Research Article GU J Sci 35(1): 92-111 (2022) DOI: 10.35378/gujs.682388

Gazi University

Journal of Science

http://dergipark.gov.tr/gujs

Genetic Algorithm Application for Permutation Flow Shop Scheduling Problems

Oguzhan Ahmet ARIK *

Nuh Naci Yazgan University, Industrial Engineering Department, 38170 Kayseri, Turkey

Highlights

- Genetic algorithm applied to permutation flow shop scheduling problems.
- Parameters of proposed GA are calibrated in view of optimality and elapsed time.
- Crossover probability has no effect on optimality and elapsed time.
- Well-known test instances have been solved with the proposed GA.

Article Info

Received: 30 Jan 2020
Accepted: 03 Mar 2021

Keywords

Genetic algorithm
Permutation flow shop
Scheduling
Makespans

Abstract

In this paper, permutation flow shop scheduling problems (PFSS) are investigated with a genetic algorithm. PFSS problem is a special type of flow shop scheduling problem. In a PFSS problem, there are n jobs to be processed on m machines in series. Each job has to follow the same machine order and each machine must process jobs in the same job order. The most common performance criterion in the literature is the makespan for permutation scheduling problems. In this paper, a genetic algorithm is applied to minimize the makespan. Taillard's instances including 20, 50, and 100 jobs with 5, 10, and 20 machines are used to define the efficiency of the proposed GA by considering lower bounds or optimal makespan values of instances. Furthermore, a sensitivity analysis is made for the parameters of the proposed GA and the sensitivity analysis shows that crossover probability does not affect solution quality and elapsed time. Supplementary to the parameter tuning of the proposed GA, we compare our GA with an existing GA in the literature for PFSS problems and our experimental study reveals that our proposed and well-tuned GA outperforms the existing GA for PFSS problems when the objective is to minimize the makespan.

1. INTRODUCTION AND LITERATURE REVIEW

Permutation flow shop scheduling (PFSS) is the consideration of the permutation order of jobs in the classical flow shop scheduling problem. In a PFSS problem, there are n jobs to be processed at m machines by following the same machine order and each machine must process jobs with the same job order. The total number of possible sequences is n! for a PFSS problem. The problem is classified as Np-Hard. By using the triple scheduling notations ($\alpha/\beta/\gamma$), the problem is classified as $F_m|perm|Cmax$. Non-approximate solution methods such as branch-and-bound take plenty of solution times to find the optimum sequence for any objective function while the problem size increases. Therefore, effective algorithms such as NEH algorithm [1] or metaheuristic algorithms such as genetic algorithm (GA), tabu search, or particle swarm optimization can be used to find a near-optimal solution in an endurable solution time. The parameter quality of proposed solution approaches of heuristics or metaheuristics for combinatorial optimization problems is so significant to obtain a near-optimal solution in a bearable time period. In this paper, a GA is proposed for PFSS problems to minimize the makespan (the maximum completion time). The parameters of the proposed GA such as crossover probability, mutation probability, and population size are analyzed with ANOVA. In order to evaluate the proposed GA's optimality and speed, Taillard's [2] flow shop instances including 20, 50, and 100 jobs with 5, 10, and 20 machines and their best-known solutions are used in this paper.

*e-mail:oaarik@nny.edu.tr

Researchers have proposed exact or heuristic methods for the problem for more than 40 years in the literature. Some researchers conducted literature surveys for the problem in the literature. Some of these are conducted by Yenisey and Yagmahan [3], Reza et al. [4], Framinan et al. [5] and Framinan et al. [6]. Because of the computational complexity of the problem, PFSS problem is one of the most investigated problems in the scheduling literature. Table 1 presents a detailed literature review about PFSS problems with the makespan ($Cmax$). As a summary for readers, we can say that the most effective algorithms for the problem are iterated greedy algorithms with accelerations schemas, tie-breaking mechanism, insertion-based local search operators for the complete solution, and partial solution.

Table 1. *Literature review for PFSS problems with makespan criterion*

Authors	**Method**	**Objectives and Constrains**	**Summary**
Tasgetiren et al. [7]	Particle swarm optimization (PSO)	$Cmax$ and the total flowtime	New best solutions were found.
Wang and Tang [8]	PSO	$Cmax$ with the blocking constrain	New best solutions were found.
Chen et al. [9]	PSO	$Cmax$	Their revised discrete PSO outperforms all the existing PSO algorithms for the problem.
Li and Deng [10]	PSO	$Cmax$	
Rajendran and Zieger [11]	Ant-colony optimization (ACO)	$Cmax$ and the total flowtime	New best solutions were found.
Ahmadizar [12]	ACO	$Cmax$	His proposed ACO outperforms well-known ACO algorithms in the literature for the problem.
Ruiz and Stützle [13]	Iterated Greedy (IG)	$Cmax$	Their IG outperforms all existing solution methods so far in the literature.
Ruiz and Stützle [14]	IG	$Cmax$ and total weighted tardiness	
Ribas et al. [15]	IG	$Cmax$ with the blocking constrain	
Minella et al. [16]	A new algorithm based on an IG algorithm	$Cmax$, tardiness, and flow time	
Grabowski and Wodecki [17]	Tabu search (TS)	$Cmax$	
Varadharajan and Rajendran [18]	Simulated Annealing (SA)	$Cmax$ and the total flowtime	
Grabowski and Pempera [19]	TS	$Cmax$ with blocking constrain	They proposed a specific neighborhood of algorithms that allows multi moves in an iteration.
Zobolas et al. [20]	GA and variable neighborhood search (VNS)	$Cmax$	
Tseng and Lin [21]	GA and local search	$Cmax$ and the total flowtime	
Pasupathy et al. [22]	GA	$Cmax$ and the total flowtime	

Chen et al. [23]	GA	$Cmax$	New strategy for combining global statistical information from population and local location information from each individual in the population.
Haq et al. [24]	GA and artificial neural network	$Cmax$	
Nagano et al. [25]	GA	$Cmax$	
Rad et al. [26]	New initial solution heuristic	$Cmax$	They introduced five new methods that outperform the NEH algorithm.
Dong et al. [27]	An improved NEH algorithm with the tie-breaking mechanism	$Cmax$	
Kalczynski and Kamburowski [28]	An improved NEH algorithm with the tie-breaking mechanism	$Cmax$	
Vázquez-Rodríguez and Ochoa [29]	GA	$Cmax$, the sum of tardiness, the sum of weighted tardiness, the sum of completion times, and the sum of weighted completion times	They obtained new NEH heuristic variants by using genetic programming.
Dubois-Lacoste et al. [30]	An algorithm consisted of two-phases local search and Pareto local search	$Cmax$, the sum of completion times and both of the weighted and no-weighted total tardiness	
Chiang et al. [31]	Memetic algortihm with NSGA-II	$Cmax$ and the total flowtime	
Zheng and Yamashiro [32]	Quantum differential evolutionary algorithm	$Cmax$, total flow time, and the maximum lateness of jobs	
Vallada and Ruiz [33]	New cooperative metaheuristic methods	Total tardiness and $Cmax$	
Lin and Ying [34]	SA	$Cmax$ and the total flowtime	
Ribas et al. [35]	A three-step heuristic algorithm	$Cmax$	
Laha and Chakraborty [36]	SA and NEH	$Cmax$	
Saravanan et al. [37]	Scatter search	$Cmax$	
Tzeng and Chen [38]	Distribution algorithm with ACO	$Cmax$	
Dasgupta and Das [39]	Cuckoo Search	$Cmax$ and mean flow time	
Chen et al. [40]	Heuristic	$Cmax$	

Moslehi and Khorasanian [41]	VNS algorithm with SA	$Cmax$ with limited buffer	New best solutions were found.
Rajendran et al. [42]	Heuristic rules for tie-breaking mechanism within NEH algorithm	$Cmax$	They reported their heuristic as the best-known heuristic rule in the literature.
Fernandez-Viagas and Framinan [43]	A new tie-breaking mechanism for heuristic and metaheuristic algorithms	$Cmax$	They reported their heuristic as the best-known heuristic rule in the literature.
Dubois-Lacoste et al. [44]	New local search mechanism for partial solutions in current metaheuristics	$Cmax$	They reported their heuristic as the best-known heuristic rule in the literature.
Abdel-Basset et al. [45]	A hybrid whale optimization algorithm	$Cmax$	
Benavides and Ritt [46]	Heuristics	$Cmax$	They stated that their new heuristics are more successful than the NEH algorithm as initial solution algorithms.
Chen et al. [47]	Quantum-inspired ACO	$Cmax$	
Kizilay et al. [48]	Variable block insertion heuristic	$Cmax$	They used their new heuristic within well-known metaheuristics and stated that their new heuristic is well fitted with well-known metaheuristics.
Fernandez-Viagas and Framinan [49]	A best-of-breed IG algorithm	$Cmax$	It is reported as the best-so-far approximate method for the problem.
Arık [50]	Artificial bee colony	$Cmax$	The best component of IG combined with an artificial bee colony algorithm.
Gyms et al. [51]	A new node decomposition scheme that combines dynamic branching and lower bound refinement strategies	$Cmax$	
Arık [52]	Population-based TS	$Cmax$	Hybrid solution method with crossover and mutation strategies for the problem under effects of learning and deterioration.

In this paper, we propose a GA for PFSS problems with the makespan criterion that is the most common performance criterion in the literature. GA drives random search operations within its structure inspiring by the evolutionary process. Like any metaheuristic algorithm for a combinatorial optimization problem, GA's strength and applicability depend on its design and parameter tuning to the problem. In this study, we tune parameters of GA considering optimality and elapsed time until finding a near-optimal / optimal solution for the PFSS problem that has lots of real-life examples. For our proposed GA, we determine the best parameter levels and effects of GA parameters on both optimality and elapsed time.

2. MATHEMATICAL MODEL

In this section a mixed-integer linear programming model for PFSS is given for the readers as follows:

Indices
$i: job\ index, i = 1 \ldots . n$
$j: machine\ index, j = 1 \ldots . m$
$r: common\ position\ index\ in\ all\ machines\ r = 1 \ldots . n$
Parameters
$P_{i,j}: basic\ processing\ time\ of\ job\ i\ on\ machine\ j$
Decision Variables
$X_{i,r}: if\ job\ i\ is\ assigned\ on\ position\ r\ of\ all\ machines, then\ it's\ 1, otherwise\ 0$
$P_{[r],j}: processing\ time\ of\ the\ job\ assigned\ on\ position\ r\ in\ machine\ j$
$C_{[r],j}: completion\ time\ of\ the\ job\ assigned\ on\ position\ r\ in\ machine\ j$
$S_{[r],j}: starting\ time\ of\ the\ job\ assigned\ on\ position\ r\ in\ machine\ j$
$C_{max}: makespan\ of\ the\ schedule$
Model

$$Min\ z = C_{max} \quad (1)$$

s. t.:

$$C_{max} \geq C_{[n],m} \quad (2)$$
$$\sum_i^n X_{i,r} = 1\ \forall r \quad (3)$$
$$\sum_r^n X_{i,r} = 1\ \forall i \quad (4)$$
$$C_{[r],j} \geq S_{[r],j} + P_{[r],j} \forall r, j \quad (5)$$
$$S_{[r],j} \geq C_{[r],j-1}\ \forall r, j = 2, \ldots, m \quad (6)$$
$$S_{[r],j} \geq C_{[r-1],j}\ \forall j, r = 2, \ldots, m \quad (7)$$
$$P_{[r],j} = \sum_i^n X_{i,r} * P_{i,j}\ \forall\ r, j \quad (8)$$
$$C_{[0],1} = 0 \quad (9)$$
$$C_{max} \geq 0 \quad (10)$$
$$C_{[r],j}, P_{[r],j}, S_{[r],j} \geq 0\ \forall r, j \quad (11)$$
$$X_{i,r} \in \{0,1\}\ \forall\ i, r \quad (12)$$

The objective function of model (1) minimizes the maximum completion time of the schedule. Constraint (2) is to determine the makespan. Constraints (3) and (4) guarantee that each job must be assigned to only one position and each position must be used for only one job. Constraint (5) shows the relationship among completion, start, and processing times of the job assigned to the position r. Constraint (6) shows that a job's starting time in machine j must be greater than or equal to the completion time of the same job in machine $j-1$. Constraint (7) shows that starting time of the job in position r must be greater than or equal to the completion time of the job in position $r-1$ in the same machine. Constraint (8) determines the calculation of the processing time of the job in position r in machine j. Constraint (9) expresses that all jobs can be processed at time zero in the first machine. Constraints (10-12) show necessary domains of decision variables.

3. GENETIC ALGORITHM

There are several existing GA methods in the literature for PFSS problems. Some of these were conducted by Pasupathy et al. [22], Chen et al. [23], and Nagano et al. [25]. Pasupathy et al. [22] proposed a multi-objective GA for scheduling in flow shops to minimize the makespan and total flowtime of jobs. They used binary-tournament selection, single-point crossover, and shift mutation mechanisms in their proposed GA. They did not tune the parameters of their proposed multi-objective GA. Chen et al. [23] proposed a self-guided GA for PFSS problems with the makespan criteria. They used a quality determination function for

solutions in the solution population and they used quality values of solutions. They used binary selection, self-guided two-point crossover, self-guided swap-based mutation mechanism in their proposed GA. Considering other GA methods for PFSS problems with the makespan criterion, our proposed GA with its components is distinct. We use a roulette wheel selection mechanism, two-point crossover with repair function, and inversion-based mutation within our proposed GA. With its distinct features, we tune the parameters of our proposed GA to increase its performance for PFSS problems with the makespan criterion.

In this section, pseudo-codes and details of the proposed GA are given to the readers. GA is one of the most known metaheuristics for combinatorial optimization problems. GA is inspired by the evaluation process in nature. GA has abilities such as stochastic best solution search and generating new solutions from best-known solutions in its solution pool. The general schema or pseudo-code of the proposed GA is illustrated in Figure 1.

The selection operation in this proposed GA is the roulette wheel selection method. The pseudo-codes of evaluation and selection operators are shown in Figure 2. The crossover operator is a stochastic solution generation method using existing solutions in the matching pool of GA and it takes place after the selection operator. A solution pair is selected from the matching pool considering their fitness values after selection operation. Then some substrings of these solutions are exchanged to generate new alternative solutions for the next step of GA, if a generated random number is less than or equal to the crossover probability of GA. While exchanging substrings between solutions, newly generated solutions may be disrupted and unfeasible. Therefore, a repair operator that fixes unfeasible solutions may be needed to have feasible and good candidate solutions. The pseudo-code of the crossover operator in this proposed GA is illustrated in Figure 3.

The encoding scheme of this proposed GA is permutation encoding. In this encoding scheme, each chromosome (solution) is a string of job indices that are illustrated with numbers between 1 and n. The repair operator in this GA, simply counts how many times a job is assigned to a solution and then replaces unassigned jobs to first places of multiple-times assigned jobs. This process goes on until there are no assigned or multiple-times assigned jobs to remain. The pseudo-code of repair operation of the proposed GA is illustrated in Figure 4.

The mutation operator is another step of GA and it assures the diversity of the population. In this paper, the mutation operator is a kind of order chancing or inversion mutation. The pseudo-code of mutation is illustrated in Figure 5.

```
1.  Start
2.  Read GA parameters
3.  Read Problem Data
4.  Set NI, Number of Iterations
5.  Generate Initial Population
6.  Evaluate Solutions in Initial Population
7.  Select Best Makespan and Best Schedule
8.  For i =1 to NI
9.      Select solutions from population via roulette wheel selection
10.     Send selected solution pairs to single point crossover operator
11.     Repair unfeasible solutions after crossover operator
12.     Send solutions to order changing mutation operator
13.     Evaluate solutions in the current population
14.     Select best makespan and best schedule from the current population
15.     If best makespan of current population is less than best makespan, then
    Set new best makespan and best schedule
16. Next i
17. Display best makespan and schedule
18. End
```

Figure 1. *The pseudo-code of the proposed genetic algorithm*

```
1.  Start
2.  Get solution population
3.  Calculate makespan values of solutions in the population
4.  Evaluate fitness values of solutions in the population
5.  Calculate cumulative selection probabilities of solutions in the population
6.  For i =1 to number of solutions in the population
7.      Generate a random number between 0 and 1
8.      Find and select the solution considering generated random number
9.      Transmit the selected solution into matching pool for crossover
10. Next i
11. End
```

Figure 2. *The pseudo-code of evaluation and selection operator*

```
1.  Start
2.  Read solutions from matching pool
3.  Convert solutions to an array Pop(t,k)
    where t = Population Size and k = number of jobs
4.  For j =1 to Population Size
5.      Generate a random number r between 1 and Population Size
6.      For i =1 to r
7.          Generate a random number r1 between 0 and 1
8.          If r1 ≤ Crossover probability, then
9.              Generate two numbers, a = 0 and b = 0
10.             If r > Population Size/2 then
11.                 If i≠n then
12.                     a = Pop(j, k - i - 1)
13.                     b = Pop(j + 1, k - i - 1)
14.                     Pop(j, k - i - 1) = b
15.                     Pop(j + 1, k - i - 1) = a
16.                 Else
17.                     a = Pop(j, i)
18.                     b = Pop(j + 1, i)
19.                     Pop(j, i) = b
20.                     Pop(j + 1, i) = a
21.                 End if
22.             Else
23.                 a = Pop(j, i)
24.                 b = Pop(j + 1, i)
25.                 Pop(j, i) = b
26.                 Pop(j + 1, i) = a
27.             End if
28.         End if
29.     Next i
30. Next j
31. Send solutions to repair operator
32. End
```

Figure 3. *The pseudo-code of crossover operator*

```
1.  Start
2.  Get the population after the crossover operation
3.  For i=1 to Population Size
4.      Read i^th solution from population
5.      Calculate how many times a job assigned to i^th solution
6.      Repeat
7.          Assign the unassigned jobs to the first places of multiple-times assigned
        to i^th solution
8.      Until There is no job assigned multiple-times or there is no job unassigned
9.  Next i
10. Send the population for the mutation operator
11. End
```

Figure 4. *The pseudo-code of the repair operator*

```
1.  Start
2.  Read solutions from after repair operator
3.  Convert solutions to an array Pop(t,k)
    where t = Population Size and k = number of jobs
4.  For  t =1 to Population Size
5.      Generate a random number pm between 0 and 1
6.      If pm ≤ mutation probability
7.          Generate a new array DummyPop(k)
8.          For i=1 to k
9.              DummyPop(i) = Pop(t,i)
10.         Next k
11.         Generate two new integers a and b (a>b) between 1 and k
12.         Generate a new integer y = 1
13.         For i=1 to a - b
14.             If  b + i -1 ≠ a
15.                 Pop(t, b - 1 + i) = DummyPop(a + 1 - y)
16.                 y +=1
17.             Else
18.                 Pop(t, a) = DummyPop(b)
19.             End If
20.         Next i
21.     End if
22. Next  t
23. Send solutions to evaluation step
24. End
```

Figure 5. *The pseudo-code of the mutation operator*

4. PARAMETER ANALYSIS FOR GENETIC ALGORITHM

In this section, a sensitivity analysis is made for obtaining the most suitable GA parameters that effect solution quality and elapsed time. The first instance (T001) of Taillard's [2] flow shop instance is preferred to make sensitivity analysis. Different population sizes and mutation/crossover probabilities are used 20 times to generate a test database for sensitivity analysis. In the sensitivity analysis; search for mutation probability p_m was started from 0.01 and 0.15 with 0.02 increments, for p_c was started from 0.8 to 0.95 with 0.05 increments, for population size was started from 30 to 150 with 30 increments. The total number of executions with these different parameters is 3200. Test results are analyzed with ANOVA and Tukey procedure for makespan values and elapsed times of executions. The ANOVA results for makespan values obtaining by using different parameters are given in Table 2.

Table 2*. Anova results of groups' population sizes, crossover, and mutation probabilities for* C_{max}

Source	**DF**	**Adj. SS**	**Adj. MS**	**F-Value**	**P-Value**
Population size	4	32184	8046.1	47.55	0.000
Crossover Prob.	3	305	101.8	0.6	0.614
Mutation Prob.	7	538955	193323.4	114.19	0.000
Error	3185	135264	169.2		
Lack-of-fit	145	384402	264.8	1.61	0.000
Pure Error	3040	500553	164.7		
Total	3199	706708			

As understood from Table 2, the factor of crossover probability has not a significant difference for makespan values by comparing it with the other two factors because the P-value of the factor of crossover probability is greater than 0.05. On the other hand, the factors of population size and mutation probability have significant differences because these factors' P-values are less than 0.05. The same induction can be made by seeing the main effects plot for makespan values for these factors illustrated in Figure 6.

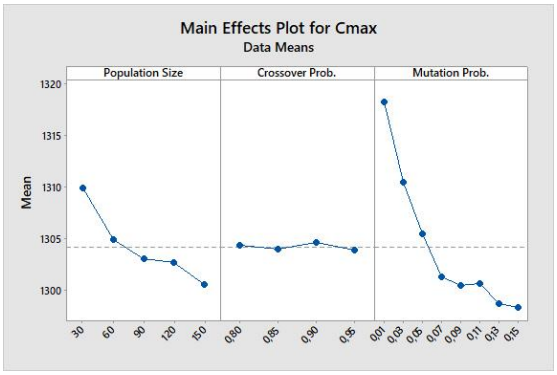

Figure 6. *The main effect plot for makespan values of factors*

Figure 6 shows that the most significant effect is made by the factor of mutation probability and the factor of crossover probability does not have an effect on makespan values. Furthermore, there are illustrations representing the results of Tukey procedures for these factors as seen in Figure 7.

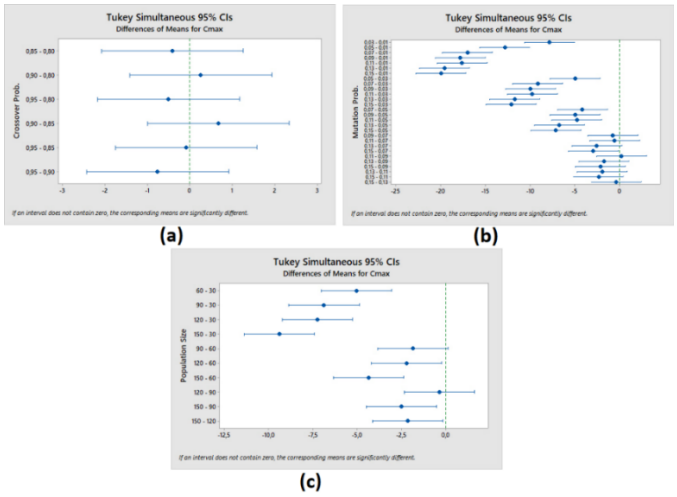

Figure 7. *Results of Tukey Procedures for the factors of (a) crossover probability, (b) mutation probability, and (c) population size*

Figure 7 shows that any crossover probability between 0.80 and 0.95 does not affect the makespan. The remaining two factors except crossover probability have effects on the makespan value. The interval plot of makespan considering different mutation probabilities and population sizes in Figure 8 shows that the least average makespan value is found where mutation probability is 0.15 and population size is 150.

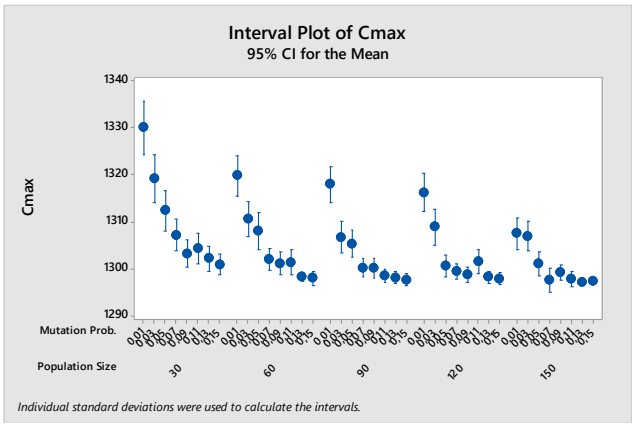

Figure 8. *Interval plot of makespan values*

After these analyses for makespan values obtained by using different GA parameters, the factor crossover probability may not has a significant difference for makespan. If the same analysis is made for elapsed times of the same instance with the same intervals of parameters, the ANOVA results in Table 3 show that the factor of crossover probability has not a significant difference for elapsed times of executions.

Table 3. *Annova results of groups' population sizes, crossover, and mutation probabilities for elapsed times*

Source	**DF**	**Adj. SS**	**Adj. MS**	**F-Value**	**P-Value**
Population size	4	12333.3	3083.33	34837.28	0.000
Crossover Prob.	3	0.2	0.07	0.75	0.522
Mutation Prob.	7	4458.7	636.95	7196.64	0.000
Error	3185	281.9	0.09		
Lack-of-fit	145	209.7	1.45	60.95	0.000
Pure Error	3040	72.2	0.02		
Total	3199	17074.1			

The main effect plot for elapsed times for these factors is given in Figure 9 and it shows that the most significant factor for elapsed times is the population size and the factor of crossover probability has no effect on elapsed times.

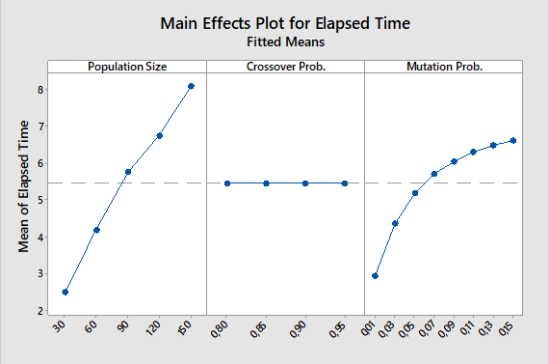

Figure 9. *The main effects plot for elapsed time*

The results of Tukey procedures of these factors for elapsed times are given in Figure 10 and they show that the factor of crossover probability has no effect on elapsed times.

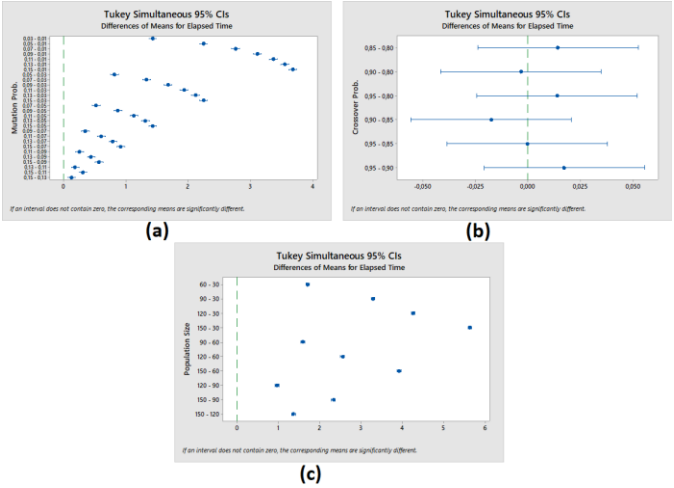

Figure 10. *Results of Tukey Procedures for the factors of (a) mutation probability, (b) crossover probability, and (c) population size*

The interval plot of elapsed times considering different mutation probabilities and population sizes in Figure 11 shows that the least average elapsed time is found where mutation probability is 0.01 and population size is 30.

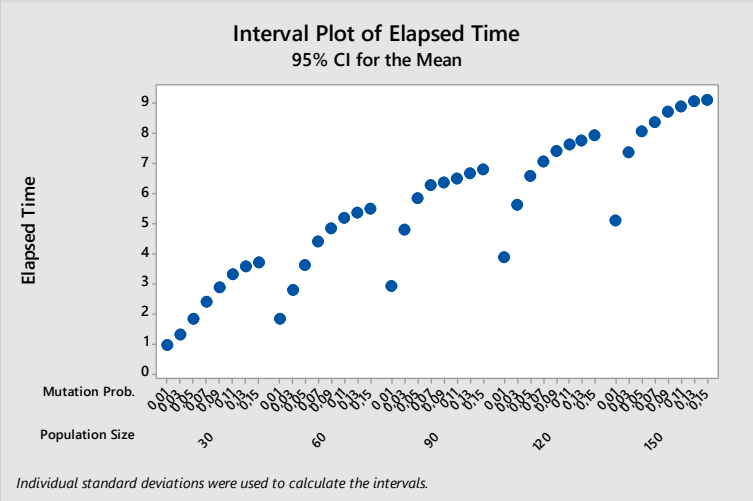

Figure 11. *Interval plot of elapsed times*

As a conclusion of these analyses, the factor of crossover probability between 0.80 and 0.95 has no effect on elapsed time and makespan value of PFSS problems. The main question still exists. Which parameters are good enough to find a near-optimal solution in a reasonable time period? To find an answer to this question, the intervals for mutation probabilities and population size must be considered simultaneously because the analyses show that the factor of crossover probability has no effect. Tables 4 and 5 show average makespan values and elapsed times of different mutation probabilities and population sizes. By using this data, Figures 12 and 13 show the interaction of these parameters in view of elapsed times and makespan values.

Table 4. *Average makespan values and elapsed times obtaining by using different population times*

Population Size	Cmax Avg.	Elapsed Time Avg.
30	1309.917	2.472562
60	1304.891	4.179554
90	1303.045	5.768405
120	1302.677	6.736424
150	1300.53	8.102433

Table 5. *Average makespan values and elapsed times obtaining by using different mutation probabilities*

Mutation Prob.	Cmax Avg.	Elapsed Time Avg.
0.01	1318.2825	2.932559
0.03	1310.445	4.365031151
0.05	1305.47	5.178598573
0.07	1301.28	5.69600607
0.09	1300.49	6.04521352
0.11	1300.6875	6.301976961
0.13	1298.7175	6.48636089
0.15	1298.3535	6.60922855

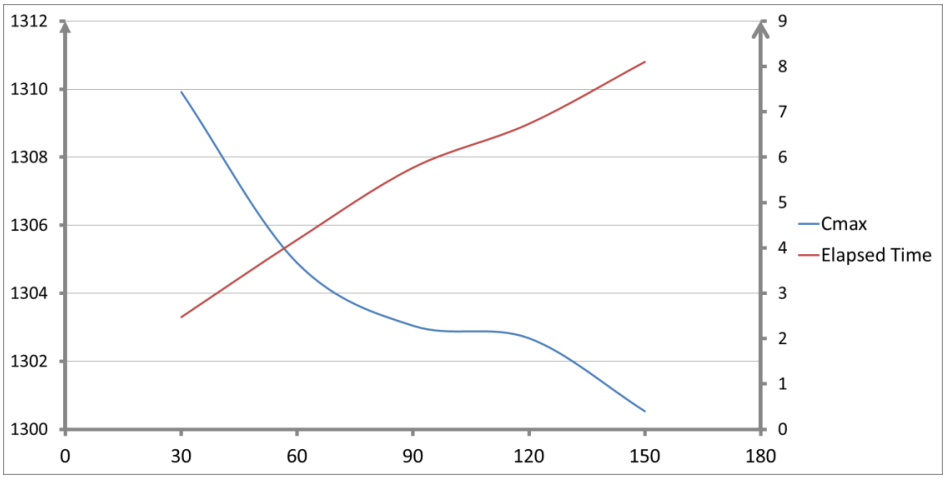

Figure 12. *Comparison of population sizes in view of makespan and elapsed time*

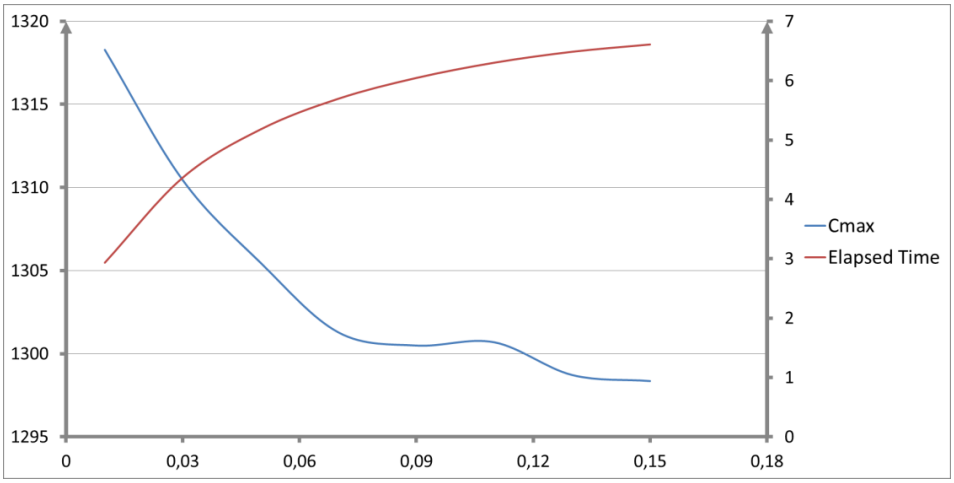

Figure 13. *Comparison of mutation probabilities in view of makespan and elapsed time*

Figure 12 shows the reasonable population size should be less than or equal to 60 and Figure 13 shows that the reasonable mutation probability is 0.03. While dealing with combinatorial optimization, the speed and optimality of the solution algorithm are so significant. The decision about the tradeoff between speed and optimality can be biased in view of a decision-maker. Therefore, this paper suggests that mutation probability should be 0.15 and population size should be 60. While the number of solutions (population size) in a population increases, the number of searches and other operations such as evaluation, selection, crossover, and mutation will be increased and this leads the elapsed time of the algorithm to increase. The best elapsed time is found where the population size is 30 but the worst makespan value is found with the same population size. In order to make a tradeoff among 30, 60, 90,120, and 150 population sizes for elapsed time and makespan, Figure 12 shows interaction among these values, and 60 seems good enough to have a near-optimal solution in a reasonable time period. The worst makespan and the best elapsed time is found where the mutation probability is 0.01. If a reader considers Figure 13, he/she can say that 0.03 seems good enough to have a near-optimal solution. However, this paper suggests being biased about mutation probability because the best average makespan value is found where mutation probability is 0.15 for all population sizes. The same approach can be considered for setting population size as 150 but the average makespan value where mutation probability is 0.15 is a bit less than the average makespan value where population size is 150.

4. PERFORMANCE OF THE PROPOSED GENETIC ALGORITHM

In this section, the results of optimality and speed test of proposed GA by using Taillard's [2] test instances including 20, 50, and 100 jobs with 5, 10, and 20 machines are given. The parameters of the proposed GA are; mutation probability is 0.15, crossover probability is 0.80 and population size is 60. There are *n!* possible schedules for a PFSS so the number of iterations should be related to *n.* Also, the number of machines *m* must be considered while solving instances that have the same *n* value and different *m* values. Therefore, this paper suggests the number of iterations can be set as *50(n+m).* All test instances were successively solved fifty times on the same computer. Each time while solving an instance, the proposed GA used only one core of the CPU. The proposed GA was coded by using the VB.NET programming language and MS Access database. Also, all test instances were solved and the sensitivity analyses were made on a standard desktop with i5-7500 CPU and 8GB RAM. Tables 6, 7, and 8 give average makespan values, best makespan values, the average elapsed times, and optimality of the proposed GA considering both best and average makespans for all test instances including 20, 50, and 100 jobs, respectively. The optimality value can be calculated as follows:

$$optimality = 1 - \frac{C_{max} - best\ known\ makespan}{best\ known\ makespan} \quad . \tag{13}$$

Equation (13) uses the best-known makespan values in the literature. In each of Tables 6,7 and 8, if the problem has an optimal makespan value, it is marked with an asterisk. If the problem has not an optimal makespan so far, then the lower bound [53] of that problem is used to calculate the optimality of the proposed GA.

Table 6. *Results of test instances including 20 jobs and different machine numbers*

n/m	Problem	Opt*/ LB	Avg. Sol.	Best Sol.	Avr. elapsed time	Avg. Optimality	Optimality
20/5	TL001	1278*	1297.4	1278	2.282	0.9848	1.0000
	TL002	1359*	1371.22	1360	2.227	0.9910	0.9993
	TL003	1081*	1104.92	1088	2.150	0.9779	0.9935
	TL004	1293*	1323.84	1305	2.229	0.9761	0.9907
	TL005	1235*	1262.6	1244	2.253	0.9777	0.9927
	TL006	1195*	1222.68	1195	2.240	0.9768	1.0000
	TL007	1234*	1252.08	1239	2.224	0.9853	0.9959
	TL008	1206*	1229.74	1206	2.244	0.9803	1.0000
	TL009	1230*	1261.58	1233	2.245	0.9743	0.9976
	TL010	1108*	1134.46	1111	2.239	0.9761	0.9973
					Avr.	**0.9800**	**0.9967**
20/10	TL011	1582*	1649.16	1609	4.504	0.9575	0.9829
	TL012	1659*	1735.78	1684	4.553	0.9537	0.9849
	TL013	1496*	1568.18	1528	4.384	0.9518	0.9786
	TL014	1377*	1449.96	1390	4.569	0.9470	0.9906
	TL015	1419*	1501.72	1441	4.565	0.9417	0.9845
	TL016	1397*	1451.56	1416	4.596	0.9609	0.9864
	TL017	1484*	1540.48	1510	4.544	0.9619	0.9825
	TL018	1538*	1619.2	1574	4.582	0.9472	0.9766
	TL019	1593*	1648.66	1612	4.569	0.9651	0.9881
	TL020	1591*	1647.32	1619	4.532	0.9646	0.9824
					Avr.	**0.9551**	**0.9837**
20/20	TL021	2297*	2396.38	2318	10.666	0.9567	0.9909

TL022	2099*	2179.54	2135	10.671	0.9616	0.9828
TL023	2326*	2411.32	2365	10.302	0.9633	0.9832
TL024	2223*	2315.38	2254	10.689	0.9584	0.9861
TL025	2291*	2377.96	2328	10.653	0.9620	0.9838
TL026	2226*	2302.8	2247	10.633	0.9655	0.9906
TL027	2273*	2357.38	2317	10.188	0.9629	0.9806
TL028	2200*	2285.78	2228	10.675	0.9610	0.9873
TL029	2237*	2340.7	2288	10.656	0.9536	0.9772
TL030	2178*	2287.48	2216	10.615	0.9497	0.9826
					Avr. 0.9595	**0.9845**

Table 7. *Results of test instances including 50 jobs and different machine numbers*

n/m	**Problem**	**Opt*/ LB**	**Avg. Sol.**	**Best Sol.**	**Avr. elapsed time**	**Avg. Optimality**	**Optimality**
50/5	TL031	2724*	2745.5	2729	22.111	0,9921	0.9982
	TL032	2834*	2874.32	2838	22.097	0.9858	0.9986
	TL033	2621*	2658.66	2629	21.928	0.9856	0.9969
	TL034	2751*	2797.7	2762	21.855	0.9830	0.9960
	TL035	2863*	2893.5	2864	21.813	0.9893	0.9997
	TL036	2829*	2851.54	2832	21.738	0.9920	0.9989
	TL037	2725*	2762.38	2735	21.808	0.9863	0.9963
	TL038	2683*	2716.66	2694	21.752	0.9875	0.9959
	TL039	2552*	2592.06	2565	21.777	0.9843	0.9949
	TL040	2782*	2795.16	2782	21.733	0.9953	1.0000
					Avr.	**0.9881**	**0.9975**
50/10	TL041	2991*	3184.8	3126	40.176	0.9352	0.9549
	TL042	2867*	3069.2	3003	39.829	0.9295	0.9526
	TL043	2839*	3067.18	2976	39.563	0.9196	0.9517
	TL044	3063*	3205.72	3133	39.503	0.9534	0.9771
	TL045	2976*	3184.12	3104	39.487	0.9301	0.9570
	TL046	3006*	3182.16	3082	39.317	0.9414	0.9747
	TL047	3093*	3277.68	3201	39.490	0.9403	0.9651
	TL048	3037*	3182.48	3124	39.487	0.9521	0.9714
	TL049	2897*	3071.58	3008	39.204	0.9397	0.9617
	TL050	3065*	3251.2	3185	39.238	0.9392	0.9608
					Avr.	**0.9381**	**0.9627**
50/20	TL051	3771	4164.56	4070	82.418	0.8956	0.9207
	TL052	3668	4029.56	3937	81.903	0.9014	0.9267
	TL053	3591	3982.3	3897	82.065	0.8910	0.9148
	TL054	3635	4038.9	3906	85.700	0.8889	0.9254
	TL055	3553	3997.12	3875	82.392	0.8750	0.9094
	TL056	3667	4009.52	3875	82.203	0.9066	0.9433
	TL057	3672	4033.62	3936	82.496	0.9015	0.9281
	TL058	3627	4040.82	3908	81.571	0.8859	0.9225
	TL059	3645	4073.66	3975	81.818	0.8824	0.9095
	TL060	3696	4071.94	3946	81.682	0.8983	0.9324
					Avr.	**0.8927**	**0.9233**

Table 8. *Results of test instances including 100 jobs and different machine numbers*

n/m	**Problem**	**Opt*/ LB**	**Avg. Sol.**	**Best Sol.**	**Avr. elapsed time**	**Avg. Optimality**	**Optimality**
100/5	TL061	5493*	5511.78	5495	146,138	0.9966	0.9996
	TL062	5268*	5299.48	5284	145.800	0.9940	0.9970
	TL063	5175*	5222.92	5200	145.037	0.9907	0.9952
	TL064	5014*	5040.3	5029	144.737	0.9948	0.9970
	TL065	5250*	5285.12	5255	142.779	0.9933	0.9990
	TL066	5135*	5162.1	5137	144.586	0.9947	0.9996
	TL067	5246*	5309.34	5264	145.902	0.9879	0.9966
	TL068	5094*	5141.06	5105	145.078	0.9908	0.9978
	TL069	5448*	5508.06	5473	145.368	0.9890	0.9954
	TL070	5322*	5369.84	5342	145.746	0.9910	0.9962
					Avr.	**0.9923**	**0.9973**
100/10	TL071	5770*	5990.32	5891	250.429	0.9618	0.9790
	TL072	5349*	5550.42	5456	248.158	0.9623	0.9800
	TL073	5676*	5850.38	5761	246.338	0.9693	0.9850
	TL074	5781*	6037.88	5942	250.578	0.9556	0.9722
	TL075	5467*	5741.7	5614	250.805	0.9498	0.9731
	TL076	5303*	5486.2	5412	248.455	0.9655	0.9794
	TL077	5595*	5774.4	5715	249.876	0.9679	0.9786
	TL078	5617*	5837.6	5777	247.177	0.9607	0.9715
	TL079	5871*	6035.18	5967	250.288	0.9720	0.9836
	TL080	5845*	6036.5	5954	250.179	0.9672	0.9814
					Avr.	**0.9632**	**0.9784**
100/20	TL081	6106	6790.18	6636	486.289	0.8879	0.9132
	TL082	6183*	6783.66	6654	485.826	0.9029	0.9238
	TL083	6252	6850.46	6745	481.684	0.9043	0.9211
	TL084	6254	6817.54	6689	482.572	0.9099	0.9304
	TL085	6262	6871.62	6741	477.946	0.9026	0.9235
	TL086	6302	6917.22	6787	498.876	0.9024	0.9230
	TL087	6184	6894.78	6763	475.453	0.8851	0.9064
	TL088	6315	7012.7	6895	483.238	0.8895	0.9082
	TL089	6204	6884.08	6758	484.152	0.8904	0.9107
	TL090	6404	6971.6	6830	476.228	0.9114	0.9335
					Avr.	**0.8986**	**0.9194**

In order to be sure about which value is the best between 0.03 and 0.15 mutation probabilities, the same set of GA parameters except for mutation probability where it is 0.03 are used and the average optimality of average makespan values was found as 0.9467 and the average optimality of best solutions was found as 0.9646. After sensitivity analyses, the average optimality of average makespan values is 0.9567 and the average optimality for best solutions is 0.9715. Therefore, setting mutation probability as 0.15 is better than the number of 0.03.

GA has been one of the strongest metaheuristics for optimization problems. GA has an easily-appliable structure to optimization problems and three basic parameters. These parameters are population size,

crossover probability, and mutation probability. Since its performance for any problem depends on the parameter settings. Therefore, we tuned the parameters of our proposed GA. While the problem's size (the numbers of jobs and machines) increases the optimality of the proposed GA decrease gradually because of the termination criterion. The termination criterion is the number of iterations having a formula of *50(n+m)*. As understood from Tables 6,7 and 8, the number of machines in the problem dramatically effects the optimality. We selected this criterion in the parameter tuning because we wanted a well-tuned parameter set that can be used in a short running time for the problem.

In order to test the performance of our proposed GA against an existing GA for PFSS problems, we coded the GA of Pasupathy et al. [22] that was originally coded for a multi-objective PFSS problem. In their GA, they used binary tournament selection, single-point crossover, and shift mutation mechanisms in their proposed GA. They suggested crossover probability as 1.0 and mutation probability as 0.1. They did not state the population size so we use 60 as population size in our version of the GA of Pasupathy et al. [22]. We coded the GA of Pasupathy et al. [22] by using the VB.NET programming language and MS Access database. And this performance comparison between two algorithms was made on a standard desktop with i5-7500 CPU and 8GB RAM. For performance evaluation, we selected Taillard's [2] test instances including 20, 50, and 100 jobs with 5, 10, and 20 machines. There are 90 instances in our experiment. We executed each GA five times for each instance. For the termination criterion, we used a well-known elapsed time calculation formula ($t.n.m/2$) for FPSS problems. In that formula, t is a constant for elapsed time calculation in milliseconds. For performance indicator, we use the formula called average relative derivation (ARD) from the best known makespan value as follows:

$$\frac{1}{90}\sum_{k=1}^{90}\frac{\overline{Cmax}_k - best\ known\ makespan\ of\ instance\ k}{best\ known\ makespan\ of\ instance\ k} \tag{14}$$

where $\overline{Cmax}_i$ is the average makespan of instance k ($k \in \{1,2,3,\dots,90\}$) after 5 executions of any GA method. As understood from Equation (14), the GA method having the lowest ARD score must be better than the other because it devives less from the best makespan values of instance than the other does.We used three t values ($t \in \{30, 60, 90\}$)) for performance comparison between our proposed GA and GA of Pasupathy et al. [22]. The ARD values of the two compared algorithms are given in Table 9. As seen from this experiment, our proposed GA outperforms the other GA proposed by Pasupathy et al. [22]. There was no parameter setting in the study of Pasupathy et al. [22] for their proposed GA. They just suggested crossover and mutation probabilities for their proposed GA. Since we tuned GA parameters such as population size, crossover probability, and mutation probability to our proposed GA, our GA's performance was expected to be satisfactory by comparing with another GA without parameter tuning.

Table 9. *The ARD values of compared algorithms*

Running time	**Our proposed GA**	**The GA of Pasupathy et al. [22]**
$30.n.m/2$ ms	0.0645	0.1400
$60.n.m/2$ ms	0.0583	0.1368
$90.n.m/2$ ms	0.0547	0.1347

5. CONCLUSION

In this paper, a genetic algorithm for permutation flow shop scheduling problems where the objective is to minimize makespan is proposed and the parameter quality of the proposed GA is tried to increase with sensitivity analyses. Using the right combinations of parameters in which each of them has different effects on optimality and elapsed time of the proposed algorithm is so significant for any solution approach proposed for combinatorial optimization problems. The sensitivity analyses show surprisingly that the factor of crossover probability does not have an effect on makespan values or elapsed times of the proposed

GA. Furthermore, the analyses show that the factor of mutation probability has a more significant effect on the makespan values than others and the factor of population size has a more significant effect on elapsed times of the proposed algorithm than others. As a conclusion of the discussion after sensitivity analyses, mutation probability, crossover probability, and population size are suggested as 0.15, 0.80, and 60, respectively. Using this parameter set in the proposed GA, well-known 90 instances are solved in order to evaluate the proposed GA's performance in view of optimality and speed. As a result of this performance test, the proposed GA has an averagely %97.15 optimality. Furthermore, we compared our proposed GA with another existing GA for PFSS problems and the experimental results revealed that our GA outperforms the existing GA in the literature. For future researches, these parameters can be used in the performance evaluation of other future genetic algorithm approaches. Furthermore, this proposed GA can be investigated with its suggested parameters for fuzzy flow shop scheduling or other flow shop scheduling problems with external effects such as learning and deterioration.

CONFLICTS OF INTEREST

No conflict of interest was declared by the author.

REFERENCES

[1] Nawaz, M., Enscore Jr, E. E., Ham, I., "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem", Omega, 11(1): 91–95, (1983).

[2] Taillard, E., "Benchmarks for basic scheduling problems", European Journal of Operational Research, 64(2): 278-285, (1993).

[3] Yenisey, M. M., Yagmahan, B., "Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends", Omega, 45: 119–135, (2014).

[4] Hejazi, S. R., Saghafian, S., "Flowshop-scheduling problems with makespan criterion: A review", International Journal of Production Research, 43(14): 2895–2929, (2005).

[5] Framinan, J. M., Gupta, J. N. D., Leisten, R., "A review and classification of heuristics for permutation flow-shop scheduling with makespan objective", Journal of the Operational Research Society, 55(12): 1243–1255, (2004).

[6] Framinan, J. M., Leisten, R., Ruiz-Usano, R., "Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation", European Journal of Operational Research, 141(3): 559–569, (2002).

[7] Tasgetiren, M. F., Liang, Y.-C., Sevkli, M., Gencyilmaz, G., "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem", European Journal of Operational Research, 177(3): 1930–1947, (2007).

[8] Wang, X., Tang, L., "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking", Applied Soft Computing, 12(2): 652–662, (2012).

[9] Chen, C. L., Huang, S. Y., Tzeng, Y. R., Chen, C. L., "A revised discrete particle swarm optimization algorithm for permutation flow-shop scheduling problem", Soft Computing, 18(11): 2271–2282, (2014).

[10] Li, D., Deng, N., "Solving Permutation Flow Shop Scheduling Problem with a cooperative multi-swarm PSO algorithm", Journal of Information and Computational Science, 9(4): 977–987, (2012).

[11] Rajendran, C., Ziegler, H., "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs", European Journal of Operational Research, 155(2): 426–438, (2004).

[12] Ahmadizar, F., "A new ant colony algorithm for makespan minimization in permutation flow shops", Computers & Industrial Engineering, 63(2): 355–361, (2012).

[13] Ruiz, R., Stützle, T., "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem", European Journal of Operational Research, 177(3): 2033–2049, (2007).

[14] Ruiz, R., Stützle, T., "An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives", European Journal of Operational Research, 187(3): 1143–1159, (2008).

[15] Ribas, I., Companys, R., Tort-Martorell, X., "An iterated greedy algorithm for the flowshop scheduling problem with blocking", Omega, 39(3): 293–301, (2011).

[16] Minella, G., Ruiz, R., Ciavotta, M., "Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems", Computers & Operations Research, 38(11): 1521–1533, (2011).

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

[18] Varadharajan, T. K., Rajendran, C., "A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs", European Journal of Operational Research, 167(3): 772–795, (2005).

[19] Grabowski, J., Pempera, J., "The permutation flow shop problem with blocking. A tabu search approach", Omega, 35(3): 302–311, (2007).

[20] Zobolas, G. I., Tarantilis, C. D., Ioannou, G., "Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm", Computers & Operations Research, 36(4): 1249–1267, (2009).

[21] Tseng, L.-Y., Lin, Y.-T., "A hybrid genetic local search algorithm for the permutation flowshop scheduling problem", European Journal of Operational Research, 198(1): 84–92, (2009).

[22] Pasupathy, T., Rajendran, C., Suresh, R. K., "A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs", International Journal of Advanced Manufacturing Technology, 27(7–8): 804–815, (2006).

[23] Chen, S.-H., Chang, P.-C., Cheng, T. C. E., Zhang, Q., "A Self-guided Genetic Algorithm for permutation flowshop scheduling problems", Computers & Operations Research, 39(7): 1450–1457, (2012).

[24] Haq, A. N., Ramanan, T. R., Shashikant, K. S., Sridharan, R., "A hybrid neural network-genetic algorithm approach for permutation flow shop scheduling", International Journal of Production Research, 48(14): 4217–4231, (2010).

[25] Nagano, M. S., Ruiz, R., Lorena, L. A. N., "A Constructive Genetic Algorithm for permutation flowshop scheduling", Computers and Industrial Engineering, 55(1): 195–207, (2008).

[26] Rad, S. F., Ruiz, R., Boroojerdian, N., "New high performing heuristics for minimizing makespan in permutation flowshops", Omega, 37(2): 331–345, (2009).

[27] Dong, X., Huang, H., Chen, P., "An improved NEH-based heuristic for the permutation flowshop problem", Computers & Operations Research, 35(12): 3962–3968, (2008).

[28] Kalczynski, P. J., Kamburowski, J., "An improved NEH heuristic to minimize makespan in permutation flow shops", Computers & Operations Research, 35(9): 3001–3008, (2008).

[29] Vázquez-Rodríguez, J. A., Ochoa, G., "On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming", Journal of the Operational Research Society, 62(2): 381–396, (2011).

[30] Dubois-Lacoste, J., Lpez-Ibez, M., Sttzle, T., "A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems", Computers & Operations Research, 38(8): 1219–1236, (2011).

[31] Chiang, T.-C., Cheng, H.-C., Fu, L.-C., "NNMA: An effective memetic algorithm for solving multiobjective permutation flow shop scheduling problems", Expert Systems with Applications, 38(5): 5986–5999, (2011).

[32] Zheng, T., Yamashiro, M., "Solving flow shop scheduling problems by quantum differential evolutionary algorithm", The International Journal of Advanced Manufacturing Technology, 49(5–8): 643–662, (2010).

[33] Vallada, E., Ruiz, R., "Cooperative metaheuristics for the permutation flowshop scheduling problem", European Journal of Operational Research, 193(2): 365–376, (2009).

[34] Lin, S.-W., Ying, K.-C., "Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm", Computers & Operations Research, 40(6): 1625–1647, (2013).

[35] Ribas, I., Companys, R., Tort-Martorell, X., "Comparing three-step heuristics for the permutation flow shop problem", Computers & Operations Research, 37(12): 2062–2070, (2010).

[36] Laha, D., Chakraborty, U. K., "An efficient hybrid heuristic for makespan minimization in permutation flow shop scheduling", The International Journal of Advanced Manufacturing Technology, 44(5–6): 559–569, (2009).

[37] Saravanan, M., Noorul, H. A., Vivekraj, A. R., Prasad, T., "Performance evaluation of the scatter search method for permutation flowshop sequencing problems", The International Journal of Advanced Manufacturing Technology, 37(11–12): 1200–1208, (2008).

[38] Tzeng, Y.-R., Chen, C.-L., "A hybrid EDA with ACS for solving permutation flow shop scheduling", International Journal of Advanced Manufacturing Technology, 60(9–12): 1139–1147, (2012).

[39] Dasgupta, P., Das, S., "A discrete inter-species cuckoo search for flowshop scheduling problems", Computers & Operations Research, 60: 111–120, (2015).

[40] Chen, C.-L., Tzeng, Y.-R., Chen, C.-L., "A new heuristic based on local best solution for permutation flow shop scheduling", Applied Soft Computing, 29: 75–81, (2015).

[41] Moslehi, G., Khorasanian, D., "A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion", Computers & Operations Research, 52: 260–268, (2014).

[42] Rajendran, S., Rajendran, C., Leisten, R., "Heuristic rules for tie-breaking in the implementation of the NEH heuristic for permutation flow-shop scheduling", International Journal of Operational Research, 28(1): 87–97, (2017).

[43] Fernandez-Viagas, V., Framinan, J. M., "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem", Computers & Operations Research, 45: 60–67, (2014).

[44] Dubois-Lacoste, J., Pagnozzi, F., Stützle, T., "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem", Computers & Operations Research, 81: 160–166, (2017).

[45] Abdel-Basset, M., Manogaran, G., El-Shahat, D., Mirjalili, S., "A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem", Future Generation Computer Systems, 85: 129–145, (2018).

[46] Benavides, A. J., Ritt, M., "Fast heuristics for minimizing the makespan in non-permutation flow shops", Computers & Operations Research, 100: 230–243, (2018).

[47] Chen, Z., Zheng, X., Zhou, S., Liu, C., Chen, H., "Quantum-inspired ant colony optimisation algorithm for a two-stage permutation flow shop with batch processing machines", International Journal of Production Research, 58(19): 5945-5963, (2020).

[48] Kizilay, D., Tasgetiren, M. F., Pan, Q.-K., Gao, L., "A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion", Algorithms, 12: (5), (2019).

[49] Fernandez-Viagas, V., Framinan, J. M., "A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective", Computers & Operations Research, 112, (2019).

[50] Arık, O. A., "Artificial bee colony algorithm including some components of iterated greedy algorithm for permutation flow shop scheduling problems", Neural Computing and Applications, 33: 3469–3486, (2021).

[51] Gmys, J., Mezmaz, M., Melab, N., Tuyttens, D., "A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem", European Journal of Operational Research, 284(3): 814–833, (2020).

[52] Arık, O. A., "Population-based Tabu search with evolutionary strategies for permutation flow shop scheduling problems under effects of position-dependent learning and linear deterioration", Soft Computing, 25(2): 1501–1518, (2021).

[53] Taiilard, E., "Benchmarks for basic scheduling problems." http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt. Access date: 30.03.2018