Matheuristic algorithms for minimizing total tardiness in the m-machine flow-shop scheduling problem

Quang Chieu Ta, Jean-Charles Billaut, Jean-Louis Bouquard

J Intell Manuf
DOI 10.1007/s10845-015-1046-4

Matheuristic algorithms for minimizing total tardiness in the *m*-machine flow-shop scheduling problem

Quang Chieu Ta · Jean-Charles Billaut · Jean-Louis Bouquard

Received: 30 June 2014 / Accepted: 22 January 2015

Abstract We consider in this paper the *m*-machine permutation flow-shop problem with total tardiness minimization. We propose several matheuristic algorithms, which are an hybridization of a local search and an exact resolution method. The matheuristics are compared to a genetic algorithm. Computational experiments are performed on benchmark instances and the results show the good performances of the matheuristic algorithms. Finally, some future research directions are given.

Keywords Scheduling · Flow-shop · Matheuristic · Genetic algorithm

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 $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs to schedule on a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ 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, and preemption is not allowed. Variable C_j denotes the completion time of job J_j and variable T_j its tardiness, defined by $T_j = \max(C_j - d_j, 0), \forall j$, $1 \leq j \leq n$.

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 $F|prmu|\sum T_j$ (Pinedo 1995), 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 $m \geq 2$ (Lenstra 1977; Du and Leung 1990).

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 (see Sen et al. 1989; Kim 1993; Pan and Fan 1997). In Pan et al. (2002), instances with up to 24 jobs can be solved to optimality, which shows the difficulty 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 (Nawaz et al. 1983), and a shifting bottleneck procedure (Koulamas 1998). Some metaheuristics have also been proposed in the literature, such as simulated annealing (Osman and Potts 1989), tabu search algorithms (Kim 1993; Nowicki and Smutnicki 1996; Grabowski and Wodecki 2004) genetic algorithms (Onwubolu and Mutingi 1999), particle swarm optimization (Tasgetiren et al. 2007; Liao et al. 2007), etc.

For the *m*-machine flow-shop scheduling problem, Onwubolu and Mutingi propose in Onwubolu and Mutingi (1999) a genetic algorithm minimizing a combination of the total tardiness and the number of tardy jobs. In the survey of Vallada et al. (2008), 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

Q. C. Ta · J.-C. Billaut (✉) · J.-L. Bouquard
CNRS, LI EA 6300, OC ERL CNRS 6305, Université François Rabelais de Tours, 64 Avenue Jean Portalis, 37200 Tours, France
e-mail: jean-charles.billaut@univ-tours.fr

Q. C. Ta
e-mail: quang-chieu.ta@univ-tours.fr

J.-L. Bouquard
e-mail: jean-louis.bouquard@univ-tours.fr

Published online: 05 February 2015

of the most efficient methods. In Vallada and Ruiz (2010), 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. 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 minimization in Vallada et al. (2008).

Since several years, a new type of approximated algorithms, including exact resolution inside heuristic approaches, has received a great interest in the literature, because of their very good performances for solving some difficult problems (Maniezzo et al. 2010; Talbi 2013). These methods are called *matheuristics* or "hybrid heuristics". For example in the scheduling literature, Della Croce et al. (2011) embed a mixed integer programming formulation solved by a commercial solver into a neighborhood search algorithm, for the two-machine flow-shop problem with total completion time minimization. In Ta et al. (2013), the authors proposed a matheuristic algorithm for the $F2||\sum T_j$ scheduling problem. Pessan et al. (2008) propose a branch-and-bound algorithm and a genetic algorithm for solving a parallel machine scheduling problem. The methods are performed in parallel and synchronized and exchange some information for improving the resolution. But the hybridization of exact and approximate methods is not so recent. Portmann, Vignier, Dardilhac and Dezalay propose in Portmann et al. (1998) a branch-and-bound algorithm crossed with a genetic algorithm for solving an hybrid flow-shop problem with makespan minimization. The genetic algorithm is used during the search of the branch-and-bound in order to improve the upper bound. Jouglet, Oguz and Sevaux consider in Jouglet et al. (2009) an hybrid flow-shop scheduling problem and use a constraint programming based branch-and-bound algorithm as a local search engine of a memetic algorithm.

In this paper, we propose a genetic algorithm and several matheuristic algorithms, initialized by the solution of a greedy algorithm. The solutions of the matheuristic methods are compared to the solutions of the genetic algorithm. For the evaluation, 108 benchmark instances proposed in Vallada et al. (2008) have been used.

The rest of the paper is organized as follows. In "Genetic algorithm" section, the genetic algorithm is described. In "MILP formulation" section, an MILP formulation of the problem is given. Then, the neighborhood search method that includes the exact resolution of some subproblems is described in "Matheuristic algorithms" section. Several methods are proposed for the optimization of a partial sequence. "Computational experiments" section reports the settings of the methods and the computational results. A conclusion and some future research directions are proposed in "Conclusion" section.

Genetic algorithm

We remember in this section some basic notions on Genetic Algorithms and then we describe our implementation of the crossover and of the mutation operators.

Principles of a genetic algorithm

Genetic Algorithms have been originally proposed by Holland (1975) and further developed by Goldberg (1989). 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 evaluated 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 introduce random modifications. Some individuals are randomly selected for crossover, some individuals 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 individual has been found.

Genetic Algorithms have been largely used for solving scheduling problems. The main steps of the Genetic Algorithm that has been developed, 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} are 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 modified 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 and 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 Portmann and Vignier (2008).

Genetic operators

Coding

The crucial step in designing a Genetic Algorithm is to define an encoding, i.e. a way to represent a solution. In the case of the m-machine 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 (Earliest Due Date first, i.e. sort the jobs in d_j non decreasing order), the method is called GA_{EDD}. If the initial sequence is given by applying an adaptation of *NEH* algorithm Nawaz et al. (1983) (described in Algorithm 1), the method is called GA_{NEH}. Finally, if one initial sequence is given by the best sequence among *EDD* and *NEH*, the method is called GA_{EN}.

Fitness

The *fitness* of an individual S is the value of the objective function $\sum T_j(S)$ of the corresponding sequence.

Algorithm 1 NEH algorithm

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

Crossover

Several crossover operators have been tested. Finally, two *crossover operators* are used: the *one-point crossover* (X1) and the *linear order crossover* (LOX) Werner (1984):

- 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.
- 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 Fig. 1.

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

Mutation

Two *mutation operators* have been used, called Backward and Forward Insertion Della Croce et al. (2004) and denoted by *BI* and *FI* in the following. Two positions i and j ($j > i$) are randomly chosen in a sequence $S = A/S_{[i]}/B/S_{[j]}/C$, with A, B and C three subsequences and $S_{[i]}$ and $S_{[j]}$ the jobs in positions i and j in S. In the backward insertion, the job in position j is sequenced before the job in position i, leading to sequence $S' = A/S_{[j]}/S_{[i]}/B/C$ and in the forward insertion, the job in position i is sequenced after the job in position j, leading to a sequence $S' = A/B/S_{[j]}/S_{[i]}/C$.

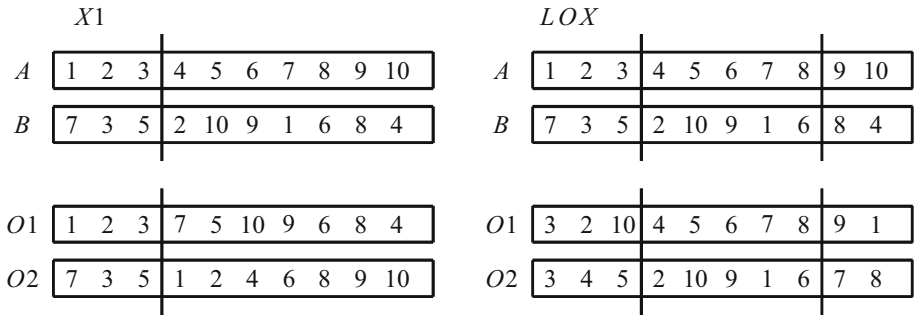

Fig. 1 Illustration of two crossover operators: backward insertion and forward insertion

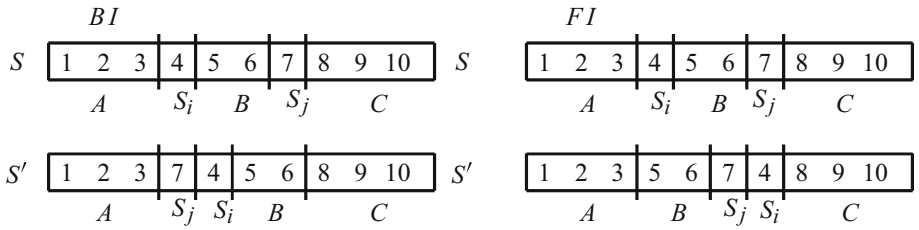

Fig. 2 Illustration of two mutation operators

The two mutation operators are illustrated in Fig. 2.

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*.

MILP formulation

In this section, we propose an MILP formulation of the problem based on positional variables (firstly introduced in Wagner 1959). Binary variable $x_{j,k}$ is equal to 1 if job J_j is in position k in the sequence, and 0 otherwise, $\forall j \in \{1, \ldots, n\}$, $\forall k \in \{1, \ldots, n\}$. Continuous variable $C_{i,k} \geq 0$ is the completion time of the job in position k on machine M_i, $\forall i \in \{1, \ldots, m\}$, $\forall k \in \{1, \ldots, n\}$ and $T_k \geq 0$ is the tardiness of the job in position k, $\forall k \in \{1, \ldots, n\}$.

The MILP model is the following:

$$\text{Minimize} \sum_{k=1}^{n} T_k \tag{1}$$

$$\text{subject to} \sum_{k=1}^{n} x_{j,k} = 1, \quad \forall j \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{j=1}^{n} x_{j,k} = 1, \quad \forall k \in \{1, \ldots, n\} \tag{3}$$

$$C_{1,1} = \sum_{j=1}^{n} p_{1,j} x_{j,1} \tag{4}$$

$$C_{1,k} = C_{1,k-1} + \sum_{j=1}^{n} p_{1,j} x_{j,k}, \quad \forall k \in \{2, \ldots, n\} \tag{5}$$

$$C_{i,1} = C_{i-1,1} + \sum_{j=1}^{n} p_{i,j} x_{j,1}, \quad \forall i \in \{2, \ldots, m\} \tag{6}$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} p_{i,j} x_{j,k}, \quad \forall i \in \{2, \ldots, m\}, \quad \forall k \in \{1, \ldots, n\} \tag{7}$$

$$C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} p_{i,j} x_{j,k}, \quad \forall i \in \{2, \ldots, m\}, \quad \forall k \in \{1, \ldots, n\} \tag{8}$$

$$T_k \geq C_{m,k} - \sum_{j=1}^{n} d_j x_{j,k}, \quad \forall k \in \{1, \ldots, n\} \tag{9}$$

Constraints (2) and (3) ensure that there is exactly one job per position and one position per job. Constraints (4) and (6) compute the completion times on machine M_1. Constraints (5), (7) and (8) determine the completion times on machine

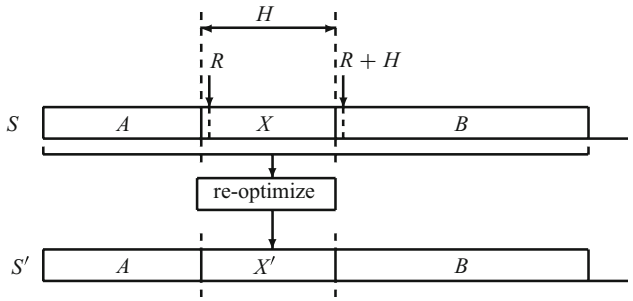

Fig. 3 Illustration of the matheuristic algorithm for sequence "*AXB*"

M_i. Constraints (9) determine the total tardiness. This model contains n^2 binary variables, $n(m+1)$ continuous variables and $m + 4n + 2nm$ constraints.

Matheuristic algorithms

In this section, we propose several matheuristic algorithms, based on iterative calls of the MILP presented in "MILP formulation" section, for the exact resolution of sub-problems.

General framework "*AXB*"

Given a sequence S=*AXB*, with an index denoted by R and a size window denoted by H, in the method, the sequence of jobs from position 1 to position $R-1$ (sequence A) and the sequence of jobs from position $R+H$ to the end (sequence B) are unchanged. The sequence of jobs between position R and position $R+H-1$ (sequence X) is re-optimized, giving sequence X'. The sequence $S' = AX'B$ is a new sequence, hopefully better than S, that replaces sequence S. The process iterates for other values of R, up to $n-H$. When all the values for R have been tested, a neighborhood based on swaps of jobs is applied to the sequence S, in order to modify and possibly improve this solution, and the procedure iterates from $R=1$, until the stopping criterion is reached. The stopping criterion is a time limit called *TimeLimMH*. This process is illustrated in Fig. 3 and the general algorithm of the method is given in Algorithm 2.

In Algorithm 2 and Algorithm 3, $S_{[k]}$ denotes the job of S in position k. If an improvement is found at a given iteration for the jobs in positions $[R, R+H-1]$, then the positions considered for the next iteration are in the interval $[R+H, R+2H-1]$, if $R+2H-1 \leq n$ (i.e. $(R-1)+H \leq n-H$. Otherwise, the positions considered for the next iteration are in the interval $[R+1, R+H]$.

In Algorithm 3, pairs of jobs are swapped if the difference between the positions of the two jobs does not exceed $n/2$. If a swap improves the solution, the current sequence is immediately updated and swaps continue.

Algorithm 2 The general Matheuristic algorithm

Input: S = initial solution
improved = true
while (CPU $\leq TimeLimMH$) and (improved = true) **do**
  improved = false; $R = 0$
  while $R \leq n - H$ **do**
    $A = (S_{[1]}, S_{[2]}, ..., S_{[R-1]})$; $X = (S_{[R]}, S_{[R+1]}, ..., S_{[R+H-1]})$
    $B = (S_{[R+H]}, S_{[R+H+1]}, ..., S_{[n]})$
    X' = re-optimization of X
    $S' = AX'B$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      improved = true; $S = S'$
      if $(R + H \leq n - H)$ **then**
        $R = R + H - 1$
      end if
    end if
    $R = R + 1$
  end while
  S' = Swap(S)
  if $(\sum T_j(S') < \sum T_j(S))$ **then**
    improved = true; $S = S'$
  end if
end while
return(S)

Algorithm 3 Swap (S)

Input: S = initial solution
for $i = 1$ **to** $n-1$ **do**
  $j = i + 1$
  while $(j \leq n)$ and $(j - i \leq n/2)$ **do**
    $S' = (S_{[1]}, ..., S_{[i-1]}, S_{[j]}, S_{[i+1]}, ..., S_{[j-1]}, S_{[i]}, S_{[j+1]}, ...S_{[n]})$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      $S = S'$
    end if
    $j = j + 1$
  end while
end for
return(S)

Notice that Algorithm 2 can be used with any initial solution.

Several versions of this algorithm are derived, depending on how sequence X' is obtained, and on the application, or not, of an additional neighborhood search. The solver for the MILP model is called iteratively.

Matheuristic algorithm $MH_{XB}(S)$

The simplest way to re-optimize S in $MH(S)$ is to introduce the following constraints into the model:

$$x_{S_{[k]},k} = 1, \forall k \in \{1, \ldots, R-1\} \cup \{R+H, \ldots, n\} \qquad (10)$$

These $n-H$ constraints ensure that sequences A and B will not be changed in S'. However, because sequence A is known, there is no need to give to the solver a complete MILP model with n^2 binary variables. Therefore, in order to reduce the size of the MILP, we compute the completion

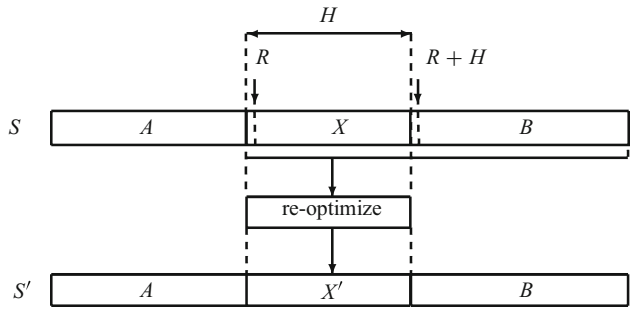

Fig. 4 Illustration of the $MH_{XB}(S)$ algorithm

times of the last job of sequence A on each machine and we only re-optimize XB, within an MILP model with only $(n - R + 1)^2$ binary variables. This reduction is interesting for large values of R. We denote by CA_i the completion time of the last job of sequence A on machine M_i and by $\sum T_j(A)$ the total tardiness of the jobs in A. New constraints are added to the model related to the CA_i. We only indicate here these new constraints. The rest of the model is unchanged, except for the definition of indices: the problem which is solved is smaller than before and only the $n - R + 1$ jobs which are not in A are sequenced from position 1 (i.e. R) to position $n - R + 1$ (i.e. n). Notice that the indices in the expression of the constraints (11) and (12) refer to the complete sequence S.

$$C_{1,R} = CA_1 + \sum_{j=1}^{n} p_{1,j} x_{j,R} \tag{11}$$

$$C_{i,R} \geq CA_i + \sum_{j=1}^{n} p_{i,j} x_{j,R}, \quad \forall i \in \{2, \ldots, m\} \tag{12}$$

Remember that $C_{i,R}$ is the completion time on machine M_i of the job in position R in S, i.e. of the first job of XB. If we denote by $\sum T_j(XB^*)$ the value of the optimal solution of this model, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(XB^*)$$

The process of $MH_{XB}(S)$ algorithm is illustrated in Fig. 4.

In Della Croce et al. (2011), a similar method is proposed for the $F2||\sum C_j$ problem, where the initial solution is given by a Recovering Beam Search algorithm, and a different method is used for fixing the parameters of the MILP.

Matheuristic algorithm $MH_{XB_1}(S)$

This method is similar to $MH_{XB}(S)$, but a neighborhood operator [among SWAP, BI and FI (see "Genetic operators" section)] is applied to the sequence of jobs from position 1 to position $R - 1$ (i.e. sequence A), leading to a new sequence A'. The objective function of the neighborhood operator is such that sequence A' will not penalize too much sequence

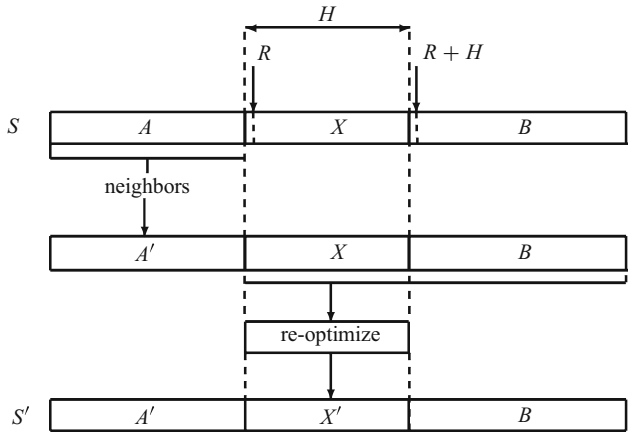

Fig. 5 Illustration of the MH_{XB_1} algorithm

X, i.e. it is a linear combination of the total tardiness of the jobs of A' and the makespan of A'. The expression of this objective function to minimize is:

$$Z = \alpha \left(\sum_{i \in A'} T_i \right) + (1 - \alpha) C_{m,R-1} \tag{13}$$

where $C_{m,R-1}$ denotes the completion time on machine M_m of the job in position $R - 1$, i.e. the last job of sequence A', and $0 \leq \alpha \leq 1$.

The process of $MH_{XB_1}(S)$ is illustrated in Fig. 5.

The total tardiness of A' and the completion time of the last job of A' on each machine are computed and denoted by CA_i, the constraints with CA_i are introduced as in (11) and (12) for algorithm $MH_{XB}(S)$. The value of S' is given by:

$$\sum T_j(S') = \sum T_j(A') + \sum T_j(XB^*)$$

where $\sum T_j(XB^*)$ is computed as in "Matheuristic algorithm $MH_{XB}(S)$" section.

Matheuristic algorithm $MH_X(S)$

In order to continue reducing the size of the problem to be solved by the solver, we limit now the problem exactly to the optimization of sequence X. Of course, minimizing the total tardiness of the jobs of X (sequenced after the jobs of A) and after that, sequencing the jobs of B leads to a worse solution than minimizing the total tardiness of the jobs of XB, even if B is fixed. Therefore, we introduce another criterion in the objective function for the optimization of X. More precisely, in order to finish the jobs of X not too late, which could be penalizing for B, we change the objective function for a linear combination of the total tardiness of the jobs of X and the makespan of X. Therefore, the objective function is equal to:

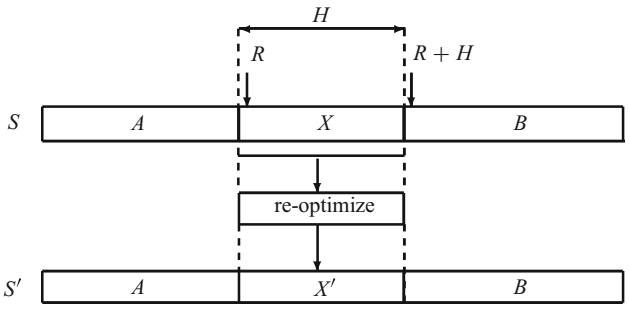

Fig. 6 Illustration of the $MH_X(S)$ algorithm

$$\text{Minimize } \alpha\left(\sum_{k=R}^{R+H-1} T_k\right) + (1-\alpha)C_{m,R+H-1} \tag{14}$$

where $C_{m,R+H-1}$ denotes the completion time on machine M_m of the job in position $R+H-1$, i.e. the last job of sequence X.

The process of $MH_X(S)$ is illustrated in Fig. 6.

First, the total tardiness of A and the completion time of the last job of A on each machine are computed and the constraints with CA_i are introduced in the model (same way as in MH_{XB}). Then, the optimization of X is done by the solver, and a sequence $X' = X^*$ is obtained. The completion times of the last job of X' on each machine are denoted by CX_i. Then, sequence B is scheduled after X, taking the CX_i values into account. Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(X^*) + \sum T_j(B)$$

Matheuristic algorithm $MH_{X_1}(S)$

We propose another matheuristic method of type MH_X called $MH_{X_1}(S)$, based also on a partial resolution of the MILP (see "MILP formulation" section), similarly as in "General framework "*AXB*" "section. Firstly, neighborhood operators (among SWAP, BI and FI) are applied to sequence A and the obtained sequence is called A'. Secondly, the sequence X is re-optimized, giving sequence X'. Finally, neighborhood operators are also applied to sequence B, giving sequence B'. The sequence $S' = A'X'B'$ is a new sequence, hopefully better than S, and the process iterates for other values of R, up to $n-H$. This process is performed until the stopping criterion is reached, i.e. a time limit called *TimeLimMH*. This matheuristic algorithm is illustrated in Fig. 7 and the algorithm is given in Algorithm 4.

The objective function for finding the best possible subsequence A' is a linear combination of the total tardiness of the jobs of A' and of the makespan of A'. This objective function is equal to:

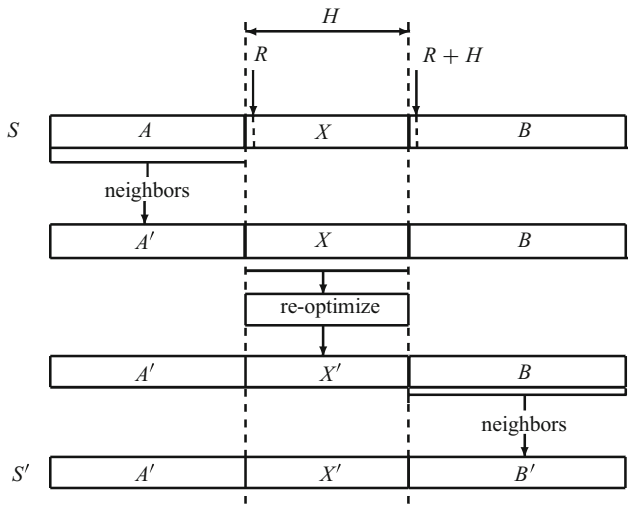

Fig. 7 Illustration of the $MH_{X_1}(S)$ algorithm

Algorithm 4 The $MH_{X_1}(S)$ algorithm

```
1: Input: S = initial solution
2: improved = true
3: while (CPU ≤ TimeLimMH) and (improved = true) do
4:   improved = false ; R = 0
5:   while R ≤ n − H do
6:     A = (S[1], S[2], ..., S[R−1]); A′ = Neighborhood operator(A)
7:     X = (S[R], S[R+1], ..., S[R+H−1]); X′ = re-optimization of X
8:     B = (S[R+H], S[R+H+1], ..., S[n]); B′ = Neighborhood
       operator(B)
9:     S′ = A′X′B′
10:    if (∑ Tj(S′) < ∑ Tj(S)) then
11:      improved = true ; S = S′
12:      if (R + H ≤ n − H) then
13:        R = R + H − 1
14:      end if
15:    end if
16:    R = R + 1
17:  end while
18: end while
19: return(S)
```

$$Z_A = \alpha\left(\sum_{i=1}^{R-1} T_i\right) + (1-\alpha)C_{m,R-1} \tag{15}$$

where $C_{m,R-1}$ denotes the completion time on machine M_m of the job in position $R-1$, i.e. the last job of sequence A'.

For the same reasons, the objective function for finding the best possible subsequence X' is also a linear combination of the total tardiness of the jobs of X' and of the makespan of X'. This objective function is equal to:

$$Z_X = \alpha\left(\sum_{k=R}^{R+H-1} T_k\right) + (1-\alpha)C_{m,R+H-1} \tag{16}$$

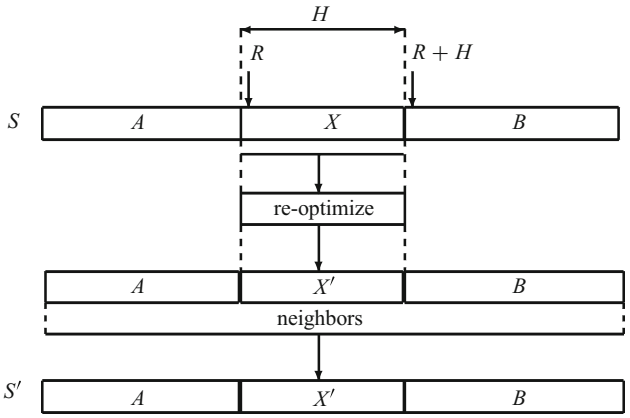

Fig. 8 Illustration of the $MH_{X_2}(S)$ algorithm

where $C_{m,R+H-1}$ denotes the completion time on machine M_m of the job in position $R + H - 1$, i.e. the last job of sequence X'.

First, the total tardiness of A' and the completion time of the last job of A' on each machine are computed and denoted by CA_i, the constraints with CA_i are introduced in the model (same way as in $MH_{XB}(S)$). Then, the optimization of X is done by the solver, and a sequence $X' = X^*$ is obtained. The completion times of the last job of X' on each machine are denoted by CX_i. Then, sequence B' is scheduled after X, taking the CX_i values into account with the following objective function:

$$Z_B = \sum_{k=R+H}^{n} T_k$$

Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A') + \sum T_j(X^*) + \sum T_j(B')$$

Matheuristic algorithm $MH_{X_2}(S)$

Another matheuristic method of type $MH_X(S)$ is called $MH_{X_2}(S)$. Similarly as in "General framework "*AXB*" " section, the sequence A and the sequence B are unchanged. The sequence X is re-optimized, giving sequence X'. The sequence $S' = AX'B$ is a new sequence. Then, the neighborhood operators (among SWAP, BI and FI) are applied to sequence S', in order to modify it and possibly improve this solution. The procedure iterates from $R = 1$, until the time limit called *TimeLimMH*. This process is illustrated in Fig. 8. The algorithm is given in Algorithm 5.

Similarly as in "General framework "*AXB*" " section, the value of S' is given by:

$$\sum T_j(S')^* = \sum T_j(A) + \sum T_j(X^*) + \sum T_j(B)$$

Algorithm 5 The MH_{X_2} algorithm

```
1: Input: S = initial solution
2: improved = true
3: while (CPU ≤ TimeLimMH) and (improved = true) do
4:   improved = false ; R = 0
5:   while R ≤ n − H do
6:     A = (S[1], S[2], ..., S[R−1])
7:     X = (S[R], S[R+1], ..., S[R+H−1]); X′ = re-optimization of X
8:     B = (S[R+H], S[R+H+1], ..., S[n])
9:     S′ = AX′B
10:    if (∑ Tj(S′)* < ∑ Tj(S)) then
11:      improved = true ; S = S′
12:    end if
13:    S′= Neighborhood operator(S)
14:    Compute (∑ Tj(S′))
15:    if (∑ Tj(S′) < ∑ Tj(S)) then
16:      improved = true ; S = S′
17:      if (R + H ≤ n − H) then
18:        R = R + H − 1
19:      end if
20:    end if
21:    R = R + 1
22:  end while
23: end while
24: return(S)
```

Matheuristic algorithm $MH_{POS}(S)$

With this method, the hypothesis is that the positions of jobs in S are not that bad, and only few changes in the sequence X are sufficient to improve the solution. More precisely, we assume that a job in a position $k \in \{R, \ldots, R + H - 1\}$ may only be scheduled at a position between $k - \delta$ and $k + \delta$.

The following constraints are added to the MILP:

$$\sum_{\ell=k-\delta}^{k+\delta} x_{S_{[k]},\ell} = 1, \forall k \in \{R, R + H - 1\}, k - \delta \geq 1, k + \delta \leq n \quad (17)$$

The process is the same as in MH_{XB}, i.e. sequence XB is re-optimized, with this limitation for the possible positions of each job. The idea is that with this limitation in the position changes, it will be possible to increase the size of H. Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(XB^*)$$

Computational experiments

Settings

The algorithms have been tested on a PC Intel $core^{TM}i5$ CPU 2.4 GHz. 108 benchmark instances proposed in Vallada et al. (2008) have been used for the evaluation. Nine instances of these benchmark instances are used for each

Table 1 Comparison of genetic algorithms

$n \times m$	GA_{EDD}			GA_{EN}			GA_{NEH}		
	Best	Cpu(s)	Δ_{EDD} (%)	Best	Cpu(s)	Δ_{EN} (%)	Best	Cpu(s)	Δ_{NEH} (%)
50 × 10	**5**	22.00	**0.79**	3	22.00	3.68	**5**	22.00	**2.78**
50 × 30	3	67.01	2.59	3	67.00	3.92	3	67.00	2.70
50 × 50	**5**	112.01	**1.22**	4	112.01	1.61	0	112.01	2.36
150 × 10	**6**	67.02	**0.49**	2	67.02	6.58	**5**	67.02	**4.35**
150 × 30	**7**	202.04	**1.53**	2	202.05	4.70	2	202.02	3.46
150 × 50	**5**	337.05	**1.39**	3	337.05	1.93	2	337.03	5.81
250 × 10	**8**	112.03	**0.08**	4	112.07	5.76	3	112.04	4.60
250 × 30	**7**	337.05	**0.23**	4	337.12	5.40	2	337.05	7.09
250 × 50	**5**	562.08	**1.07**	3	562.06	2.15	3	562.06	5.21
350 × 10	**8**	157.09	**0.58**	4	157.07	4.47	2	157.03	17.13
350 × 30	**8**	472.13	**0.06**	3	472.10	4.20	1	472.06	15.83
350 × 50	**6**	787.09	**0.23**	4	787.14	3.00	1	787.11	6.22
Sum/avg	73	269.55	0.86	39	269.56	3.95	29	269.54	6.46

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 a uniform distribution between $P(1-\tau-\rho/2)$ and $P(1-\tau+\rho/2)$ following the method of Potts and Van Wassenhove 1982 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 Vallada et al. (2008) for each tuple (n, m, τ, ρ) has been used for the tests, which gives the 108 instances.

For the genetic algorithms, some preliminary experiments have conducted to the following parameters settings:

- $TimeLimGA = (n(m/2) \times 90)/1000$ s (as defined in Vallada et al. 2008),
- $PopSize = |P_k| = 150$ individuals,
- $CrossSize = |C_k| = 200$ individuals,
- $MutSize = |M_k| = 100$ individuals.

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 $<3\,\%$.

The matheuristic algorithms have been tested with several initial solutions. The best solutions have been obtained with the initial sequence EDD. In the following, all the matheuristic algorithms start their process with this sequence of jobs.

The time limit of the matheuristic algorithms has been fixed to $TimeLimMH = (200 + n + m)$ seconds. The size window has been fixed to $H = 6$, the coefficient α in the linear combination for $MH_{XB}(S)$, $MH_{XB_1}(S)$, $MH_X(S)$, $MH_{X_1}(S)$ and $MH_{X_2}(S)$ have been fixed to $\alpha = 0.5$, and the coefficient δ in $MH_{POS}(S)$ has been fixed to $\delta = 3$. The solver that has been used for solving the MILP model is CPLEX v12.2.

In the 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. In each line, the number in bold corresponds to the best value in the line.

Comparison of the genetic algorithms

The three Genetic Algorithms are compared in terms of quality. In Table 1, column 'Best' for 'GA_χ' ($\chi \in \{EDD, EN, NEH\}$) indicates the number of times the method GA_χ outperforms the other methods, column Cpu(s) indicates the average computation time of GA_χ per nine instances, column 'Δ_χ' indicates the average deviation between GA_χ and the best method between GA_{EDD}, GA_{EN} and GA_{NEH}.

$$\Delta_\chi = \frac{GA_\chi - \min(GA_{EDD}, GA_{EN}, GA_{NEH})}{GA_\chi}$$

As we can see from Table 1, the Genetic Algorithm where the initial population 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 0,86 %. This value is around 3,95 % for GA_{EN} and 6,46 % for GA_{NEH}. Algorithm GA_{EDD} has been used in the following for the comparisons with the matheuristic algorithms.

Comparison of the matheuristics

The six matheuristic methods (MH_X, MH_{X_1}, MH_{X_2},MH_{XB}, MH_{XB_1} and MH_{POS}) are compared in terms of quality. In Tables 2 and 3, similarly to the previous table, column 'Best' for 'MH_χ' ($\chi \in \{XB, XB_1, X, X_1, X_2, POS\}$) indicates

Table 2 Comparison of matheuristic algorithms (1)

$n \times m$	MH_X			MH_{X_1}			MH_{X_2}		
	Best	Cpu(s)	Δ_X (%)	Best	Cpu(s)	Δ_{X_1} (%)	Best	Cpu(s)	Δ_{X_2} (%)
50 × 10	3	260.11	2.32	4	260.11	13.34	2	260.09	7.03
50 × 30	4	280.23	8.69	1	280.17	10.06	1	280.15	10.89
50 × 50	2	300.56	1.78	4	300.40	0.48	2	300.39	2.87
150 × 10	3	360.30	5.11	**5**	360.21	**11.82**	3	360.04	5.96
150 × 30	3	380.18	2.87	**7**	380.61	**3.96**	1	380.03	6.46
150 × 50	1	400.56	5.52	**7**	402.81	**3.99**	1	400.21	5.07
250 × 10	3	461.88	3.03	**9**	460.73	**0.00**	3	460.02	2.87
250 × 30	2	486.62	5.75	**8**	483.64	**11.11**	1	480.05	17.25
250 × 50	1	504.20	15.88	**8**	503.42	**0.08**	0	500.11	17.00
350 × 10	3	588.03	14.26	**6**	562.58	**11.37**	4	560.01	1.28
350 × 30	4	608.87	3.63	**6**	591.63	**12.20**	1	580.08	16.11
350 × 50	4	636.10	1.19	0	606.78	27.18	**6**	600.04	**3.46**
Sum/avg	33	438.97	5.84	**65**	432.76	**8.80**	25	430.10	8.02

Table 3 Comparison of matheuristic algorithms (2)

$n \times m$	MH_{XB}			MH_{XB_1}			MH_{POS}		
	Best	Cpu(s)	Δ_{XB} (%)	Best	Cpu(s)	Δ_{XB_1} (%)	Best	Cpu(s)	Δ_{POS} (%)
50 × 10	3	260.23	13.95	2	260.25	21.08	1	260.45	18.02
50 × 30	2	281.08	8.94	0	281.24	17.62	1	281.10	5.02
50 × 50	1	304.26	4.36	0	301.80	11.62	0	302.45	6.32
150 × 10	3	360.41	4.79	1	360.61	42.48	2	361.25	11.21
150 × 30	1	384.19	13.59	0	383.76	41.18	0	385.56	38.44
150 × 50	0	418.15	16.81	0	415.52	28.73	0	406.76	35.96
250 × 10	3	462.14	10.86	1	462.36	50.74	1	462.95	48.76
250 × 30	1	487.57	35.18	1	485.83	42.18	1	489.21	44.69
250 × 50	0	524.79	44.13	0	523.70	44.49	0	520.97	45.42
350 × 10	2	562.87	37.89	2	565.43	44.82	2	564.95	45.59
350 × 30	1	594.79	45.02	1	598.22	45.32	1	594.84	45.72
350 × 50	0	636.69	45.96	0	668.86	45.99	0	665.74	46.43
Sum/Avg	17	439.76	23.46	8	442.30	36.35	9	441.35	32.63

the number of times method MH_χ outperforms the other methods, column Cpu(s) indicates the average computation time of MH_χ per nine instances and column 'Δ_χ' indicates the average deviation between MH_χ and the best matheuristic.

$$\Delta_\chi = \frac{MH_\chi - \min\left(MH_X, MH_{X_1}, MH_{X_2}, MH_{XB}, MH_{XB_1}, MH_{POS}\right)}{MH_\chi}$$

The results clearly show the domination of MH_{X_1}, mainly for instances with $(n \times m) = (150 \times 10)$ up to $(n \times m) = (350 \times 30)$. For $(n \times m) = (350 \times 30)$ instances, MH_{X_2} performs better. We can also see that the MH_X, MH_{X_1}, MH_{X_2} are better than the $MH_{XB}, MH_{XB_1}, MH_{POS}$ algorithms. The main reason is due to the computation time required by CPLEX for solving each MILP model. For MH_{XB} and MH_{XB_1} CPLEX takes time for loading the problem and solving it, as soon as the problem size increases, which limits the number of neighbors explored. The fact that some variables are already decided clearly helps, but is not sufficient for making this method competitive. The advantage of MH_X, MH_{X_1}, MH_{X_2} are that the problem given to CPLEX is very small and very quickly solved. Therefore, the algorithm can perform several times the loop (starting several times with $R = 1$), before reaching quickly a better solution. The method MH_{POS} is not able to find good solutions before the end of the algorithm for the same reasons. Clearly, the quality of the matheuristic is strongly related to its ability of solving quickly to optimality a big number of partial sequences.

Method MH_{X_1} is kept in the next section for the comparison with GA_{EDD}.

Table 4 Comparison of the best matheuristic and the best genetic algorithm

$n \times m$	MH_{X_1}			GA_{EDD}		
	Best	Cpu(s)	Δ_{MH} (%)	Best	Cpu(s)	Δ_{GA} (%)
50 × 10	**5**	260.11	**11.91**	**5**	260.01	**−1.30**
50 × 30	**5**	280.17	**0.06**	4	280.01	−0.20
50 × 50	**6**	300.40	**−0.62**	3	300.02	0.59
150 × 10	**7**	360.21	**6.30**	3	360.02	4.06
150 × 30	**7**	380.61	**−1.21**	3	380.02	1.14
150 × 50	**6**	402.81	**1.35**	3	400.05	−3.67
250 × 10	**9**	460.73	**−3.28**	3	460.03	3.08
250 × 30	**8**	483.64	**8.07**	2	480.05	2.90
250 × 50	**9**	503.42	**−3.59**	1	500.07	3.43
350 × 10	**7**	562.58	**7.39**	4	560.08	3.46
350 × 30	**6**	591.63	**7.42**	4	580.10	3.31
350 × 50	0	606.78	25.36	**9**	600.10	**−22.68**
Sum/avg	75	432.76	4.93	44	430.05	−0.49

Comparison between genetic algorithm and matheuristic

The best proposed matheuristic algorithms (MH_{X_1}) is now compared to the best proposed Genetic Algorithm (GA_{EDD}). The time limit of the Genetic Algorithm and of the matheuristic algorithm have been fixed to $(200 + n + m)$ seconds.

In Table 4, column 'Best' for MH_{X_1} indicates the number of times the method outperforms GA_{EDD}. Column 'Δ_{MH}' indicates the average deviation between MH_X and GA_{EDD}:

$$\Delta_{MH} = \frac{MH_{X_1} - GA_{EDD}}{MH_{X_1}}$$

Column 'Best' for GA_{EDD} indicates the number of times the method GA_{EDD} outperforms method MH_{X_1}, column 'Δ_{GA}' indicates the average deviation between GA_{EDD} and MH_{X_1}.

$$\Delta_{GA} = \frac{GA_{EDD} - MH_{X_1}}{GA_{EDD}}$$

The results clearly show that the matheuristic outperforms the Genetic Algorithm in most of the cases. However, the performance of the Genetic Algorithm is better than the performance of the matheuristic for $(n \times m) = (350 \times 50)$, We also notice that for $n = 50$ jobs, the two methods are equivalent.

Conclusion

We consider in this paper the m-machine flow-shop scheduling problem, with the objective to minimize the total tardiness. We propose several matheuristic algorithms that are compared to a Genetic Algorithm. The computational experiments show that the matheuristic algorithms are competitive with the Genetic Algorithm and that the best matheuristic algorithm outperforms the Genetic Algorithm, except for problems of size (350, 50).

Several research directions can be considered for a future work. The first idea is to embed the resolution of the MILP model into the Genetic Algorithm 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 and the neighborhood search in some matheuristic algorithms. Some first experiments have already been conducted in this sense without a significant success. A third idea is to use the solution of the Genetic Algorithm as an initial solution for the matheuristic algorithm. Finally, the matheuristic 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.

Acknowledgments The authors are very grateful for the financial support of the Vietnamese government.

References

Della Croce, F., Grosso, A., & Salassa, F. (2011). A matheuristic approach for the total completion time two-machines permutation flow-shop problem. In *Lecture Notes in Computer Science, 6622 LNCS*, (pp. 38–47).

Della Croce, F., Ghirardi, M., & Tadei, R. (2004). Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, *10*, 89–104.

Du, J., & Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, *19*, 483–495.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Welsey.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow-shop problem with makespan criterion. *Computers and Operations Research*, *31*, 1891–1909.

Holland, J. A. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan.

Jouglet, A., Oguz, C., & Sevaux, M. (2009). Hybrid flow-shop: A memetic algorithm using constraint-based scheduling for efficient search. *Journal of Mathematical Modelling and Algorithms*, *8*, 271–292.

Kim, Y. D. (1993). A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers and Operations Research*, *20*, 391–401.

Kim, Y. D. (1993). Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, *44*, 19–28.

Koulamas, C. (1998). A guaranteed accuracy shifting bottleneck algorithm for the two-machine flowshop total tardiness problem. *Computers and Operations Research*, *25*, 83–89.

Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343–362.

Liao, C. J., Tseng, C. T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers and Operations Research*, *34*, 3099–3111.

Maniezzo, V., Stutzle, T., & Voß, S. (Eds.). (2010). Matheuristics: Hybridizing metaheuristics and mathematical programming. *Annals of Information Systems* (Vol. 10). Springer.

Nawaz, M., Enscore, E, Jr, & Ham, T. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, *11*(1), 91–95.

Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, *91*, 160–175.

Onwubolu, G. C., & Mutingi, M. (1999). Genetic algorithm for minimizing tardiness in flow-shop scheduling. *Production Planning and Control*, *10*, 462–471.

Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, *17*, 551–557.

Pan, J. C. H., & Fan, E. T. (1997). Two-machine flowshop scheduling to minimize total tardiness. *International Journal of Systems Science*, *28*, 405–414.

Pan, J. C. H., Chen, J. S., & Chao, C. M. (2002). Minimizing tardiness in a two-machine flow-shop. *Computers and Operations Research*, *29*, 869–885.

Pessan, C., Bouquard, J. L., & Néron, E. (2008). Genetic branch-and-bound or exact genetic algorithm? In N. Monmarché, et al. (Eds.), *EA 2007, LNCS 4926* (pp. 136–147).

Pinedo, M. (1995). *Scheduling theory, algorithms, and system*. Upper Saddle River: Prentice Hall.

Portmann, M. C., Vignier, A., Dardilhac, D., & Dezalay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, *107*, 389–400.

Portmann, M. C., & Vignier, A. (2008). Chapter 4: Genetic algorithm and scheduling. In P. Lopez & F. Roubellat (Eds.), *Production scheduling*. New York: Wiley.

Potts, C. N., & Van Wassenhove, L. N. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, *1*, 177–81.

Sen, T., Dileepan, P., & Gupta, J. N. D. (1989). The two-machine flowshop scheduling problem with total tardiness. *Computers and Operations Research*, *16*, 333–340.

Ta, Q. C., Billaut, J. C., & Bouquard, J. L. (2013). Recovering beam search and Matheuristic algorithms for the $F2\|\sum T_j$ scheduling problem. In *11th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'13)*. France: Pont à Mousson.

Talbi, E. G. (Ed.). (2013). *Hybrid metaheuristics, studies in computational intelligence*. Berlin: Springer.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, *177*, 1930–1947.

Vallada, E., Ruiz, R., & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers and Operations Research*, *35*, 1350–1373.

Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, *38*, 57–67.

Wagner, R. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, *6*(2), 131–140.

Werner, F. (1984). On the solution of special sequencing problems. In *Ph.D. Thesis, TU Magdeburg in German*.