Solving Re-entrant No-wait Flexible Flowshop Scheduling Problem; Using the Bottleneck-based Heuristic and Genetic Algorithm

Sara Habibi[1]*, Shahin Ordikhani[2], Ahmad Reza Haghighi[3]
[1]School of Engineering, Urmia University, Oroumieh, West Azerbaijan Province, Oroumieh, Iran
[2]Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
[3]Technical and Vocational University, Tehran, Tehran Province, Iran
*Email of Corresponding Author: sara.habibi1987@yahoo.com
Received: December 25, 2018; Accepted: March 3, 2019

Abstract
In this paper, we study the re-entrant no-wait flexible flowshop scheduling problem with makespan minimization objective and then consider two parallel machines for each stage. The main characteristic of a re-entrant environment is that at least one job is likely to visit certain stages more than once during the process. The no-wait property describes a situation in which every job has its own processing sequence with the constraint that no waiting time is allowed among operations within any jobs. This study develops a bottleneck-based heuristic (BBFFL) to solve a flexible flowshop problem including a bottleneck stage. Also, a genetic algorithm (GA) based on heuristics for the problem is presented. First, the mathematical model for the problem is proposed, and then the suggested algorithms are explained. For small-scale, the results of the BBFFL and GA are compared to the results derived from the GAMS. For large-scale problems, the results of the GA and BBFFL are compared with each other. For small-scale problems, the algorithms have a close performance but the BBFFL is likely to generate much better in finding solutions in large-scale problems.

Keywords
Flexible Flowshop, No-wait, Re-entrant, Bottleneck, Genetic Algorithm

1. Introduction

Flow shop scheduling problems have a very extensive application in different industries and have attracted many researchers towards themselves. In a flow shop scheduling problem, there is a set of n jobs, tasks or parts which should be processed on a set of m machines or processors in a similar order. Also in the classic flow shop, it is assumed that there are infinite storages between steps and hence the unfinished jobs can be stopped between the neighbor machines. In many sorts of flow shops, jobs are processed without passing a step or a working station more than once. However, in some industries like semiconductors fabrication, the production design may be in a way that jobs in workshop revolve again on the machines from the beginning or pass some of the steps or working stations more than once. Generally, this kind of flow shop sorting is called the re-entrant flow shop in which at least one job should meet one or several stages more than once. In manufacturing industries, there are so many samples in re-entrant flow shops. For instance, photolithography is one of the most complex steps in the wafer fabrication process of semiconductor production.

Photolithography is an optical process used for mapping multiple layers of circuit patterns on silicon wafers and during this process; it can visit this stage more than once. Other examples are assembling and testing the electronic circuits which are put on each other. When ever a new circuit is added to the collection, it should pass again through some machinery. Some other examples in this regard can be printed circuit boards (PCB), dual machine cyclical workshops, signals processing and manufacturing scheduling for axle hub facilities

In no-wait flow shop problems, performing steps of a job on machines from the beginning to the end are carried out uninterruptedly which means when the processing of a job on the first machine begins, it should be moved on the machines continuously and uninterruptedly until its procedure is completed on the last machine. Uninterrupted limitations naturally are created based on the specifications of jobs or the processing environment. The examples consist of the processes related to extraction and melting metals in which the iron should be remained hot during the operations, chemical processes with unsustainable intermediate productions, the absence of a central warehouse, computer systems, foodstuffs processing, pharmaceutical industry or semiconductors test facilities. A perfect review of about modeling and integration of planning, scheduling and equipment configuration about semiconductors can be found in [1].

In the real world, the bottleneck phenomenon occurs repeatedly in the majority of manufacturing systems. Goldratt and Cox stated that the bottleneck sources would evaluate the overall performance. Bottleneck management is a critical task in workshops and is very useful in manufacturing scheduling. Scheduling approaches for flow shop problems and workshop manufacturing by considering bottleneck steps consist of three steps as follows:

1) Identifying the bottleneck step
2) Scheduling the bottleneck step
3) Scheduling the non-bottleneck step

Drum-Buffer-Rope scheduling method or DBR presented by Goldratt and Fox has been concluded from limitations theory. This method concentrates on limited resources scheduling (bottleneck resources) and normally treats the other resources (non-bottleneck resources). However, Canoy stated that usually, the exact scheduling of secondary resources for temporal support assurance was one of the most essential limited resources. Therefore, several researchers considered all of the resources and presented some methods for their solution. The bottleneck in a process occurs when the input reaches faster than the completion of the next step.

In this paper, no-wait re-entrant flow shop scheduling problem with considering back warding in each sequence for jobs is investigated, and for its solution, an algorithm based on bottleneck and meta-heuristic genetic algorithm is applied. Other parts of the paper are organized as follows:

In the second section, a literature review about the investigated subject in this paper is presented. In the third part of the article, the definition of the problem and its modeling are investigated. The fourth and fifth parts of this paper are related to solving the proposed problem using the proposed algorithms. The seventh part is related to the obtained results produced by running the proposed algorithms. Also, the seventh part examines the validity of the proposed algorithm in this paper and finally in the eighth part, conclusion and suggestions for future studies are presented.

2. Literature Review

Many papers in flow shop scheduling traits with considering the re-entrant feature of the environment by the no-wait system have been printed so far. However considering these two features simultaneously according to the broad application which is used in robotic industries, is not noticed by researchers. Robotic flow shops are used extensively in the steal and electronic industries in which according to the features of the technology itself, after that the processing is completed on a machine, it should be separated from the machine immediately and transformed uninterruptedly to the next machine in the process. Otherwise, defective items may be produced.

In the re-entrant system trait, Choi and Kim in took into account the minimization problem of maximum completion time of jobs and proposed several heuristic efficient algorithms for the problems [2]. Chen et al. also suggested a combined genetic algorithm for the same problem [3-4]. Investigated dual machine re-entrant flow shop scheduling problem with minimization of the maximum completion time of jobs objectives and developed several heuristic methods for solving the problem [5]. Chu et al. in presented a hill climbing algorithm as well as an adapted genetic algorithm for solving re-entrant flowshop scheduling problem with various resource considering qualification matching [6]. Authors in presented a heuristic algorithm for solving the re-entrant flow shop problem using the k-insertion technique for minimization of total flow [7]. Choi and Kim used a branch and bound and also three heuristic methods for solving a problem with the objective function of total delay [8]. Tasouji Hassanpour et al. considered the production environment of no-wait re-entrant flow shop with the objective of minimizing makespan of the jobs and developed a mathematical model and solved it using three algorithms including simulated annealing, genetic algorithm and a bottleneck based heuristic algorithms [9]. Adressi et al. in studied a group scheduling problem in no-wait flexible flowshop considering two stages with group sequence-dependent setup times and random breakdown of the machines [10]. Minimizing the makespan for a re-entrant hybrid flow shop scheduling problem with time window constraints was the focus of using a genetic algorithm hybridized ant colony optimization [11].

Regarding the no-wait trait, studied the group scheduling problem of jobs in a single stage no-wait flow shop environment in which setup times are sequence dependent using both genetic and simulated annealing algorithms [12]. Many pieces of research have been done on solving the no-wait flow shop scheduling problem, considering different criteria like maximum completion time of jobs and the time performed in the total flow which is led to present many heuristic and meta-heuristic algorithms. There are many investigations which were carried out in no-wait flow shop scheduling problem and continued until today such as the research of which studied the minimization of makespan in a two-machine no-wait flow shop problem with separable setup times and single server-side constraints [13]. A mathematical formulation presented for the problem and a hybrid algorithm are developed by using the variable neighborhood search and Tabu search. Could be pointed out as one of the recent jobs in this regard [14]. He studied dual machine no-wait flow shop scheduling problem, considering group setting time. Maya et al. studied the problem of minimizing the makespan of jobs in a no-wait flowshop environment, considering two batch processing machines. In this research, jobs with different sizes have been batched together, without surpassing the capacity of manufacturing machines. Another assumption of this research is that the start of a batch processing is subject to the availability of all the jobs existing in the batch. They

developed a greedy randomized adaptive search procedure (GRASP) algorithm to tackle the problem [15].

According to the performed research, the importance of research in no-wait re-entrant flow shop scheduling trait is observed. As one of the most recent works in no-wait scope, addressed the no-wait flow shop scheduling problem under makespan and flowtime criteria utilizing a hybrid meta-heuristic algorithm based on ant colony optimization and simulated annealing algorithm [16].

3. Definition of the Investigated Problem

In the flow shop scheduling problem (FSS), it is assumed that there is a set of jobs J={1,….,n} that should be processed on a set of machines M={1,….,m}. In this problem, there are m machines in series that each of the tasks is to be processed on the machines. All jobs have the same processing route, meaning that each job is processed first on machine 1, then on machine 2 and so on until the last machine finishes its work on the job. Another assumption considered for this problem is the limitation of the waiting time that this assumption can occur in multi machines environments like flowshop and jobshop. This assumption causes processing of jobs on the machines to be performed uninterruptedly and without any delays between machines. In addition to the above assumptions, there is an other assumption in which jobs can go backward several steps and perform their processing job. The problem considering these assumptions is called integrated re-entrant flow shop scheduling and no-wait time problem which is a kind of Hybrid Flow Shop Scheduling problem (HFSS).

This problem based on notation is shown as $HFSS|pmu, rcrc, no-wait|C_{max}$ [17]. The objective function considered in the investigated problem is the minimization of the maximum completion time of jobs on the machines.

In addition to the previous assumptions, the following assumptions are also considered for the problem:

Two operations of a job are not performable simultaneously. There is no interruption, which means a job remains on its related machine until completing its process. During the performance of jobs, there is no pre-emption, which means if the operation of a job is processed; the next operations of the job should also be processed. The processing time of each job is independent of the order of performing jobs. The preparing time is independent of the order of performing jobs and is considered in the jobs processing time. The transportation time between machines is negligible. Breakdown, maintenance time and costs are not considered in the model. Machines may be inactive for some time. Each device does not process more than one job simultaneously. Technical limitations are known and invariable. There is no random mode, which means processing times, setup times, entry parts times and the number of jobs has definite values. Machines are available continuously during the planning horizon.

4. Problem Modeling

The symbols used for modeling of the problem are introduced as follows:

Indexes:

i job index (i=1,...,n)

j operation index $j = \{1, 2, ..., n_i\}$

k machine index (k=1,...,m)

l machine index in job stations $l = \{1,2,\dots,l\}$

h order of jobs on each machine $h = \{1,2,\dots,h_k\}$

Parameters:

p_{ij} processing time of j^{th} operation of the job i

a_{jk} 1 if the j^{th} operation of a job is performed on machine k, 0 otherwise

Re_j 1 if an operation includes re-entrant condition after the j^{th} operation, 0 otherwise

$S\,\mathrm{Re}_j$ if an operation includes re-entrant condition after j^{th} operation that equals the machine index, 0 otherwise

M a very big number which can be considered as a sum of processing times of all the operations

Variables:

$C_{\max}$ maximum completion time of jobs

s_{ij} starting time of the j^{th} operation of an i^{th} job

pb_{kh} processing time of the job positioned in an h^{th} order of ak^{th} machine

sb_{klh} starting time for processing the job positioned in the h^{th} order of the k^{th} machine in an l^{th} station

r_{ijk}^{h} 1 if the j^{th} operation of the i^{th} job which needs to be operated on ak^{th} workstation is positioned in the h^{th} order of an l^{th} machine, 0 otherwise

$$\min imize\, C_{\max} \tag{1}$$

$$\sum_{h} r_{ijk}^{lh} = a_{jk;\,\forall i,j,k} \tag{2}$$

$$\sum_{i}\sum_{j} r_{ijk}^{lh} \le 1;\ \forall k,h,l \tag{3}$$

$$\sum_{l} r_{i,j-1,k-1}^{lh} = \sum_{l} r_{ijk}^{lh};\ \forall i,j>1,k>1,h \quad \mathrm{Re}(j-1) <> 1 \tag{4}$$

$$\sum_{l} r_{i,j-1,k-1}^{lh} = \sum_{l} r_{i,j,k};\ \forall i,j,k>1,k',h' \ \mathrm{Re}(j-1)=1, S\,\mathrm{Re}(j)=k' \tag{5}$$

$$sb_{k,l,h-1} + \sum_{k}\sum_{l}\sum_{h} p_{i,j} * r_{i,j,k}^{lh-1} \le sb_{k,l,h};\ \forall i,j>1 \tag{6}$$

$$s_{i,j-1} + \sum_{k}\sum_{l}\sum_{h} p_{i,j-1} * r_{i,j-1,k}^{lh} = s_{i,j;}\ \forall i,j>1 \tag{7}$$

$$s_{ij} \le (1 - r_{ijk}^{lh}) * M + sb_{klh};\ \forall i,j,k,l,h \tag{8}$$

$$sb_{klh} \le (1 - r_{ijk}^{lh}) * M + s_{ij};\ \forall i,j,k,l,h \tag{9}$$

$$C_{\max} \ge s_{ij} + \sum_{k}\sum_{l}\sum_{h} p_{ij} * r_{ijk}^{lh};\ \forall i,j \tag{10}$$

$$r_{ijk}^{h} = \{0,1\}, s_{ij} \ge 0, sb_{kh} \ge 0, sb_{kh} \ge 0, \forall i,j,k,h \tag{11}$$

Objective function consists of minimizing the maximum completion time of the jobs. Equation (2) denotes that when the operation of one job is assigned to any particular machine, this operation can be positioned in any order of the machine. Constraints (4) and (5) are added to the problem to comply with the assumption of the permutation flow shop problem. Constraint set (6) is attached to the model to set the starting time of jobs on each machine in stations. This constraint states that as long as the process of the previous job is not completed, the current work process cannot be started. Constraint set (7) adjusts the starting time of operations which are positioned in the processing route, in other words, it makes sure that successive operations of any machine are performed after the preceding ones. Constraints (8) and (9) have been added to the model to adjust the starting time of any operations of each job and starting time of jobs on machines. Constraint (10) calculates the maximum completion time of the jobs. Finally, the constraint set (11) determines the nature of the model variables.

5. Solving No-wait Re-entrant Flow Shop Scheduling Problem

Authors in [3] showed that the re-entrant flow shop scheduling is even for problems with two machines; Np-hard problem if minimization objective function of the maximum time of completing jobs is considered. It can be shown that the addressed problem in this paper is also Np-hard. Thus, for solving the problem, in addition to the exact solution for small-scale problems, the heuristic algorithm based on bottleneck and the meta-heuristic genetic algorithm is proposed for solving the small-scale problems and large-scale problems as well.

5.1 The Proposed Bottle Neck-based Algorithm

The Bottleneck-Based Flexible Flow Line? (BBFFL) heuristic method is proposed for solving the problem in this paper. The proposed heuristic method belongs to the category of heuristic methods for producing a solution.

The main idea is in such a way that jobs scheduling in bottleneck step can have some effects on the performance of the heuristic method for scheduling of jobs in all the steps hence BBFFL produces the scheduling schedule based on the produced scheduling programs in the bottleneck station.

The heuristic method consists of three steps:

1) Identifying the bottleneck workstation: In this step first, the workload in each workstation is defined as the total average of processing times of all of the processed activities in that workstation which is divided to the number of machines in that step. For example, the workload in the workstation J is calculated by the following relation:

$$R_j = \left(\sum_{i=1}^{n} \bar{P}_{ij} \right) \Big/ m_j \qquad (12)$$

The workstation that has a bigger R_j is defined as the bottleneck station.

2) Generating an initial sequence of the jobs by a bottleneck-based initial sequence generator (BBISG)

In this step, the working stations are divided into three subsystems:

Up-stream sub-system is consists of the previous stations of the bottle neck station and the bottle neck sub-system which includes the bottle neck station and down-stream sub-system which comprises the working stations after the bottle neck station. BBISG produces a sequence of jobs

based on total processing times in up-stream sub-system and down-stream sub-system of jobs. The steps of the algorithm are as follows:

- Step 1: Set Ω to.
- Step 2: Divide the system into three up-stream, down-stream, and bottleneck system. Compute the total minimum processing time values for the up-stream sub-system (fp_i^{min}) and the down-stream sub-system (lp_i^{min}).
- Step 3: If $fp_i^{min} \leq lp_i^{min}$, allocate jobs to U, otherwise allocate them to L.
- Step 4: If U=∅, go to step 5. Select job with a minimum value for fp_i^{min} for i □ U. If there is more than one job for the minimum value of fp_i^{min}, select the job with maximum processing average time in the bottleneck step. If more than one job has this characteristic again, select randomly. Add the selected job to Ω set and omit from U. Do step 4 again.
- Step 5: If L=∅, go to step 6. Select job with a maximum value for lp_i^{min} for i □ L. If there is more than one job for the maximum value of lp_i^{min}, select the job with maximum average processing time in the bottleneck step. If more than one job has this characteristic again, select randomly. Add the selected job to Ω set and omit it from L set. Do step 5 again.
- Step 6: Obtain an initial sequence of jobs in Ω.
- Step 7: Stop.

In this algorithm, the parameters are defined as follows:

fp_i^{min}: Total minimum processing time required for job i before the bottleneck stage b

lp_i^{min}: Total minimum processing time required for job *i* after the bottleneck stage

1) Applying a bottleneck-based multiple insertion procedure (BBMIP) to the initial sequence to generate the final schedule. This step of the algorithm includes the following steps:

- Step 1: Select the first job in the initial sequence generated by BBISG and let it be the current partial sequence.
- Step 2: Select the next job in the initial sequence and insert the job into the positions before, between, and after every two consecutive jobs of the current partial sequence.
- Step 3: Calculate makespan for each partial sequence produced in Step 2 while adjusting jobs' entering sequence at the bottleneck stage to be the same as the one at the first stage.
- Step 4: Select the partial sequence with minimum makespan and let the partial sequence be the current partial sequence.
- Step 5: If the current partial schedule includes all the n jobs, then stop; otherwise go to Step2.

5.2 The Proposed GA Approach

The first step in this algorithm is linking the main problem with the genetic algorithm (GA) structure. Chromosome in this problem has two parts. The first part indicates the priority of processing of jobs on workstations and the second part suggests that jobs in each workstation are processed on the available machines in the station. In the following, each section will be explained.

- Part 1: For the investigated problem in this study, according to the mapping feature which exists for the problem, the chromosome which is used for showing the solution of the problem has a

length equal to the number of jobs of the problem and is independent of the number of workstations. In other words, the chromosome is shown by a sequence of jobs number as $\sigma = ([1], [2], \dots, [n])$where the number of each job is repeated only once in each chromosome. The priority of processing of each job on the stations is in the order of their appearance on the chromosome from left to right or in another word, the priority of jobs operation processing on each machine is based on their earliest occurrence in the sequence vector σ. A chromosome for a problem with six jobs is shown in Figure 1. In this chromosome, the order of jobs processing on workstations is in a way that at first, job 1 is processed on all of the working stations and then job 6 is processed according to the completion processing time of job 1 on each workstation and this characteristic that job has no waiting time between each workstation, is processed on all of the workstations. This approach is continued until all of the jobs are processed and completed on stations.

1	6	4	5	2	3

Figure1. Representation of chromosome

- Part 2:

This part of chromosome helps to show that every job in each workstation is processed on which machines.

For this part of the chromosome, a length equal to the total number of operations of jobs is considered. That means if it is assumed that there are N jobs and the number of operations of each job is equal to O, the length of this chromosome will be equal to N*O. The first O genes of this chromosome indicate the number of machines which the job with a priority of 1 should be processed on them.

This way of coding causes each mapping of chromosomes genes to is converted into feasible scheduling for the problem on each machine.

The role of a fitness function is to show the fitness of each chromosome. The fitness function constitutes the foundation of the selected phase. For the investigated problem in this paper, since the objective function is the minimization problem, the fitness value of each chromosome in each generation is considered equal to the deviation of the objective function corresponding to the chromosome, with the worst objective function in that generation plus 1.

$$FF_i = OF_w - OF_i + 1 \quad (13)$$

Amongst the most common performed methods, the two-point intersection can be pointed out which is used in this paper.

In the proposed problem, for implementation of mutation operator, two genes of a chromosome which have different values are selected randomly, and their values are displaced.

In this paper for selecting the survivals, three approaches are implemented: crossover operator, mutation operator and elites (chromosomes transferred to the next generation without any change). In order to generate better solutions, a local search approach is performed on 50 percent of the new generation.

Preparation of the algorithm means identifying the initial population number, intersection percentage, mutation and reproduction (elite percent) and these values are different for problems having different objectives. These preparations will be performed for the algorithm in the parameter

setting for the investigated problem. Reaching a certain number of generations is considered as the final conditions for the algorithm.

Different termination conditions can be applied to the GA algorithm. For the proposed GA algorithm, one of the following conditions causes the algorithm to reach its end:

1. Generating a specified number of generations.
2. No improvement is observed during a specified period of generations.

5.3 Setting the Genetic Algorithm Parameter

In this paper, the sample problems are divided and tested into two categories of small and large scales as shown in Table 1. For setting the parameters related to the genetic algorithm used in this paper, the Taguchi method has been used in which the optimum parameters for small and large-scale problems are shown in Table 2 and 3:

Table1. Small and large-scale problems

Small-scale	Large-scale
m×n	m×n
3×4	10×20
5×4	15×20
7×4	20×20
3×6	10×30
5×6	15×30
7×6	20×30
3×8	10×40
5×8	15×40
7×8	20×40

Table2. Parameter tuning for small-scale problems

Initial population	Number of generation	Crossover percentage	Mutation percentage	Elite percentage	Number of local searches
100	50	0.8	0.13	0.07	5

Table3. Parameter tuning for large-scale problems

Initial population	Number of generation	Crossover percentage	Mutation percentage	Elite percentage	Number of local searches
150	100	0.7	0.15	0.15	10

6. Computational Results

The mathematical model developed for the problem is simulated by the GAMS software and is solved by CPLEX solver. Also for coding, the presented solving methods by the C^{++} programming language are used. All the computations are performed on the computer with 4GBRAM, IntelCore2DuoP7550CPU, 2.26 GHz.

The processing times of jobs follow a uniform distribution between 1 and 100. The genetic algorithm is run four times for the proposed problems, and the average of solutions and the best

solution are produced by four times running. The average running times for problems are shown in columns of the computational results table. For evaluating the performance of the presented methods in the small-scales, the obtained results by the GAMS, bottleneck heuristic algorithm and genetic algorithm are compared with each other as summarized in Table 4. For the large-scale problems, since the solution of the problem is not obtained by a mathematical model in a reasonable time, the proposed solving methods are evaluated.

Table 4 shows the computational results related to small-scale problems, in accordance with Table 1. Ascan be concluded from the table, the genetic algorithm and bottleneck algorithm are respectively capable of finding 6 and four optimal solutions. There are four problems in which the solutions based on the bottleneck algorithm are better than or equal to the solutions produced by the GA. Times related to every two algorithms are negligible but the overlay of the times of the proposed algorithm is better than the GA. Regarding the objective function, both values have a negligible difference with solutions of the GAMS, but the GA algorithm produced better solutions for small-scale problems overall. The summary of the obtained results for the small-scale problems is reported in Table 5. The average error percentage of each solution method concerning obtained solutions of the GAMS, number of obtained optimum solutions from each algorithm and average times of solution methods and the GAMS in seconds are reported in columns of this table respectively.

Tables 6 and 7 respectively indicate computational results of the large-scale problems and summary of results obtained from the large-scale problems. As can be seen from the table, in six problems, the proposed algorithm outperforms the GA, and the total average of its solutions concerning the mean average of the solutions of the GA is 2.7% better. Briefly, the obtained solutions of bottleneck algorithm in the large-scale problems have better quality, but regarding the computational time, they need more time to find the solution. In total, it can be concluded that for small-scale and large-scale problems, the genetic algorithm and the bottleneck algorithm are more appropriate respectively.

The nonparametric Kruskal-Wallis test is used for validation of the algorithm in this paper. Both obtained results by the two algorithms were compared according to the abnormalities of the distribution in the confidence level of 95% using Minitab 16 software. As can be seen in Figure 2, it can be concluded that equality of the values obtained from the proposed algorithm cannot be rejected at the 95% confidence level.

Table4. Computational results for the small-scale problems for the GAMS, GA, and BB

m×*n*	GAMS		GA			BB	
	Solution	Time	Average Solution	Best Solution	Time	Solution	Time
3×4	441	5.08	441	441	0.344	441	0.051
5×4	632	10.25	632	632	0.378	632	0.007
7×4	654	11.723	677.5	671	0.421	841	0.012
3×6	598	788.11	598	598	0.393	598	0.023
5×6	637	1440.24	678.25	637	0.416	690	0.037
7×6	818	2164.12	844	827	0.468	884	0.065
3×8	729	3606	749	755	0.467	729	0.067
5×8	884	3602	884	884	0.567	927	0.144
7×8	925	3603.45	941.5	925	0.663	1147	0.215

Table5. Computational results for the small-scale problems (1)

Percent of optimal solutions		Average error of methods compared to the GAMS		Average computational time (seconds)		
GA	BB	GA	BB	GA	GAMS	BB
66.67	44.44	0.019	0.082	0.457	1692.33	0.0.69

Table6. Computational results for the large-scale problems for the GA and BB

m×*n*	GA			BB	
	Average Solution	Best Solution	Average Time	Solution	Time
3×20	2573.75	2464	0.866	2482	8.498
5×20	3331	3265	1.136	3279	13.705
7×20	3593	3581	1.358	3545	20.935
3×30	4032	4479	2.265	4444	39.944
5×30	4629	4479	3.023	4676	91.199
7×30	5116.25	5038	3.686	4950	163.740
3×40	5075.25	4983	2.964	5054	186.040
5×40	5678.75	5489	3.370	5721	331.508
7×40	6465.75	6442	4.846	6330	555.884

Table7. Computational results for the large-scale problems [1]

Average error of the BB compared to the GA	The BB average computational time (seconds)	The GA average computational time (seconds)
-0.0279	156.829	2.612

```
Kruskal-Wallis Test on Response

factor    N  Median  Ave Rank      Z
1        18    1758      18.4  -0.06
2        18    1815      18.6   0.06
Overall  36              18.5

H = 0.00  DF = 1  P = 0.950
H = 0.00  DF = 1  P = 0.950  (adjusted for ties)
```

Figure2. Results for the Kruskal-Wallis test

7. Conclusion and Suggestion for the Next Studies

In this paper, the bottleneck-based heuristic and meta-heuristic genetic algorithms were proposed for solving the flexible flowshop problem with considering re-entrant and no-wait specifications for the manufacturing environment. The mathematical model developed for the problem was modeled by the GAMS software and solved by the CPLEX solver. For coding the presented solution methods, the C^{++} programming language is used. Also, for comparing the two algorithms in terms of equality of the average of the obtained solutions from each technique, then on parametric Kruskal-Wallis was used. For the small-scale problems, the results obtained by the GAMS software are compared with the obtained results by the bottleneck and genetic algorithms. Likewise, for the large-scale problems, the results obtained by the bottleneck and genetic algorithm are compared with each other. The computational results show that the bottleneck-based algorithm shows better results for the large-scale problems and genetic algorithm for the small-scale problems.

Solving the proposed problem using different meta-heuristic algorithms and comparing the obtained results in this paper and also improvement of the existed algorithm in this paper is suggested for the next studies. Also, considering case studies to make the problem more applicable and setup times for jobs and limitations of maintenance and repairing of machinery can be some of the researching routes. The heuristic methods for obtaining a lower limit or solution of the problem are also proposed.

8. References

[1] Fordyce, K., Milne, R.J., Wang, C.T. and Zisgen, H. 2015. Modeling and Integration of Planning, Scheduling, and Equipment Configuration in Semiconductor Manufacturing: Part I. Review of Successes and Opportunities. International Journal of Industrial Engineering. Theory, Applications and Practice. 22(5): 575-600.

[2] Choi, S.W. and Kim, Y.D. 2008. Minimizing makespan on an m-machine re-entrant flowshop. Computers & Operations Research. 35(5): 1684-1696.

[3] Chen, J.S., Pan, J.C.H. and Lin, C.M. 2008. A Hybrid Genetic Algorithm for the Re-entrant Flow-shop Scheduling Problem. Expert Systems with Applications. 34(1): 570-577.

[4] Amin-Naseri, M. R. 2017. Solving Re-entrant No-wait Flow Shop Scheduling Problem. Journal of Industrial Engineering Research in Production Systems. 4(9): 271-279.

[5] Jing, C., Huang, W. and Tang, G. 2011. Minimizing Total Completion Time for Re-entrant Flow Shop Scheduling Problems. The Oretical Computer Science. 412(48): 6712-6719.

[6] Chu, F., Liu, M., Liu, X., Chu, C. and Jiang, J. 2018. Reentrant Flow Shop Scheduling Considering Multiresource Qualification Matching. Scientific Programming. 2018: 1-9.

[7] Jing, C., Tang, G. and Qian, X. 2008. Heuristic Algorithms for Two Machine Re-entrant Flow Shop. The oretical Computer Science. 400(1-3): 137-143.

[8] Choi, S.W. and Kim, Y.D. 2009. Minimizing Total Tardiness on a Two-machine Re-entrant Flowshop. European Journal of Operational Research. 199(2): 375-384.

[9] Hassanpour, S.T., Naseri, M.A. and Nahavandi, N. 2015. Solving Re-entrant no-Wait Flow Shop Scheduling Problem. International Journal of Engineering-Transactions C: Aspects. 28(6): 903-912.

[10] Adressi, A., Hassanpour, S. and Azizi, V. 2016. Solving Group Scheduling Problem in No-wait Flexible Flowshop with Random Machine Breakdown. Decision Science Letters. 5(1): 157-168.

[11] Chamnanlor, C., Sethanan, K., Gen, M. and Chien, C.F. 2017. Embedding Ant System in Genetic Algorithm for Re-entrant Hybrid Flow Shop Scheduling Problems with Time Window Constraints. Journal of Intelligent Manufacturing. 28(8): 1915-1931.

[12] Adressi, A., Bashirzadeh, R., Azizi, V. and Tasouji Hassanpour, S. 2014. Solving Group Scheduling Problem in No-wait Flow Shop with Sequence Dependent Setup Times. Journal of Modern Processes in Manufacturing and Production. 3(1): 5-16.

[13] Samarghandi, H. and ElMekkawy, T.Y. 2013. Two-machine, no-wait job shop problem with separable setup times and single-server constraints. The International Journal of Advanced Manufacturing Technology. 65(1-4): 295-308.

[14] Pang, K.W. 2013. A Genetic Algorithm based Heuristic for Two Machine no-wait Flowshop Scheduling Problems with Class Setup Times that Minimizes Maximum Lateness. International journal of production economics. 141(1): 127-136.

[15] Maya, J., Muthuswamy, S., Vélez-Gallego, M.C. and Rojas-Santiago, M. 2014. Minimising Makespan in a No-wait Flow Shop with Two Batch Processing Machines: a Grasp Algorithm. International Journal of Industrial and Systems Engineering. 17(2): 152-169.

[16] Riahi, V. and Kazemi, M. 2018. A New Hybrid Ant Colony Algorithm for Scheduling of No-wait Flowshop. Operational Research. 18(1): 55-74.

[17] Graham, R.L., Lawler, E.L., Lenstra, J.K. and Kan, A.R. 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In Annals of discrete mathematics. 5: 287-326.