Research Article

A new memetic global and local search algorithm for solving hybrid flow shop with multiprocessor task scheduling problem

Batuhan Eren Engin[1] · **Orhan Engin**[2]

Received: 28 January 2020 / Accepted: 16 November 2020 / Published online: 23 November 2020

Abstract
Hybrid flow shop (HFS) scheduling problem is combining of the flow shop and parallel machine scheduling problem. Hybrid flow shop with multiprocessor task (HFSMT) scheduling problem is a special structure of the hybrid flow shop scheduling problem. The HFSMT scheduling is a well-known NP-hard problem. In this study, a new memetic algorithm which combined the global and local search methods is proposed to solve the HFSMT scheduling problems. The developed new memetic global and local search (MGLS) algorithm consists of four operators. These are natural selection, crossover, mutation and local search methods. A preliminary test is performed to set the best values of these developed new MGLS algorithm operators for solving HFSMT scheduling problem. The best values of the MGLS algorithm operators are determined by a full factorial experimental design. The proposed new MGLS algorithm is applied the 240 HFSMT scheduling instances from the literature. The performance of the generated new MGLS algorithm is compared with the genetic algorithm (GA), parallel greedy algorithm (PGA) and efficient genetic algorithm (EGA) which are used in the previous studies to solve the same set of HFSMT scheduling benchmark instances from the literature. The results show that the proposed new MGLS algorithm provides better makespan than the other algorithms for HFSMT scheduling instances. The developed new MGLS algorithm could be applicable to practical production environment.

Keywords Hybrid flow shop scheduling · Multiprocessor task · Memetic algorithm · Local search · Makespan

1 Introduction

At the flow shop scheduling, each stage, only one machine is available and in the same order of machines, the job visits the stages of the shop [1]. But in the hybrid flow shop scheduling, at each stage, one or more identical parallel machines are available and in the same order of machines at each stage, job visits the stages of the shop and the job is processed only one machine in the parallel environment [1–4]. Gupta [5] and Hoogeveen et al. [6] showed that two-stage HFS scheduling problem is NP-hard in the strong sense even if there is only one machine on the first stage and two machines on the second stage. Arthanari and Ramamurty [7] proposed the first model of the HFS scheduling from the literature, and they developed a branch and bound method to solve the HFS scheduling problems. The HFS scheduling can be found in the different production environments such as printed circuit board and integrated circuit packaging [8] and machine-vision systems [9].

The HFSMT is a special structure of the HFS scheduling problem. In the HFSMT scheduling, a job requires one or several processors simultaneously at each stage. HFSMT scheduling is proven to be NP-hard in the strong sense [10]. In this context, there have been few attempts in the literature to solve the HFSMT scheduling to optimality by providing exact algorithms for small size instances. Since the exact algorithms may not provide solutions in

✉ Orhan Engin, orhanengin@yahoo.com | [1]Invent Analytics Co, Ankara, Turkey. [2]Department of Industrial Engineering, Faculty of Engineering and Natural Science, Konya Technical University, 42079 Konya, Turkey.

reasonable time as the computational effort increases exponentially as the problem size increase, studies in the literature dealing with HFSMT scheduling mostly focused on developing heuristics. During the last decades, some studies about the HFSMT scheduling problems are given as follows. Edwin, Ansari and Ren [11] proposed a genetic algorithm for solving the multiprocessor scheduling problems. Oğuz and Ercan [12] developed a constructive heuristic algorithm to solve the multiprocessor tasks in a two stage flow shop environment scheduling problems. Şerifoğlu and Tiryaki [13] generated a simulated annealing approach for solving multiprocessor task scheduling in multistage hybrid flow shop scheduling problems. Oğuz, Ercan, Cheng and Fung [14] proposed a constructive heuristic algorithm to solve the multiprocessor task scheduling in a two stage hybrid flow shop scheduling problems. Oğuz, Zinder et al. [15] developed a tabu search approach for solving the hybrid flow shop scheduling with multiprocessor task systems. Şerifoğlu and Ulusoy [16] generated a genetic algorithm for solving the multiprocessor task scheduling in multistage hybrid flow shop scheduling problems. Oğuz and Ercan [17] proposed a genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. Kwok and Ahmad [18] developed an optimal algorithm for solving multiprocessor task scheduling problems. Zinder, Do and Oğuz [19] showed the computational complexity of multiprocessor task scheduling problems. Coll, Ribeiro and Souza [20] studied the tasks related to precedence constraints to a set of processors for multiprocessor scheduling problems. Shenassa and Mahmoodi [21] used genetic algorithm and generated a novel intelligent method for solving task scheduling in multiprocessor scheduling problems. Ying and Lin [22] proposed an ant colony system for solving the multiprocessor task scheduling in multistage hybrid flow shops scheduling problems. Engin, Yılmaz and Ceran [23] generated a data mining to find patterns in genetic algorithm for solving hybrid flow shops scheduling with multiprocessor task problems. Kuszner and Malafiejski [24] developed a polynomial algorithm for some preemptive multiprocessor task scheduling. Hwang, Gen and Katayama [25] proposed a genetic algorithm for multiprocessor task scheduling with communications costs. Tseng and Liao [26] developed a particle swarm optimization algorithm to solve the hybrid flow shop scheduling with multiprocessor tasks problems. Cheng, Shiau, Huang and Lin [27] generated a genetic algorithm for multiprocessor tasks with resource and timing constraints. Ying [28] proposed an iterated greedy algorithm for solving the multistage hybrid flow shop with multiprocessor tasks problems. Kahraman, Engin, Kaya and Öztürk [1] present a parallel greedy algorithm (PGA) to solve the HFSMT scheduling problems. They applied the PGA by two phases iteratively, called destruction and construction. They compared their computational results with the Oğuz et al. [17]'s GA. Also Engin, Ceran and Yılmaz [10] proposed an efficient genetic algorithm (EGA) to solve HFSMT problems. They developed a new mutation operator. They compared their computational results with the Oğuz et al. [17]'s GA and Kahraman et al. [1]'s PGA. Wang, Chou and Wu [29] generated a simulated annealing algorithm to solve the hybrid flowshop scheduling with multiprocessor tasks problems. Xu, Wang, Liu and Wang [30] proposed a shuffled frog-leaping algorithm for solving the hybrid flow shop scheduling with multiprocessor tasks. Lin, Ying and Huang [31] developed an artificial bee colony algorithm for multiprocessor task scheduling in multistage hybrid flow shops. Lahimer, Lopez and Haouari [32] generated a discrepancy search and climbing algorithm for hybrid flow shop scheduling with multiprocessor tasks. Akkoyunlu, Engin and Büyüközkan [33] proposed a harmony search algorithm for solving the hybrid flow shop scheduling with multiprocessor tasks problems. Rani and Zoraida [34] developed a discrete firefly algorithm for hybrid flow shop with multistage multiprocessor task scheduling problems. Engin and Engin [35] proposed a memetic algorithm to solve HFSMT with earliness and tardiness penalties scheduling problems. They added the time windows to the jobs in hybrid flow shop with multiprocessor task scheduling problems. Kurdi [36] proposed a new social spider optimization algorithm to solve the HFSMT scheduling problems with the objective of makespan minimization. The author tested the proposed algorithm on benchmark instances and compared with other existing algorithm from the literature. Also Kurdi [37] developed an ant colony system with a novel non-daemon actions procedure algorithm to solve multiprocessor task scheduling hybrid flow shop scheduling problems with the objective of makespan minimization.

In this paper, a new memetic global and local search algorithm is developed to solve HFSMT scheduling problems. The effectiveness of the proposed method is tested with the Oğuz [38] benchmark instances from the literature. The computational results are compared with the earlier studies of Oğuz et al. [17]'s GA and Kahraman et al. [1]'s PGA and Engin et al. [10]'s EGA.

Currently, there is no reported paper on the memetic global and local search algorithm to solve the hybrid flow shop with multiprocessor task scheduling problems. The contributions of this study are summarized as follows:

- The proposed new memetic global and local search algorithm is first used to solve the HFSMT scheduling problems.
- A full factorial design is done to determine the best parameter sets for the MGLS to solve the HFSMT scheduling problems.

- The 240 HFSMT instances from the literature are solved the proposed MGLS algorithm.
- The new best makespan values are found for HFSMT benchmark instances from the literature.
- The developed MGLS algorithm is analyzed. The computational results of the MGLS algorithm are compared with the earlier studies of Oğuz et al. [17]'s GA and Kahraman et al. [1]'s PGA and Engin et al. [10]'s EGA.

The results showed that the proposed MGLS provides better makespan than GA, PGA and EGA.

The paper organized as follows: Sect. 2 gives information about the HFSMT scheduling, Sect. 3 provides the MGLS algorithm, Sect. 4 contains computational results and Sect. 5 presents the conclusion and future research.

2 Hybrid flow shop with multiprocessor task scheduling

HFSMT contains a set of *n* jobs with the *k-stage* flow shop scheduling. At stage (*i*), identical parallel processors (m_i) are available. The number of machines which the jobs require at each stage is indicated with $size_{ij}$. The HFSMT scheduling problem dealt in this study is denoted as $F_k(P_{m1},\ldots,P_{mk})|size_{ij}|C_{max}$.

The assumptions in the HFSMT scheduling problem are given as follows [1, 10].

- The number of jobs, stages, machines at each stage are known and fixed,
- All processors, machines and jobs are available through the planning horizon,
- The processing time of each job on each stage is given before,
- The processing times are included the setup times of the jobs,
- The objective is minimizing the makespan values.

The following mathematical model is developed by Lin and Zhang [8]. Let's denote the following variables [35]:

Denote the following variables:

n, number of jobs to be sequenced.

s, number of stages in the shop.

m_i, number of machines at stage i.

p_{ij}, processing time of job j on machine i.

$size_{ij}$, number of processors required to process.

PH, planning horizon.

Let's denote the following decision variables:

SPT_{ij}, starting processing time of job j at stage i.

C_{ij}, completion time of job j at stage i.

C_{max}, makespan

$$x_{i,j,t} = \begin{cases} 1 \text{ if job } j \text{ is processed at stage } i \text{ in } t \text{ timeperiod} \\ 0 \text{ otherwise} \end{cases}$$

The mathematical formulation is given as follows [39]:

$$\min z = C_{\max} \tag{1}$$

s.t.:

$$\sum_{j=1}^{n} X_{ijt} size_{ij} \le m_i, i = 1,..,s, t = 1,\ldots,PH \tag{2}$$

$$C_{i-1,j} \le C_{ij} - p_{ij}, i = 2,..,s, j = 1,\ldots,n \tag{3}$$

$$C_{ij} - S_{ij} + 1 = p_{ij}, i = 1,..,s, j = 1,\ldots,n \tag{4}$$

$$\sum_{t=1}^{PH} X_{ijt} = p_{ij}, i = 1,..,s, j = 1,\ldots,n \tag{5}$$

$$SPT_{ij} \le t + PH\left(1 - X_{ijt}\right), i = 1,..,s, j = 1,..,n, t = 1,\ldots,PH \tag{6}$$

$$tX_{ijt} \le SPT_{ij} + p_{ij} - 1, i = 1,\ldots,s, j = 1,\ldots,n, t = 1,\ldots,PH \tag{7}$$

$$C_{max} \ge C_{sj}, j = 1,\ldots,n \tag{8}$$

$$X_{ijt} \in \{0,1\} \tag{9}$$

$$SPT_{ij} \in \{1,\ldots,PH\}, i = 1,\ldots,s, j = 1,\ldots,n, t = 1,\ldots,PH \tag{10}$$

$$C_{i0} = 0, i = 1,\ldots,s \tag{11}$$

$$C_{ij} \in \{1,\ldots,PH\}, i = 1,\ldots,s, j = 1,\ldots,n, t = 1,\ldots,PH \tag{12}$$

The objective function (1) is minimized the makespan. Constraint (2) defines the maximum number of machines at each stage; Constraint (3) determines a task of job to start. Constraint (4) defines the starting time; Constraint (5) denotes the time occupation of each job at each stage; Constraint (6) and Constraint (7) are denoted the required number of processors; Constraint (8) calculates the C_{max}. Finally, constraints (9)–(12) denote the decision variables.

3 Memetic global and local search algorithm

Combinatorial optimization problems are frequently encountered in industrial areas such as assignment, job employee scheduling, routing, network design and other areas of industrial, economic and scientific importance. Methods used to solve these problems are generally classified into three different types, i.e., exact, approximation and heuristic algorithms. Exact methods guarantee to find optimal solution to the problem in hand, but the computation time significantly increases depending on the input size. Using heuristics in solving large size problems aims finding good solution in reasonable time.

Memetic algorithm (MA) was first addressed in [40]. He defined MAs as the population-based heuristics coupled with individual learning and local improvement. Ong and Keane [41] state that MAs can balance well between exploration (searching promising areas in search space) and exploitation (searching neighborhoods of individuals in hope of finding better results). It is suggested in [42] that MAs are the extension of GAs developed to further improve fitness of offspring in an attempt to reduce premature convergence likelihood. Burke [43] suggests that the main benefit of using MA is the reduction of feasible solution space into subspaces of local optima. They also added that the addition of local improvement methods to genetic algorithm may increase the computational effort, but this can be justified with the reduction in search space to reach good solutions.

The proposed new MGLS algorithm is combined the global and local search methods. The proposed new MGLS algorithm used four operators. These are natural selection, crossover, mutation and local operators.

During the last year, memetic algorithm used some scheduling problems in the literature: for parallel machine scheduling [44], flow shop [45–47] job shop [48] open shop [49] staff shop [50].

Genetic operators and the local search method are briefly explained below.

3.1 Initial population

The initial population randomly generated individuals to ensure a good exploration in solution space. The initial population size is kept constant through each generation. The proposed MGLS algorithm is started with an initial solution that generated by Nawaz, Enscore and Ham [51] heuristic algorithm solution.

3.2 The Nawaz, Enscore and Ham (NEH) heuristic

NEH heuristic is known to perform well for the permutation flow shop under the makespan minimization objective. It is based on the idea that the jobs with higher total processing times at each stage on all machines should be scheduled as early as possible to reduce the makespan. The steps into NEH heuristic are given as follows:

Step 1 Calculate the total processing times of all jobs at each stage.

Step 2 Sort the jobs in descending order of their total processing times.

Step 3 Take job j, $j = 1,\ldots, n$ and find the best schedule by placing it in all the possible jth positions among the sequence of jobs that are already scheduled.

3.3 Encoding of solutions

Encoding scheme used to represent solutions in memetic algorithms is very crucial step in designing memetic based meta-heuristics, which affects the performance of algorithms. By definition, the information of sequence of n jobs, at each stage is needed to be carried by each individual. This type of encoding scheme is known as permutation encoding and used very frequently in job scheduling problems. In this paper, encoding schema is used.

3.4 Decoding of solutions

Studies in the literature dealing with HFSMT scheduling used different types of decoding methods. One method adopts the permutation at the first stage, and uses the same permutation for the remaining stages. This method is known as permutation scheduling. Another method, known as list scheduling (LS) method, decodes each chromosome right after a stage to a full schedule and re-sequences the jobs for the stage in non-decreasing order of their completion times at previous stage [17, 29, 31]. A formal description of algorithm LS is given in [17]. In this paper, a list scheduling decoding method is used.

Example An example is given below about the working of decoding method. In this example, 8 jobs are considered with two stages and four processors at both stages. For each jobs, the processing times (pt) and number of parallel machines (pm) are given in Table 1.

For a permutation ($\pi_1 = 5, 8, 1, 7, 2, 3, 6, 4$), the Gantt chart is given in Fig. 1.

Table 1 The processing times for jobs

Jobs	J1	J2	J3	J4	J5	J6	J7	J8
pt_{1j}	1	1	4	2	3	3	4	3
pm_{1j}	2	2	3	3	3	2	3	1
pt_{2j}	2	1	2	1	3	4	4	2
pm_{2j}	4	1	1	2	4	1	4	3

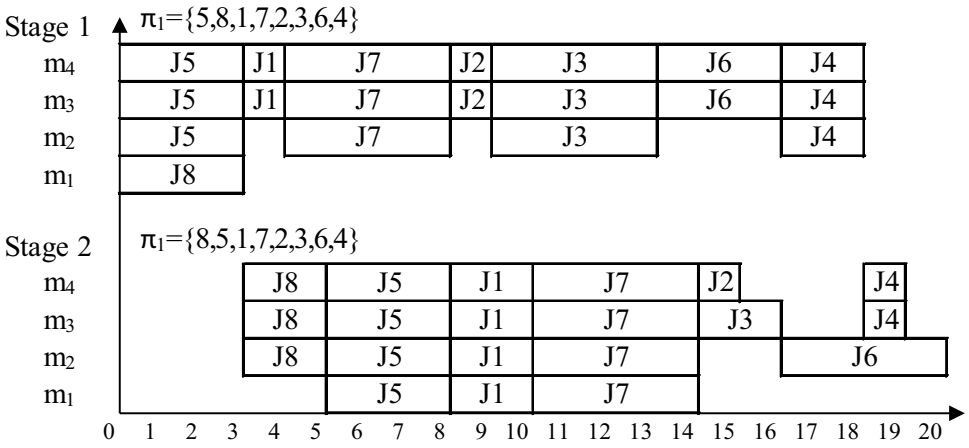

Fig. 1 The schedule of the permutation π_1

Permutation of jobs at stage 1 is determined by a metaheuristic, i.e., the memetic global and local search algorithm in this case. List scheduling method decodes each individual, i.e., the information of permutation of jobs, right after all the jobs are completed at each stage. Here, the list scheduling assigns each job, one by one ($\pi_1 = 5, 8, 1, 7, 2, 3, 6, 4$), to the processors available, starting from the beginning of time. It must be addressed that a job cannot start to be processed before its preceding jobs in the permutation, yet they can start at the same time if there is enough number of idle processors. For instance, job 2 might have started at the same time with the job 1 at stage 1, but it cannot be allowed to do so as it would yield the same results with the permutation of ($\pi^* = 5, 8, 1, 2, 7, 3, 6, 4$). This type of allowance would hamper the algorithm's performance, because many different permutations referring to the same solution might occur.

3.5 Fitness function

Fitness function is a type of objective function that is used as single figure of merit, in memetic algorithms. In this paper, the fitness of an individual carrying the information of permutation of n jobs is defined by C_{max}.

3.6 Selection method

Individuals from the current population are selected based on their fitness value through the tournament selection method in order to generate offspring by crossover operator. Steps to tournament selection are listed below [42]:

Step 1 Define the size of the reproduction pool,

Step 2 Select two individuals from the main population with replacement,

Step 3 Choose the best one from these selected two and keep it in the mating pool,

Step 4 Repeat the *Steps* 1 and 2 until the reproduction pool is full of individuals selected.

3.7 Crossover methods

Crossover is the main operator that enables the memetic algorithms to search different parts of the solution area by mixing the features of individuals. Parents might produce offspring or not within the probability called crossover rate (c). Since mating pool may contain more than two potential parents, in each step, two parents are selected by the same selection method, i.e., tournament selection, which is used to form the mating pool. Also, since individuals are represented as permutation schedule, crossover operator is carried out considering the fact that a job number cannot be repeated in a chromosome. Three different crossover methods coded in order to decide which one gives better outcomes for the problem in hand which

are Position Based Crossover (PBX), Order Crossover (OX) and New Crossover Operator (NXO), introduced by [10, 17].

3.7.1 PBX

The PBX proposed by Syswerda [52]. The steps of the PBX are summarized as follows:

Step 1 Select a subset of positions from the parent A,

Step 2 Genes at these positions are copied to the offspring with the same positions,

Step 3 The other positions are filled with remaining genes with the same relative order as in the parent B.

3.7.2 OX

The OX proposed by Davis [53]. The steps of the OX are summarized as follows:

Step 1 Choose a swath consecutive alleles at random from parent A,

Step 2 For offspring 1, copy the swath and fill the blanks with the genes in the same order that they appear in parent B,

Step 3 For offspring 2, copy the swath and fill the blanks with the genes in the same order that they appear in parent A

3.7.3 NXO

NXO is developed by Oğuz and Ercan [17] for HFSMT problems. The details of the NXO can be seen in Oğuz and Ercan [17]'s research.

3.8 Mutation operators

Mutation operator is essential to offer the opportunity to escape from trapping the local optima and thus to increase variability and diversity in the population to search different part of the solution area, by mutating genes in chromosomes with several different methods from the individuals randomly chosen. A randomly chosen chromosome might be exposed to mutation within the probability called mutation rate (m). The best parameter set for the crossover and mutation rate is determined with experimental design. Most frequently used mutation operators in the literature for the permutation encoding would be inversion mutation and arbitrary three genes change mutation.

3.9 Inversion mutation

General steps for inversion mutation can be given as follows:

Steps 1 Choose a random individual to make it exposed to mutation,

Steps 2 Select two positions at random, at reverse the sequence between them,

Steps 3 Include the mutated individual to the population, regardless of whether it has better or worse value of objective function than its version before the mutation.

3.10 Arbitrary three genes change mutation

General steps for "arbitrary three genes change mutation" can be given as follows:

Steps 1 Choose a random individual to make it exposed to mutation,

Steps 2 Select three positions at random, at exchange their positions,

Steps 3 Include the mutated individual to the population, regardless of whether it has better or worse value of objective function than its version before the mutation.

3.11 Local search

Local search procedure integrated in MGLS algorithm uses the idea of neighborhood that searches the neighbors around the best solution that can be reached within one move, which is a systematic change of jobs position in strings called insertion. By using insertion local search applied to the best known solution in iteration, all schedules are obtained by removing a job at the ith position and inserting it in the jth position where i and j are not equal.

3.12 Preventing convergence

One of the major difficulties in solving scheduling problems is the chance of premature convergence to local optima. At some point, most of the individuals in population may have the same schedule and objective function, leading to a clone offspring generated in crossover to its parents, therefore causing inefficient search and decrease in quality of the solutions found. To prevent this, we implement the random offspring generation (ROG) before the crossover operation under a certain circumstance. The idea behind the ROG method is that if both parents are

the same, instead of generating an individual which would be clone to its parents, a random solution on the problems domain is generated. This method is provided by Rocha and Neves [54]. Also, a re-born strategy is added to the algorithm which ensures that the population is restarted if there is no improvement in the best found solution so far for the predetermined consecutive number of generation, i.e., 250 generations in this case, while preserving only the best solution found in the hope of exploration in different solution area.

3.12.1 Stopping condition

The algorithm stops searching if it reaches a certain CPU time spent for each problem, that is 1800s, the same with Engin et al. [10], or if a situation occurs in which the best known solution found so far cannot be improved in 250 consecutive iterations. The comparisons of stopping conditions in compared studies are given in Table 2.

The steps of the proposed MGLS algorithm are given below:

Step 1 Adopt the parameter sizes of MGLS algorithm; such as population size, maximum iteration number, the mutation and crossover rate are determined by full factorial experimental design,

Step 2 Generate initial population consisting of random individual, including one individual generated by NEH algorithm; evaluate fitness of each individual in the population.

Step 3 Check the algorithm stopping criteria. If the condition holds, stop the algorithm and report the best solution and fitness values. Otherwise, proceed to Step 4,

Step 4 Select parents by using tournament selection method,

Step 5 Generate new offspring with crossover operator and apply mutation procedure to the generated offspring,

Step 6 Perform local search using the best known solution,

Table 2 The CPU times of the comparative algorithms

Methods	Configuration of a system	Limit of CPU time(min)
EGA	Pentium IV, 3.00 GHz, 512 K	25.00
PGA	Pentium IV, 3.00 GHz, 512 K	25.00
GA	Pentium IV, 3.00 GHz, 256 K	30.00
MGLS	Pentium IV,1.70 GHz, 1024 K	30.00

Step 7 If the stopping criteria do not hold, go to *Step 2*; otherwise, stop the algorithm.

4 Computational results

In this study, Oğuz [38]'s 240 HFSMT scheduling benchmark instances are used. These instances are divided into two types: type *P* and *Q*. The instances are combinations of *n-job* and *k-stage* ($n = 10, 20, 50, 100$; $k = 2, 5, 8$), and each combination contains 10 problems.

For type *P* and *Q* problems, the numbers of processors at each stage are uniformly distributed ($U(1, 5)$). This type of problems was proven to be NP-hard even with 2 stages by Oğuz and Ercan [17]. These problems are decoded as "problem type_number of jobs_number of stages_problem index." There are 10 instances in each category, in which the number of jobs and the number of stages are the same, therefore 240 instances in hand. The proposed MGLS algorithm is coded in C# language on the Visual Studio environment.

4.1 Lower bound

The benchmark instances lower bounds are calculated by Oğuz et al. [15] through the formula (13) below:

$$LB = \max_{i \in M} \left\{ \min_{j \in J} \left\{ \sum_{k=1}^{i-1} p_{kj} \right\} + \frac{1}{m_i} \sum_{j \in J} p_{ij} size_{ij} + \min_{j \in J} \left\{ \sum_{k=i+1}^{m} p_{kj} \right\} \right\} \tag{13}$$

where *j* and *m* denote the set of jobs and stages, respectively, p_{kj} denotes the processing time of job *J* at stage *k*, and $size_{ij}$ denotes the number of parallel machines for job *j* at stage *i*. The performance of MGLS indicating how much the results found by MGLS deviates from the lower bound will be assessed by relative percentage deviation (RPD). The RPD is calculated with Eq. 14.

$$RPD(L) = \frac{BestC_{max}(L) - Lowerbound(L)}{Lowerbound(L)} \times 100 \tag{14}$$

4.2 Parameter test of MGLS algorithm

The performance of the proposed MGLS algorithm is dependent on the selected parameter sets. These parameter sets are crossover and mutation rate, population and iteration size and parent selection rate. Therefore, a full factorial experimental design (FFED) is carried out to determine the best parameter set for each instance, solving a predetermined problem with each possible parameter combination. Table 3 shows the parameter choices.

Table 3 Possible parameters of MGLS algorithm

Parameters	Possible values
Population size	20; 50
Crossover rate	0.8; 0.9
Mutation rate	0.1; 0.2
Parent selection rate	0.1; 0.2
Iteration size	500; 1000
Crossover methods	PBX, OX, NXO
Mutation methods	Reverse, three genes swap

For each problem category, an instance is chosen out of 10 instances solved by using every possible parameter combination. The results are given in Table 4.

Table 4 indicates that NXO clearly outperforms the other crossover methods for most of the problem instances. The best parameter sets in Table 4 are generalized for the rest of the instances belonging to the same instance class.

4.3 Performance comparison of MGLS algorithm with GA, EGA and PGA

Table 5 shows the comparison of MGLS algorithm to the genetic algorithm by Oğuz and Ercan [17], parallel greedy approach (PGA) by Kahraman et al. [1] and efficient genetic algorithm (EGA) by Engin et al. [10] in terms of average relative percentage deviation (ARPD) from the lower bounds. The ARPD is evaluated with (15). The number of instances for each problem group is defined as I ($L = 1,\ldots,I$) at 15 [55].

$$ARPD = \frac{\sum_{L=1}^{I} RPD(L)}{I} \tag{15}$$

Table 6 shows that, in terms of the number of better solutions found, MGLS outperforms GA developed by Oğuz and Ercan [17] and Parallel Greedy algorithm (PGA) developed by [1].

Most importantly, in terms of the ARPD, developed MGLS outperforms GA, PGA and EGA, as it can be seen in Table 7.

The results given in Table 7 show that the proposed MGLS algorithm outperforms than GA, PGA and EGA in

Table 4 The result of FFED

Problem	Pop. size	Iteration	Selection ratio	Crossover method	Crossover ratio	Mutation method	Mutation ratio	RPD
P10-S2-T05	20	500	0.2	NXO	0.9	Three genes	0.1	0.000
P20-S2-T05	20	500	0.2	NXO	0.9	Reverse	0.1	0.000
P50-S2-T05	50	1000	0.1	NXO	0.9	Three genes	0.2	0.646
PH1-S2-T05	50	1000	0.1	NXO	0.9	Three genes	0.2	0.000
P10-S5-T05	20	500	0.2	NXO	0.9	Reverse	0.1	3.684
P20-S5-T05	50	500	0.2	NXO	0.9	Three genes	0.1	1.893
P50-S5-T05	50	1000	0.1	NXO	0.9	Three genes	0.2	0.000
PH1-S5-T05	50	1000	0.1	NXO	0.9	Three genes	0.2	0.000
P10-S8-T05	20	500	0.2	NXO	0.9	Reverse	0.2	4.212
P20-S8-T05	50	1000	0.1	NXO	0,9	Three genes	0.2	2.641
P50-S8-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	0.872
PH1-S8-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	0.000
Q10-S2-T05	20	500	0.1	NXO	0.9	Reverse	0.1	19.209
Q20-S2-T05	50	1000	0.1	NXO	0.9	Reverse	0.1	1.547
Q50-S2-T05	50	1000	0.1	NXO	0,9	Reverse	0.1	0.000
QH1-S2-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	0.029
Q10-S5-T05	20	500	0.2	NXO	0.9	Reverse	0.1	2.574
Q20-S5-T05	50	1000	0.2	NXO	0.9	Reverse	0.1	− 3.397
Q50-S5-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	13.528
QH1-S5-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	13.329
Q10-S8-T05	50	500	0.1	NXO	0.9	Reverse	0.2	19.635
Q20-S8-T05	50	1000	0.2	NXO	0,9	Reverse	0.2	20.510
Q50-S8-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	11.444
QH1-S8-T05	50	1000	0.1	NXO	0.9	Reverse	0.2	12.120

Table 5 The comparison of GA, PGA, EGA and MGLS algorithm in terms of RPD

Problem	RPD				Problem	RPD			
	GA	PGA	EGA	MGLS		GA	PGA	EGA	MGLS
P10-S2-T01	0.000	0.000	0.000	0.000	P10-S5-T01	4.53	4.878	4.53	5.052
P10-S2-T02	0.000	0.000	0.000	0.000	P10-S5-T02	6.022	6.022	6.022	6.022
P10-S2-T03	5.96	5.96	5.96	5.96	P10-S5-T03	7.54	7.54	7.54	9.695
P10-S2-T04	0.000	0.000	0.000	0.000	P10-S5-T04	8.609	11.755	11.755	12.583
P10-S2-T05	0.000	0.000	0.000	0.000	P10-S5-T05	3.684	3.684	3.684	3.684
P10-S2-T06	0.000	0.000	0.000	0.000	P10-S5-T06	0.000	0.000	0.000	0.000
P10-S2-T07	0.000	0.000	0.000	0.000	P10-S5-T07	8.21	8.21	8.21	8.21
P10-S2-T08	0.000	0.000	0.000	0.000	P10-S5-T08	2.742	2.258	2.258	2.419
P10-S2-T09	0.000	0.000	0.000	0.000	P10-S5-T09	9.076	9.076	9.076	10.191
P10-S2-T10	0.000	0.000	0.000	0.000	P10-S5-T10	10.417	12.179	12.66	12.179
P20-S2-T01	1.997	1.69	2.611	2.611	P20-S5-T01	1.567	1.567	1.567	1.567
P20-S2-T02	2.368	1.275	1.639	1.639	P20-S5-T02	7.165	6.858	7.165	7.165
P20-S2-T03	0.000	0.000	0.000	0.000	P20-S5-T03	2.561	6.17	2.561	2.561
P20-S2-T04	0.000	0.000	0.000	0.000	P20-S5-T04	2.268	0.113	0.000	0.000
P20-S2-T05	0.000	0.000	0.000	0.000	P20-S5-T05	2.313	3.049	1.682	1.893
P20-S2-T06	0.000	0.000	0.000	0.000	P20-S5-T06	2.65	2.479	1.88	2.137
P20-S2-T07	0.341	0.341	0.341	0.341	P20-S5-T07	0.000	0.000	0.000	0.000
P20-S2-T08	0.278	0.278	0.278	0.278	P20-S5-T08	3.519	3.519	3.128	3.128
P20-S2-T09	0.000	0.000	0.000	0.000	P20-S5-T09	0.000	0.000	0.000	0.000
P20-S2-T10	0.000	0.000	0.000	0.000	P20-S5-T10	7.165	7.165	6.858	7.165
P50-S2-T01	1.753	1.088	0.605	0.665	P50-S5-T01	0.911	0.835	0.911	0.911
P50-S2-T02	0.000	0.000	0.000	0.000	P50-S5-T02	4.539	0.532	1.099	0.39
P50-S2-T03	0.59	0.215	0.000	0.000	P50-S5-T03	0.998	0.998	0.998	0.998
P50-S2-T04	1.238	1.423	0.681	0.557	P50-S5-T04	0.618	4.325	2.523	2.781
P50-S2-T05	0.353	0.588	0.235	0.235	P50-S5-T05	2.577	0.000	0.000	0.000
P50-S2-T06	0.000	0.000	0.000	0.000	P50-S5-T06	0.000	0.000	0.000	0.000
P50-S2-T07	2.716	2.13	0.692	2.503	P50-S5-T07	1.100	0.477	0.513	0.33
P50-S2-T08	0.000	0.000	0.000	0.000	P50-S5-T08	2.447	0.745	1.099	1.099
P50-S2-T09	0.000	0.000	0.000	0.000	P50-S5-T09	5.096	0.668	0.000	0.209
P50-S2-T10	0.262	0.052	0.052	0.052	P50-S5-T10	0.271	1.264	0.587	0.316
P1H-S2-T01	0.534	0.507	0.721	0.107	P1H-S5-T01	3.493	0.056	0.000	0.000
P1H-S2-T02	0.000	0.000	0.000	0.000	P1H-S5-T02	1.543	0.000	0.000	0.000
P1H-S2-T03	0.814	0.603	0.693	0.814	P1H-S5-T03	3.819	2.272	0.674	1.248
P1H-S2-T04	0.000	0.000	0.000	0.000	P1H-S5-T04	1.425	1.548	1.13	1.376
P1H-S2-T05	0.000	0.000	0.000	0.000	P1H-S5-T05	2.347	0.000	0.000	0.000
P1H-S2-T06	0.000	0.000	0.000	0.000	P1H-S5-T06	3.591	4.488	2.924	3.078
P1H-S2-T07	0.926	0.538	0.657	0.837	P1H-S5T07	0.300	0.000	0.000	0.000
P1H-S2-T08	0.02	0.000	0.000	0.000	P1H-S5-T08	0.673	0.000	0.000	0.000
P1H-S2-T09	0.105	0.394	0.683	0.499	P1H-S5-T09	1.379	0.000	0.000	0.000
P1H-S2-T10	0.097	0.242	0.097	0.000	P1H-S5-T10	3.248	3.389	2.994	3.417
P10-S8-T01	21.268	23.662	21.268	25.634	Q10-S2-T01	0.000	0.000	0.000	0.000
P10-S8-T02	26.179	28.455	31.87	28.455	Q10-S2-T02	5.012	5.012	5.012	5.012
P10-S8-T03	20.027	21.789	20.027	22.869	Q10-S2-T03	9.859	0.000	0.000	− 23.944
P10-S8-T04	13.091	14.038	17.35	14.038	Q10-S2-T04	0.000	0.000	0.000	0.000
P10-S8-T05	1.902	4.212	4.212	5.707	Q10-S2-T05	0.000	19.209	0.000	19.209
P10-S8-T06	0.409	0.409	0.409	0.409	Q10-S2-T06	5.817	5.817	5.817	6.094
P10-S8-T07	24.757	26.214	30.097	26.214	Q10-S2-T07	0.286	0.286	0.286	0.571
P10-S8-T08	19.479	21.933	19.479	18.865	Q10-S2-T08	12.333	93.33	93.333	93.333
P10-S8-T09	0.83	2.075	2.075	2.075	Q10-S2-T09	4.076	4.076	4.076	6.522

Table 5 (continued)

Problem	RPD				Problem	RPD			
	GA	PGA	EGA	MGLS		GA	PGA	EGA	MGLS
P10-S8-T10	19.589	20	19.589	20.274	Q10-S2-T10	7.331	7.331	7.331	7.331
P20-S8-T01	8.163	9.448	9.599	4.913	Q20-S2-T01	7.193	7.193	7.193	7.193
P20-S8-T02	0.000	0.000	0.000	0.000	Q20-S2-T02	0.892	35.414	35.414	35.414
P20-S8-T03	2.573	4.117	2.573	2.573	Q20-S2-T03	4.69	46.529	4.69	46.529
P20-S8-T04	1.593	3.805	7.08	3.628	Q20-S2-T04	0.000	0.000	0.000	0.000
P20-S8-T05	3.152	2.641	2.300	2.641	Q20-S2-T05	0.844	0.703	1.547	1.547
P20-S8-T06	8.163	9.675	4.913	5.291	Q20-S2-T06	0.426	0.426	1.42	0.426
P20-S8-T07	23.795	27.487	28.821	26.974	Q20-S2-T07	0.000	0.000	0.000	0.000
P20-S8-T08	3.791	2.729	3.791	3.791	Q20-S2-T08	0.369	0.369	0.369	0.369
P20-S8-T09	0.000	0.000	0.000	0.000	Q20-S2-T09	0.000	0.000	0.000	0.000
P20-S8-T10	4.117	4.803	3.945	4.031	Q20-S2-T10	1.826	1.674	1.826	1.674
P50-S8-T01	2.709	6.578	5.288	2.88	Q50-S2-T01	0.161	0.161	0.161	0.161
P50-S8-T02	2.75	2.75	0.000	0.000	Q50-S2-T02	0.329	0.329	0.329	0.329
P50-S8-T03	0.936	1.161	0.787	0.075	Q50-S2-T03	1.651	4.068	4.068	4.068
P50-S8-T04	4.76	3.917	2.696	2.148	Q50-S2-T04	0.623	0.187	0.249	0.249
P50-S8-T05	3.639	2.691	0.986	0.796	Q50-S2-T05	0.000	0.000	0.000	0.000
P50-S8-T06	1.875	1.641	0.313	0.000	Q50-S2-T06	2.353	21.799	22.215	21.869
P50-S8-T07	5.436	3.701	2.544	2.12	Q50-S2-T07	0.000	0.000	0.000	0.000
P50-S8-T08	2.434	1.917	1.881	0.221	Q50-S2-T08	3.292	7.081	7.267	7.267
P50-S8-T09	4.76	4.465	3.412	3.665	Q50-S2-T09	2.747	2.816	2.129	2.266
P50-S8-T10	5.371	4.726	3.187	2.363	Q50-S2-T10	2.567	9.949	10.334	10.847
P1H-S8-T01	2.877	1.509	0.000	0.664	Q1H-S2-T01	2.25	2.179	2.214	2.286
P1H-S8-T02	3.568	0.801	0.157	0.157	Q1H-S2-T02	1.904	1.666	1.36	2.278
P1H-S8-T03	1.987	1.472	0.644	0.405	Q1H-S2-T03	3.849	3.952	4.364	3.849
P1H-S8-T04	2.149	0.498	0.103	0.103	Q1H-S2-T04	0.000	5.008	0.000	5.756
P1H-S8-T05	1.944	0.731	0.019	0.000	Q1H-S2-T05	0.029	0.029	0.029	0.029
P1H-S8-T06	3.422	2.907	1.104	1.27	Q1H-S2-T06	0.911	1.058	0.705	2.615
P1H-S8-T07	2.426	1.203	0.019	0.000	Q1H-S2-T07	1.891	2.247	1.891	2.11
P1H-S8-T08	7.294	8.451	5.584	7.193	Q1H-S2-T08	0.118	0.355	0.118	0.591
P1H-S8-T09	1.951	1.334	0.02	0.319	Q1H-S2-T09	1.642	0.000	0.000	− 4.58
P1H-S8-T10	3.838	0.985	0.185	0.185	Q1H-S2-T10	1.885	1.885	1.778	2.347
Q10-S5-T01	9.291	9.291	9.291	10.135	Q10-S8-T01	15.86	25.134	25.134	26.344
Q10-S5-T02	1.536	1.877	2.56	2.56	Q10-S8-T02	7.57	2.789	2.789	2.258
Q10-S5-T03	5.664	7.434	9.027	7.434	Q10-S8-T03	8.005	7.349	7.349	6.168
Q10-S5-T04	6.919	14.992	14.827	14.333	Q10-S8-T04	2.289	2.169	2.169	4.096
Q10-S5-T05	2.574	2.574	2.574	2.757	Q10-S8-T05	19.635	20.7	19.635	21.005
Q10-S5-T06	8.155	10.485	10.485	10.874	Q10-S8-T06	13.018	17.456	17.456	18.195
Q10-S5-T07	15.146	15.146	15.146	15.146	Q10-S8-T07	14.944	32.432	32.432	31.002
Q10-S5-T08	3.811	0.000	0.000	− 18.598	Q10-S8-T08	13.58	14.938	13.58	16.049
Q10-S5-T09	0.000	0.000	0.000	− 14.732	Q10-S8-T09	18.053	19.937	18.053	20.879
Q10-S5-T10	13.163	13.163	13.163	14.145	Q10-S8-T10	7.261	4.331	4.331	5.86
Q20-S5-T01	3.533	2.257	2.257	2.257	Q20-S8-T01	11.168	10.656	11.168	10.553
Q20-S5-T02	7.970	7.970	7.846	7.846	Q20-S8-T02	18.653	21.634	24.283	21.192
Q20-S5-T03	13.626	13.626	12.915	12.915	Q20-S8-T03	10.169	10.829	7.439	9.04
Q20-S5-T04	4.277	9.364	9.017	11.445	Q20-S8-T04	9.436	9.347	8.201	5.732
Q20-S5-T05	13.907	0.000	0.000	-3.397	Q20-S8-T05	18.469	19.898	21.327	19.592
Q20-S5-T06	7.093	10.93	12.093	11.395	Q20-S8-T06	11.168	10.143	10.041	9.836
Q20-S5-T07	4.580	4.580	4.580	4.580	Q20-S8-T07	40.285	24.808	21.625	21.734
Q20-S5-T08	11.578	11.323	7.761	10.433	Q20-S8-T08	21.613	22.366	21.613	19.462

Table 5 (continued)

Problem	RPD				Problem	RPD			
	GA	PGA	EGA	MGLS		GA	PGA	EGA	MGLS
Q20-S5-T09	1.647	3.074	1.207	1.427	Q20-S8-T09	20.128	18.522	20.128	20.771
Q20-S5-T10	5.345	5.122	4.12	5.011	Q20-S8-T10	14.924	13.909	13.807	13.198
Q50-S5-T01	15.558	13.707	11.163	11.278	Q50-S8-T01	20.491	19.151	18.705	20.268
Q50-S5-T02	9.406	7.839	8.567	7.167	Q50-S8-T02	14.607	15.236	14.607	13.874
Q50-S5-T03	10.042	0.000	0.000	– 2.208	Q50-S8-T03	9.231	1.921	0.585	0.877
Q50-S5-T04	11.83	0.100	0.000	– 2.607	Q50-S8-T04	24.14	22.742	19.677	18.172
Q50-S5-T05	17.784	15.685	13.644	13.528	Q50-S8-T05	22.415	14.016	11.916	11.444
Q50-S5-T06	12.231	11.665	9.966	10.702	Q50-S8-T06	13.731	13.068	10.985	13.258
Q50-S5-T07	8.294	3.622	1.417	2.205	Q50-S8-T07	9.445	4.035	0.688	0.275
Q50-S5-T08	11.68	0.359	0.41	-2.51	Q50-S8-T08	24.389	31.505	32.048	31.233
Q50-S5-T09	2.479	0.000	0.000	-8.77	Q50-S8-T09	14.828	16.624	12.827	14.828
Q50-S5-T10	2.723	1.733	0.05	-0.198	Q50-S8-T10	15.652	8.807	7.851	5.435
Q1H-S5-T01	18.948	10.089	7.813	8.705	Q1H-S8-T01	10.581	9.298	6.784	9.56
Q1H-S5-T02	15.451	0.000	0.000	-10.335	Q1H-S8-T02	8.463	6.153	2.414	3.946
Q1H-S5-T03	16.207	2.502	0.227	1.024	Q1H-S8-T03	14.856	14.691	9.711	10.426
Q1H-S5-T04	18.948	11.75	8.213	8.982	Q1H-S8-T04	16.203	12.035	7.787	8.102
Q1H-S5-T05	24.866	13.658	10.729	12.971	Q1H-S8-T05	15.998	15.157	9.87	12.12
Q1H-S5-T06	17.349	7.61	4.855	6.11	Q1H-S8-T06	16.969	16.601	13.994	15.467
Q1H-S5-T07	24.796	9.139	7.974	7.421	Q1H-S8-T07	15.692	13.286	8.967	9.267
Q1H-S5-T08	18.322	4.895	2.657	3.105	Q1H-S8-T08	15.458	17.952	15.197	16.009
Q1H-S5-T09	21.104	3.276	0.443	0.561	Q1H-S8-T09	11.234	2.808	0.000	– 0.426
Q1H-S5-T10	22.71	8.952	6.609	6.879	Q1H-S8-T10	9.151	9.71	5.693	9.178

terms of the ARPD. On average, MGLS deviates 5.610% from lower bounds for all problem set, while GA, PGA and EGA deviate 6.450%, 6.469% and 5.643% from lower bounds, respectively. Table 7 shows that the proposed MGLS algorithm achieves best results in terms of ARPD for the problem sets P20S8, P50S5, P50S8, P1HS2, Q10S5, Q20S8, Q50S5, Q50S8 and Q1HS5. This result showed that the implementation of NEH heuristic and local search approach in MGLS algorithm increased the chance of finding good solutions.

Also, MGLS algorithm improved the best known solutions so far for 48 instances out of 240 and also found 12 solutions which improve LB, meaning that whose objective functions are lower than the calculated LB, as shown in Table 8. Gantt chart of a solution found by MGLS to the problem Q10S2T03 is drawn to assure that it is possible to find solutions whose objective values might be lower than defined LB (In this instance, LB is 497 for Q10S2T03; however, the objective value is obtained as 378, which is less than LB, as given in Fig. 2).

Table 6 Comparison of MGLS, GA, PGA and EGA

Statement	GA	PGA	EGA
MGLS is better than	113	110	69
MGLS is equal to	64	76	90
MGLS is worse than	63	54	81
Total	240	240	240

The proposed MGLS algorithm found better solution than the others metaheuristic methods for HFSMT scheduling problems from the literature. The reasons of this are explained as follows.

- The proposed MGLS algorithm is a population-based heuristics coupled with individual learning and local improvement.
- MGLS algorithm has been a good balance between exploration and exploitation.
- MGLS reduced the feasible solution space into subspaces of local optima.
- The proposed MGLS algorithm is combined the global and local search methods.
- The MGLS algorithm used natural selection, crossover, mutation and local operators.
- A FFED was carried out to determine the best parameter set for each instance, solving a predetermined problem with each possible parameter combination.

Table 7 The comparison of GA, PGA, EGA and MGLS in terms of ARPD

	Jobs	Stages	GA	PGA	EgA	MGLS
P	10	2	0.596	0.596	0.596	0.596
		5	6.083	6.560	6.574	7.004
		8	14.753	16.279	16.638	16.454
	20	2	0.498	0.358	0.487	0.487
		5	2.921	3.092	2.484	2.561
		8	5.535	6.471	6.302	5.269
	50	2	0.580	0.550	0.227	0.401
		5	1.856	0.984	0.773	0.703
		8	3.467	3.355	2.109	1.406
	100	2	0.250	0.228	0.285	0.226
		5	2.182	1.175	0.772	0.830
		8	3.146	1.989	0.784	0.924
Q	10	2	4.471	13.506	11.586	11.413
		5	6.626	7.496	7.707	4.405
		8	12.022	14.724	14.293	15.186
	20	2	1.624	9.231	5.246	9.315
		5	7.356	6.825	6.180	6.767
		8	17.601	16.211	15.963	15.014
	50	2	1.372	4.639	4.675	4.706
		5	10.203	5.471	4.522	2.750
		8	16.893	14.711	12.989	12.836
	100	2	1.448	1.838	1.246	1.672
		5	19.870	7.187	4.952	4.503
		8	13.461	11.769	8.042	9.207
Average of all			6.450	6.469	5.643	5.610

Table 8 Solutions that improve lower bounds

Problem	RPD	Problem	RPD
Q10S2T03	−23.944	Q50S5T08	−2.510
Q10S5T08	−18.598	Q50S5T09	−8.770
Q10S5T09	−14.732	Q1HS5T02	−10.101
Q20S5T05	−3.397	QH1S2T09	−4.580
Q50S5T03	−2.208	QH1S8T09	−0.426
Q50S5T04	−2.607	Q50S5T10	−0.198

5 Conclusion and future research

In this research, hybrid flow shop with multiprocessor task problems, i.e., Fk$(Pm_1,\ldots,,Pm_k)|size_{ij}|C_{max}$, is solved by proposed memetic global and local search algorithm. Three different types of crossover methods, which are order, position based and new crossover operator, and two different types of mutation methods, which are inversion and three arbitrary genes exchange mutation, are implemented in the algorithm. Also, for more intense search near the neighborhood of good solutions, an iterative local search applied for the best solution in

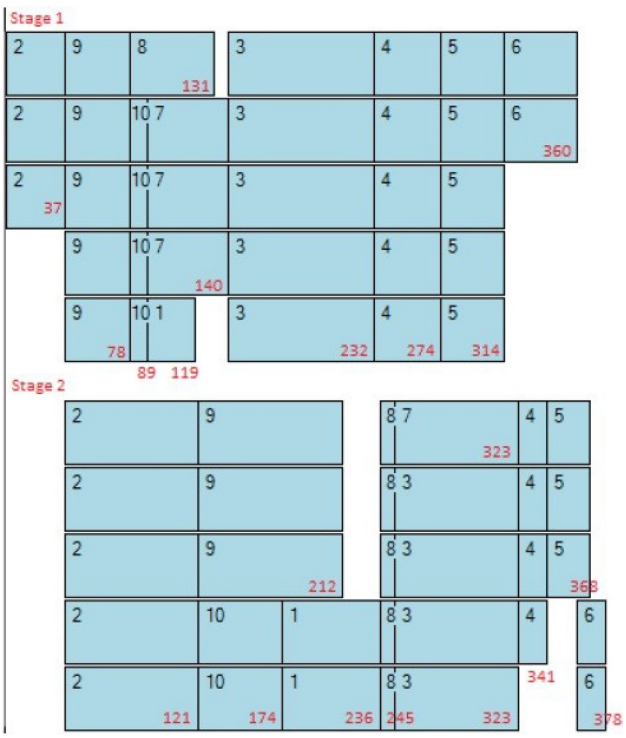

Fig. 2 The Gantt chart of the Q10S2T03 problem's solution

each iteration is implemented in the algorithm. The performance of the proposed MGLS algorithm is dependent on the parameter sets such as crossover and mutation methods and rate, population and iteration size and parent selection rate. A full factorial experimental design is done to determine the best parameter sets for solving the HFSMT scheduling problems. The Oğuz [38]'s benchmark HFSMT instances are solved by the proposed new MGLS algorithm with the best parameter sets. The results showed that the proposed MGLS outperformed GA proposed by Oğuz and Ercan [17], PGA proposed by Kahraman et al. [1] and EGA approach proposed by Engin et al. [10] in finding good solutions in terms of ARPD. Besides, MGLS algorithm improved the best known solutions so far for 48 instances out of 240. The experiment results showed that the MGLS algorithm is a better approach to Fk$(Pm_1,\ldots,Pm_k)|size_{ij}|C_{max}$ than its predecessors in the literature.

For future work, we are considering to improve the lower bound proposed by Oğuz [38] and provide different heuristics to compare with each other.

Compliance with ethical standards

Conflict of interest The authors declare that they have no competing interests.

References

1. Kahraman C, Engin O, Kaya I, Öztürk RE (2010) Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach. Appl Soft Comput 10(4):1293–1300
2. Engin O, Doyen A (2004) A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Gener Comput Syst 20:1083–1095
3. Alaykıran K, Engin O, Doyen A (2007) Using ant colony optimization to solve hybrid flow shop scheduling problems. Int J Adv Manuf Technol 35:541–550
4. Kahraman C, Engin O, Kaya I, Yılmaz MK (2008) An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. Int J Comput Intell Syst 1(2):134–147
5. Gupta JND (1988) Two stage hybrid flow shop scheduling problem. J Oper Res Soc 39(4):359–364
6. Hoogeveen JA, Lenstra JK, Vettman B (1996) Preemptive scheduling in a two-stage multiprocessor flowshop is NP-hard. Eur J Oper Res 89:172–175
7. Arthanari TS, Ramamurthy KG (1971) An extension of two machines sequencing problem. Opsearch 8:10–22
8. Linn R, Zhang W (1999) Proceedings of the 24th international conference on computers and industrial engineering hybrid flow shop scheduling: a survey. Comput Ind Eng 37(1):57–61
9. Chou FD (2013) Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. Int J Prod Econ 141(1):137–145
10. Engin O, Ceran G, Yilmaz MK (2011) An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Appl Soft Comput 11(3):3056–3065
11. Edwin SHH, Ansari N, Ren H (1994) A genetic algorithm for multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 5(2):113–120
12. Oğuz C, Ercan FM (1997) Scheduling multiprocessor tasks in a two stage flow shop environment. Comput Ind Eng 33:269–272
13. Şerifoğlu SFS, Tiryaki IU (2002) Multiprocessor task scheduling in multistage hybrid flow-shops: a simulated annealing approach, Proceedings of the 2nd international conference on responsive manufacturing, pp 270–274
14. Oğuz C, Ercan MF, Cheng TCE, Fung YF (2003) Heuristic algorithms for multiprocessor task scheduling in a two stage hybrid flow shop. Eur J Oper Res 149:390–403
15. Oğuz C, Zinder Y, Do VH, Jania A, Lichtenste M (2004) Hybrid flow shop scheduling problems with multiprocessor task systems. Eur J Oper Res 152:115–131
16. Şerifoğlu SFS, Ulusoy G (2004) Multiprocessor task scheduling in multistage hybrid flow shops: a genetic algorithm approach. J Oper Res Soc 55:504–512
17. Oğuz C, Ercan MF (2005) A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. J Sched 8:323–351
18. Kwok YK, Ahmad I (2005) On multiprocessor task scheduling using efficient state space search approaches. J Parallel Distrib Comput 65:1515–1532
19. Zinder Y, Do VH, Oğuz C (2005) Computational complexity of some scheduling problems with multiprocessor tasks. Discret Optim 2:391–408
20. Coll PE, Ribeiro CC, Souza CC (2006) Multiprocessor scheduling under precedence constraints: polyhedral results. Discret Appl Math 154:770–801
21. Shenassa MH, Mahmoodi M (2006) A novel intelligent method for task scheduling in multiprocessor systems using genetic algorithm. J Franklin Inst 343:361–371
22. Ying KC, Lin SW (2006) Multiprocessor task scheduling in multistage hybrid flowshops: an ant colony system approach. Int J Prod Res 44(16):3161–3177
23. Engin O, Yılmaz MK, Ceran G (2006) Using data mining to find patterns in genetic algorithm solutions to hybrid flow shop scheduling problems with multiprocessor task systems. Paper presented at international conference on modelling and simulation, AMSE, Konya, Turkey, pp 1041–1045, 2006
24. Kuszner L, Malafiejski M (2007) A polynomial algorithm for some preemptive multiprocessor task scheduling problems. Eur J Oper Res 176:145–150
25. Hwang R, Gen M, Katayama H (2008) A comparison of multiprocessor task scheduling algorithms with communication costs. Comput Oper Res 35:976–993
26. Tseng CT, Liao CJ (2008) A particle swarm optimization algorithm for hybrid flowshop scheduling with multiprocessor tasks. Int J Prod Res 46(17):4655–4670
27. Cheng SC, Shiau DF, Huang YM, Lin YT (2009) Dynamic hard-real time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints. Expert Syst Appl 36:852–860
28. Ying KC (2009) An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. J Oper Res Soc 60:810–817
29. Wang HM, Chou FD, Wu FC (2011) A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. Int J Adv Manuf Technol 53(5):761–776
30. Xu Y, Wang L, Liu M, Wang SY (2013) An effective shuffled frog-leaping algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Int J Adv Manuf Techno 68(5):1529–1537
31. Lin SW, Ying KC, Huang CY (2013) Multiprocessor task scheduling in multistage hybrid flowshops: a hybrid artificial bee colony algorithm with bi-directional planning. Comput Oper Res 40(5):1186–1195

32. Lahimer A, Lopez P, Haouari M (2013) Improved bounds for hybrid flow shop scheduling with multiprocessor tasks. Comput Ind Eng 66(4):1106–1114
33. Akkoyunlu MC, Engin O, Büyuközkan K (2015) A harmony search algorithm for hybrid flow shop scheduling with multiprocessor task problems. In: Proceedings of the 6th international conference on modeling, simulation, and applied optimization (ICMSAO), Istanbul, Turkey, May 2015, pp. 1–3
34. Rani ADC, Zoraida BSE (2016) Multistage multiprocessor task scheduling in hybrid flow shop problems using discrete firefly algorithm. Int J Adv Intell Paradig 8(4):377–391
35. Engin B, Engin O (2018) Hybrid flow shop with multiprocessor task scheduling based on earliness and tardiness penalties. J Enterp Inf Manag 31(6):925–936
36. Kurdi M (2018) A social spider optimization algorithm for hybrid flow shop scheduling with multiprocessor task. In: Proceedings of the 12th international NCM conference, Ankara, Turkey, pp. 38–4411–12 Sep 2018
37. Kurdi M (2019) Ant colony system with a novel non-daemon actions procedure for multiprocessor task scheduling in multistage hybrid flow shop. Swarm Evolut Comput 44:987–1002
38. Oğuz C (2006) Data for hybrid flow-shop scheduling with multiprocessor tasks, Koc University. Available from http://home.ku.edu.tr/~coguz/
39. Ying KC, Lin SW (2018) Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. Expert Syst Appl 92:132–141
40. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts, towards memetic algorithms. Tech. rep, California Institute of Technology
41. Ong YS, Keane AJ (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Trans Evolut Comput 8(2):99–110
42. Garg P (2010) A comparison between memetic algorithm and genetic algorithm for the cryptanalysis of simplified data encryption standard algorithm. CoRR abs/1004.0574
43. Burke EK, Newall JP, Weare RF (1996) A memetic algorithm for university exam timetabling. Springer, Berlin , pp 241–250
44. Wu X, Che A (2019) A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. Omega 82:155–165
45. Zhang G, Xing K (2018) Memetic social spider optimization algorithm for scheduling two-stage assembly flowshop in a distributed environment. Comput Ind Eng 125:423–433
46. Wang H, Fu Y, Huang M, Huang GQ, Wang J (2017) A NSGA-II based memetic algorithm for multiobjective parallel flowshop scheduling problem. Comput Ind Eng 113:185–194
47. Deng J, Wang L (2017) A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. Swarm Evolut Comput 32:121–131
48. Mencía R, Sierra MR, Mencía C, Varela R (2015) Memetic algorithms for the job shop scheduling problem with operators. Appl Soft Comput 34:94–105
49. Engin BE, Baysal ME, Engin O, Sümbül MO, Sarucan A (2015) A memetic algorithm to solve the open shop scheduling problem. In: Proceedings of the 6th international conference on modeling, simulations and applied optimization, Yıldız Technical University, İstanbul, Turkey
50. Soukour AA, Devendeville L, Lucet C, Moukrim A (2013) A memetic algorithm for staff scheduling problem in airport security. Expert Syst Appl 40(18):7504–7512
51. Nawaz M, Enscore EE, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11(1):91–95
52. Syswerda G (1991) Scheduleoptimizationusinggeneticalgorithms. In: Davis L (ed) Hand- book of genetic algorithms. Van Nostrand Reinhold, New York
53. Davis L (1985) Applying adaptive algorithms to epistatic domains. In: International joint conference on artificial intelligence.
54. Rocha M, Neves J (1999) Preventing premature convergence to local optima in genetic algorithms via random offspring generation. Springer, Berlin Heidelberg, pp 127–136
55. Engin O, Güçlü A (2018) A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. Appl soft comput 72:166–176