparison of genetic algorithms of case 2

n × m	GA_2(EDD)			GA_2(NEH)			GA_2(EN)		
	Best	Cpu(s)	$\Delta GA2_{EDD}$	Best	Cpu(s)	$\Delta GA2_{NEH}$	Best	Cpu(s)	$\Delta GA2_{EH}$
50 × 10	7	22.03	0.23%	4	22.02	2.66%	4	22.03	2.66%
50 × 30	5	67.11	1.71%	4	67.04	2.00%	4	67.05	2.00%
50 × 50	6	112.04	0.67%	3	112.04	2.25%	3	112.05	2.25%
150 × 10	7	67.12	0.59%	2	67.09	12.10%	2	67.16	6.67%
150 × 30	8	202.09	0.16%	2	202.13	4.49%	3	202.14	3.78%
150 × 50	6	337.15	0.49%	3	337.17	5.83%	3	337.14	2.59%
250 × 10	8	112.19	0.15%	2	112.38	21.08%	3	112.32	7.47%
250 × 30	7	337.18	9.95%	2	337.35	6.30%	2	337.21	16.18%
250 × 50	6	562.26	0.23%	3	638.75	4.36%	3	562.36	2.68%
350 × 10	8	157.21	0.78%	2	157.62	24.84%	2	157.35	14.29%
350 × 30	8	472.24	0.23%	2	472.27	6.29%	1	472.45	17.59%
350 × 50	7	787.39	0.70%	1	787.43	7.01%	1	787.76	5.47%
Sum/avg	**83**	**269.66**	**1.32%**	**30**	**276.10**	**8.27%**	**31**	**269.75**	**6.97%**

Table 7: Comparison of best genetic algorithms of case 1 and case 2

n × m	GA_1(EDD)			GA_2(EDD)		
	Best	Cpu(s)	Δ_1EDD	Best	Cpu(s)	Δ_2EDD
50 × 10	6	22,00	2,79%	5	22,03	2,79%
50 × 30	3	67,01	1,48%	6	67,11	1,48%
50 × 50	2	112,01	0,12%	7	112,04	0,12%
150 × 10	6	67,02	1,29%	5	67,12	1,29%
150 × 30	4	202,04	0,49%	6	202,09	0,49%
150 × 50	2	337,05	0,07%	7	337,15	0,07%
250 × 10	6	112,03	1,33%	6	112,19	1,33%
250 × 30	6	337,05	1,86%	4	337,18	1,86%
250 × 50	6	562,08	2,57%	4	562,26	2,57%
350 × 10	9	157,09	9,90%	2	157,21	9,90%
350 × 30	6	472,13	1,61%	5	472,24	1,61%
350 × 50	6	787,09	2,04%	4	787,39	2,04%
Sum/avg	**62**	**269,67**	**1,39%**	**61**	**269,55**	**2,96%**

In **Table 7**, column 'Best' for '$GA_x(EDD)$' ($x \in \{1, 2\}$) indicates the number of times the method $GA_x(EDD)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $GA_x(EDD)$ per nine instances, column 'Δ_x' indicates the average deviation between $GA_x(EDD)$ and the best method between

$GA_1(EDD)$ and $GA_2(EDD)$. We can see that the genetic algorithm ($GA_1(EDD)$) with the initial population given by EDD rule and the parameters of case 1 (see **Table 4**) leads to the best results. The average deviation between the solutions returned by this method and the best solutions is 1,39%. This value is around 2,96% for $GA_2(EDD)$. Algorithm $GA_1(EDD)$ has been used in the following for the comparisons with the Tabu search algorithms.

$$\Delta_x(EDD) = \frac{GA_x(EDD) - min\left(GA_1(EDD), GA_2(EDD)\right)}{GA_x(EDD)} \quad (2)$$

3.2. Comparison of Tabu search algorithms

For the Tabu search algorithms, a lot of preliminary experiments have conducted to the following parameters settings. The time limit of TS is $TimeLimTS = (n(m/2) \times 90)/1000$ seconds. The TS methods have been executed with four Tabu list parameters $\lambda \in \{20, 40, 60, 120\}$ and the initial solutions EDD, NEH or EN rule. The parameter ($\lambda = 40$) leads to the best results all three initial solutions.The three best TS methods are compared in terms of quality. In **Table 8**, column 'Best' for '$TS_\lambda(T)$' (with λ is a number element of Tabu list ($\lambda = 40$), T is an initial solution ($T \in$ {EDD, EN, NEH}) which indicates the number of times the method $TS_\lambda(T)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $TS_\lambda(T)$ per nine instances, column 'Δ_T' indicates the average deviation between $TS_\lambda(T)$ and the best method between $TS_{40}(EDD)$, $TS_{40}(EN)$ and $TS_{40}(NEH)$.

Table 8: Comparison of Tabu search algorithms

n × m	TS_{40}(EDD)			TS_{40}(NEH)			TS_{40}(EN)		
	Best	Cpu(s)	Δ_{EDD}	Best	Cpu(s)	Δ_{NEH}	Best	Cpu(s)	Δ_{EH}
50 × 10	6	22,01	0,23%	4	22,00	0,72%	5	22,01	0,63%
50 × 30	7	67,02	0,15%	2	67,03	1,93%	0	67,06	1,81%
50 × 50	7	112,04	0,28%	1	112,06	1,14%	2	112,05	1,12%
150 × 10	5	67,18	1,53%	6	67,02	0,88%	2	67,22	6,26%
150 × 30	8	202,28	2,60%	1	202,62	5,74%	2	202,74	2,11%
150 × 50	8	337,78	0,01%	1	338,18	8,30%	0	337,72	0,69%
250 × 10	7	112,67	0,75%	4	112,74	1,83%	4	113,10	1,51%
250 × 30	7	338,47	0,05%	3	339,38	3,93%	4	339,10	3,97%
250 × 50	7	565,79	0,22%	3	566,30	2,15%	2	565,65	4,73%
350 × 10	9	159,00	0,00%	4	158,28	2,20%	2	159,08	3,88%
350 × 30	6	476,29	1,30%	3	478,27	5,30%	2	475,08	4,06%
350 × 50	8	793,20	0,08%	2	797,05	4,94%	2	799,20	6,33%
Sum/avg	**85**	**271,14**	**1,43%**	**34**	**271,74**	**4,09%**	**27**	**271,33**	**4,76%**

We can alsosee from **Table 8**, that the Tabu search algorithm where the initial solution is given by EDD rule

leads to the best results. On average, the deviation between the solutions returned by this method and the best solutions is 1,43%. These values are around 4,09% for TS_{40}(EN) and 4,76% for TS_{40}(NEH).

$$\Delta_T = \frac{TS_\lambda(T) - min(TS_{40}(EDD), TS_{40}(NEH), TS_{40}(EN))}{TS_{40}(G)} \quad (3)$$

3.3. Comparison of the best algorithm among GA and TS

Now, the best algorithms GA and TS are compared. The results are presented in **Table 9**. Column 'Best' for '*Algo*' with *Algo* ∈ {GA_1(EDD) TS_{40}(EDD)} which indicates the number of times the method *Algo* outperforms the other methods, column Cpu(s) indicates the average computation time of *Algo* per nine instances, column 'Δ*Algo*' indicates the average deviation between GA_1(EDD) and TS_{40}(EDD).

Table 9: Comparison of the best GA and TS algorithm

n × m	GA_1(EDD)			TS_{40}(EDD)		
	Best	Cpu(s)	Δ_1EDD	Best	Cpu(s)	Δ_2EDD
50 × 10	2	22.00	7.54%	9	22.01	0.00%
50 × 30	0	67.01	8.94%	9	67.02	0.00%
50 × 50	0	112.01	4.60%	9	112.04	0.00%
150 × 10	2	67.02	10.93%	9	67.18	0.00%
150 × 30	2	202.04	6.49%	8	202.28	0.01%
150 × 50	0	337.05	13.00%	9	337.78	0.00%
250 × 10	3	112.03	5.02%	9	112.67	0.00%
250 × 30	3	337.05	1.89%	8	338.47	0.13%
250 × 50	3	562.08	1.57%	7	565.79	0.67%
350 × 10	2	157.09	17.76%	9	159.35	0.00%
350 × 30	2	472.13	4.18%	8	476.19	11.11%
350 × 50	2	787.09	3.19%	8	793.20	0.38%
Sum/avg	**21**	**269.55**	**7.09%**	**102**	**271.17**	**1.03%**

$$\Delta_{Algo} = \frac{Algo - min(GA_1(EDD), TS_{40}(EDD))}{Algo} \quad (4)$$

The best results are given by the Tabu search initialized by EDD rule. The average deviation between TS_{40}(EDD) and the best solution is 1,03%, the average computation time of TS_{40}(EDD) per 108 instances is 271,17 seconds. These values are 7,09% and 269,55 seconds for GA_1(EDD).

4. Conclusion

We consider in this paper the m-machine flow shop scheduling problem, with the objective to minimize the total tardiness. We propose two greedy algorithms (EDD and NEH), GA and TS algorithms which are initiated by

EDD, NEH, EN solution. The neighborhood operators have also applied for the GA and TS method. The algorithms are tested and evaluated from 108 benchmark instances of Vallada and his colleagues [21]. Many parameters for each method have been tested. The results obtained by the proposed algorithms show that TS method outperforms GA. The algorithms initiated by EDD heuristic are always better than the algorithms initiated by EN or NEH. Several research directions can be considered for a future work. The first idea is to embed the resolution of the MILP (Mixed Integer Linear Programming) model into the GA, TS or into another metaheuristic, as a new neighborhood operator. A second idea is to find better crossover and mutation operators, in order to improve the genetic algorithm. A third idea is to propose a simulated annealing algorithm to be compared to the GA, TS algorithms for m-machine permutation flow shop scheduling problem. Finally, the metaheuristic methods that are proposed here can be used for minimizing the total tardiness in more complicated scheduling problems such as an integrated flow shop scheduling and vehicle routing problem.

Acknowledgements

The authors gratefully acknowledge the financial support of the project (T17-41) from Ha Noi University of Mining and Geology, Viet Nam. We would like to thank to Professor Jean-Charles Billaut of University of Tours, France who provided insight and expertise that greatly assisted the research.

References

[1] M. L. Pinedo, Scheduling: Theory, algorithms, and systems. Springer 2008.

[2] J. Du and J. Y. T. Leung, “Minimizing total tardiness on one machine is NP-hard,” Mathematics of Operations Research, vol. 19, pp. 483–495, 1990.

[3] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, “Complexity of machine scheduling problems,” Annals of Discrete Mathematics, vol. 1, pp. 343–362, 1977.

[4] Y. D. Kim, “A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops,” Computers & Operations Research, vol. 20(4), pp. 391–401, 1993.

[5] J. C.-H. Pan and E.-T. Fan, “Two-machine flowshop scheduling to minimize total tardiness,” International Journal of Systems Science, vol. 28, no. 4, pp. 405–414, 1997.

[6] T. Sen, P. Dileepan, and J. N. D. Gupta, “The two-machine flowshop scheduling problem with total tardiness,” Computers & Operations Research, vol. 16(4), pp. 333–340, 1989.

[7] J. C. H. Pan, J. S. Chen, and C. M. Chao, “Minimizing tardiness in a two-machine flow-shop,” Computers & Operations Research, vol. 29, pp. 869–885, 2002.

[8] M. Nawaz, E. Enscore, and I. Ham, “A heuristic algorithm for the m-machine n-job flow shop sequencing problem,” Omega, vol. 11, pp. 91–95, 1983.

[9] C. Koulamas, "A guaranteed accuracy shifting bottleneck algorithm for the two-machine flowshop total tardiness problem," Computers & Operations Research, vol. 25, pp. 83–89, 1998.

[10] I. Osman and C. Potts, "Simulated annealing for permutation flow-shop scheduling," Omega, vol. 17, no. 6, pp. 551–557, 1989.

[11] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion," Computers & Operations Research, vol. 31, pp. 1891–1909, 2004.

[12] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flowshop problem," European Journal of Operational Research, vol. 91, pp. 160–175, 1996.

[13] V. A. Armentano and D. P. Ronconi, "Tabu search for total tardiness minimization in flowshop scheduling problems," Computers & Operations Research, vol. 26, pp. 219–235, 1999.

[14] M. Ben-Daya and M. Al-Fawzan, "A tabu search approach for the flow shop scheduling problem," European Journal of Operational Research, vol. 109, pp. 88–95, 1998.

[15] J. Brandão and A. Mercer, "A tabu search algorithm for the multi-trip vehicle routing and scheduling problem," European Journal of Operational Research, vol. 100, pp. 180–191, 1997.

[16] G. C. Onwubolu and M. Mutingi, "Genetic algorithm for minimizing tardiness in flow-shop scheduling," Production Planning and Control, vol. 10, pp. 462–471, 1999.

[17] C.-J. Liao, Chao-Tang Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," Computers & Operations Research, vol. 34, no. 10, pp. 3099–3111, 2007.

[18] M. F. Tasgetiren, Y. C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," European Journal of Operational Research, vol. 177, no. 3, pp. 1930–1947, 2007.

[19] F. Della Croce, A. Grosso, and F. Salassa, "A matheuristic approach for the total completion time two-machines permutation flow shop problem," Lecture Notes in Computer Science, 6622 LNCS, pp. 38–47, 2011.

[20] Q. C. Ta, J. C. Billaut, and J. L. Bouquard, "Recovering beam search and Matheuristic algorithms for the F2|| ∑Tj scheduling problem," 11th Workshop on Models and Algorithms for Planning and Scheduling Problems, 2013, pp. 218–220.

[21] E. Vallada, R. Ruiz, and G. Minella, "Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics," Computers & Operations Research, vol. 35, pp. 1350–1373, 2008.

[22] E. Vallada and R. Ruiz, “Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem,” European Journal of Operational Research, vol. 38, pp. 57–67, 2010.

[23] V. Fernandez-Viagas and J. M. Framinan, “NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness,” Computers & Operations Research, vol. 60, pp. 27-36, 2015.

[24] V. Fernandez-Viagas, R. Ruiz, and J. M. Framinan, “A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation,” European Journal of Operational Research. vol. 257, pp. 707-721, 2017.

[25] J. A. Holland, Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, 1975.

[26] D. E. Goldberg, Genetic algorithms in Search, Optimization and Machine Learning. Addison-Welsey, Reading Mass, 1989.

[27] M. C. Portmann and A. Vignier. "Chapter 4: Genetic algorithm and scheduling". In Production Scheduling, P. Lopez and F. Roubellat Eds, Wiley-ISTE, 2008.

[28] R. T. F. Della Croce, M. Ghirardi, “Recovering Beam Search: en- hancing the beam search approach for combinatorial optimization problems,” J. Heuristics, vol. 10, pp. 89–104, 2004.

[29] R. Ruiz, C. Maroto, and J. Alcaraz, “Two newrobust genetic algorithms for the flowshop scheduling problem,” Omega, vol. 34, pp. 461–476, 2006.

[30] F. Della Croce, M. Ghirardi, and R. Tadei, “Recovering Beam Search : Enhancing the Beam Search Approach for Combinatorial,” Journal of Heuristics, vol. 10, pp. 89–104, 2004.

[31] C. C. Wu, W. C. Lee, and T. Chen, “Heuristic algorithms for solving the maximum lateness scheduling problem with learning considerations,” Computers & Industrial Engineering, vol. 52, pp. 124–132, 2007.

[32] F. Glover, “Tabu Search - Part I,” ORSA Journal on Computing, vol. 1, pp. 190–206, 1989.

[33] F. Glover, “Tabu Search - Part II,” ORSA Journal on Computing, vol. 2, pp. 4–32, 1990.

[34] E. Vallada, R. Ruiz, and G. Minella, “Minimising total tardiness in the m-machine flowshop problem : A review and evaluation of heuristics and metaheuristics,” Computers & Operations Research, vol. 35, pp. 1350–1373, 2008.

[35] C. N. Potts and L. N. Van Wassenhove, “A decomposition algorithm for the single machine total tardiness problem,” Operations Research Letters, vol. 1, pp. 177–181, 1982.