1-1-2022

A parameter tuned hybrid algorithm for solving flow shop scheduling problems with parallel assembly stages

Mona Jabbari

Madjid Tavana
La Salle University, tavana@lasalle.edu

Parviz Fattahi

Fatemeh Daneshamooz

Sustainable Operations and Computers 3 (2022) 22–32

Contents lists available at ScienceDirect

Sustainable Operations and Computers

journal homepage:
http://www.keaipublishing.com/en/journals/sustainable-operations-and-computers/

A parameter tuned hybrid algorithm for solving flow shop scheduling problems with parallel assembly stages

Mona Jabbari [a], Madjid Tavana [b,c,*], Parviz Fattahi [d], Fatemeh Daneshamooz [e]

[a] Department of Finance, Providence College, Providence, Rhode Island
[b] Business Systems and Analytics Department, Distinguished Chair of Business Analytics, La Salle University, Philadelphia, PA 19141
[c] Business Information Systems Department, Faculty of Business Administration and Economics, University of Paderborn, Paderborn, Germany
[d] Department of Industrial Engineering, Alzahra University, Tehran, Iran
[e] Department of Industrial Engineering, Bu-Ali Sina University, Hamedan, Iran

ARTICLE INFO

Keywords:
Flow shop
Parallel assembly stages
Scheduling
Metaheuristic
Taguchi

ABSTRACT

In this paper, we study the scheduling problem for a customized production system consisting of a flow shop production line with a parallel assembly stage that produces various products in two stages. In the first stage of the production line, parts are produced using a flow shop production line, and in the second stage, products are assembled on one of the parallel assembly lines. The objective is to minimize the time required to complete all goods (makespan) using efficient scheduling. A mathematical model is developed; however, the model is NP-hard and cannot be solved in a reasonable amount of time. To solve this NP-hard problem, we propose two well-known metaheuristics and a hybrid algorithm. To calibrate and improve the performance of our algorithms, we employ the Taguchi method. We evaluate the performance of our hybrid algorithm with the two well-known methods of Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) and demonstrate that our hybrid algorithm outperforms both the GA and PSO approaches in terms of efficiency.

1. Introduction

Machine scheduling is a significant challenge in the planning and control of manufacturing operations. The optimization of machine scheduling requires an effective and efficient assignment of production activities to a limited number of available machines. These problems are computationally challenging, and therefore subject of many research articles [24,29,37]. Flow shop scheduling with assembly operation is a widely applied machine scheduling problem in manufacturing. A large number of these problems are traditional two-stage assembly flow shop scheduling problems [[19],[27]]. Lee et al. [19] conducted the first study on flow shop scheduling with assembly operations. They demonstrated that the problem is NP-hard and proposed a branch and bound and three heuristics to solve it.

Cheng and Wang [5] solved a makespan minimization problem in a two-machine flow shop system considering a special structure. In their study, there are two types of components known as unique components and common components. Unique components are processed individually, and common components are processed in batches, which require different times for each type of product. The first machine produces both unique and common components in the production system, and the second machine assembles components into products. Yokoyama [36] studied the scheduling problem of a hybrid production system that includes machining and assembly operations. He proposed a branch and bound to solve the problem. Sun et al. [38] studied a flow shop scheduling problem with an assembly operation. They presented a series of heuristic algorithms based on Johnson's and Gupta's algorithms. Yokoyama and Santos [39] studied a modified flow shop scheduling in which processed parts are assembled to form the products in a subsequent assembly stage. Each part was produced on a flow shop line with two machines, and then products were completed on a single assembly stage. They proposed an efficient branch and bound method to minimize the weighted sum of the product completion time. Allahverdi and Al-Anzi [1] studied an assembly scheduling problem with two stages. In this problem, parts are produced using m machines in the first stage and assembled with an assembly machine in the second stage. They considered setup times and process times separately. They employed a hybrid tabu search, a self-adaptive differential evolution (SDE), and a new self-adaptive differential evolution (NSDE) to solve the problem. Hatami et al. [13] solved a distributed assembly permutation flow shop scheduling problem with a makespan minimization objective. They proposed a mixed-integer linear programming model, three constructive algorithms, and a variable neighborhood descend algorithm to solve the problem. Mahdavi et al.

* Corresponding author at: Business Systems and Analytics Department, Distinguished Chair of Business Analytics, La Salle University, Philadelphia, PA 19141, United States.
E-mail addresses: Mona.jabbari@providence.edu (M. Jabbari), tavana@lasalle.edu (M. Tavana).

https://doi.org/10.1016/j.susoc.2021.09.002
Received 18 May 2021; Received in revised form 7 July 2021; Accepted 8 September 2021
Available online 15 September 2021

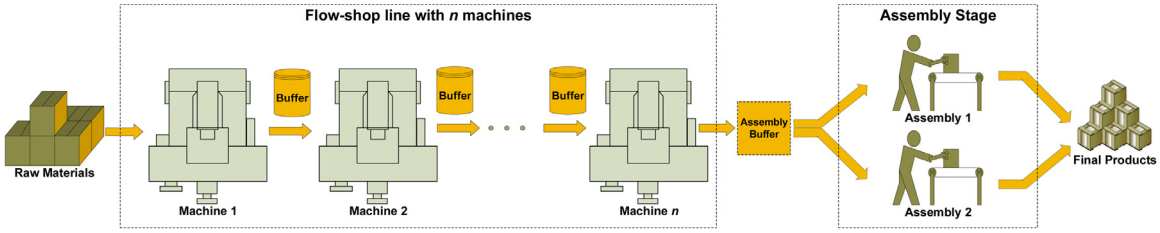

Fig. 1. Flow shop with a parallel assembly stage.

[21] studied a production system that consists of a hybrid flow shop production line followed by an assembly stage to minimize makespan. They modeled the production setup as an integer programming problem and offered two metaheuristics to solve the problem. Fattahi et al. [8] considered a compound production system with a hybrid flow shop line at the first stage and an assembly line at the second stage. They presented a mathematical model for the problem and a series of heuristic algorithms based on Johnson's algorithm. They evaluated the final solution with two improved lower bounds. Tajbakhsh et al. [32] studied a multi-objective assembly permutation flow shop scheduling problem, which includes three parts: machining, assembly, and batch processing. They proposed a mathematical model of the problem, and a metaheuristic algorithm was applied to solve the problem. Fattahi et al. [9] studied a hybrid flow shop scheduling problem with setup time and assembly operations. They extended a hierarchical branch and bound algorithm to solve the problem and reported three upper and lower bound. Komaki and Keyvanfar [16] suggested a metaheuristic algorithm based on a gray wolf optimizer algorithm for solving the two-stage assembly flow shop scheduling problem with predetermined release times. Maria Gonzalez-Neira et al. [12] studied the scheduling problem processed through two stages in a company and solved the problem with a hybrid algorithm. The problem is a distributed assembly permutation flow shop problem in which the production line consists of a set of distributed flow shop lines in the first stage and an assembly line in the second stage. Framinan and Perez-Gonzalez [10] addressed the two-stage assembly scheduling problem and used some constructive heuristic and metaheuristic algorithms to solve the problem. Allahverdi et al. [2,3] proposed an algorithm for solving the two-stage assembly-scheduling problem to minimize total tardiness with setup times. Their proposed algorithm reduces the error of the best previous algorithms by about 50%. Komaki et al. [17] presented some dispatching rules, a lower bound, and a metaheuristic algorithm based on the cuckoo optimization algorithm for solving the three-stage assembly flow shop scheduling problem. Lee [20] proposed six lower bounds and four heuristic algorithms for solving a two-stage assembly problem to minimize the total completion time.

Sabouni and Logendran [30] proposed a new mathematical model, lower bounds, and an efficient algorithm for solving a real case flow shop problem in the electronic industry. Sheikh et al. [31] proposed some optimal polynomial solutions and some multi-objective-based metaheuristic algorithms for solving a multi-objective assembly flow shop with release time.

In this paper, we will discuss a modified type of production system that is used in modern industries. In many production lines, many procedures are performed on each part repeatedly. The majority of the time, pieces go through the same series of processes in the same order. In this case, machines are placed in a series setup known as a flow shop [26]. In our problem, the first stage consists of k ($k = \{1, 2, ..., K\}$) machines that are arranged as a flow shop, and in the second stage, produced parts are assembled on two parallel assembly lines. This production line has the flexibility of producing several products with different parts. The required parts are manufactured in the first stage to produce each product. The parts are then assembled to form the final product. It should be noted that the two parallel assembly lines in the second stage are identical. Therefore, the assembly time of each product is the same on both assembly lines. It is assumed that all parts and machines are available at the beginning of the production time horizon, and each part can be processed by only one machine at a time; machines cannot operate more than one product simultaneously. Setup time and required time for the transmission of parts between flow shop and assembly stages are negligible. Also, preemption is not authorized. The number of parts and required machine timing are deterministic, and there is no space limitation between stages.

The remainder of the paper is organized as follows. The problem formulation and the mathematical model are presented in Section 2. Section 3 presents the algorithms proposed in this study. In Section 4, the parameters of the algorithms are designed, and some example problems are solved using the proposed solution procedures. The results are also compared based on different criteria. Finally, the summary of the results and directions for future researches are discussed in Section 5.

2. Problem description

This paper studies a modified flow shop scheduling production line with assembly operations. Assume a set of T products $t = \{1, .., T\}$ has to be produced. Each product consists of different parts (i) which should be processed through a flow shop line. Different machines are arranged as a flow shop at the first stage. Each part is processed at the first stage, and then parts are assembled at either assembly line one or two to produce the final product. Fig. 1 shows a simplified view of the explained problem. As depicted in Fig. 1, the first stage involves the processing of raw materials or pieces. When the process is complete, the produced parts are assembled to form the final products in the second stage. In this paper, the following conditions are considered:

a Assembly operation cannot start unless all its related component parts are ready to assemble.
b The processing time of part i, $i = \{1, .., n\}$, are given as constants on each machine at a flow shop.
c Assembly times of the product t, $t = \{1, .., T\}$, are available, and they are the same on both assembly stages.
d Machines cannot process more than one operation at the same time.
e All machines are available and ready to use at the beginning of the scheduling horizon and never break down during the scheduling period.
f Setup times are independent of each other and are included in the total processing time for each operation. Therefore, the order in which the parts are manufactured has no effect on the setup durations.
g The objective function is the minimization of the makespan (maximum completion time).
h At the first stage, the sequence of production is the same for all machines. In other words, the production line is a permutation flow shop.
i Different products are ordered, and only one of each product is required.

The notations of the mathematical model of the problem are as follows:

T	Total number of products
t	Product index, $t = \{1, .., T\}$
n	Total number of parts
i	Part index, $i = \{1, .., n\}$
n_t	Number of parts in product t
k	Machine index at the first stage (flow shop) $k = \{1, 2, ..., K\}$

p_{ik} Process time of part i on the machine k at the first stage
m Index of stages in the assembly line, $m = \{1, 2\}$
AT_t Assembly time of product t
a_{it} 1 if the part i is a component of the product t; 0 otherwise
j Index of sequence on flow shop machines, j={1, .., n}
p Index of sequence in assembly lines
M A large positive number, The mathematical modeling of the problem has the following variables:
x_{ij} 1, if the part i is processed at sequence j on flow shop; 0 otherwise
a'_{jt} 1, if the part in the sequence j is a component of the product t; 0 otherwise
R_t Ready time of the product t to be assembled, i.e., all component parts of the product t have passed through stage 1
p'_{jk} Processing time of part in sequence $j$on the machine k at the first stage
c_{jk} Completion time of the job in sequence $j$on the machine k at the first stage
z_{mpt} 1, if the product t is assembled in priority p on the assembly line m
f_j Completion time of processing part in sequence $j$at the first stage (complete flow shop)
s_{mp} The start time of machine m to assemble the product in priority p
$c_{\max}$ Makespan

Based on the aforementioned problem and notations, the mathematical model is presented as follows:

$$Min \;\; z = c_{\max} \tag{1}$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \forall j \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \forall i \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} \times p_{ik} = p'_{jk} \forall j, k \tag{4}$$

$$c_{jk} = p'_{jk} j = 1, k = 1 \tag{5}$$

$$c_{jk} = p'_{jk} + c_{(j-1)k} j \geq 2, k = 1 \tag{6}$$

$$c_{jk} = p'_{jk} + c_{j(k-1)} j = 1, k \geq 2 \tag{7}$$

$$c_{jk} = \max\left\{c_{(j-1)k}, c_{j(k-1)}\right\} + p'_{jk} k \geq 2, j \geq 2 \tag{8}$$

$$c_{jk} \leq f_j \forall j, k \tag{9}$$

$$\sum_{i=1}^{n} x_{ij} \times a_{it} = a'_{jt} \forall j, t \tag{10}$$

$$f_j \leq R_t + M(1 - a'_{jt}) \forall j, t \tag{11}$$

$$R_t \leq s_{mp} + M(1 - z_{mpt}) \forall t, m, p \tag{12}$$

$$S_{mp} \leq R_t + M(1 - z_{mpt}) \forall t, \forall m, p = 1 \tag{13}$$

$$S_{m(p-1)} + AT_t \times z_{m(p-1)t} \leq S_{mp} \forall t, \forall m, p \geq 2 \tag{14}$$

$$\sum_{m=1}^{2} \sum_{p=1}^{P} z_{mpt} = 1 \forall t \tag{15}$$

$$\sum_{t=1}^{n} z_{mpt} \geq \sum_{t=1}^{n} z_{m(p+1)t} \forall m, p \tag{16}$$

$$c_{\max} \geq s_{mp} + \sum_{t=1}^{T} AT_t \times z_{m(p-1)t} \forall m, p \tag{17}$$

$$x_{ij}, a'_{jt} \; and \; z_{mpt} \in \{0, 1\} \tag{18}$$

Eq. (1) shows the objective function that is makespan minimization. Constraints (2), (3) imply that each part should be processed in one priority. In particular, Constraint (2) ensures that there should be only one part in each priority, and Constraint (3) ensures that each part should be allocated to one priority. Eq. (4) calculates the processing time of the job for the j_{th} priority. The required process time of each job on each machine is indicated by Constraints (5)–(7), and [7]. More precisely, Constraint (5) shows the completion time of the first job on the first flow shop machine. Constraint (6) shows the completion time of other jobs on the first flow shop machine. Eq. (7) calculates the completion time of the first priority job on the other machines, and Constraint (8) shows the completion time of other jobs on the other machines. Constraint (9) guarantees that the completion time of each job in stage one (flow shop) is greater than its completion time on each machine.

Eq. (10) shows the position of product $T's$ parts in the flow shop sequence. Eq. (11) guarantees that the product T will be ready to assemble only when all parts are processed at stage one. Eqs. (12) and (13) take care of the assembly start time of the products at stage two. Eq. (12) ensures that the assembly operation can only start when all parts of the product are ready to be assembled. Eq. (13) shows that the assembly process of the products on the first priority of each assembly line should start after all the parts are ready. At the same time, Constraint (14) calculates the assembly start time of products in other priorities. Eq. (15) guarantees that each product should be allocated to just one assembly line and only on one priority. Constraint (16) arranges the sequences in the assembly lines. Constraint (17) ensures that the makespan is greater or equal to the completion time of each product. Finally, Constraint (18) specifies the specifications of the decision variables.

3. Proposed algorithms

In this problem, parts machining and assembly operation planning should be treated simultaneously. As mentioned previously, Lee et al. [39],[40] proved that a two-stage manufacturing system with two machines at the first stage and a single assembly line at the second stage is strongly NP-Hard. Therefore, the more complicated problem considered in this paper is also NP-hard. In this case, the mathematical model can only find the optimum solutions for a small-scale scheduling problem in reasonable computation time. As the problem gets larger and more complicated, the computation time increases. Therefore, in most cases, the mathematical model cannot find the optimum answer to real-world problems in a reasonable amount of time. Many scholars have successfully employed metaheuristic algorithms (*i.e.*, the genetic algorithm, Particle swarm optimization, etc.) have been successfully employed by many scholars in solving complex scheduling problems [[4],[33],[15]]. We propose three algorithms to find near-optimal solutions to this problem. These developed algorithms are described in the following section.

3.1. Genetic algorithm

Holland first introduced the Genetic Algorithm (GA) (1970). Genetic algorithms are a type of evolutionary algorithm, which is a larger class of metaheuristics. GAs employ techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover, to generate solutions for optimization problems [11]. Genetic algorithms have been widely used to solve different optimization problems [[22],[35],[23]].

Section 1	5	1	9	4	10	7	8	2	6	3
Section 2	3	8	1	5	10	7	2	9	4	6

Fig. 2. A possible chromosome of GA.

Assembly line 1	Product 3	Product 2
Assembly line 2	Product 1	Product 4

Fig. 3. Decoding section two of the example chromosome.

A GA starts with a set of chromosomes called a population that defines an initial solution. A new population is generated from the former population. We hope to improve the initial solution by producing new populations. Each solution is evaluated based on the fitness criteria, and the most fitted solutions are chosen to form new solutions. New offspring solutions are created by a crossover operator. This operator combines the information of two parents by exchanging selected parts of the solutions of the parents. The other operator called mutation is employed in the algorithm to maintain population diversity by slight changes in the selected solutions (Gen and Cheng, 2000). At the beginning of the process, the properties of the operators (fitness criteria, crossover and etc.) should be specified, and then the GA algorithm is applied to initialize the first population of solutions; then mutation, crossover, inversion, and selection operators help to improve the initial solution through a repetitive process. This repetitive cycle is applied to the population until some termination condition is satisfied (e.g., no improvement in the population, a specific number of generations, and a certain pre-defined value for objective function). In the following, each GA part adapted to the scheduling problem is described.

3.1.1. Encoding

The chromosome of the proposed GA is constructed of two sections. Section 1 is a string of n integers, representing the parts of the product. In each chromosome, value i at the position j in Section 1 indicates the part i belongs to sequence j on the flow shop machines.

Section 2 illustrates the product assembly sequence on each assembly line using n integers. The value i denotes the product and values greater than the number of products are utilized to shift the sequence between assembly line 1 and assembly line 2. The assembly schedule begins with assembly line 1. We use an example to demonstrate the GA chromosome encoding technique. Fig. 2 illustrates a hypothetical chromosome structure for a problem with 10 parts and 4 products.

In this example, Section 1 represents the production sequence on the flow shop machines. Parts are manufactured in the following order on each flow shop machine: {5 – 1 – 9 – 4 – 10 – 7 – 8 – 2 – 6 – 3}. Section 2 details the assembly sequence for each product on each assembly line, as seen in Fig. 3. Before discussing Section 2 of the chromosome, note that we have four products and must determine the assembly priority of these four products on two assembly lines (T={1,2,3,4}). Section 2 begins with the number 3 in the first position; 3 is a member of Set T. Product 3 is allocated to assembly line 1 at its first priority because it is a member of Set T. The following number is 8, which is not a member of set T; as such, it does not represent an item; instead, it shifts the sequence between assembly lines 1 and 2; in other words, when the priority is scheduling on assembly line 1, a number that is not a member of set T, such as 8, will shift the scheduling to the second assembly line, and vice versa. Because assembly line 1 was scheduled before observing 8, the assembly scheduling will now be changed to assembly 2. Because the following number is 1, product 1 is given the first priority on assembly line 2. Following that, the next number is 5, which does not belong to the set T; as a result, the scheduling is shifted from assembly line 2 to assembly line 1. The following number is 10, which is not a part of set T, so scheduling is transferred from assembly line 1 to assembly line 2-similarly, number 7 redirects scheduling to assembly line 1. Because the next number, 2, is a member of set T, product 2 will be placed on assembly line 1′s second priority, after product 3. The next number is 9, which shifts scheduling to assembly line 2. Finally, assembly line 2′s second priority is assigned to product 4. The chromosome is encoded using this proposed approach, as illustrated in Fig. 3. We can capture every possible state of the problem using this encoding strategy.

3.1.2. Selection strategy

A fraction of the existing population is chosen to reproduce the new generation. Different methods are proposed to select parents (Reeves and Rowe, 2003). In this paper, selecting parents is based on the fitness of the chromosomes. Half of the parents are randomly selected from 40% of the best chromosomes in the population, and the remaining is randomly selected from the entire population.

3.1.3. Crossover operator

There are many crossover techniques for different data structures. In this paper, a one-point crossover is selected. In this method, a single crossover point is selected on the chromosomes of both parents, and the data after this crossover point is swapped between the parents. The new chromosomes are called the children.

When the crossover operator is used, several of the children's chromosomes become infeasible due to repeated sequences of the same integers. Accordingly, the following strategy is planned to cope with this problem:

a Find out repetitive numbers in each child's chromosome.
b Replace the first repetitive number of Child 1 with the first repetitive number of Child 2.
c Repeat this procedure for all chromosomes until children are feasible.

Once this method is implemented, all of the children's encoding becomes feasible. Fig. 4 illustrates a sample crossover operator using the suggested technique applied to the same example covered in Section 3.1.1.

In the Fig. 4 example, we apply the crossover operator and employ the suggested strategy until all the children are feasible. After applying the crossover operator, the initial step is to detect the repeating number in each offspring, *i.e.*, 1–4, 7–10). The first repeated offspring of Child 1 is then replaced with the first repetitive offspring of Child 2. If we look at this example again, the first repeated number in child one is 1, while the first repeated number in child two is 8. Following the proposed strategy, we substitute 1 for 8. It should be noted that the second 1 and the second 8 in each child have remained unchanged. The second repeated number in child 1 is a 9, while the second repeated number in child 2 is a 2. As a result, the first 9 and 2 observed in the offspring are replaced with each other, but the second 9 and 2 observed in the offspring remain unchanged. In child 1, the third repeated number is an 8, and in child 2, the third repeated number is a 9. As a result, the first "8″ and "9″ in the offspring are replaced by each other, but the second 8 and 9 in the offspring stay untouched. This procedure is repeated for the fourth set of repeated numbers (*i.e.*, 3, 4), the fifth set of repeated numbers (*i.e.*, 5, 6), and the sixth set of repeated numbers (*i.e.*, 10, 7).

3.1.4. Mutation operator

Mutation helps to keep genetic diversity from one generation to the next generation in genetic algorithm chromosomes. Mutation happens by a defined mutation probability during the evolution process. The mutation operator in GA preserves and introduces diversity in a generation. In this problem, the mutation operator is employed in two steps, one in the sequence of the flow shop and the other in the priority of assembly. In both steps, two numbers are randomly selected and replaced with each other.

Parent 1	5	1	9	4	10	**Crossover Point**	7	8	2	6	3
	8	3	5	1	10		7	2	9	4	6
Parent 2	8	10	2	4	5		6	7	3	9	1
	9	4	6	7	1		8	10	3	2	5
Produced offspring											
Offspring 1	5	1	9	4	10	6	7	3	9*	1*	
	8	3	5	1	10	8*	10*	3*	2	5*	
Offspring 2	8	10	2	4	5	7	8*	2*	6	3	
	9	4	6	7	1	7*	2	9*	4*	6*	
Exchange strategy for making offspring feasible											
offspring 1	5	8	2	4	10	6	7	3	9*	1*	
	9	4	6	1	7	8*	10*	3	2	5*	
offspring 2	1	10	9	4	5	7	8*	2*	6	3	
	8	3	5	10	1	7*	2	9*	4*	6*	

Note: * *Represents repeated offspring*

Fig. 4. Example of crossover operator.

Priority in flow shop	0.901	0.384	0.184	0.910	0.197	0.035	0.328	0.721	0.898	0.887
Discrete swarm	9	5	2	10	3	1	4	6	8	7

Fig. 5. Discrete PSO strategy.

Once children are generated and mutation is applied to them, their fitness is calculated, and if their fitness function is better, they are replaced with their parents.

3.1.5. Termination criterion

There are different termination criteria. In this paper, three termination criteria are proposed:

a The number of iterations exceeds the pre-defined number assigned by the user.
b The computational time of the program exceeds the time limit assigned by the user.
c The convergence of the chromosomes in two following iterations becomes less than the limit assigned by the user. The convergence criterion is calculated by Eq. (19)

$$\alpha_1 \times \left| \frac{mean_{t-1} - mean_t}{mean_{t-1}} \right| + \alpha_2 \times \left| \frac{\min_{t-1} - \min_t}{\min_{t-1}} \right| \le \varepsilon \tag{19}$$

where α_1, α_2 are random numbers and $\varepsilon$is the convergence limit. $mean_t$ is the average value of solutions in t iteration and $\min_t$is the minimum value obtained in the iteration t.

3.2. Proposed particle swarm optimization

Particle Swarm Optimization (PSO) is a stochastic optimization technique introduced by Eberhart and Kennedy [6]. PSO is based on the social behavior metaphor and has many similarities with evolutionary computation techniques such as Genetic Algorithms [25]. Many scholars have successfully used PSO for solving complicated optimization problems [[18],[14],[34]]. Since the PSO algorithm is a population-based algorithm, it starts with a population of random solutions, and a set of potential solutions evolves to a suitable solution (or set of solutions) for a problem. Each solution is called a "particle," and the set of the population is called the "swarm." The position of each particle stands for the potential solution. Each particle changes its position to follow these three principles:

a To maintain its inertia.
b To change based on its best position so far.
c To change based on the swarm's best position.

The particle's best position is named P_{best}, and the swarm's best solution is named G_{best}. The position of a particle is updated by each iteration, and the new particle is calculated using the formula, $X_i^{t+1} = X_i^t + V_i^{t+1}$ where X_i^t represents the particle position and V_i^t is the velocity of the particle i at the iteration t. The velocity is calculated according to this formula,$V_i^{t+1} = \omega_0 * V_i^t + C_1 * rand_1 * (P_{best_i} - X_i^t) + C_2 * rand_2 * (G_{best} - X_i^t)$, where ω_0 is the inertia weight controlling the movement of the particle. C_1, C_2 are called cognitive and social parameters, which determine the balance between convergence to P_{best_i} or G_{best} [25]. It is obvious that standard PSO equations use real-valued positions and velocities, and they cannot be used to generate a discrete job permutation.

3.2.1. Discrete PSO

Standard PSO is usually used for problems with real values. Since the solution space of this problem is discrete, at this stage, a strategy is introduced to adapt the problem with the standard PSO. When designing the PSO algorithm, one of the key issues is to establish a suitable way of encoding a schedule (or solution) in a way that particles have the necessary information. In this problem, each particle is divided into two sections. Section one represents the sequence of parts on flow shop machines, and section two shows the priority of product assembly on each assembly line. Section one is made of n (number of parts) random numbers between (0,1). The smallest number in this code is representative of part 1, and the largest number represents part n. So the numerical order shows parts number, and the value i in position j represents the sequence of parts machining. An example of this encoding process is shown in Fig. 5.

The same strategy is used to encode the second section. This section is interpreted exactly the same as Section 3.1.1. Applying this strategy, PSO can be used for the flow shop with assembly operation. Each particle is updated by the given formula and in each iteration P_{best} and G_{best} are calculated. The main loop of the algorithm is repeated until the termination criterion is met.

3.3. Proposed hybrid algorithm

As mentioned before, the PSO algorithm is one of the most efficient evolutionary algorithms known until now. However, this algorithm is used for problems with a continuous solution space. The proposed hybrid algorithm is assigned to take advantage of the PSO algorithm and become compatible with the discrete solution space of the problem. The hybrid algorithm is a combination of PSO and GA, with the main PSO structure. However, GA operators are applied to particles in each iteration. In other words, the hybrid algorithm upgrades a swarm of particles using GA operators such as crossover and mutation in each iteration. Therefore, the algorithm can search for the solution area more practically and find better solutions. Moreover, following the P_{best} and G_{best},

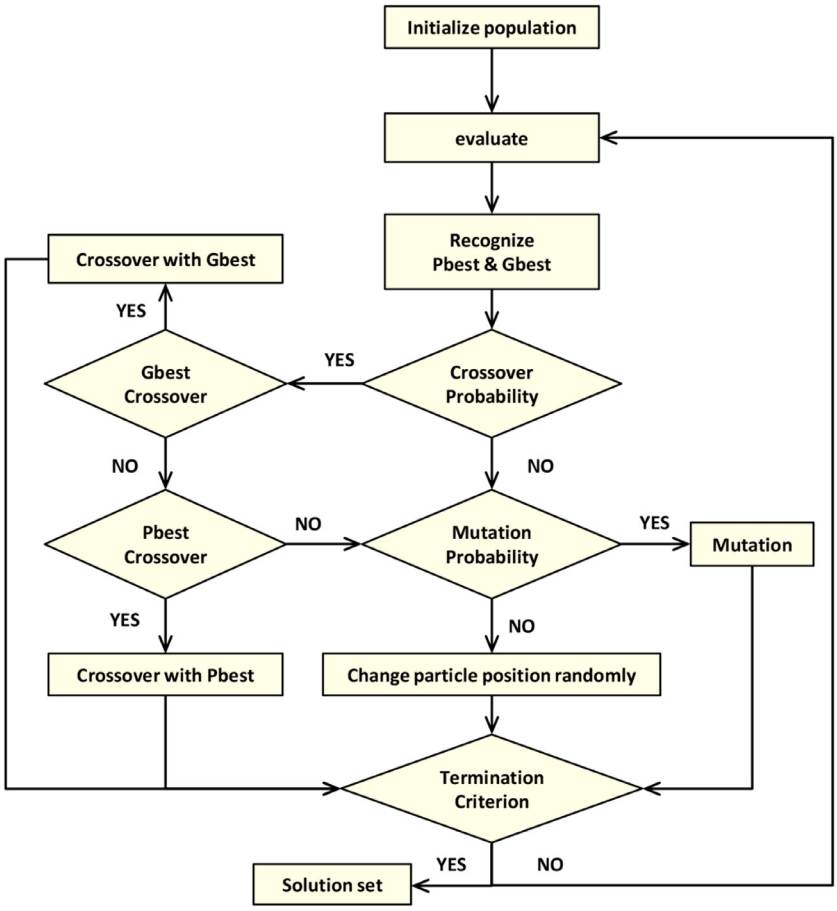

Fig. 6. Flowchart of the proposed hybrid algorithm.

in each iteration, the algorithm reaches a better solution. The flowchart of the proposed hybrid algorithm is shown in Fig. 6.

As illustrated in Fig. 6, the algorithm begins with generating a new population to serve as the initial population. The population's performance is next analyzed, and similar to what we proposed in PSO, the P_{best} and G_{best} values are calculated. The initial population will then be modified based on a given probability, either via a crossover operator, a mutation operator, or a random change in particle placements. When the crossover operator is applied to the initial population, the procedure is identical to what is described in part 3.1.3 of the GA algorithm. However, in this scenario, one of the crossover operator's parents should be the G_{best} or P_{best}. In this step, a random process is used to select either P_{best} or G_{best} as the parent, after which a new generation is created using that parent. When the mutation operator updates the initial solution, the mutation process is identical to the one used by the PSO algorithm. If neither the Crossover nor Mutation operators are used, the initial population is updated randomly, with particle positions changing arbitrarily. After each update of the initial chromosome, the termination criterion is evaluated, and the procedure is repeated as long as the termination criteria are met. All the encoding strategy and termination criterion is the same as the proposed strategy introduced for GA.

4. Computational results

There is no benchmark for the flow shop with parallel assembly stages. Therefore, random instances are made using MATLAB software. The problems are classified into three categories: small problems, medium problems, and large ones. The classificatiIn this paper, GA, PSO, and hybrid metaheuristic parameters have been tuned to optimize their performance. One of the best-known ways to tune the parameters is the Taguchi method. Therefore this method has been used in this study. We used the Minitab software to design the experiments and assign the best level for each size of the problems. Calibrated GA parameters are population size, maximum iteration, convergence factors $(\alpha_1, \alpha_2 | \alpha_1 + \alpha_2 = 1)$, convergence limit (ε),number of crossovers, crossover boundaries, and the probability of mutation (p_m).The Taguchi's preferred design for GA is the $L_{27}(3^7)$ orthogonal array. This array is designed to handle seven parameters in three levels. Each experiment is performed, and the results are shown in Table 1. The PSO parameters consist of swarm size, maximum iteration, convergence factors (α_1, α_2), convergence limit (ε),inertia weight (ω),cognitive and social parameters (C_1, C_2), and maximum and minimum velocity $(V_{\max}, V_{\min})$. The Taguchi's preferred design for PSO is the $L_{27}(3^7)$ orthogonal array. Table 2 shows the detailed information of the Taguchi method for the PSO procedure and the proper rates. Parameters of the hybrid algorithm are swarm size, maximum iteration, convergence factors (α_1, α_2), convergence limit (ε), probability of crossover (P_c), probability of crossover with Gbest $(P_{c-Gbest})$, probability of crossover with Pbest $(P_{c-Pbest})$, crossover boundaries, and mutation chance (P_m). The Taguchi design for the hybrid algorithm is $L_{27}(3^9)$, which handles nine parameters in three levels. Parameter levels of the proposed hybrid algorithm and the results of the Taguchi method are shown in Table 3.on strategy is based on the required time to solve the MIP model using GAMS software. If the required time is less than 600 s, the problem is defined as a small problem; if the required time is between 600 s and 5400 s, the problem is medium size, and finally, if it takes more than 5400 s, the problem is classified as a large problem. Details of each instance are shown in Table 4.

Several experiments are performed to evaluate the performance of the proposed solution methods. The MIP model is solved by the modeling language GAMS-IDE. The proposed algorithms are coded in MATLAB (R2013a) software. A personal computer with a 2.4 GHz Intel Core i5 processor and 16 GB of RAM memory is used to run the programs.

Table 1
GA parameters levels and result of the Taguchi method.

Parameter	Lower limit	Upper limit	Appropriate quantity according to Taguchi method		
			Small	Medium	Large
Population size	40	100	100	100	100
Maximum iteration	75	125	75	125	125
(α_1, α_2)	0.1	0.9	[0.5,0.5]	[0.3,0.7]	[0.3,0.7]
ϵ	10^{-5}	10^{-4}	4×10^{-5}	2×10^{-5}	10^{-5}
Number of crossovers	10	30	30	30	30
Crossover boundaries	0.1	0.9	$\begin{bmatrix} 0.5 & 0.9 \\ 0.5 & 0.9 \end{bmatrix}$	$\begin{bmatrix} 0.2 & 0.8 \\ 0.5 & 0.9 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0.9 \\ 0.5 & 0.9 \end{bmatrix}$
P_m	0.01	0.03	0.01	0.01	0.03

Table 2
PSO parameters levels and result of the Taguchi method.

Parameter	Lower limit	Upper limit	Appropriate quantity according to Taguchi method		
			Small	Medium	Large
Swarm size	60	100	80	100	100
Maximum iteration	75	125	75	125	125
(α_1, α_2)	0.1	0.9	[0.5,0.5]	[0.3,0.7]	[0.3,0.7]
ϵ	10^{-5}	10^{-4}	2×10^{-5}	2×10^{-5}	10^{-5}
ω	0.4	0.9	30	30	30
(C_1, C_2)	1.5	2	(2,2)	(2,2)	(2,2)
$(V_{\min}, V_{\max})$	$0.1 \times (\max_{variation} - \min_{variation})$	$\max_{variation} - \min_{variation}$	$0.1 \times (\max_{variation} - \min_{variation})$	$\max_{variation} - \min_{variation}$	$0.1 \times (\max_{variation} - \min_{variation})$
$V_{\max}$	$0.1 \times (\max_{variation} - \min_{variation})$	$\max_{variation} - \min_{variation}$	$0.1 \times (\max_{variation} - \min_{variation})$	$\max_{variation} - \min_{variation}$	$0.1 \times (\max_{variation} - \min_{variation})$
$V_{\min}$			-maximum velocity	-maximum velocity	-maximum velocity

Table 3
Hybrid parameters levels and result of the Taguchi method.

Parameter	Lower limit	Upper limit	Appropriate quantity according to Taguchi method		
			Small	Medium	Large
swarm size	60	100	100	100	100
Maximum iteration	75	125	125	125	125
(α_1, α_2)	0.1	0.9	[0.5,0.5]	[0.5,0.5]	[0.7,0.3]
ϵ	10^{-5}	10^{-4}	10^{-5}	10^{-5}	10^{-5}
P_c	0.2	0.8	0.5	0.5	0.75
$P_{c-Gbest}$	0.2	0.8	0.5	0.75	0.75
$P_{c-Pbest}$	0.2	0.8	0.25	0.25	0.25
crossover boundaries	0.1	0.9	$\begin{bmatrix} 0.2 & 0.8 \\ 0.3 & 0.7 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0.9 \\ 0.5 & 0.9 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0.9 \\ 0.2 & 0.8 \end{bmatrix}$
P_m	0.2	0.7	0.5	0.7	0.7

In order to evaluate the proposed algorithms, two performance criteria are calculated. The first criterion is named relative percentage deviation (RPD). It is calculated using Eq. (20). The second one is an improvement factor surveying the algorithm performance and is calculated by Eq. (21).

$$RPD = \left| \frac{Alg_{Best} - Best_{sol}}{Best_{sol}} \right| \tag{20}$$

$$\mathrm{Im}p = \left| \frac{Alg_{Initial-sol} - Alg_{Final-sol}}{Alg_{Initial-sol}} \right| \tag{21}$$

When multiple measurements are made across different samples, the relative percentage deviation is advantageous since it allows the output to be more generalized.

Alg_{Best} in Eq. (20) denotes the best solution found by each algorithm while $Best_{sol}$ indicates the best solutions obtained by all of these algorithms. In other words, it is the best of the bests. The performance metric specified in Eq. (21) quantifies an algorithm's capacity to improve its initial solution. This performance metric is critical for this problem since it allows us to assess the new algorithm's capability to explore the solution space and locate the optimal solution. When the objective function is not zero, RPD is quite beneficial. RPD is a frequently used performance metric for scheduling problems, particularly when makespan is considered as the objective function [28].

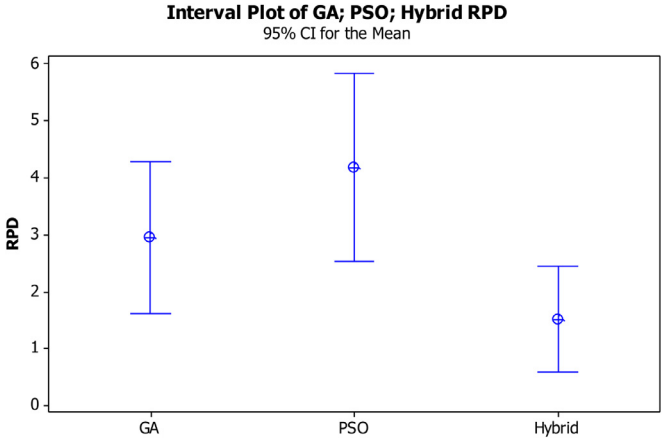

Fig. 7. Mean and interval plot of GA, PSO, and hybrid RPDs.

Due to minimize the errors, each problem is solved ten times by each procedure, and the mean of the results are shown in Table 5. Since the required time to solve large problems by the exact method is indefi-

Table 4
Detailed information on instances.

Problem Number		Number of parts	Number of products	Number of machines in flow shop
Small	P1	10	3	4
	P2	6	3	5
	P3	10	3	3
	P4	8	3	5
	P5	10	5	5
	P6	9	4	6
	P7	8	4	5
	P8	11	5	5
	P9	8	4	7
	P10	12	4	4
Medium	P11	16	5	6
	p12	14	5	5
	p13	15	6	4
	p14	15	7	5
	p15	12	6	7
	p16	14	7	3
	p17	14	6	5
	p18	15	6	5
	p19	16	5	7
	p20	18	6	6
Large	p21	30	10	8
	p22	25	5	8
	p23	24	6	5
	p24	23	5	6
	p25	22	7	5
	p26	30	6	5
	p27	25	10	6
	p28	30	15	10
	p29	35	10	10
	p30	40	10	15

Table 5
Computational results.

Problem		C^{max}				**CPU Time**				*RPD*			Imp		
		GAM S	***G A***	**PS O**	**Hybri d**	**GAM S**	**GA**	**PSO**	**Hybri d**	**GA**	**PSO**	**Hybri d**	**GA**	**PSO**	**Hybri d**
Small	P1	136	13 6	136	136	4.038	3.4	12.0 1	31.2	0.000	0.000	0.000	10.52 6	3.546	11.688
	P2	120	12 0	121	120	2.236	1.33	8.5	28.23	0.000	0.833	0.000	4.762	2.419	2.439
	P3	152	15 2	152	152	3.207	2.6	8.6	30.2	0.000	0.000	0.000	5.000	3.797	5.590
	P4	155	15 5	155	155	3.357	2.4	8.7	31	0.000	0.000	0.000	4.321	5.488	7.186
	P5	177	18 1	177	177	40.61 2	3.5	9.05	33.5	2.260	0.000	0.000	12.13 6	13.65 9	12.808
	P6	164	16 5	165	164	6.2	4.1	14.3	16.2	0.610	0.610	0.000	11.29 0	9.836	7.865
	P7	159	16 0	159	159	9.179	2.5	8.7	31.3	0.629	0.000	0.000	6.433	8.092	6.471
	P8	i70	i7 5	i70	i70	39.0i	3.8	2.4	35.6	2.94i	0.000	0.000	i4.2i 6	i5.42 3	i4.i4i
	P9	i79	i7 9	i80	i79	4i.3i	3.0i	3.6	34.8	0.000	0.559	0.000	3.243	4.255	6.77i
	P1 0	i64	i6 4	i64	i64	59.8	2.8	7.8	23.6	0.000	0.000	0.000	9.392	ii.82 8	i3.684
Mediu m	Pi 1	20i	22 5	224	2i8	3236	i0.6	32	23	i 0.94	ii.44	8.458	i7.27 9	i8.84 i	i8.959
	Pi 2	2i4	22 2	232	223	465	4	3	2.7	3.738	8.4ii	4.206	i2.25 3	7.570	i3.900
	Pi 3	230	23 8	243	236	i0i6	3.4	2.2	3.7	3.478	5.652	2.609	8.8i2	8.302	9.23i
	Pi 4	248	25 6	256	254	4895	6	24	2i	3.226	3.226	2.4i9	7.58i	4.833	i.550
	Pi 5	i93	2i 6	220	203	i007	i3.8	26.6	23	ii.9i	i3.99	5.i8i	i 0.83 7	8.333	i5.768
	Pi 6	i 56	i6 4	i64	i65	348i	5	8	7	5.i28	9.6i5	5.769	9.890	9.043	9.836
	Pi 7	208	2i 7	2i5	2i4	676	9	i5.7	2i	4.327	8.i73	2.885	i2.i4 6	8.i63	i2.295
	Pi 8	224	23 3	234	234	i339	7.4	i4.8	25.5	4.0i8	5.804	4.464	i0.03 9	8.846	i0.687
	Pi 9	279	29 6	304	285	769	i3	28.5	22.2	6.093	8.96i	2.i5i	8.923	9.792	i2.308
	P2 0	246	27 6	28i	274	3605. 2	ii.6	32	i9	i2.i9 5	i4.22 8	ii.382	i 0.25 4	8.i70	ii.327
Large	P2 i	–	50 2	503	504	–	24.6	73	65	0.000	0.i99	0.398	i2.39 i	i0.97 3	i0.480
	P2 2	–	40 9	4i9	398	–	2i	40	48	2.764	5.276	0.000	8.50i	4.773	i0.962
	P2 3	–	33 6	339	334	–	i3.3	40	30	0.599	i.497	0.000	8.i97	6.6i2	i0.695
	P2 4	–	33 5	342	332	–	i2.3	29.6	24.5	0.904	3.0i2	0.000	9.459	7.568	i0.027
	P2 5	–	30 0	3ii	286	–	i0.5	34	30	4.895	8.74i	0.000	8.8i5	3.ii5	i2.805
	P2 6	–	40 0	398	39i	–	i2.3	40	40.4	2.302	i.790	0.000	6.542	7.870	9.908
	P2 7	–	39 6	409	393	–	i7.i 8	35.7	38.26	0.763	4.07i	0.000	i0.20 4	7.256	ii.685
	P2 8	–	49 8	5i3	490	–	32	54.7	5i	i 0.633	4.694	0.000	i2.i6 9	i0.78 3	i5.5i7
	P2 9	–	56 8	573	559	–	32	74.5	63.3	i.6i0	2.504	0.000	i 0.93 8	9.335	ii.4i0
	P3 0	–	70 6	7i8	703	–	79.5	i2i. 8	99.3	0.427	2.i34	0.000	9.834	6.02i	8.225

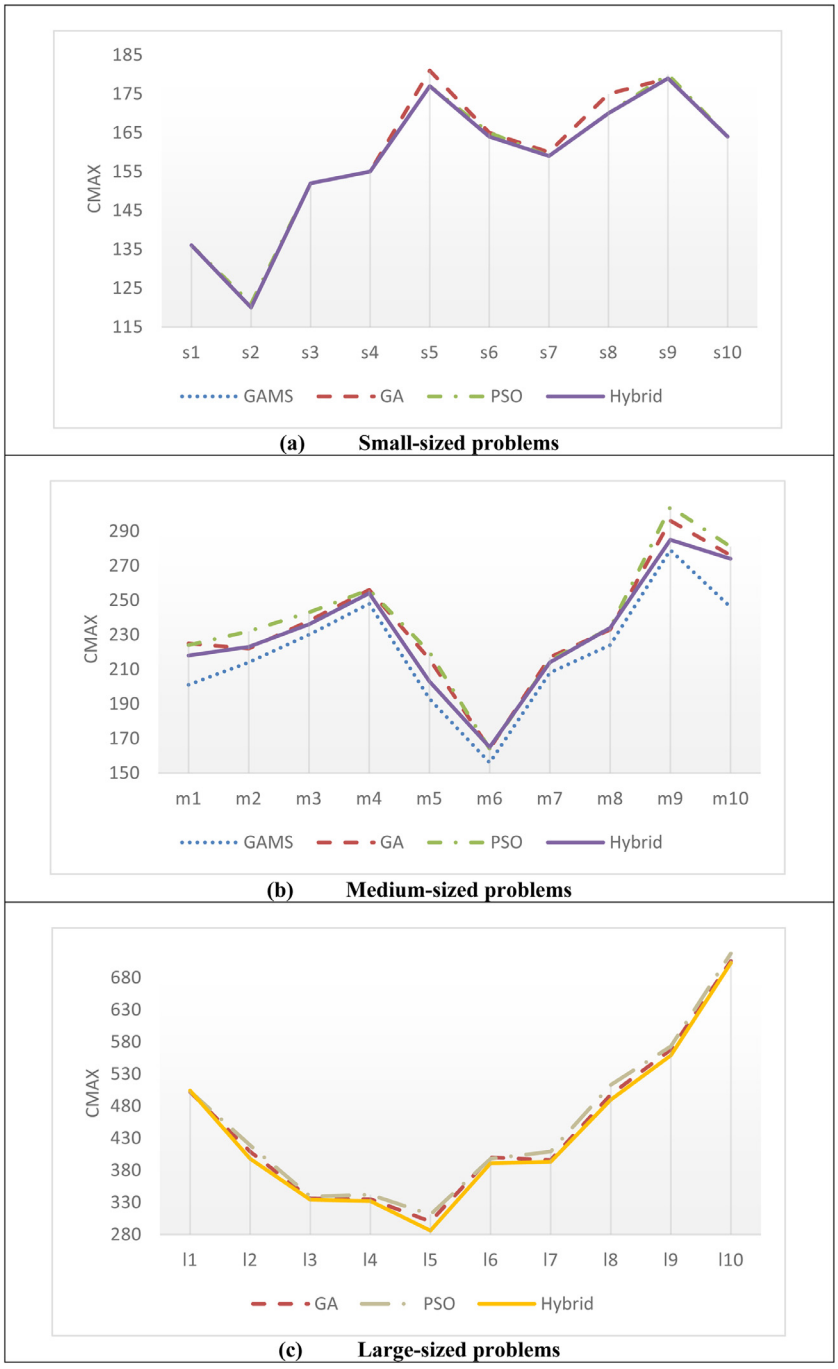

Fig. 8. A comparison of completion time with different algorithms.

nite, only small and medium problems are solved using mixed-integer programming.

As shown in Table 5, the hybrid algorithm usually achieves the best solution. It is obvious that the hybrid algorithm is capable of finding better solutions in large problems; furthermore, when the GAMS software is not able to find the optimal solution after 5400 s of computation time, the hybrid algorithm gets to a satisfactory solution in a fraction of minute. Fig. 7 shows the mean and interval plot of GA, PSO, and hybrid solutions RPD. In this figure, the hybrid solution has the least RPD.

Fig. 8 presents a comparison of different solution procedures for small-, medium-, and large-sized problems. (As shown in Fig. 8(a), all procedures have approximately the same performance, and they reach the optimal solution using the exact solution method. For medium-sized problems in Fig. 8(b), the GAMS algorithm performs best, and for large-sized problems. Fig. 8(c) shows the performance of three algorithms for large-sized problems. As mentioned before, these problems are not solvable by GAMS in a reasonable time. A comparison of the three algorithms shows the hybrid algorithm has the best performance.

In Fig. 9, the required CPU times are shown. This graph shows that GA is the fastest algorithm. The hybrid algorithm and the PSO algorithm reach to the solution gradually. Although the hybrid algorithm has the best performance in finding solutions, it performs much slower than the GA algorithm. As the problem gets larger, this difference gets bigger. In most cases, PSO has the worst performance with regards to CPU time required for solving the problems.

The improvement factor shows the ability of the algorithm to search the solution space and improve the initial solution. Fig. 10 compares the algorithms in terms of the solution space survey. As is obvious from

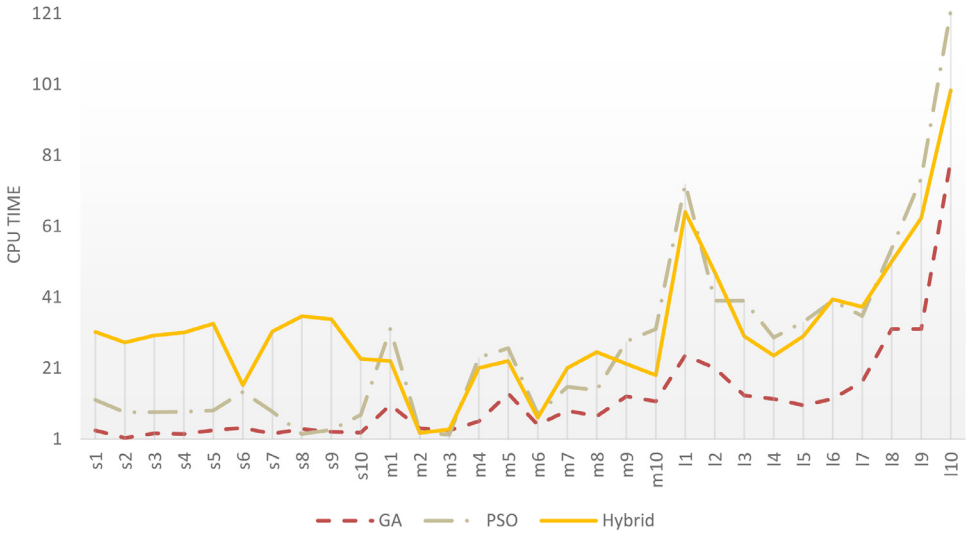

Fig. 9. Comparison of solution procedures based on CPU time.

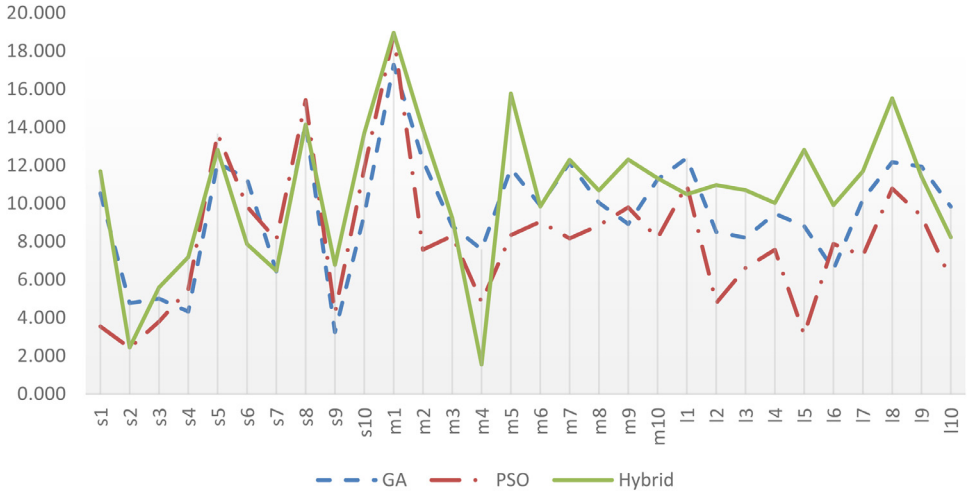

Fig. 10. Comparison of solution algorithms based on improvement factor.

Fig. 10, the hybrid solution searches the solution space in a more practical manner and improves the initial solution better than the other algorithms.

5. Conclusion

In this paper, the scheduling of a flow shop with parallel assembly stages was studied. In this production system, various products are produced. The required parts of each product are manufactured in a flow shop stage, and when the parts are ready, products are assembled in one of two assembly lines. The objective function is to minimize the makespan. A mixed-integer linear programming model of the problem is formulated and coded in GAMS optimization software to solve the problem. Exact solution methods can only solve small (and medium) size problems due to the complexity of the problem. Therefore, GA swarm optimization was used to solve medium and large-size problems. Moreover, a hybrid algorithm was proposed to solve the proposed problem. The hybrid solution algorithm is approximately similar to particle swarm optimization that utilizes GA operators for better performance. Since the appropriate design of parameters has a great effect on metaheuristics algorithm accuracy, the parameters of each algorithm are calibrated using the Taguchi method. All of the proposed metaheuristics can find appropriate solutions. The numerical experiment shows that, for every scale of the problem, the proposed hybrid algorithm effectively finds better solutions compared to other algorithms. However, GA usually has better performance regarding computational time. It was shown that the hybrid algorithm searches the solution space more effectively and can improve the initial solution better than the other algorithms.

Solving this problem considering other objective functions can be suggested for further researches. In this research, it is supposed that the demand for each product is stable and known in advance, which is not always realistic. Moreover, maintenance time has an important role in scheduling; therefore, solving the problem without the current assumptions is also recommended for further research.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Dr. Madjid Tavana is grateful for the partial support he received from the Czech Science Foundation (GAˇCR19–13946S) for this research.

References

[1] A. Allahverdi, F.S. Al-Anzi, Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times, Int. J. Prod. Res. 44 (22) (2006) 4713–4735.
[2] A. Allahverdi, H. Aydilek, A. Aydilek, Two-stage assembly scheduling problem for minimizing total tardiness with setup times, Appl. Math. Model. 40 (17–18) (2016) 7796–7815.

[3] M.R. Bonyadi, Z. Michalewicz, Analysis of stability, local convergence, and transformation sensitivity of a variant of the particle swarm optimization algorithm, IEEE Trans. Evolut. Comput. 20 (3) (2016) 370–385.
[4] A. Branda, D. Castellano, G. Guizzi, V. Popolo, Metaheuristics for the flow shop scheduling problem with maintenance activities integrated, Comput. Ind. Eng. 151 (2021) 106989.
[5] T.C.E. Cheng, G. Wang, Batching and scheduling to minimize the makespan in the two-machine flowshop, IIE Trans. 30 (5) (1998) 447–453.
[6] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the MHS'95 Sixth International Symposium on Micro Machine and Human Science, IEEE, 1995, pp. 39–43.
[7] W. Fang, J. Sun, H. Chen, X. Wu, A decentralized quantum-inspired particle swarm optimization algorithm with cellular structured population, Inf. Sci. 330 (2016) 19–48.
[8] P. Fattahi, S.M.H. Hosseini, F. Jolai, A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem, Int. J. Adv. Manuf. Technol. 65 (5–8) (2013) 787–802.
[9] P. Fattahi, S.M.H. Hosseini, F. Jolai, R. Tavakkoli-Moghaddam, A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations, Appl. Math. Model. 38 (1) (2014) 119–134.
[10] J.M. Framinan, P. Perez-Gonzalez, The 2-stage assembly flowshop scheduling problem with total completion time: efficient constructive heuristic and metaheuristic, Comput. Oper. Res. 88 (2017) 237–246.
[11] M. Gen, R. Cheng, Genetic algorithms and engineering optimization, John Wiley & Sons, 2000 (Vol. 7).
[12] E.M. Gonzalez-Neira, D. Ferone, S. Hatami, A.A. Juan, A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times, Simul. Model. Pract. Theory 79 (2017) 23–36.
[13] S. Hatami, S. Ebrahimnejad, R. Tavakkoli-Moghaddam, Y. Maboudian, Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup times, Int. J. Adv. Manuf. Technol. 50 (9–12) (2010) 1153–1164.
[14] M.A. Islam, Y. Gajpal, T.Y. Elmekkawy, Hybrid particle swarm optimization algorithm for solving the clustered vehicle routing problem, Appl. Soft. Comput. (2021) 107655, doi:10.1016/j.asoc.2021.107655.
[15] A. Khare, S. Agrawal, Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. (2020) 1–17, doi:10.1080/00207543.2020.1837982.
[16] G.M. Komaki, V. Kayvanfar, Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time, J. Comput. Sci. 8 (2015) 109–120.
[17] G.M. Komaki, E. Teymourian, V. Kayvanfar, Z. Booyavi, Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem, Comput. Ind. Eng. 105 (2017) 158–173.
[18] N. Kumar, A.K. Manna, A.A. Shaikh, A.K. Bhunia, Application of hybrid binary tournament-based quantum-behaved particle swarm optimization on an imperfect production inventory problem, Soft. comput. (2021) 1–23.
[19] C.Y. Lee, T.C.E. Cheng, B.M.T. Lin, Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem, Manag. Sci. 39 (5) (1993) 616–625.
[20] I.S. Lee, Minimizing total completion time in the assembly scheduling problem, Comput. Ind. Eng. (2018).
[21] I. Mahdavi, G.M. Komaki, V. Kayvanfar, Aggregate hybrid flowshop scheduling with assembly operations, in: Proceedings of the 18th IEEE, Conference on Industrial Engineering & Engineering Management (IE&EM), 2011, September, pp. 663–667.
[22] M.J. Mayer, A. Szilágyi, G. Gróf, Environmental and economic multi-objective optimization of a household level hybrid renewable energy system by genetic algorithm, Appl. Energy 269 (2020) 115058.
[23] M. NoParast, M. Hematian, A. Ashrafian, M.J.T. Amiri, H. AzariJafari, Development of a non-dominated sorting genetic algorithm for implementing circular economy strategies in the concrete industry, Sustain. Prod. Consum. 27 (2021) 933–946.
[24] H. Öztop, M.F. Tasgetiren, D.T. Eliiyi, Q. Pan, L. Kandiller, An energy- efficient permutation flowshop scheduling problem, Expert Syst. Appl. 150 (2020) 113279, doi:10.1016/j.eswa.2020.113279.
[25] K.E. Parsopoulos (Ed.), Particle Swarm Optimization and Intelligence: Advances and Applications: Advances and Applications, IGI global, 2010.
[26] M.L Pinedo, Scheduling: Theory, algorithms and systems, 4, 4th ed., Springer Science & Business Media, New York, 2012 Springer, doi:10.1007/978-1-4614-2361-4.
[27] C.N. Potts, S.V. Sevast'Janov, V.A. Strusevich, L.N. Van Wassenhove, C.M. Zwaneveld, The two-stage assembly scheduling problem: complexity and approximation, Oper. Res. 43 (2) (1995) 346–355.
[28] M. Rabiee, R.S. Rad, M. Mazinani, R. Shafaei, An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines, Int. J. Adv. Manuf. Technol. 71 (5) (2014) 1229–1245.
[29] R. Ruiz, Q.K. Pan, B. Naderi, Iterated Greedy methods for the distributed permutation flowshop scheduling problem, Omega 83 (2019) 213–222 (Westport).
[30] M.Y. Sabouni, R. Logendran, Lower bound development in a flow shop electronic assembly problem with carryover sequence-dependent setup time, Comput. Ind. Eng. (2018).
[31] S. Sheikh, G.M. Komaki, V. Kayvanfar, Multi objective two-stage assembly flow shop with release time, Comput. Ind. Eng. 124 (2018) 276–292.
[32] Z. Tajbakhsh, P. Fattahi, J. Behnamian, Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm, J Oper Res Soc 65 (10) (2014) 1580–1592, doi:10.1057/jors.2013.105.
[33] E. Vallada, F. Villa, L. Fanjul-Peyro, Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem, Comput. Oper. Res. 111 (2019) 415–424.
[34] G. Wang, Integrated supply chain scheduling of procurement, production, and distribution under spillover effects, Comput. Oper. Res. 126 (2021) 105105.
[35] Y.B. Woo, B.S. Kim, A genetic algorithm-based matheuristic for hydrogen supply chain network problem with two transportation modes and replenishment cycles, Comput. Ind. Eng. 127 (2019) 981–997.
[36] M. Yokoyama, Hybrid flow-shop scheduling with assembly operations, Int. J. Prod. Econ. 73 (2) (2001) 103–116.
[37] F. Zhao, L. Zhang, J. Cao, J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem, Comput. Ind. Eng. 153 (2021) 107082.
[38] Xi Sun, Kazuko Morizawa, Hiroyuki Nagasawa, Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling, European Journal of Operational Research Volume 146 (3) (2003) 498–516, doi:10.1016/S0377-2217(02)00245-X.
[39] M. Yokoyama, D.L. Santos, Three-stage flow-shop scheduling with assembly operations to minimize the weighted sum of product completion times European, Journal of Operational Research 161 (3) (2005) 754–770, doi:10.1016/j.ejor.2003.09.016.
[40] S. Lee, Li Lin, Ni Jun, Markov-based maintenance planning considering repair time and periodic inspection, Journal of Manufacturing Science and Engineering 135 (3) (2013) 031013.