A Simulated Annealing Algorithm for Multi Objective Flexible Job Shop Scheduling with Overlapping in Operations

Mehrzad Abdi Khalife[a], Babak Abbasi[b], Amir Hossein Kamali Dolat Abadi[a,*]

[a] *Islamic Azad University, Qazvin Branch, Department of Industrial Engineering, Qazvin, Iran*

[b] *Sharif University of Technology, Department of Industrial Engineering, Tehran, Iran*

Received 5 Oct., 2009; Revised 23 Oct., 2009; Accepted 2 Nov., 2009

Abstract

In this paper, we considered solving approaches to flexible job shop problems. Makespan is not a good evaluation criterion with overlapping in operations assumption. Accordingly, in addition to makespan, we used total machine work loading time and critical machine work loading time as evaluation criteria. As overlapping in operations is a practical assumption in chemical, petrochemical, and glass industries, we used simulated annealing algorithm for multi-objective flexible job shop scheduling problem with overlapping in operations to find a suitable solution. To evaluate performance of the algorithm, we developed a mixed integer linear programming model, and solved it with the classical method (branch and bound). The results showed that in small size problems, the solutions of the proposed algorithm and the mathematical model were so close, and in medium size problems, they only had lower and upper bounds of solution and our proposed algorithm had a suitable solution. We used an experimental design for improving the proposed algorithm.

Keywords: Flexible job shop, Scheduling, Overlapping, Multi-objective optimization, Simulated Annealing, Combinatorial optimization.

1. Introduction

In the factory environment, time and resource management is an important matter. The initial way of managing time and resource is to schedule operations in manufacturing systems. Due to the competitiveness nature of today's markets, factories must use flexible manufacturing equipments, and thus flexible job shop scheduling finds a new and important place in factories. Through considering flexibility assumption in job shop, we can see and describe more diversification and complex problems with polynomial algorithms. Many researchers are working in job shop environment. Garey et al. [8] are first who introduced job shop scheduling problems. Some researchers like Brandimart [2] and Paulli [14] use dispatching rules for solving flexible job shop scheduling problems. Attention to size proved that job shop scheduling problems are NP-Hard (Garey et al. [8]) and with added flexibility increase complexity more than job shop.

After Garey et al., Xia and Wu [18] and Fattahi et al. [6] proved that flexible job shop scheduling problems are NP-Hard too.

Since flexible job shop scheduling problems are NP-Hard, heuristic and Meta heuristic algorithms are used for solving them. Hurink et al. [10], Dauzere-peres and Paulli [3], Mastrolill and Gambardella [13], and Rigoa [16] use Tabu search algorithm for addressing flexible job shop scheduling problems. Besides, Genetic algorithm is one of the most frequently used full algorithms that many researchers including Kacem et al. a,b [11][12], Zandieh et al. [20], Gholami and Zandieh [9], Gao et al. [7], and Pezzella et al. [15] use for solving flexible job shop scheduling problems.

Another heuristic and meta heuristic algorithm is PSO algorithm which was used by Xia and Wu [18] for assigning operations to machines and simulated annealing algorithm was used for scheduling those operations.

To relate flexible job shop scheduling problems to the real world, we should add assumptions like sequence

[*] Corresponding author E-mail: amir_kamaly2002@yahoo.com

dependent setup time (Saidi Mehrabad and Fattahi [17]), fuzzy logic (Kacem et al. b [12]) and overlapping in operations (Alvarez-Valdes et al. [1] and Fattahi et al. [5]) to problems. In this paper, we draw on the approach suggested by Alvarez-Valdes et al. [1] and Fattahi et al. [5] to solve multi-objective flexible job shop scheduling problems with overlapping in operations.

As in the current market, the demand is in batches, we can consider overlapping in operations, for instance, if an order of 500 glasses of a given model is placed, the production order would consist of decorating and packing these 500 glasses, and the previous production order would consist of producing 10,000 glasses, 500 of which would proceed to the original order and the remaining 9500 would be stored. This scheme shows relationships and priorities between production orders.

The operations to be performed on the pieces can, and in fact should, overlap. If an order consists of decorating and packing 500 glasses, we do not have to wait until all the glasses are decorated to start packing them. Sometimes this overlapping is automatic, that is, as soon as the first piece is processed on one machine, it goes directly to the next machine. On other occasions, the overlapping is limited by structural constraints such as the size of the box to be packed or the capacity of the container used to move the pieces from one machine to another. Overlapping can even happen between operations of different orders linked by precedence constraints.

For solving more than two flexible job shop scheduling problems, we can use two different approaches: integrated approach and decomposition approach. In the integrated approach, operations are assigned to machine and are sequenced in one stage. Hurink et al. [10], Dauzere-Peres and Paulli [3], and Mastrololli and Gambardella [13] used this approach in their studies.

The decomposition approach in which operations are assigned to machines and are sequenced in two separate stages is developed for reducing complexity of problems. This approach includes two stages: first, operations are assigned to available machines and then the assigned operations are scheduled in machines and after solving to stage respectively we have the final solutions. The decomposition approach in flexible job shop scheduling problems was firstly introduced by Brandimart [2]. They solved a general problem with dispatching rules and improved the solutions with tabu search algorithm. Following Brandimarte [2], many researchers used the decomposition approach in flexible job shop scheduling problems (e.g. Xia and Wu [18], Fattahi et al. [5][6], and Fattahi [4]).

Many objective functions have been considered in flexible job shop scheduling programs. The main focus of attention has been on minimizing makespan that Fattahi et al. [5][6], Saidi Mehrabad and Fattahi [17], and Zandieh et al. [20] use as an objective function and if it describes dynamic flexible job shop scheduling problems, the researchers can determine total tardy, mean tardy and the number of tardy jobs (Gholami and Zandieh [9], Fattahi [4]). Kacem et al. a,b [11][12], Xia and Wu [18], Zhang et al. [21], and Gao et al. [7] describe flexible job shop scheduling problems with three objective functions including makespan, critical machine work loading time, and total work loading time. Then Xing et al. [19] describe these three objective functions in a flexible job shop environment with simulation method.

The present paper with the following organization aims to present meta heuristic method simulated annealing algorithm for solving multi-objective flexible job shop scheduling problems with overlapping in operations: section 2 presents multi-objective flexible job shop scheduling problems with overlapping in operations and a mathematical model for solving the problems with usual deterministic mathematical approaches. In this paper branch and bound method is used. Section 3 introduces simulated annealing algorithm briefly and section 4 explains how this algorithm is used in this paper. In this section, both approaches to assignment and scheduling of operations are illustrated. Proposed algorithm parameters tuned with use experimental design are presented in section 5. Section 6 presents the numerical results and section 7 includes the conclusion and suggestions for further research.

2. Problem description and formulation

We formulate the FJSS problem with overlapping in operations as follows. The problem has m machines and n jobs. Each job consists of a sequence of operations$O_{j,h}$, $h = 1,\ldots,h_j$, where$O_{j,h}$, and h_j denote the hth operation of job j and the number of operations required for job j, respectively. The machine set is represented as M, $M = \{M_1, M_2, \ldots, M_m\}$. Unless stated otherwise, index i denotes a machine, index j denotes jobs and index h denotes operations throughout the paper. The execution of each operation h of a job j (denoted as$O_{j,h}$) requires one machine out of a set of given machines called$M_{j,h} \subset M$, and a process time, $p_{i,j,h}$, for each alternative machine. The set$M_{j,h}$ is defined by $a_{i,j,h}$ as described below. Index k which is assigned to each machine determines the sequence of the assigned operations on it (Fattahi et al. [6]).

We consider multi-objective functions. The weighted sum of the three objective values taken as objective functions is as follows:

F(c) =0.5* F1(c) +0.3* F2(c) +0.2 * F3(c)

Where F(c) denotes the objective value of schedule c, F1(c), F2(c) and F3(c) denote the makespan, Critical machine work loading time and Total machine work loading time of schedule c, respectively. We consider WL_i for Work loading time for machine i and TWL for Total work loading time for all machines. The weight of different objectives is determined empirically. If the decision maker pays more attention to a certain objective, we can define a large weight for it. Otherwise, a small weight is defined for it. There are three advantages in using weighted sum approach to deal with the multi-objective optimization. Firstly, it is easy for decision makers to understand the weighted sum method. Second, developers can implement it conveniently. Thirdly, the weight of different objectives can be modified in order to satisfy the requirements of decision makers. In the present study, the importance of objectives F1(c), F2(c) and F3(c) are considered as 'the most important', 'more important' and 'important', so the weight of them is defined as 0.5, 0.3 and 0.2, respectively (Xing et al. [19]).

We develop the overlapping model suggested by Fattahi et al. [5] to formulate the FJSP with overlapping in operations. The operations to be performed on the jobs can, and in fact should, overlap. The extent of this overlapping is limited by a coefficient, $ov_{j,h}$, defined as the proportion of operation $O_{j,h}$ that has to be processed before its successor $O_{j,h+1}$ can start. This coefficient is 1 if no overlapping is allowed, and is very small, near to 0, if there is automatic overlapping in which as soon as the first unit of operation $O_{j,h}$, is processed, it becomes part of the next operation and is immediately processed. The overlapping, $ov_{j,h}$, of two operations determines the earliest starting time of the second operation, so that it is guaranteed that a sufficient proportion of $O_{j,h}$, is already processed and also that $O_{j,h+1}$ does not finish before the last units coming from $O_{j,h}$, have arrived.

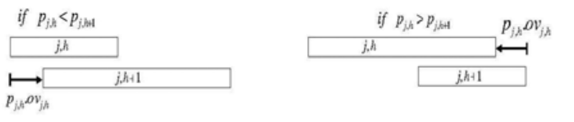

Fig. 1. Types of overlapping in operations

In the left hand part of Fig. 1, the first condition determines the earliest starting time of $O_{j,h+1}$ and in the right hand part of this Fig., the second condition determines the earliest finishing time of $O_{j,h+1}$.

Under these assumptions and notations, the problem is to determine both an assignment and a sequence of the operations on all machines that minimizes the makespan given n, m, $O_{j,h}$, $a_{i,j,h}$, $ov_{j,h}$ and $P_{i,j,h}$. Henceforth, this problem is referred to as flexible job shop scheduling problems (FJSP) with overlapping operations. The following additional notations are used in the mixed integer linear program formulation of FJSP with overlapping operations (Fattahi et al. [5]).

$$a_{i,j,h}\begin{cases}1 & \text{if } O_{j,h} \text{ can be performed on machine } i\\ 0 & \text{otherwise}\end{cases}$$

$$y_{i,j,h}\begin{cases}1 & \text{if machine i selected for operation} O_{j,h}\\ 0 & \text{otherwise}\end{cases}$$

$$r_{j,h}\begin{cases}1 & if \quad P_{j,h} \le P_{j,h+1}\\ 0 & if \quad P_{j,h} > P_{j,h+1}\end{cases}$$

L: A large number ($\propto$).

$$x_{i,j,h,k}\begin{cases}1 & \text{if } O_{j,h} \text{ can be performed on machine i in priority k}\\ 0 & \text{otherwise}\end{cases}$$

Min: F(c) =0.5* F1(c) +0.3* F2(c) + 0.2* F3(c)
F1(c) =Makespan time
F2(c) =critical machine work loading time
F3(c) =Total work loading time of all machines
$C_{\max}$: Makespan time
WL_i : Work loading time for machine i.
TWL : Total work loading time for all machines
$t_{j,h}$: Starting time of the processing of operation $O_{j,h}$.
$TM_{i,k}$: Start of working time for machine i in priority k.
k_i : The number of assigned operations to machine i.
$PS_{j,h}$: Processing time of operation $O_{j,h}$ after selecting a machine.

A mixed integer linier program for the FJSP with overlapping operations is as follows:

Min: F(c) =0.5* F1(c) +0.3* F2(c) + 0.2* F3(c)
F1(c) = $C_{\max}$
F2(c) = max (WL_i) i=1,…,m
F3(c) = $TWL = \sum_i WL_i$

s.t.

$$C_{\max} \ge t_{j,k_j} + ps_{j,k_j} \quad \text{for } j=1,...,n \tag{1}$$

$$WL_i = \sum_k x_{i,j,h,k}.ps_{j,h} \quad \text{for } i=1,...,m \tag{2}$$

$$\sum_i y_{i,j,h}.p_{i,j,h} = ps_{j,h} \quad \text{for } j=1,...,n;\ h=1,...,h_j \tag{3}$$

$$t_{j,h} + ps_{j,h}.ov_{j,h} \le t_{j,h+1}(1-r_{j,h})L \quad \text{for } j=1,...,n;\ h=1,...,h_{j-1} \tag{4}$$

$$t_{j,h} + ps_{j,h}.ov_{j,h} \le t_{j,h+1} + ps_{j,h+1} - ps_{j,h}.ov_{j,h} + r_{j,h}.L \quad \text{for } j=1,...,n;\ h=1,...,h_{j-1} \tag{5}$$

$$Tm_{i,k} + ps_{j,h}.x_{i,j,h,k} \le Tm_{i,k+1} \quad \text{for } i=1,...,m;\ j=1,...,n;\ h=1,...,h_j;\ k=1,...,k_i-1 \tag{6}$$

$$Tm_{i,k} \le t_{j,h} + (1 - x_{i,j,h,k}).L \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j\ ;\ k=1,\dots,\ k_i \tag{7}$$

$$Tm_{i,k} + (1 - x_{i,j,h,k}).L \ge t_{j,h} \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j\ ;\ k=1,\dots,\ k_i \tag{8}$$

$$y_{i,j,h} \le a_{i,j,h} \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j \tag{9}$$

$$\sum_j \sum_h x_{i,j,h,k} = 1 \quad \text{for } i=1,\dots,m;\ k=1,\dots,\ k_i \tag{10}$$

$$\sum_i y_{i,j,h} = 1 \quad \text{for } j=1,\dots,n;\ h=1,\dots,h_j \tag{11}$$

$$\sum_k x_{i,j,h,k} = y_{i,j,h} \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j \tag{12}$$

$$t_{j,h} \ge 0 \quad \text{for } j=1,\dots,n;\ h=1,\dots,h_j \tag{13}$$

$$x_{i,j,h,k} \in \{0,1\} \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j\ ;\ k=1,\dots,\ k_i \tag{14}$$

$$y_{i,j,h} \in \{0,1\} \quad \text{for } i=1,\dots,m;\ j=1,\dots,n;\ h=1,\dots,h_j \tag{15}$$

Constraint (1) determines the makespan, that is, the first objective. Constraint (2) determines work loading times of all machines, so we can search for a critical machine with maximum work loading time to determine second and third objectives. Constraint (3) determines the processing time of operation $O_{j,h}$ in the selected machine. Constraints (4) and (5) enforce each job to follow a specified operation sequence and consider the overlapping constraints. Constraint (6) makes it compulsory for each machine to process one operation at a time. Constraints (7) and (8) guarantee that each operation $O_{j,h}$ can start when its assigned machine is idle and the previous operation $O_{j,h-1}$ is completed. Constraint (9) determines the alternative machines for each operation. Constraint (10) assigns the operations to a machine and sequences assigned operations on all machines. Constraints (11) and (12) assure that each operation can be performed only on one machine and at one priority. Results of $x_{i,j,h,k}$ yield the assignment of each operation on a machine and sequence assigned operations on all machines.

3. Simulated annealing algorithm

We want to use simulated annealing algorithm (SA algorithm) for solving problems. SA algorithm is inspired metal cooling process. In this process, with reducing temperature gradually, we aim to get closer to optimal solution. SA algorithm searches current solution neighborhoods for a better solution and uses it for many complementary problems. Some researchers like Fattahi et al. [5][6] and Zandieh et al. [20] use SA algorithm in flexible job shop environment. SA algorithm generates an initial solution randomly and then neighborhood of this solution and afterward compares objective functions of them with each other: if new solution is better than current solution, it substitutes current solution with new solution and then generates another neighborhood. To guarantee algorithm diversification, SA algorithm accepts worse

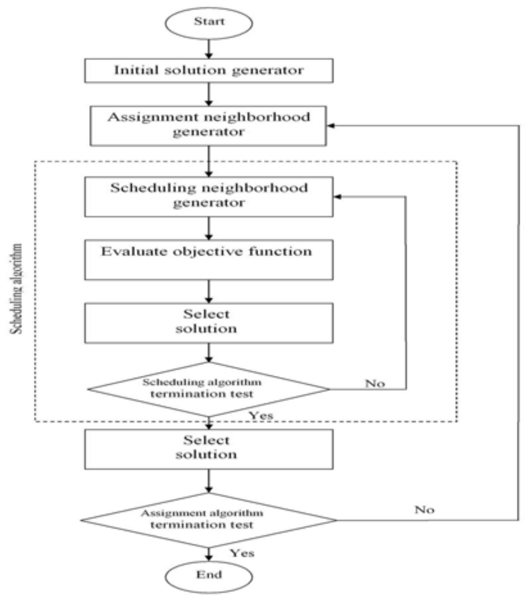
Start
Initial solution generator
Assignment neighborhood generator
Scheduling neighborhood generator
Evaluate objective function
Select solution
Scheduling algorithm termination test
No
Yes
Scheduling algorithm
Select solution
Assignment algorithm termination test
No
Yes
End

Fig. 2. Solution approach flowchart

solution with probability p=$\exp^{(-\Delta/T_i)}$ where Δ is different and then generates another neighborhood. To guarantee algorithm diversification, SA algorithm accepts worse solution with probability p=$\exp^{(-\Delta/T_i)}$ where Δ is different between current solution and new solution objective functions; T_i is iTh algorithm stage temperature that is obtained from cooling schema presented in equation 16. That denotes T as initial temperature and T_0 as frozen temperature as SA algorithm moves from T to T_0 in passing time. Finally, stop condition is met when the algorithm temperature becomes T_0.

$$T_i = T - i.\frac{T - T_0}{N} \tag{16}$$

In equation 16, N denotes the maximum number of iterations of SA algorithm. The initial temperature of SA algorithm is high enough that the first answer acceptance probability is larger than 80 percent.

4. The Proposed algorithm

In this paper, we use the decomposition approach that yields two algorithms, namely, assignment and scheduling algorithms in separate stages. Fig. 2 depicts the flowchart of the solving approach, and both algorithms are described in the next section.

4.1. assignment algorithm

As mentioned earlier, we use the decomposition approach for objective function optimization. At its first stage which includes assigning operations to machines, assignment algorithm is used. Using scheduling algorithm, we want to establish an appropriate assignment algorithm solution and after scheduling algorithm process, we select the best solution.

We develop Fattahi et al. [5] solution seed and neighborhood generator structure, and in neighborhood structure, we select one operation randomly and change the assignment SA algorithm pseudo-code which is illustrated in Fig. 3 where N_{Assign} denotes number of iterations in each SA algorithm temperature, S is current solution that is generated randomly for initial solutions; Sc is S neighborhood and S* is the best solution obtained.

```
Parameters setting
input temperature T, final temperature $T_0$ , number of iteration
$N_{Assign}$ , and L
generate initial solution randomly = S
evaluate S = F
n=1, $n_l$=1
while $T_i$ < T
   while n < $N_{Assign}$
      while $n_l$<L
      select a neighborhood $S^c$ of S
      evaluate $S^c$ with scheduling algorithm = $F^c$
      compute $\Delta = F^c - F$
      if $\Delta <= 0$
            then S = $S^c$ ; F = $F^c$
      else
            generate a random variable $P \sim (0,1)$
            if $e^{(-\Delta/T_i)} > P$
                  then S = $S^c$ ; F = $F^c$
            end if
      end if
      $n_l$=$n_l$+1
      end while
      n = n + 1
   end while
   update $T_i$
end while
if F < $F^*$
   $F^*$ = F ; $S^*$ = S
end if
```

Fig. 3. Assignment algorithm pseudo-code

4.2. scheduling algorithm

As mentioned earlier, after assignment algorithm, we use the scheduling algorithm to schedule operations on machines. The generated Sc is the input to the scheduling algorithm, algorithm solutions evaluated by total objective function that provide three different objectives:

Min: F(c) =0.5* F1(c) +0.3* F2(c) + 0.2* F3(c)

Scheduling algorithm initial solution is output of the assignment algorithm. In this stage, we use the solution seed by Fattahi et al. [5] again. For neighborhood generator, select one machine randomly and change priority between operations and then evaluate objective function. The scheduling algorithm pseudo-code is presented in Fig. 4 where $N_{Schedul}$ denotes number of iterations in each SA algorithm, temperature for algorithm better performance, its parameters had to set hence in the next section, we treat algorithm. Overlappings in operations could be classified in three different classes: first class, second class, and third class. In the first class, all operations in one job have the same overlapping coefficient that is represented by $ov_{j,h}$. $ov_{j,h}$ is between zero and one. Thus, if overlapping coefficient for a job is

```
parameters setting
input temperature T, final temperature $T_0$ , number of iteration
$N_{Schedul}$ , and L
input solution from assignment algorithm = S
evaluate S = F
n=1, $n_l$=1
while $T_f$< T
   while n < $N_{Schedul}$
      while $n_l$<L
      select a neighborhood $S^c$ of S
      evaluate $S^c$ with function
      $(0.5* F_1(c) +0.3* F_2(c) + 0.2* F_3(c))$ = $F^c$
      compute $\Delta = F^c - F$
      if $\Delta <= 0$
            then S = $S^c$ ; F = $F^c$
      else
            generate a random variable $P \sim (0,1)$
            if $e^{(-\Delta/T_i)} > P$
                  then S = $S^c$ ; F = $F^c$
            end if
      end if
      $n_l$=$n_l$+1
      end while
      n = n + 1
   end while
   update T
end while
if F < $F^*$
   $F^*$ = F ; $S^*$ = S
end if
```

Fig. 4. Scheduling algorithm pseudo-code

0.1 and the operation process time is 20 units after passing (20*0.1=2) unit could start next operation of that job on an idle machine. If overlapping in operations is to be overlooked, overlapping coefficient must be set equal to one for all jobs. The second class of overlapping considers different overlapping coefficients for different jobs, thus each job consists of one overlapping coefficient and 1*N matrix is made. The third class of overlapping considers different overlapping coefficients for each operation so it made a matrix. In Fig. 10, we can see an example of this matrix.

5. Tuned proposed algorithm

To improve performance of the proposed algorithm, we needed to set algorithm parameters. Efficient algorithm parameters can be classified into two major classes: assignment algorithm parameters and scheduling algorithm parameters. In addition, they can be presented as follows: 1- maximum iteration defined by N 2- initial temperature of SA algorithm defined by T 3- frozen temperature defined by T_0 4- number of iterations in each algorithm temperature defined by L .

Parameters tuned with the use experimental design for maximum neighborhood in both algorithms assignment and scheduling. Experiments result in changes affiliated to consideration problem size. To examine the proposed algorithm, 24 problems are defined and then solved by the mathematical model presented in section 2 and branch and bound approach. One-hour CPU time restriction is set for solving problems. If problems are solved in this time restriction, they are classified as small size; if this time restriction is just lower or upper bound distinguish, problems are classified as medium size and if in the identified time, the branch and bound approach cannot enter a feasible space, problems are classified as large size. For this classification, 9, 9, and 6 random problems were generated, respectively. For parameter setting, a problem from each class was selected randomly: number 6 for small size class, problem number 14 for medium size and problem number 19 for large size class.

If objective function does not reduce more than $\varepsilon = 0.01$ in 100 algorithm iterations, algorithm will stop to decrease running time risk.

To apply experiments and algorithm running, other parameters are set empirically and the experiment level is indicated in Table 1. Table 2 shows a designed ANOVA test for three pattern problems.

Table 1

DOE, assignment and scheduling algorithms parameter

Problem size	Problem number	T_0	T_f	L	Level of assignment algorithm neighborhood	Level of scheduling algorithm neighborhood
Small	6	100	0.1	10	40 , 50 , 60 , 70	20 , 30 , 40 , 50
Medium	14	150	0.1	15	60 , 75 , 90 , 105	30 , 45, 60 , 75
Large	19	200	0.1	20	80 , 100 , 120 , 140	40 , 60 , 80 , 100

Table 2
Experimental design ANOVA with two factors

		ss	df	ms	f	P-Value
Small size problem	A	174.845	3	58.28167	0.666667	0.5847
	B	524.535	3	174.845	2	0.1546
	interaction	524.535	9	58.28167	0.666667	0.7271
	Error	1398.76	16	87.4225		
	Total	2622.675	31			
Medium size problem	A	6772.44	3	2257.48	1.134648	0.3649
	B	10694.3	3	3564.766	1.791711	0.1893
	interaction	20816.85	9	2312.984	1.162545	0.3792
	Error	31833.39	16	1989.587		
	Total	70116.99	31			
Large size problem	A	11021.38	3	3673.793	4.988057	0.0125
	B	28686.85	3	9562.282	12.9831	0.0001
	interaction	2037.473	9	226.3859	0.307373	0.9612
	Error	11784.28	16	736.5178		
	Total	53529.98	31			

Table 3
Proposed algorithm parameters setting

Number of scheduling algorithm neighborhood	Number of assignment algorithm neighborhood	L	T_f	T_0	Problem number	Problem size
20	40	10	0.1	100	6	Small
30	60	15	0.1	150	14	Medium
60	100	20	0.1	200	19	Large

Considering the first ANOVA experiment, there is no significant difference between small size problems with the defined level in Table 1, therefore reducing problems running time uses a lower level of neighborhoods that is 40 neighborhoods for assignment algorithm and 20 neighborhoods for scheduling algorithm. For medium size problems, its ANOVA indicates that there is no significant difference between neighborhoods levels of assignment and scheduling algorithms, and both algorithms use lower level of neighborhood. The numbers of neighborhoods in medium size problems for assignment and scheduling algorithms are 60 and 30 respectively. Finally, the ANOVA experiment for large size problems shows that numbers of neighborhoods in assignment and scheduling algorithms have significant differences.

By using multi comparative experiments, it is concluded that the best number of neighborhoods in assignment algorithm is 100 and in scheduling algorithm is 60. The proposed algorithm parameters for solving problems are presented in Table 3.

6. Experimental results

As maintained in section 4, we randomly generated three classified problems with the mathematical model solving by branch and bound method solving and time observation. Using lingo software, we solved all problems by the mathematical model and branch and bound method. The result of solving the small size problems with the branch and bound method is convergent to optimal solution less than one hour, and for medium size problems, just lower and upper bounds are obtained by the branch and bound method and they cannot be convergent to optimal solution. The results of solving the third class of problems with the branch and bound method is not convergent to feasible solution space, therefore the third class of problems are solved with the mathematical model and B & B method and the results are shown in Table 4 (see Appendix).

We programmed our proposed algorithm with Matlab software and solved it with 2.4 GHz, Pentium 4, 512MB Ram PC. The results of the proposed algorithm presented in Table 4 can be compared with the results of branch and bound method.

Flexibility coleus denotes the mean number of assigned machines to each operation. We use (cv^2) index for the rate of diffusion to determine algorithm convergence represented in equation 17. (cv^2) index results are also shown in Table 4. If (cv^2) index is in low level, the algorithm has a good convergence.

$$(cv)^2 = \frac{\frac{\sum_{i=1}^{n}(f_i - \bar{f})^2}{n-1}}{(\bar{f})^2} \qquad (17)$$

In the above formula, $\bar{f}$ is the mean of objective function in algorithm repetition, N denotes number of algorithm repetition which is considered 10 in this paper, and f_i is objective function in i th algorithm run time.

Table 4

Proposed algorithm and the results of the mathematical model solved with Branch and Bound method

row	size	B&B									proposed Algorithm with overlapping					proposed Algorithm with out				flexibility	(cv)^2
		f1		f2		f3		f(c)		CPU time	f1	f2	f3	f(c)	CPU time	f1	f2	f3	f(c)		
Small size problem																					
1	2.2.2	107		107		185		122.6		1	107	107	185	122.6	17.67	107	107	185	122.6	1.5	0
2	2.2.2	66		66		127		78.2		2	66	66	127	78.2	17.81	66	66	127	78.2	2	0
3	3.2.2	221		221		432		263.2		15	221	221	432	263.2	18.6	221	221	432	263.2	1.67	0
4	3.2.2	355		355		690		422		14	355	355	690	422	18.35	355	355	690	422	1.83	0
5	3.2.2	119		119		213		153.3		187	119	119	213	153.3	18.83	119	119	213	153.3	2	0
6	3.3.3	256		250		687		340		1590	256	250	687	340	20.55	320	320	687	393.4	1.67	0
7	3.3.5	235.5		220		912		365.15		2954	235.5	220	912	365.15	20.15	397	220	912	446.9	2	0
8	3.3.4	204.7		166		584		268.95		2216	204.7	166	584	268.95	19.39	283	166	584	308.1	2	0
9	3.3.3	171.7		170		457		228.25		1980	171.7	170	457	228.25	22.87	220	220	497	257.4	2	0
Medium size problem																					
row	size	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	CPU time	f1	f2	f3	f(c)	CPU time	f1	f2	f3	f(c)	flexibility	(cv)^2
10	3.3.2	227	241.7	227	240	377	448.7	257	282.59	<3600	241.7	240	448.7	282.45	33.98	250	250	455	291	2	0
11	4.3.5	329.3	419.5	234	403	902	1688	415.25	668.25	<3600	419.5	403	1688	668.25	35.19	516	466	1537	705.2	1.75	0
12	4.3.5	324.7	450	318	440	1204	1444	498.55	645.8	<3600	417	417	1539	641.4	36.38	466	466	1459	664.6	1.92	0
13	4.3.5	336.5	416	330	416	1002	1553	467.65	643.4	<3600	355	355	1453	574.6	38.72	382	382	1396	582.8	2.2	0.000731
14	5.3.6	328.7	644	320	330	1207	1700	501.75	761	<3600	474.7	376	1717	693.55	40.94	677	446	1677	807.7	2.2	0.000613
15	5.3.7	443.7	644	302	320	1264	1608	565.25	739.6	<3600	494.6	374	1742	707.9	42.16	644	355	1747	777.9	2.6	0.000566
16	6.3.7	294	-	294	-	1087	-	452.6	-	<3600	445.8	395	2258	783	44.51	589	504	2282	902.1	2.67	0.000644
17	6.3.7	393	-	393	-	1249	-	564.2	-	<3600	559.5	430	2199	848.55	46.63	598	398	2244	867.2	2.83	0.006342
18	7.3.7	346	-	346	-	1509	-	578.6	-	<3600	547.8	479	2676	952.8	49.1	662	474	2715	1016	2.61	0.003033
Large size problem																					
19	7.3.7	-	-	-	-	-	-	-	-	<3600	580.5	503	2811	1003.4	68.87	750	542	2717	1081	2.86	0.001989
20	8.3.7	-	-	-	-	-	-	-	-	<3600	631.7	578	3400	1169.3	71.05	865	625	3222	1264.2	2.58	0.002479
21	8.4.7	-	-	-	-	-	-	-	-	<3600	987.8	845	4595	1666.4	79.9	1311	855	4519	1815.8	2.44	0.003293
22	9.4.8	-	-	-	-	-	-	-	-	<3600	1081.6	737	5126	1787.1	83.76	1299	819	4857	1866.6	2.39	0.002309
23	11.4.8	-	-	-	-	-	-	-	-	<3600	1354.2	1057	6490	2292.2	85.29	1579	1104	6516	2423.9	2.34	0.001902
24	12.4.8	-	-	-	-	-	-	-	-	<3600	1494	1009	7353	2520.3	89.81	1723	1077	7585	2701.6	2.33	0.003965

For considering overlapping assumption in the three defined classes, we use problem 16 here. As described in section 4-2, in the first class of overlapping, all overlapping coefficients are the same, that is, in this paper we use 0.1 for all numerical examples. Problem 16 is defined in Table 5, and the effect of 0.1 overlapping coefficient is depicted in Fig. 5. The second class of overlapping is defined using [0.2 0.1 0.3 0.2 0.5 0.2] matrix that from left to right represents overlapping coefficients of job number 1 to job number 6. In Fig. 6, you can see the effect of this matrix of overlapping coefficients on operations in optimal solution.

Table 5
Problem 16 operations process time

Job No.	operation	Machine						
		1	2	3	4	5	6	7
1	$O_{1,1}$	147	123	145				
	$O_{1,2}$	123	130		140			
	$O_{1,3}$				140	160	200	
2	$O_{2,1}$	214		150				
	$O_{2,2}$		66	87	99			
	$O_{2,3}$					178	95	150
3	$O_{3,1}$	87	62					
	$O_{3,2}$			180	105			145
	$O_{3,3}$				190	100	153	
4	$O_{4,1}$	87	65					
	$O_{4,2}$			250		173		
	$O_{4,3}$				145		136	
5	$O_{5,1}$	128	123	145				
	$O_{5,2}$			86	65	47		
	$O_{5,3}$					110	85	
6	$O_{6,1}$		145	320	154			
	$O_{6,2}$			123	150	192		
	$O_{6,3}$					120	240	180

The third kind of overlapping considered here is the overlapping coefficient for each operation. Its matrix is shown in Fig. 7, and the overlapping coefficient matrix solution of problem 16 is illustrated in Fig. 8.

In order to show the effect of overlapping assumption, we solve the problem with two approaches: firstly, with overlapping in operations (with 0.1 overlapping coefficient) and then without overlapping in operations.

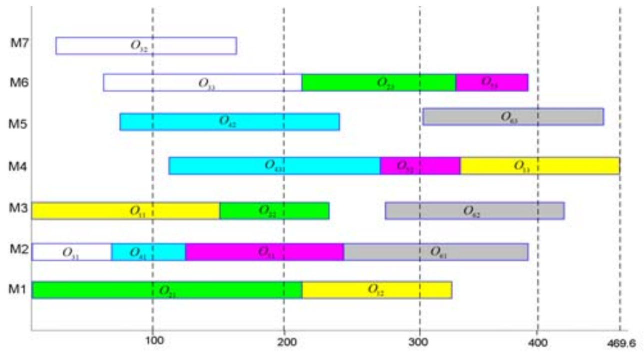

Fig. 5. Problem 16 with overlapping coefficient 0.1 Gantt chart

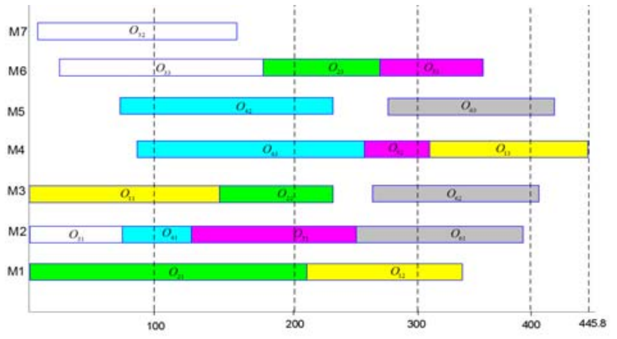

Fig. 6. Problem 16 with overlapping coefficient matrix [0.2 0.1 0.3 0.2 0.5 0.2

$$\begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.2 \\ 0.1 & 0.3 \\ 0.2 & 0.5 \\ 0.4 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}$$

Fig. 7. Overlapping coefficient for each operation in jobs

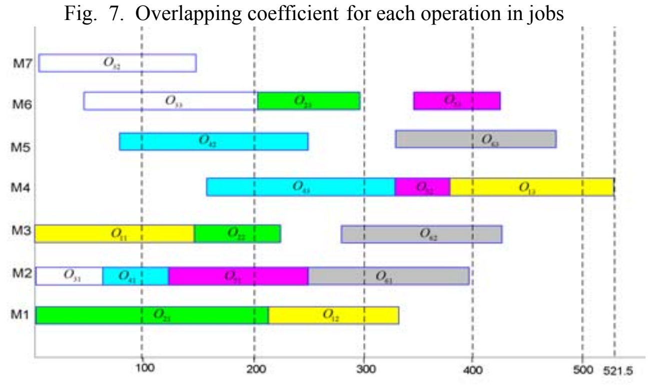

Fig. 8. Problem 16 with overlapping coefficient matrix in Fig 7 Gantt chart

In the first approach, with considering overlapping in operations, a better objective function is observed. The results of overlapping in problem 16 are shown in Fig. 5 and the results of problem 16 without overlapping are shown in Fig. 9. The results indicate that overlapping in operation reduces makespan and consequently reduces total function.

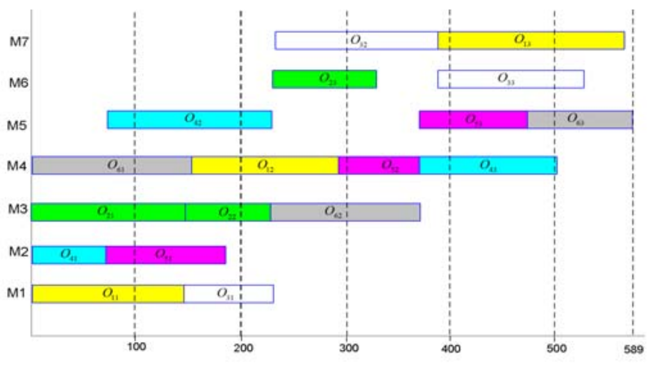

Fig. 9. Problem 16 without overlapping coefficient Gantt chart

When solving a scheduling problem with an overlapping one, a fundamental mistake may take place if one operation time is shorter than previous operation time. In this situation, maybe an operation ends before the previous operation. If in scheduling, we can use overlapping assumption (i.e. when one operation processes, there is an idle machine for next operation), to Fig. out the mistake just mentioned, we consider second operation process time equal to previous operation process time. The solution of problem 16 is in Fig. 5. In this Fig., the second operation of job number 6 after passing 14.5 time unit from first operation process is prepared to process on machine 3 (with 0.1 overlapping coefficient) but because second operation process time is shorter than first operation time, we consider the first operation process time for the second operation. This is also the same with the third operation and we consider the first operation process time for the third operation too. In this case, the initial operation process time

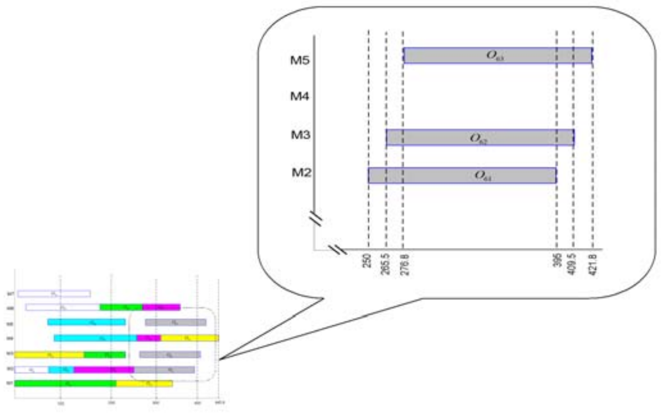

Fig. 10. Modified operation process time with overlapping in operation assumption

is considered for determining objective function. In Fig. 10, these mistake solving approaches are shown. As the manufacturing cost does not depend on the time of job completion but on machine working time, the Makespan is not an appropriate criterion, so we consider two other criteria: critical machine work loading time and all machine work loading time which are defined as objective functions. We thought that if Makespan is minimum, the two other objectives are minimized too, but it is not true. We solved problem 16 two times, first with one objective function (i.e. Makespan), and then with three objectives. As the results presented in Table 6 indicate when the Makespan is the objective function, total function increases, and if the cost depends on machine working time, the shop cost will increase too. Regarding the three objectives, the total function is lower than the first case and if the cost depends on machine working time, the shop cost will decrease. Fig. 11 shows the Gant chart of the Makespan objective function resulted from solving problem 16, and Fig. 5 shows the Gant chart of the three objective functions.

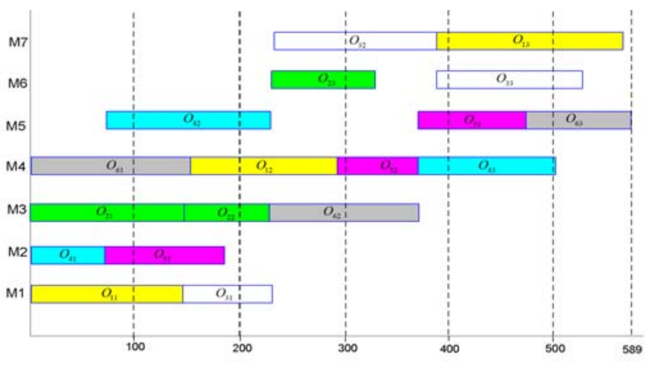

Fig. 11. Gant chart of problem 16 solution with Makespan objective function

Table 6

Multi-objective function in flexible job shop scheduling problems

Makespan	Critical machine work loading time	Total work loading time	Total function	Fig No.
426.3	395	2268	785.25	11
445.8	395	2208	783	5

Overlapping in operations assumption reduces makespan and in turn increases machines utilization. Problems due to both overlapping in operations assumption and lack of overlapping are compared in Fig. 12. Furthermore, best makespans for both of these approaches are shown. The Fig. indicates that adding overlapping to operations assumption results in improvement in both objective function and Makespan.

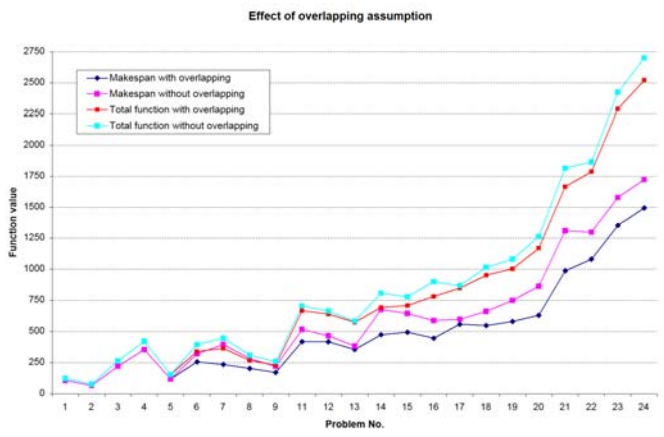

Fig. 11. Gant chart of problem 16 solution with Makespan objective function

7. Conclusion and Future Research

In this paper, multi-objective flexible job shop scheduling with overlapping in operations is described. Overlapping in operations assumption is commonplace in many industries like chemical, petrochemical and glass factories. Here, multi-objective flexible job shop scheduling is formulated with an introduced mixed integer linear programming (MILP), and using lingo software, the mathematical model is solved with the branch and bound method. As flexible job shop problem scheduling is NP-Hard and overlapping in operations increases its complexity, multi-objective flexible job shop problem scheduling with overlapping in operations is strongly NP-Hard. While these problems are NP-Hard, we can use heuristic and Meta heuristic algorithms for solving them. Thus, we introduced simulated annealing Meta heuristic algorithm. Besides, we introduced decomposition approach in order to reduce the complexity of solving problem in flexible job shop scheduling. In the first stage, we identified the assignment algorithm using SA algorithm. Assignment algorithm output was the initial solution of the scheduling algorithm. The scheduling algorithm was based on SA algorithm too. The proposed algorithm was programmed by Matlab software. To evaluate performance of the proposed algorithm, we used its comprehensive results and the mathematical model. The numerical results showed pattern problems were solved in a short time. Further, comparing performance of the proposed algorithm with the MILP model indicated that its result is very good. If the job shop cost relates to machine working time, just Makespan as the objective function cannot be a good evaluation criterion. Therefore, we defined a multi- objective function consisting of Makespan, critical machine work loading time, and total machine work loading time. To improve performance of the proposed algorithm, we used an experimental design with two factors. Finally, we introduced different overlappings in operations.

To improve the results further, this study could use another heuristic and Meta heuristic algorithm for solving the problem. To be more like the real world events, dynamic scheduling with minimized total tardy objective or previous maintenance or random breakdown in machines could be considered in the study.

References

[1] R. Alvarez-Valdes, A. Fuertes, J.M. Tamarit, G. Gimenez, and R.S. Ramos, A heuristic to schedule flexible job shop in a glass factory. European Journal of Operational Research, 165, 525-534, 2005.

[2] P. Brandimarte, Routing and scheduling in a flexible job shop by taboo search. Annals of Operations Research, 41, 157-183, 1993.

[3] S. Dauzere-Peres, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem using tabu search. Annals Operations Research, 70, 281-306, 1997.

[4] P. Fattahi, A hybrid Multi objective algorithm for flexible job shop scheduling. Proceedings of World Academy of Science Engineering and Technology val. 38 issn, 2070-3740, 2009.

[5] P. Fattahi, F. Jolai, and J. Arkat, Flexible job shop scheduling with overlapping in operations. Appl. Math. Modelling, doi: 10.1016/j.apm.2008.10.029, 2008.

[6] P. Fattahi, M. Saidi, and F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Intelligent Manufacturing, 18, 331–342, 2007.

[7] J. Gao, L. Sun, and M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research, 35(9), 2892-2907, DOI: 10.1016/j.cor.2007.01.001, 2008.

[8] M. R. Garey, D. S. Johnson, and R. Sethi, The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 1, 117-129, 1976.

[9] M. Gholami, M. Zandieh, Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. Journal Intell Manuf, DOI 10.1007/s 10845-008-0150-0, 2008.

[10] E. Hurink, B. Jurisch, and M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machines. Operations Research Spektrum, 15, 205-215, 1994.

[11] I. Kacem, S. Hammadi, and P. Borne, a. Approach by localization and multiobjective evolutionary for flexible job shop scheduling

problems. IEEE transactions on systems, man and cybernetic part C, 32(1), 1-13, 2002.

[12] I. Kacem, S. Hammadi, and P. Borne, b. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Mathematics and Computers in Simulation, 60, 245–276, 2002.

[13] M. Mastrololli, L. M. Gambardella, Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling, 3(1), 3-20, 2002.

[14] J. Paulli, A hierarchical approach for the FMS scheduling problem. Eur. J. Operatres, 89, 32-42, 1995.

[15] F. Pezzella, G. Morganti, and G.Ciaschetti, A genetic algorithm for the flexible job shop scheduling problem. Computers & Operations Research, 35(10), 3202-3212, DOI: 10.1016/ j.cor.2007.02.014, 2008.

[16] C. Riago, Tardiness minimization in a flexible job shop: A tabu search approach. Journal of Intelligent Manufacturing, 15(1), 103-115, 2004.

[17] M. Saidi Mehrabad, P. Fattahi, Flexible job shop scheduling with tabu search algorithm. International Journal of Advanced Manufacturing Technology, 32, 563-570, 2007.

[18] W. Xia, Z. Wu, An effective hybrid optimization approach for multi-objective flexible jobshop scheduling problems. Computers & Industrial Engineering, 48, 409–425, 2005.

[19] L. Xing, Y. Chen, and K. Yang, Multi-objective flexible job shop schedule: Design and evaluation by simulation modeling. Applied Soft Computing, 9, 362–376, 2009.

[20] M. Zandieh, I. Mahdavi, and A. Bagheri, Solving flexible job shop scheduling problems by a genetic algorithm. Journal of Applied Sciences, 8(24), 4650-4655, 2008.

[21] G. Zhang, X. Shao, P. Li, and L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Computers & Industrial Engineering, DOI:10.1016/j.cie.2008.07.021, 2008.