Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems

Wajdi Trabelsi, Christophe Sauvey, Nathalie Sauer

Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems

Wajdi Trabelsi, Christophe Sauvey and Nathalie Sauer
LGIPM, Paul Verlaine University of Metz, France

ABSTRACT

Storage or buffer capacities between successive machines in flowshop problems may be unlimited, limited or null. The last two cases can lead to blocking situations. In flowshop scheduling literature, many studies have been performed about classical flowshop problems and also about some problems with only one blocking situation between all machines.
This paper deals with makespan minimization in flowshop scheduling problems where mixed blocking constraints are considered. After a problem description and definitions of different blocking constraints, a mathematical model is presented and heuristics are developed to propose quick solutions to these kinds of problems. Then, metaheuristics are used to improve found solutions. A comparison between heuristics and metaheuristics is then performed.

Keywords: flowshop scheduling problem, blocking constraints, heuristic, metaheuristics, makespan.

1. INTRODUCTION

In order to obtain higher profits, modern production companies usually try to maximize their productivity. The latter goal, among others, can be achieved by optimal or almost optimal jobs scheduling in the production process while reducing storage capacity and even to remove it whenever possible. Storage capacity reduction between machines can lead to situations that are known in literature as blocking situations. Scheduling models differ depending on the technology used and constraints applied in the system. The most common scheduling problem is classical flowshop where buffer space capacity between machines is considered unlimited. Other problems are characterized by only classical blocking constraint *RSb* (Release when Starting Blocking: a machine remains blocked by a job until this job starts on the next machine in routing) and some others by specific blocking constraints *RCb* or *RCb** (Release when Completing Blocking: a machine will be available to treat its next operation after its job is finished on the following machine in the process). In this paper, a general case is presented where successive machines can be subject to different types of blocking constraints, considering makespan as optimization criterion. This problem is likely to better approach industry-derived cases, which can already be seen as flowshop problems.

The first article dealing with a flowshop problem was published over fifty years ago (Johnson, 1954). Since then, many authors have focused on different aspects of this problem. We can cite a

few articles: (Bellman and Gross, 1954) and (Bellman *et al.,* 1982) for classical flowshop problem with two machines and (Nawaz *et al.,* 1983) and (Carlier and Rebaï, 1996) for general case with multiple machines. For greater sized problems, some heuristics and metaheuristics have been proposed, such as (Iyer and Saxena, 2004) and (Siarry and Michalewicz, 2007).

For problems with classical blocking constraint (*RSb*), Sawik (1993, 1995) proposed a heuristic for multi-stage flowshop problem both respectively with and without storage capacity. In (Wang *et al.*, 2006), authors developed a hybrid genetic algorithm for flowshop scheduling with limited buffers. Other papers dealing with flowshop problems with *RSb* constraint, such as (Carraffa *et al.,* 2001) and (Ronconi, 2005), can also be cited.

Regarding *RCb* constraint, an Integer Linear Programming (I.L.P) model, lower bounds and a metaheuristic are presented in (Martinez, 2005) for flowshop and hybrid flowshop cases. These problems have been solved in (Yuan and Sauer, 2007) and (Yuan *et al.*, 2009) by a metaheuristic "Electromagnetism-like Mechanism". A new blocking constraint called *RCb** has been proposed in (Trabelsi *et al.,* 2010) which is an *RCb* constraint variant. In this work, authors propose heuristics to solve jobshop problems with *RCb* and *RCb** constraints.

Finally, for articles that have dealt with different types of constraints mixed in a production system, we can cite (Martinez *et al.,* 2006) who studied complexity issues with *RSb* and *RCb* constraints. We can also cite works by (Grabowski and Pempera, 2000) which consider a real-life problem of scheduling clients' orders of concrete blocks in a factory of building industry modeled as a hybrid flowshop scheduling problem with mixed no-wait/no-store constraints and mixed bottleneck/non-bottleneck machines. To our knowledge, no other author was interested in solving flowshop problems simultaneously subjected to different types of blocking constraints on successive machines in a process.

In this paper, we describe different flowshop cases with one or more blocking constraints. We present a mathematical model, propose a heuristic method to solve mixed flowshop problems, and compare it with NEH heuristic. We also use some improvement methods for both algorithms. Then, genetic algorithms are used to improve found solutions. In the last part, a comparison between heuristics and metaheuristics is performed.

2. PROBLEM DESCRIPTION

In flowshop scheduling problem, a set of n jobs, $J = \{J_1, J_2, \ldots, J_n\}$, must be executed on a set of m machines, $M = \{M_1, M_2, \ldots, M_m\}$. All jobs J_i require the same operation order, $O_i = \{O_{i1}, O_{i2}, \ldots, O_{im}\}$, that must be executed according to the same manufacturing process. Operation O_{ij} needs an execution time P_{ij} on machine $M_j \in M$. Each machine can only execute one job at any time. Pre-emptive operation is not authorized in presented work. Objective function consists in determining best scheduling in order to reduce makespan, *i.e.* time when all operations are completed.

In the following, we present different cases of flowshop problem: classical flowshop problem without blocking constraint (Fig. 1), a flowshop scheduling problem with only one blocking constraint present between all machines (Fig. 2, 3, and 4) and an example in which different blocking constraints are mixed (Fig. 5). To illustrate these different cases, we consider an example with five jobs and five machines.

The following execution time matrix P_{ij}, has list of machines M_j in columns and the list of jobs J_i in lines.

$$P_{i,j} = \begin{pmatrix} 1 & 1 & 2 & 1 & 2 \\ 1 & 3 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 1 \\ 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 1 & 2 & 1 \end{pmatrix} \quad i, j \in \{1,..,5\}$$

In classical flowshop problem, buffer space capacity is considered to be unlimited. Therefore, there is no blocking situation and the system can be called *Wb* (Without blocking): a machine will be immediately available to execute the next operation after its operation in process is finished. In the example presented in Fig. 1, machine M_1 is available to treat job J_4 as soon as job J_3 on machine M_1 will be finished.

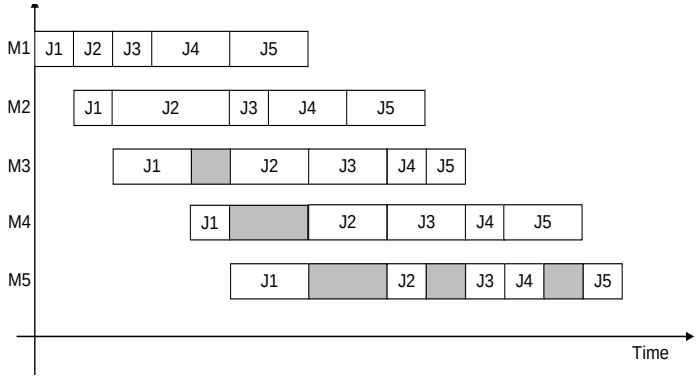


Fig. 1. Classical flowshop problem (*Wb* case)

For flowshop problem with a classical blocking constraint *RSb* (Release when Starting Blocking), a machine remains blocked by a job until this job starts on next machine in routing. This constraint is met in industrial problems where there is no intermediate storage between machines. In our example presented in Fig. 2, job J_3 remains blocked on machine M_1 as long as the following machine M_2 is not available.

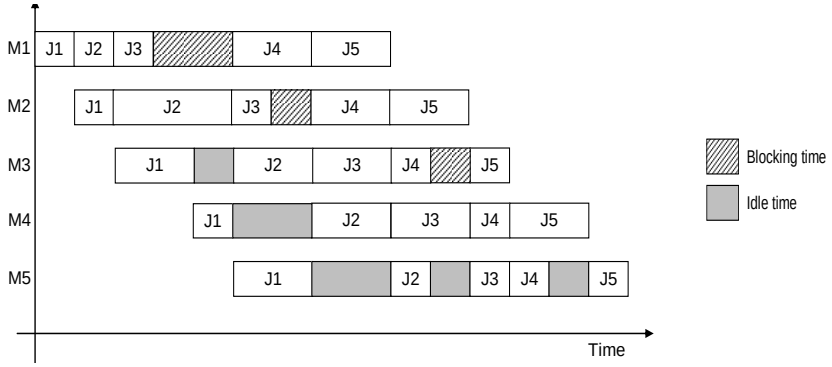


Fig. 2. Flowshop problem with classical blocking *RSb*

In specific blocking *RCb** (Release when Completing Blocking*), a machine will be immediately available to treat its next operation after its job on the following machine in process is finished without regard to whether or not it leaves the machine. In the example presented in Fig. 3, specific blocking *RCb** differs from classical blocking *RSb* by the fact that machine M_1 remains blocked by job J_3 until its operation on machine M_2 is finished. This constraint was introduced for the first time by (Trabelsi *et al*., 2010).

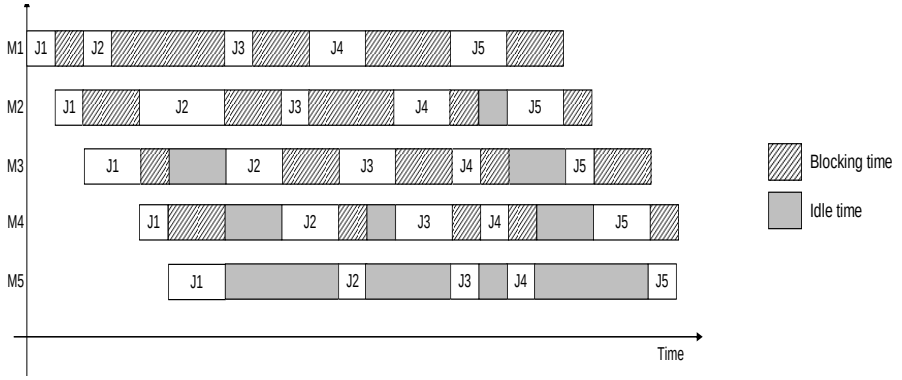


Fig. 3. Flowshop problem with specific blocking *RCb**

We come across this kind of constraint in production lines when two successive machines are depending on same resource (tool or operator…), so they cannot process at same time.

In the case of particular blocking constraint *RCb*, a machine will be immediately available to treat its next operation after the operation of its job on the following machine in process is finished and it has left this machine. This constraint was introduced for the first time by (Dauzère-Pérès *et al*., 2000).

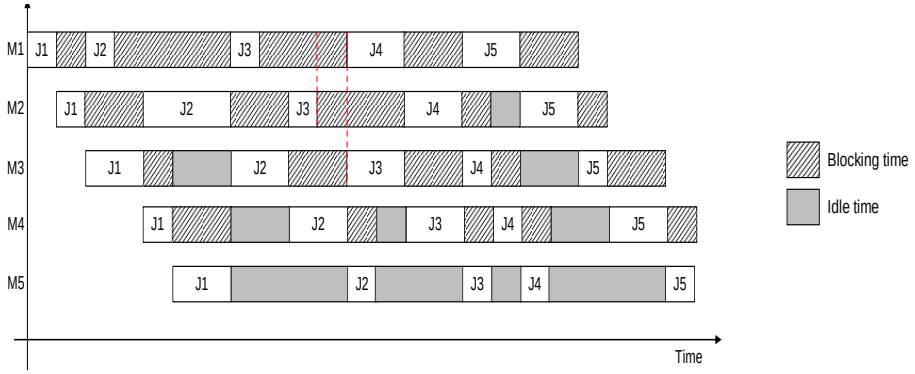


Fig. 4. Flowshop problem with specific blocking *RCb*

RCb constraint is useful in some industrial environments, such as waste treatment industry and aeronautics parts fabrication (Martinez, 2005). For cider brewing applications for example, one cannot melt apples of different customers. In a first step, apples are poured in a bath and then pressed to make apple juice. New customer apples cannot be poured in bath before all of the first customer's apples have been pressed.

The difference between *RCb* and *RCb** blocking is underlined by job J_3 (Fig. 3 and 4): in an *RCb** blocking problem, machine M_1 remains blocked by job J_3 until its following operation on machine M_2 is finished, whereas in *RCb* problem, blocking time is bigger since machine M_1 will only be available when its following operation on M_2 is completed and job J_3 leaves machine M_2. This date corresponds to further following operation on machine M_3 beginning.

In a large production line, we can encounter different types of blocking constraints which depend on intermediate storage between machines, characteristics of machines and technical constraints.

To describe a flowshop problem where different blocking constraints are mixed, we introduce a vector V that contains blocking constraints between machines. Element V_j is blocking constraint between machines M_j and M_{j+1}. This vector has m-1 elements (as many elements as the number of transitions between machines). In the previous example, if V = (*RCb*, *RSb*, *RCb**, *Wb*), then *RCb* is the blocking constraint between machines M_1 and M_2. Thereby, we construct Gantt diagram as follows (Fig. 5):

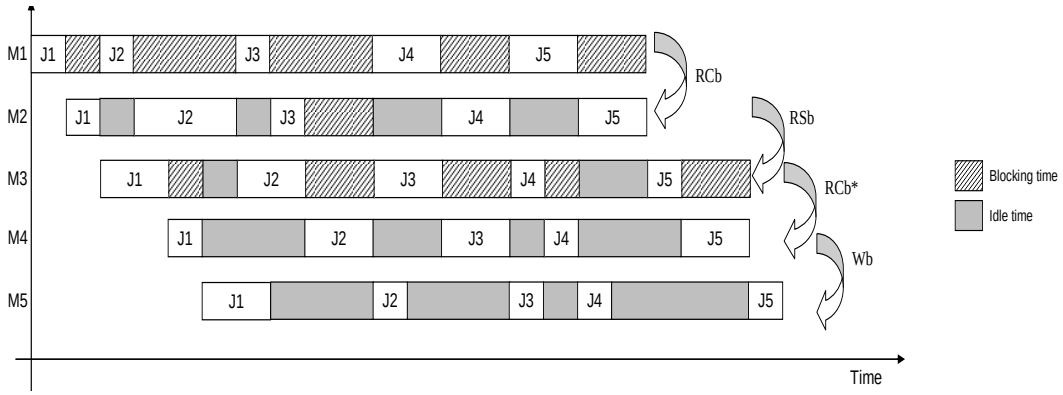


Fig. 5. Flowshop problem with mixed blocking constraints

3. MATHEMATICAL MODEL

In this section, we present a mathematical model for a flowshop problem subject to different blocking constraints. This model is detailed in (Trabelsi *et al*., 2011) in order to minimize makespan of a flowshop with mixed blocking constraint. Since jobs sequence is identical on all machines, we have to determine jobs permutation G that minimizes scheduling completion time or makespan. This model was solved by Xpress-MP optimization software and run on a PC with a 3.16 GHz Core 2 Duo' CPU.

3.1. Parameters

Parameters used in this model are as follows:

- n: Number of jobs.
- m: Number of machines.
- $P_{i,j}$: Job J_i execution time on machine M_j.
- $B_{h,j}$ = 1 if there is an h blocking constraint between machine M_j and machine M_{j+1} and 0 otherwise. With:

h = 1: if there is *Wb*, i.e. no blocking constraint between machine M_j and machine M_{j+1}.

h = 2: if there is an *RSb* constraint between machine M_j and machine M_{j+1}.

h = 3: if there is an *RCb** constraint between machine M_j and machine M_{j+1}.

h = 4: if there is an *RCb* constraint between machine M_j and machine M_{j+1}.

3.2. Variables

Variables used in this model are as follows:

- $S_{k,j}$: Starting time of job at position k in sequence G on machine M_j.
- $C_{k,j}$: Completion time of job at position k in sequence G on machine M_j.
- $G_{i,k}$ = 1 if job J_i is at position k in sequence G and 0 otherwise.

3.3. Model

Developed mathematical model is as follows:

$$Min\ C_{\max} \tag{1}$$

With following constraints:

$$C_{\max} \geq C_{i,m}\ \forall i \in \{1,..,n\} \tag{2}$$

$$S_{k,j} \geq S_{k,j-1} + \sum_{i=1}^{n} P_{i,j-1}.G_{i,k}\ ,\ \forall k \in \{1,..,n\}, \forall j \in \{2,..,m\} \tag{3}$$

$$S_{k,j} \geq C_{k-1,j}.B_{1,j} + S_{k-1,j+1}.B_{2,j} + C_{k-1,j+1}.B_{3,j} + S_{k-1,j+2}.B_{4,j} \quad \forall k \in \{2,..,n\}, \forall j \in \{1,..,m\text{-}2\} \tag{4}$$

$$S_{k,m\text{-}1} \geq C_{k-1,m\text{-}1}.B_{1,m\text{-}1} + S_{k-1,m}.B_{2,m\text{-}1} + C_{k-1,m}.B_{3,m\text{-}1}\ ,\ \forall k \in \{2,..,n\} \tag{5}$$

$$S_{k,m} \geq C_{k-1,m}\ \forall k \in \{2,..,n\} \tag{6}$$

$$C_{k,j} = S_{k,j} + \sum_{i=1}^{n} P_{i,j}.G_{i,k}\ ,\ \forall k \in \{1,..,n\}, \forall j \in \{1,..,m\} \tag{7}$$

$$\sum_{i=1}^{n} G_{i,k} = 1\ ,\ \forall k \in \{1,..,n\} \tag{8}$$

$$\sum_{k=1}^{n} G_{i,k} = 1\ ,\ \forall i \in \{1,..,n\} \tag{9}$$

$$G_{i,k} \in \{0,1\}\ ,\ \forall i \in \{1,..,n\}, \forall k \in \{1,..,n\} \tag{10}$$

$$S_{k,j} \geq 0\ ,\ \forall k \in \{1,..,n\}, \forall j \in \{1,..,m\} \tag{11}$$

$$C_{k,j} \geq 0, \forall k \in \{1,..,n\}, \forall j \in \{1,..,m\} \tag{12}$$

3.4. Equations meaning

Each model constraint is described as follows:

- Equation (1): Objective function of our problem: minimize total completion time of scheduling.
- Equation (2): Makespan value must be greater than or equal to completion time of all jobs on the last machine.
- Equation (3): Represents precedence constraint between the same job on successive operations: to start its operation on a downstream machine, a job must first finish its operation on the upstream machine.
- Equation (4): This equation models different blocking constraints represented by parameter $B_{h,j}$. For example, if there is an *RSb* constraint between machine M_j and machine M_{j+1}, equation (4) becomes: $S_{k,j} \geq S_{k-1,j+1}$
- Equation (5): This is a special case of the above equation which only deals with the penultimate machine. There is no blocking *RCb* at this stage because it depends on the two following machines.
- Equation (6): This is a special case of (4). It only deals with last machine, which can only be without blocking.
- Equation (7): This constraint calculates the completing time of jobs: *i.e.* last machine operations completion time.
- Equation (8): Each job J_i is placed at only one position k in sequence G.
- Equation (9): At each position k of sequence G is only assigned one job J_i.
- Equation (10): $G_{i,k}$ is a Boolean variable. It is equal to 1 if job J_i is at position k in sequence G and 0 otherwise.
- Equation (11): Starting time of jobs cannot be negative.
- Equation (12): Completion time of jobs cannot be negative.

4. HEURISTICS

As presented in (Trabelsi *et al*., 2011), it is not possible to obtain an exact solution for big-sized problems in a reasonable time. Therefore, it is necessary to develop approached methods and estimate their mean error with optimal solution (when it is computable) or lower bounds.

In this paper, we propose some heuristics with some local improvements in order to quickly obtain a feasible solution (in a polynomial time), but not necessarily an optimal one. In this section, we review the NEH method and present a new heuristic. We also use some improvement methods for both algorithms.

4. 1. NEH heuristic presentation

NEH heuristic is one of the most famous heuristics known for flowshop scheduling (Nawaz, Enscore and Ham, 1983). This heuristic is not only efficient, but also very simple to compute. It is a constructive algorithm that selects the longest job not yet sequenced and tries to place it at all possible positions of the partial sequence under construction. Selected position minimizes partial makespan. A lot of references took this heuristic as a reference to compare their results (Zobolas, Tarantilis and Ioannou, 2009), (Ruiz and Stützle, 2007). A pedagogical comparative study of some heuristics is performed in (Ruiz and Maroto, 2005).

Algorithm 1

Let $J = (J_1, J_2, .., J_n)$ a list of jobs ordered by decreasing total processing time
For $i = 1$ to n
    Select job J_i from J
    $C_{pmax} = \infty$
    For $k = 1$ to i
        Place job J_i at position k in partial sequence without changing relative position of other jobs already sequenced
        Calculate C_{max} of associated partial scheduling
        If $C_{max} < C_{pmax}$ then
            $Best_k = k$
            $C_{pmax} = C_{max}$
        End if
    End for
    Place job J_i at position $Best_k$ in partial sequence
End for

Adaptation of this heuristic to our problem is merely respect different blocking constraints between all machines when we calculate the corresponding makespan.

4. 2. TSS heuristic presentation

Proposed heuristic is based on a construction method. This proposal is inspired by an algorithm developed in (Trabelsi *et al*., 2010) for jobshop problem with *RCb* and *RCb** blocking. From a partial solution, it is necessary to choose which job must be placed next. For that, we chose a criterion that combines three parameters:

- *Cpmax*: Partial Makespan: *i.e.* completion time of a new placed job.
- *SomTpsIn*: Sum of inactive times: *i.e.* the time during which machines are inactive, including idle times and blocking times.
- *SomTpsEx*: Sum of execution times of placed operations in partial schedule.

To have a best possible scheduling, we trend on one hand to minimize inactive times of machines and completion time, and on the other hand to maximize machine use, which is depicted in the following criterion:

$Cr = \min (Cpmax + SomTpsIn - SomTpsEx)$.

Then, n possibilities of schedule are constructed by switching jobs to place at first position and finally we choose the one that gives smallest final makespan *Cmax*.

General structure of the proposed algorithm to solve flowshop problems with mixed constraints is given below.

Algorithm 2

Introduce problem data (jobs number, machines number, operations execution time and blocking constraints between machines)
For i = 1 to n (*we calculate all possibilities by switching job to place at first position*)
    We place as first job J_i
    As long as it still remains jobs to place, do
        For t = 1 to [n - (number of already placed jobs)]
            Calculate *Cpmax, SomTpsIn, SomTpsEx*
            Calculate C_{rt} (Criterion after placing a job J_t)
        End for
        Place job that has the smallest C_{rt}
    End while
    Calculate $C_{max,i}$ (*final makespan while placing job J_i at first*)
End for
Choose scheduling that gives the smallest final makespan $C_{max,i}$

Example: Let be a flowshop scheduling problem with four jobs and three machines.

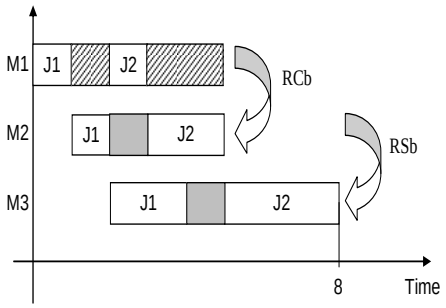
Operations execution times are as follows:

$$P_{i,j} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix} \quad \forall i \in \{1,..,4\},\ \forall j \in \{1,..,3\}$$

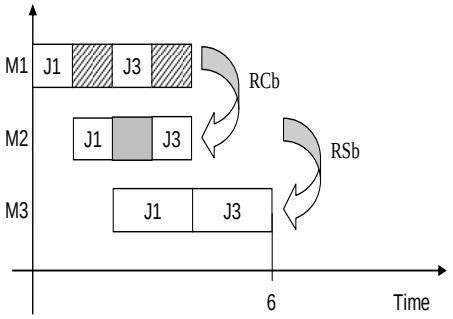
Blocking constraints vector:
V = (*RCb*, *RSb*) , *i.e. RCb* is blocking constraint between machines M_1 and M_2, and *RSb* is blocking constraint between machines M_2 and M_3.

Let be J_1 the job placed at first position. We have then three possibilities to place a following job: J_2, J_3 or J_4 (Figure 6). We use choice criterion between these three cases to choose the job that gives the smallest criterion.

A



B



C

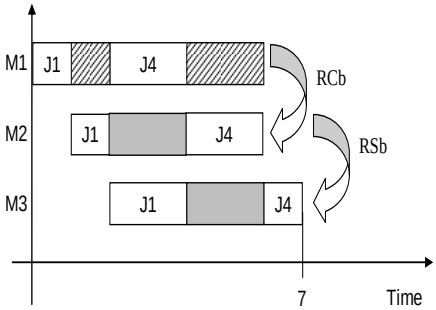


Figure 6: Three possible scheduling cases

We will explain how we calculated these different parameters through first case *i.e.* when job J_2 is placed (Figure 6-A):

- *SomTpsEx* = $\sum_{i=1}^{2} \sum_{j=1}^{3} P_{i,j}$ = 10 ut. We placed two first jobs and we calculate its operations execution times.
- *SomTpsIn* = 1+1+1+2 = 5 ut. It is the sum of inactive times with two time units as idle times (intervals [2-3] on machine M_1 and [4-5] on machine M_2) and three time units as blocking time due to *RCb* constraint between machines M_1 on M_2.
- *Cpmax* = 8 ut. This partial makespan represents minimal date from which all machines will be available.

Then, $Cr(A) = 8 + 5 - 10 = 3$ ut.

The results of each possibility (J_2, J_3 or J_4) are given in Table 1. In this example, job J_3 is placed as second job in partial sequence because criterion *Cr* is minimal with job J_3.

	Execution time « *SomTpsEx* » (ut)	Idle time + Blocking time « *SomTpsIn* » (ut)	Partial makespan « *Cpmax* » (ut)	*Cr*
A	10	2 + 3	8	3
B	**8**	**1 + 2**	**6**	**1**
C	9	4 + 3	7	5

Table 1: Choice criterion used to place next operation

Final solution given by above presented algorithm for this $F3|\text{Mixed}|C_{max}$ example is presented by the Gantt diagram (Figure 7).

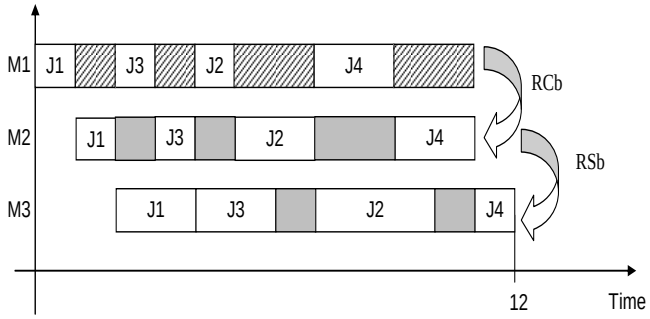


Figure 7: Complete scheduling for $F3|\text{Mixed}|C_{max}$

4. 3. NEH local improvement

This local improvement is based on the iterative NEH method. In fact, we start with the scheduling order obtained by initial heuristic and then we use the iterative part of NEH algorithm, as long as we notice a solution improvement.

Algorithm 3

Let $J = (J_1, J_2, .., J_n)$ a jobs order obtained by initial heuristic
For $i = 1$ to n
    Select job J_i from J
    $C_{pmax} = \infty$
    For $k = 1$ to n
        Place job J_i at position k in partial sequence without changing relative position of other jobs already sequenced
        Calculate C_{max} of associated partial scheduling
        If $C_{max} < C_{pmax}$ then
            $Best_k = k$
            $C_{pmax} = C_{max}$
        End if
    End for
    Place job J_i at position $Best_k$ in partial sequence
End for

4. 4. TSS local improvement

In regard to classical flowshop problems, *RCb* blocking induces supplementary blocking time, which is defined as differential blocking time. In (Sauvey and Sauer, 2010), the authors give two definitions of differential blocking time depending on whether it comes before occurrence of job J_i on machine M_j, or after.
Differential blocking time "before" job J_i on machine M_j is due to insufficient execution time of job J_i on machine M_{j-1} in front of the execution time of the preceding job in the routing on machine M_{j+1}. Differential blocking time "after" job J_i on machine M_j is due to insufficient execution time of the preceding job in the routing on machine M_{j+2}. (Figure 8)

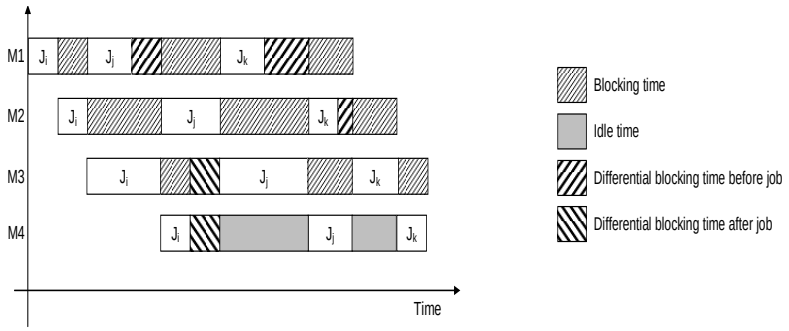


Fig. 8. Four machines problem with *RCb* blocking constraint

TSS local improvement is based on the fact that, in an obtained sequence, a job with greater differential blocking time is permuted with all other jobs of sequence and the best solution is chosen. We continue as long as permutations improve makespan.

4. 5. Experimental results

Presented works had been performed to see the effect of different improvements on obtained results by heuristics solving.

To determine whether our heuristic TSS would give good results, we compared it with the NEH algorithm in step (a), and then for both algorithms we added an NEH local improvement in step (b) and a TSS local improvement in step (c). At the end, the best solution is kept as the better makespan (d).

For each problem with fixed dimension, we generated 100 different instances (20 instances for problems bigger than 11 jobs / 10 machines) for which we know optimal solutions, obtained in works of (Trabelsi *et al*., 2011). For each of these instances, execution times were generated uniformly in interval [0, 99]. All heuristics processes are programmed in the "C++" language and run on a PC with a 3.16 GHz Core 2 Duo' CPU.

Table 2 presents the mean error between optimal solutions *Opt* and makespan C_{max}. Error percentage is calculated as follows:

$$\%\,err = \frac{C_{\max} - Opt}{Opt} \times 100$$

	mach	5		6		7		10		15		20		50		100	
jobs	heuristic	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH
	a	4,46%	3,75%	3,39%	4,61%	3,94%	3,72%	2,86%	4,25%	2,55%	3,42%	1,76%	3,70%	1,20%	2,55%	0,86%	1,73%
5	b	0,48%	0,38%	0,21%	0,17%	0,29%	0,29%	0,40%	0,32%	0,44%	0,34%	0,39%	0,22%	0,36%	0,27%	0,20%	0,12%
	c	0,45%	0,23%	0,19%	0,15%	0,23%	0,19%	0,38%	0,13%	0,38%	0,25%	0,35%	0,15%	0,32%	0,17%	0,19%	0,08%
	d	0,07%		0,10%		0,08%		0,08%		0,11%		0,08%		0,08%		0,06%	
	a	4,15%	4,02%	4,25%	3,97%	5,37%	4,58%	4,14%	3,98%	4,35%	3,82%	2,91%	4,09%	1,78%	2,56%	1,12%	1,84%
6	b	0,41%	0,74%	0,38%	0,33%	0,61%	0,43%	0,39%	0,54%	0,86%	0,81%	0,73%	0,50%	0,58%	0,24%	0,39%	0,21%
	c	0,35%	0,59%	0,37%	0,32%	0,53%	0,30%	0,36%	0,45%	0,76%	0,71%	0,67%	0,43%	0,55%	0,19%	0,35%	0,18%
	d	0,11%		0,17%		0,17%		0,22%		0,45%		0,28%		0,12%		0,11%	
	a	5,21%	4,07%	4,87%	3,72%	5,68%	4,24%	4,92%	4,02%	4,58%	3,84%	3,95%	4,01%	2,46%	2,65%	1,72%	2,07%
7	b	0,50%	0,87%	0,46%	0,46%	0,78%	0,89%	0,82%	0,68%	0,93%	0,73%	0,84%	0,58%	0,88%	0,37%	0,53%	0,26%
	c	0,48%	0,69%	0,46%	0,44%	0,66%	0,75%	0,69%	0,57%	0,84%	0,59%	0,75%	0,49%	0,85%	0,34%	0,48%	0,23%
	d	0,24%		0,25%		0,35%		0,31%		0,31%		0,32%		0,22%		0,17%	
	a	5,35%	3,87%	5,21%	3,26%	6,42%	3,77%	5,90%	3,95%	5,80%	4,10%	4,36%	3,95%	3,09%	2,82%	2,16%	2,18%
8	b	0,82%	0,95%	0,54%	0,52%	0,94%	0,79%	1,13%	1,03%	1,23%	1,00%	1,09%	0,76%	0,96%	0,52%	0,64%	0,42%
	c	0,75%	0,79%	0,50%	0,48%	0,84%	0,63%	1,02%	0,84%	1,14%	0,75%	1,01%	0,68%	0,93%	0,48%	0,61%	0,40%
	d	0,47%		0,28%		0,41%		0,57%		0,53%		0,44%		0,36%		0,28%	
	a	5,33%	3,67%	6,10%	3,45%	6,35%	4,01%	6,27%	4,12%	5,74%	4,17%	5,19%	4,25%	3,33%	3,37%	2,43%	2,42%
9	b	0,57%	0,85%	0,80%	0,73%	1,25%	1,23%	1,53%	1,05%	1,49%	1,26%	1,18%	0,87%	1,26%	0,73%	0,92%	0,62%
	c	0,53%	0,76%	0,75%	0,59%	1,14%	1,02%	1,49%	0,85%	1,41%	1,14%	1,11%	0,80%	1,13%	0,72%	0,82%	0,61%
	d	0,39%		0,38%		0,78%		0,65%		0,80%		0,59%		0,52%		0,46%	
	a	5,51%	3,72%	5,57%	3,17%	6,33%	4,02%	6,86%	4,54%	6,77%	4,56%	5,67%	4,86%	3,95%	3,20%	2,94%	2,57%
10	b	0,81%	0,94%	0,74%	0,81%	1,46%	1,27%	1,31%	1,25%	2,16%	1,63%	1,51%	1,32%	1,60%	0,92%	1,08%	0,63%
	c	0,72%	0,87%	0,68%	0,71%	1,31%	1,16%	1,22%	1,14%	2,10%	1,49%	1,43%	1,18%	1,51%	0,87%	1,01%	0,58%
	d	0,41%		0,39%		0,82%		0,74%		1,17%		0,89%		0,75%		0,47%	
	a	5,52%	3,33%	6,21%	3,19%	7,20%	4,23%	7,03%	4,39%	7,16%	5,74%	6,03%	3,89%	4,32%	3,87%	3,43%	2,56%
11	b	0,75%	0,96%	1,04%	0,95%	1,57%	1,52%	1,49%	1,30%	1,67%	1,90%	1,97%	1,44%	1,44%	1,11%	1,07%	0,72%
	c	0,65%	0,87%	0,97%	0,82%	1,44%	1,43%	1,32%	1,18%	1,57%	1,83%	1,95%	1,44%	1,43%	1,05%	1,01%	0,67%
	d	0,45%		0,53%		0,97%		0,80%		1,09%		1,13%		0,94%		0,52%	
	a	5,47%	3,31%	5,27%	3,21%	7,15%	4,07%	7,58%	5,12%	7,85%	5,57%	6,41%	4,84%	4,80%	3,64%	3,49%	2,50%
12	b	0,77%	0,81%	0,96%	0,88%	1,34%	1,46%	2,00%	1,70%	2,33%	2,02%	1,67%	1,49%	1,56%	1,11%	1,12%	0,86%
	c	0,75%	0,79%	0,94%	0,81%	1,24%	1,30%	1,85%	1,69%	2,22%	1,87%	1,46%	1,43%	1,56%	1,01%	1,06%	0,82%
	d	0,51%		0,52%		0,90%		1,15%		1,61%		0,98%		0,84%		0,58%	

Table 2 : Error percentages between optimal solutions and makespan by using different heuristics

Results presented in Table 2 prove that for small-sized problems (with 5 to 8 jobs), NEH and TSS have a very close error percentage (TSS is often better for problems with big number of machines), while for problems of larger sizes, NEH is often much better than TSS. (Table 2, lines a)

By adding an NEH local improvement on previous results, we note that error percentage becomes significantly smaller and comparable for both proposed heuristics. We also note that with TSS as a first heuristic, we only obtained best results for problems with 5 machines and some other problems (6/10, 7/7, 10/6...). (Table 2, lines b)

Since NEH local improvement gives good results, and as computing time is negligible, we can add other improvements. We added TSS local improvement, and that still improves results for both types of heuristics (Table2, lines c).

Finally, after adding different improvements to TSS and NEH heuristics, the best solution is kept for both cases (Table2, lines d) and we can note that error percentage is improving again, which means that the best solutions found with NEH or TSS heuristic taken first are often different.

We can notice the interest of varying both heuristics and local improvements:

- It gives us the possibility of tackling problems from different angles. Obtained solutions are rather different, which allows us to choose the best one.
- As computing time of problems solved by heuristics and local improvement methods is negligible, we can mix them together to improve again solutions.

We also notice that maximum error percentage by using our methods is obtained for about 15 machines.

5. META HEURISTICS

5. 1. Genetic algorithms

In order to further improve solutions obtained by heuristic methods, we propose to use genetic algorithms which have proven their efficiency on discrete optimization problems since 1962. Holland's works on adaptive problems (Holland, 1962) and its fields of application have widely spread (Holger and Stützle, 2005), (Siarry and Michalewicz, 2007).

Results obtained by genetic algorithm for flowshop scheduling problems with *RCb* blocking constraint proposed in works of (Sauvey and Sauer, 2010) encouraged us to perform this method on mixed blocking flowshop problems. We used this metaheuristic to consider different blocking constraints simultaneously.

This genetic algorithm runs with two main phases: selection and evaluation (Figure 8). An initial population is randomly generated and directly evaluated. Evaluation consists in our problem to calculate each individual's makespan in order to sort them. Then, a selection is operated on individuals in order to determine those likely to give the best results. Our selection includes crossing, mutation, and addition of new individuals at each new operation.

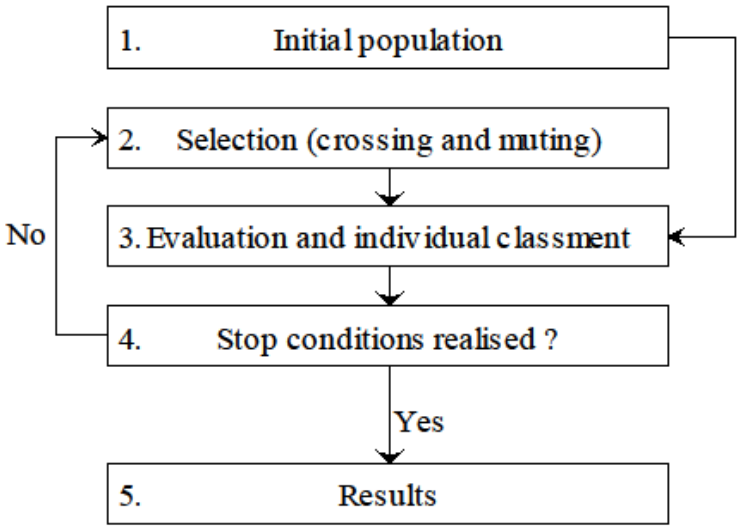


Fig. 8. Genetic algorithm presentation

In our genetic algorithm, we can impose best individuals keeping percentage (*pc_best*), new randomly generated people introduction percentage (*pc_new*), and crossing percentage (*pc_cross*). Individuals selected for crossing are taken among the preceding generation's best individuals and new randomly generated people. The remaining percentage of the created generation is created by mutation. Mutation is a random alteration of an individual's genes. In our algorithm, we select for mutation among best-kept individuals in order to see whether they further improve their good results.

As represented on Fig. 8, the loop composed of steps 2 and 3 is repeated as long as it is allowed by step 4. To perform our simulations, we chose to stop when the population no longer evolves quickly enough or stops evolving altogether. Population is then homogeneous and we can hope that it is near to optimum. The criteria we have retained to stop our simulations is the number of times when the best value was identical (*ctbvi*). It is very easy to put a loop counter to count the number of times loop 2-3-4 is performed. At each loop iteration, we hope that the evaluation of the new population will give a better individuate than best individuate evaluated up to time. Then, if the best individuate of the new current population improves the algorithm solution, it is memorized as the algorithm temporary solution and a loop counter is initialized to 0. If the algorithm temporary solution is not improved by the current loop population, this loop counter is incremented up to a maximum value defined by the parameter *ctbvi*. The returned algorithm solution is a temporary solution which has not evolved since *ctbvi* algorithm loops.

5. 2. Experimental results

Percentages of crossing, mutation, and new individuals are adjustable. To perform work presented in this paper, we set for our experiments these percentages as follows: (*pc_best* : 0,1 ; *pc_new* : 0,1 ; *pc_cross* : 0.65). The criteria we have retained to stop our simulations is the number of times when the best value was identical (*ctbvi* = 500). The genetic algorithms processes are also programmed in the "C++" language and run on a PC with a 3.16 GHz Core 2 Duo' CPU.

In table 3, we can see at first (lines a) mean error between optimal solutions and makespan obtained by using genetic algorithm with 50 individuals. We add local improvements (lines b) to TSS and NEH. Then, in (lines c) we take best solution that gives the better makespan between the last genetic algorithm with local improvements (b) and another genetic algorithm with 100 individuals. Computing time of (c) is then presented.

metaheuristic	jobs/mach	5	6	7	10	15	20	50	100
a		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
b	5	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
c		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
computing time		0	0	0	0	0	0	1	2,06
a		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
b	6	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
c		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
computing time		0	0	0	0	0	0,02	1,03	3,01
a		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
b	7	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
c		0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
computing time		0	0	0	0,02	0,1	0,11	1,42	3,45
a		0,01%	0,00%	0,03%	0,03%	0,03%	0,05%	0,00%	0,02%
b	8	0,01%	0,00%	0,01%	0,03%	0,02%	0,05%	0,00%	0,02%
c		0,00%	0,00%	0,00%	0,00%	0,00%	0,01%	0,00%	0,00%
computing time		0	0,01	0,05	0,25	0,4	0,59	1,97	5,06
a		0,04%	0,05%	0,08%	0,09%	0,15%	0,07%	0,09%	0,04%
b	9	0,03%	0,05%	0,08%	0,08%	0,14%	0,07%	0,09%	0,04%
c		0,00%	0,00%	0,00%	0,02%	0,04%	0,01%	0,02%	0,01%
computing time		0,05	0,08	0,14	0,51	0,85	1,15	3,61	7,15
a		0,11%	0,08%	0,10%	0,19%	0,21%	0,17%	0,17%	0,10%
b	10	0,08%	0,07%	0,10%	0,18%	0,21%	0,17%	0,17%	0,09%
c		0,04%	0,02%	0,06%	0,05%	0,10%	0,04%	0,05%	0,05%
computing time		0,13	0,05	0,13	0,44	0,76	1,1	3,23	6,1
a		0,15%	0,13%	0,22%	0,23%	0,36%	0,39%	0,28%	0,07%
b	11	0,12%	0,11%	0,22%	0,21%	0,36%	0,39%	0,28%	0,07%
c		0,02%	0,04%	0,09%	0,10%	0,20%	0,29%	0,12%	0,04%
computing time		0,48	0,23	0,69	0,82	2,15	2,95	6,5	9,95
a		0,19%	0,14%	0,31%	0,36%	0,64%	0,64%	0,34%	0,23%
b	12	0,17%	0,13%	0,29%	0,35%	0,57%	0,60%	0,34%	0,23%
c		0,06%	0,06%	0,17%	0,25%	0,34%	0,27%	0,23%	0,13%
computing time		0,56	0,39	0,61	1	2,35	2,65	5,3	11,15

Table 3 : Error percentages between optimal solutions and makespan obtained with different metaheuristics

We note that genetic algorithms are very efficient: 0% error for problems up to 8 jobs with a computing time that is almost negligible, and remains inferior to 0.64% error for all other problems up to 12 jobs and 100 machines with a reasonable computing time.

If we detail further results, we find that there is not much improvement of results in (b) when adding NEH and TSS local improvements. On the contrary, in (c), we see a marked improvement

on obtained solutions, allowing us to note that tackling a problem from different angles (*i.e.* use of two genetic algorithms or more) is better than adding local improvements to obtained schedules.

6. CONCLUSION AND PERSPECTIVES

In this paper, we focused on flowshop scheduling problem with mixed blocking constraints and its solving with heuristic and metaheuristic methods. Among the most effective heuristic methods, NEH has been taken to be locally improved and compared with our proposed heuristic TSS. We have tackled this problem with NEH and TSS heuristics successively, with two local improvements. First one is based on the iterative NEH method. Second is based on the permutation of the job that has the greatest differential blocking time in the sequence with all other jobs of this sequence. The best solution of the two heuristics with both improving methods is then chosen.

Our methods, when combined, improves significantly and very quickly our solutions: less than 1.61% error between optimal solutions and makespan in a time less than a second for a tested range of problems. Using metaheuristics, we further improved the results (highest error percentage is 0.34%), but it can be considerably more time consuming: up to 11.15s per problem on realized tests.

In our future works, we intend to further improve the efficiency of these methods, especially in computation time for the genetic algorithm. Then, we will try to adapt our methods on hybrid flowshop problems in order to obtain comparable results.

REFERENCES

Bellman, R. and O. Gross, (1954). Some combinatorial problems arising in the theory of multi-stage processes, pp. 175-183. Journal of Society of Industrial and Applied Mathematics 2.

Bellman, R., A.O. Esogbue and I. Nabeshima, (1982). Mathematical Aspects of Scheduling and Applications. Pergamon Press, New York.

Caraffa, V., S. Ianes, T.P. Bagchi and C. Sriskandarajah, (2001). Minimizing makespan in a blocking flow-shop using genetic algorithms, vol. 70, pp. 101-115. International Journal Production Economics.

Carlier, J. and I. Rebaï, (1996). Two branch and bound algorithms for the permutation flow, vol. 90, pp. 238-251. European Journal of Operational Research.

Dauzère-Pérès S., C. Pavageau and N. Sauer, (2000). Modélisation et résolution par PLNE d'un problème réel d'ordonnancement avec contraintes de blocage, pp. 216-217. 3ème congrès ROADEF, Nantes.

Grabowski, J. and J. Pempera, (2000). Sequencing of jobs in some production systems, vol. 125, pp. 535-550. European Journal of Operational Research.

Holger H.H. and T. Stützle, (2005). Stochastic Local Search: Foundations and Applications, Morgan Kaufmann, San Francisco, CA, USA.

Holland, J. H. (1962). Outline for logical theory of adaptive systems, vol. 3, pp. 297-314. Journal of the association of computing machinery.

Iyer, S. K. and B. Saxena, (2004). Improved genetic algorithm for the permutation flow-shop scheduling problem, vol. 31, pp. 593–606. Computers and Operations Research.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included, vol. 1, pp. 61-68. Naval Research Logistics Quarterly.

Martinez, S., S. Dauzière-Pérès, C. Guèret, Y. Mati and N. Sauer, (2006). Complexity of flowshop scheduling problems with a new blocking constraint, vol. 169(3), pp. 855-864. European Journal of Operational Research.

Martinez., S. (2005). Ordonnancement de systèmes de production avec contraintes de blocage, Ph.D thesis, IRCCyN, Nantes, France.

Nawaz, M., Enscore, E. and Ham I. (1983). A heuristic algorithm for the m-machine, n-job flowshop sequencing problem, vol. 11 (1), 91-95. OMEGA, The International Journal of Management Science.

Ronconi, D.P. (2005). A Branch-and-Bound Algorithm to Minimize the Makespan in a Flow-shop with Blocking, pp. 53-65. Annals of Operations Research.

Ruiz, R. and C. Marotto, (2005). A comprehensive review and evaluation of permutation flowshop heuristics, vol. 165, pp. 479-494. European Journal of Operational Research.

Ruiz, R., and T. Stützle, (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, vol. 177, pp. 2033-2049. European Journal of Operational Research.

Siarry, P. and Z. Michalewicz, (2007). Advances in Metaheuristics for Hard Optimization, Springer-Natural Computing Series.

Sauvey, C. and N. Sauer, (2010). A genetic algorithm with genes-association recognition for flowshop scheduling problems, Journal of Intelligent Manufacturing.

Sawik, T.J. (1993). A schedule algorithm for flexible flow lines with limited intermediate buffers, vol. 9, pp. 127-138. Applied Stochastic Models and Data Analysis.

Sawik, T.J. (1995). Scheduling flexible flow lines with no in-process buffers, vol. 33, pp. 1357-1367. Int. J. Prod. Res.

Trabelsi, W., C. Sauvey and N. Sauer, (2010). Heuristic methods for problems with blocking constraints solving jobshop scheduling. MOSIM'2010, 8th International Conference on Modelling and Simulation, Hammamet, Tunisia.

Trabelsi, W., C. Sauvey and N. Sauer, (2011). Mathematical Model and Lower Bound for Flowshop Problem With Mixed Blocking Constraints. IESM'2011, International Conference on Industrial Engineering and Systems Management, Metz, France.

Wang, L., L. Zhang and D.Zheng, (2006). An effective hybrid genetic algorithm for flowshop scheduling with limited buffers. Computers and Operations Research. Vol. 33(10), pp.2960–71.

Yuan, K. and N. Sauer, (2007). Application of EM algorithm to flow-shop scheduling problems with a special blocking. ISEM.

Yuan, K., N. Sauer and C.Sauvey, (2009). Application of EM algorithm to hybrid flow shop scheduling problems with a special blocking. Emerging Technologies and Factory Automation. IEEE International Conference.

Zobolas, G. I., Tarantilis, C. D. and G. Ioannou, (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, vol. 36, pp. 1249-1267. Computers and Operations Research.