International Journal of Industrial Engineering & Production Research (2018)

March 2019, Volume 30, Number 1
pp. 25-37

DOI: 10.22068/ijiepr.30.1.25 http://ijiepr.iust.ac.ir/

A Hybrid Genetic Algorithm and Parallel Variable Neighborhood Search for Job Shop Scheduling with an Assembly Stage

Parviz Fattahi*, Sanaz Keneshloo, Fatemeh Daneshamooz & Samad Ahmadi

Parviz Fattahi, *Professor, Department of Industrial Engineering, Alzahra University,Tehran, Iran.*
Sanaz Keneshloo*, Msc of Industrial Engineering, Bu-Ali Sina University, Hamedan, Iran.*
Fatemeh Daneshamooz, *PhD Student of Industrial Engineering, Bu-Ali Sina University, Hamedan, Iran.*
Samad Ahmadi, *Director of Uni-Soft Systems Ltd., Ingenuity Centre, University of Nottingham Innovation Park, Triumph Road, Nottingham, NG7 2TU, Department of Mathematics, University of Leicester, Leicester, LE1 7RH.*

KEYWORDS

Job shop;
Genetic Algorithm;
Parallel Variable Neighborhood Search.

ABSTRACT

In this research, a job shop scheduling problem with an assembly stage is studied. The objective function is to find a schedule that minimizes the completion time of all products. At first, a linear model is introduced to express the problem. Then, in order to confirm the accuracy of the model and to explore the efficiency of the algorithms, the model is solved by GAMS. Since the job shop scheduling problem with an assembly stage is considered as an NP-hard problem, a hybrid algorithm is used to solve the problem in medium to large sizes in a reasonable amount of time. This algorithm is based on genetic algorithm and parallel variable neighborhood search. The results of the proposed algorithms are compared with those of genetic algorithm. Computational results showed that, for small problems, both HGAPVNS and GA have approximately the same performance. In addition, in medium to large problems, HGAPVNS outperforms GA.

1. Introduction

Scheduling is one of the most important issues in the planning and operation of production systems [1] and plays an essential role in manufacturing systems [2]. In this paper, a two-stage assembly scheduling problem, named job shop scheduling problem, with an assembly stage is studied in which the first stage is solved as a job shop problem and, then, appends an assembly stage to produce products. In the job shop scheduling problem, a finite set of jobs is processed on a finite set of machines. Each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specified duration. Each machine can process at most one job at a time; once, a job initiates processing on a given machine, it must complete processing uninterrupted. The objective of the JSP is to find a schedule that minimizes makespan of the jobs.

The job shop scheduling problem has been considered a difficult combinatorial optimization problem since the 1950s [3]. Gray et al. [REF] showed that the jobshop scheduling problem was NP-hard. The first serious application of GAs to solving the JSSP was proposed by Nakano and Yamada [4]. Gen et al. [5] proposed a new method for solving JSP using genetic algorithm. They also used permutation representation for the first time in order to produce better results. Dorndorf and Pesch [6] described a class of approximation algorithms for solving the minimum makespan problem of jobshop

* **Corresponding author:** ***Parviz Fattahi***
Email: p.fattahi@alzahra.ac.ir
Received 13 May 2017; revised 02 December 2018; accepted 19 December 2018

scheduling. Generic algorithm was the basis of the proposed algorithms. They also considered sequences of dispatching rules and shifting bottleneck procedure. Moonen et al. [7] presented a Giffler-Thompson Focused Genetic algorithm in order to solve static job shop problems. The design of the genetic algorithms for the considered problem varies in the process of encoding a solution and in the process of its various operators. In this design of the genetic algorithm, a crossover operator is developed based on the principles set by Giffler and Thompson to generate active schedules. Sha and Hsu [8] proposed a hybrid particle swarm optimization (PSO) for the job shop problem. Since the solution space of the JSP is discrete, they modified the particle position representation, particle movement, and particle velocity to better suit PSO for the JSP. They modified the particle position based on preference list-based representation, particle movement based on swap operator, and particle velocity based on the tabu list concept. Moreover, they used Giffler and Thompson's heuristic to decode a particle position into a schedule. The computational results showed that the modified PSO performed better than the original design and that the hybrid PSO was better than other traditional meta-heuristics. Roshanaei et al. [9] considered the problem of scheduling a jobshop where setup times were sequence-dependent in minimizing the maximum completion times of operations. They employed a recent effective meta-heuristic algorithm, known as a variable neighborhood search, to solve the problem. The obtained results strongly support the high performance of the proposed algorithm. Fakhrzad et al. [10] presented a new multi-objective jobshop scheduling with sequence-dependent setup times. They proposed an efficient multi-objective hybrid genetic algorithm. They took a variable neighborhood search algorithm as a local improving procedure. In addition, they assigned fitness-based dominance relation. The computational results showed that the proposed HGA outperformed the SPEAII algorithm. Nowadays, according to the development of technology and diversification of customer needs, approaches to manufacturing and assembly due to customer orders have increased. Global competition and the need to control the production cost force companies to design products with modular structures. Thus, considering different stages of production simultaneously in planning and scheduling is very important. The first paper about the two-stage scheduling problems was published by lee et al. [11]. They described an application of two-stage assembly problem in a fire engine assembly plant. Moreover, Potts et al. [12] described an application of the two-stage assembly problem in personal computer manufacturing. Jabbari [13] studied a flow-shop scheduling problem with a parallel assembly stage. The objective function of the problem is to minimize the completion time of products. They suggested a hybrid algorithm for solving the problem. Computational results showed that the suggested algorithm outperformed genetic algorithm and particle swarm optimization algorithm. Wong and Ngan [14] studied the assembly jobshop scheduling with lot steaming. In this study, the system objective is defined as the makespan minimization. In addition, part sharing is used to differentiate product-specific components from common components. In order to solve the problem, they proposed a hybrid genetic algorithm (HGA) and a hybrid particle swarm optimization (HPSO). Computational results showed that HGA significantly outperformed HPSO. Cheng [15] presented an approximation method to estimate the mean of a standard deviation of job flowtime in a dynamic jobshop with assembly operations. The accuracy of the method in predicting the flow times of jobs with varying structural complexities was assessed through computer simulation. Daneshamoz et al. [16] proposed a linear model for jobshop scheduling with a parallel assembly stage in order to minimize makespan. In addition, they suggested a particle swarm optimization algorithm to solve a problem on a large scale. Their results showed that the suggested algorithm could reach near-optimal solutions in various dimensions of the problem. Sculli [17] studied the priority dispatching rules in jobshop scheduling problem with assembly operations and random delays. Zhang [18] proposed a genetic algorithm with a rule-based encoding scheme and a simulation-based solution evaluation method for solving the jobshop scheduling problem involving assembly operation. The computational experiments on a set of randomly generated test instances showed that the proposed algorithm is effective. Dimyati [19] addressed a problem of scheduling in a made-to-order jobshop with product assembly consideration. A mixed integer linear programming model was developed to solve the

model with the objective of makespan minimization.

According to the literature review, scarce investigation has been carried out on scheduling jobshop with an assembly stage in comparison to the job shop scheduling and other production systems. Since the jobshop scheduling problem with an assembly stage is a generalization of classical jobshop scheduling, this problem is at least as hard as classical jobshop scheduling. Therefore, it belongs to the NP-hard class. Therefore, it is necessary to utilize new approaches and meta-heuristics to solve the problem on medium to large scales.

The structure of this paper is organized in five sections. In the next section, the problem background and formulation are presented. Section 3 describes the proposed algorithm. Computational results are reported in Section 4. Finally, Section 5 discusses the results and concludes the paper with recommendations for future research.

2. Problem Description

This paper considers a two-stage assembly scheduling problem containing a jobshop scheduling stage and an assembly stage. In this problem, several products must be produced, and each product is made by assembling a set of several different jobs. At first, the parts are manufactured in the jobshop stage. A jobshop consists of m machines ($M_1, M_2, \dots, M_m$) that perform operations on n jobs ($J_1, J_2, \dots, J_n$) in different sizes. After manufacturing the parts, they are assembled into the products. The objective of the paper is to minimize the makespan.

2-1. Assumptions

- ✓ During the time horizon of the schedule, machines are available
- ✓ All of the operations are available at time zero
- ✓ Demand for final products is specified
- ✓ An operation once started, cannot be interrupted
- ✓ Each product consists of a list of specific jobs, and each job consists of a specific operation
- ✓ The processing times and assembly times are fixed and specified
- ✓ Setup times are included in the processing times
- ✓ When all of the jobs of a product have completed processing at the first stage, they become available for processing on the second stage
- ✓ Each machine can handle only one operation at a time
- ✓ Each operation can be processed only on one machine

2-2. Notations

P	Total number of products
p	Products index
n	Total number of jobs
J	Jobs index (1,…,n)
n_p	Number of sub-jobs of product p
J_p	Sub-jobs of product p index
m	Total number of machines at stage one
i	Machines index at stage one
$O_{j,p,h}$	Operation h of job j of product p
$t_{j,p,h}$	Starting time of operation $O_{j,p,h}$
$p_{j,p,h}$	Processing time of operation $O_{j,p,h}$
k_i	Number of operations dedicated to machine i
F_p	Completion time of all jobs belong to product p (at stage one)
C_p	Completion time of product p
A_p	Assembly time of product p
St_P	Starting assembly time of product p
$Sm_{k'}$	Starting time of assembly machine in turns k'
$Tm_{i,k}$	Starting time of machine i in turns k
$a_{i,j,p,h}$	1 if operation $O_{j,p,h}$ is processed on machine i; 0,otherwise
$x_{i,j,p,h,k}$	1 if operation $O_{j,p,h}$ is processed on machine i in turns k; 0,otherwise
$Z_{p,k'}$	1 if product p is assembled in turns k'

2-3. Mathematical model

The mathematical model of problem is presented as follows:

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

Subject to:

$$c_{max} \geq C_p \quad p=1,2,3,\ldots,P \quad (2)$$

$$t_{j,p,h} + p_{j,p,h} \leq t_{j,p,h+1} \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots,h_j-1 \end{matrix} \quad (3)$$

$$Tm_{i,k} + p_{j,p,h}.x_{i,j,p,h,k} \leq Tm_{i,k+1} \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots,h_j \\ k=1,2,3,\ldots,k_i-1 \\ i=1,2,3,\ldots,m \end{matrix} \quad (4)$$

$$Tm_{i,k} \leq t_{j,p,h} + (1 - x_{i,j,p,h,k}).L \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots h_j \\ k=1,2,3,\ldots,k_i \\ i=1,2,3,\ldots,m \end{matrix} \quad (5)$$

$$Tm_{i,k} + (1 - x_{i,j,p,h,k}).L \geq t_{j,p,h} \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots,h_j \\ k=1,2,3,\ldots,k_i \\ i=1,2,3,\ldots,m \end{matrix} \quad (6)$$

$$\sum_p \sum_j \sum_h x_{i,j,p,h,k} = 1 \quad \begin{matrix} k=1,2,3,\ldots,k_i \\ i=1,2,3,\ldots,m \end{matrix} \quad (7)$$

$$\sum_k x_{i,j,p,h,k} = a_{i,j,p,h} \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots,h_j \\ i=1,2,3,\ldots,m \end{matrix} \quad (8)$$

$$t_{j,p,h} + p_{j,p,h} \leq F_p \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \end{matrix} \quad (9)$$

$$F_p \leq St_p \quad p=1,2,3,\ldots,P \quad (10)$$

$$A_p + St_p \leq C_p \quad p=1,2,3,\ldots,P \quad (11)$$

$$Sm_{k'} + A_p.Z_{p,k'} \leq Sm_{k'+1} \quad \begin{matrix} p=1,2,3,\ldots,P \\ k'=1,2,3,\ldots, \\ k'_{i'}-1 \end{matrix} \quad (12)$$

$$Sm_{k'} \leq St_p + (1 - Z_{p,k'}).L \quad \begin{matrix} p=1,2,3,\ldots,P \\ k'=1,2,3,\ldots,k'_{i'} \end{matrix} \quad (13)$$

$$Sm_{k'} + (1 - Z_{p,k'}).L \geq St_p \quad \begin{matrix} p=1,2,3,\ldots,P \\ k'=1,2,3,\ldots,k'_{i'} \end{matrix} \quad (14)$$

$$\sum_{k'} Z_{p,k'} = 1 \quad p=1,2,3,\ldots,P \quad (15)$$

$$x_{i,j,p,h,k \in \{0,1\}} \quad \begin{matrix} p=1,2,3,\ldots,P \\ j=1,2,3,\ldots,n_p \\ h=1,2,3,\ldots,h_j \\ k=1,2,3,\ldots,k_i \\ i=1,2,3,\ldots,m \end{matrix} \quad (16)$$

$$Z_{p,k' \in \{0,1\}} \quad \begin{matrix} p=1,2,3,\ldots,P \\ k'=1,2,3,\ldots,k'_{i'} \end{matrix} \quad (17)$$

$$C_p \geq 0 \quad p=1,2,3,\ldots,P \quad (18)$$

The objective function (1) indicates the minimization of makespan. Constraint (2) expresses that the makespan is not smaller than the completion time of any product. Constraint (3) ensures a precedence relationship. Constraint (4) expresses that a machine at stage one is not able to process an operation in turns k+1 before processing the operation in turns k. Constraints (5 and 6) state that an operation cannot start before completing its preceding operation; in addition, they satisfy machine constraint. Constraints (7 and 8) define the sequence restrictions. Constraint (9) shows the maximum processing time of jobs of a product. Constraint (10) defines the earliest start time of assembly stage. Constraint (11) shows the completion time of products. Constraint (12) expresses that the machine in the assembly stage first assembles the jobs of product in priority k; then, it starts assembling the jobs of product in priority k+1. Constraints (13 and 14) express that an operation cannot start before completing its preceding operation in the second stage. Constraint (15) dedicates only an operation of a job to each machine in each turn. Constraints (16 and 17) enforce the binary requirements of the decision variables. Constraint (18) implies that the completion time of operations must be positive.

2-4. Numerical example

In this section, an example is given to report the performance of the proposed model. This example illustrates a jobshop scheduling problem with an assembly stage. It is assumed that there are 2 products, 3 jobs, 3 machines in the first stage, and only one machine in the second stage. The processing time of the operations on the machines and the assembly time of the products are listed in Table 1. In this example, the machines are selected randomly, and the related processing times are highlighted in Table 1. Then, a feasible sequence is generated, and its corresponding Gantt chart is presented in Figure 1.

3. Proposed Algorithm

The jobshop scheduling problem with an assembly stage that is an extension of the classical jobshop model known as a NP-hard problem. According to the proposed model and its several constraints and also due to the complexity of this type of problem, much time is required for solving by exact algorithms. Therefore, in order to save time in solving larger real-world problems, a hybrid algorithm is suggested. This algorithm is composed of Genetic algorithm and parallel variable neighborhood search to benefit from the advantages of both types of algorithms, which are exploration ability of genetic algorithms and exploitation capability of parallel variable neighborhood search.

Tab. 1. Processing time of the operations on the machines

Product	Part	Operation	Machine M_1	Machine M_2	Machine M_3	Assembly time
1	1	1	15			20
		2			5	
		3		9		
	2	1		*11*		
		2	6			
		3	8			
	3	1			10	
		2		13		
2	1	1			20	18
		2	*17*			
	2	1		*10*		
		2	12			
	3	1			8	
		2			13	
		3		7		

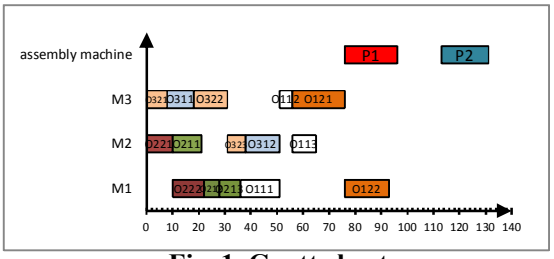

Fig. 1. Gantt chart

3-1. Genetic algorithm

Genetic algorithm is a population-based heuristic search methodology designed to mimic the natural process of evolution. This algorithm is commonly used to generate useful solutions to optimization and search problems, often by emulating a similar technique from Biology such as inheritance, mutation, selection, and crossover [20]. John Holland developed the formal theory of GA in the 1960s, and continued improvements in the performance value have made GA attractive for many problem-solving optimization methods [21]. GA is shown to perform well in mixed (i.e. continuous and discrete) combinatorial problems. However, it easily becomes trapped in local optima [22]. A GA begins with a set of solutions represented by a group of chromosomes called the population. A

new population can be generated by applying genetic operators, such as selection, crossover, and mutation, to current population.

3-2. Parallel variable neighborhood search

Variable neighborhood search (VNS) is a meta-heuristic that exploits systematically the idea of neighborhood change both in descent to local minima and in escape from the valleys that contain them. Mladenovi´c and Hansen [24] developed the variable neighborhood search algorithm in 1997. In the first step of PVNS, a set of neighborhood structures and the sequence of their implementations are determined. The application of the parallelism to a meta-heuristic can and must allow reducing the computational time or increasing the exploration in the search space. In this algorithm, parallelization is based on the application of multiple independent searches, increasing the exploration in the search space. In the initialization step of the PVNS, the neighborhood structures needed for searching the solution space are identified. In this step, a stopping condition is delineated. In addition, an initial solution is generated and is set as the current solution, y. The main part of the algorithm is comprised of internal and external loops. The internal loop is responsible for searching the solution space, whereas the external loop controls the stop condition of algorithm. The symbol k is delineated as the iteration counter of the internal loop. There are a number of processors in the internal loop that are used in the search process. In each iteration of the internal loop, every processor, with regard to input solution, performs a single iteration of a sequential VNS method including shake and local search procedures. Neighborhood structures related to N_k^S are utilized in the shake procedure and Neighborhood structures related to N_l^{ls} are employed in local search procedure. Supposedly, in this section, complete execution of shake and local search procedures via a processor is called 'run', and one completed set of runs by all processors is named as 'generation'. Therefore, in each iteration of the internal loop, one generation should be performed by processors.

In every generation, input solution for each processor equals the current solution, $\tilde{y}$. At the beginning of a generation, each processor employs a shake procedure on the input solution. In the next step, considering the solution obtained from the shake procedure, each processor should do a local search. By the commencement of the local search procedure, l and n (iteration counter of local search) begin with 1. Then, h_{th} processor, $pr_{(h)}$, employs the first neighborhood structure related to local search procedure, N_l^{ls}, to generate neighboring solution y_p from input solution y'_h, resulting from the shake procedure. In this case, y_p is compared with y'_h. If y_p is better than y'_h or both of them are equal, y'_h replaces y_p. However, when y_p is worse than y'_h, y'_h is preferred and one unit is added to the counter ($l \leftarrow l + 1$) and the next neighborhood structure is activated. When the first iteration of the local search is accomplished, one unit is added to counter n, and the next iteration begins. During the local search procedure, if Counter l equals $l_{max} + 1$, the first neighborhood structure is used again. After all processors complete a set of runs, one generation gets finished, and the updating step then commences. At the beginning of this step, coordinating process among processors is done by a central unit. Once each processor completes a run, it reports the obtained solution to the central unit. Then, in the updating step, the central unit identifies the best solution (y'') among obtained solutions of processors. In the rest of this step, y'' is compared with $\tilde{y}$ in terms of the solution quality. If y'' is better than $\tilde{y}$, $\tilde{y}$ replaces y''. Further, the search process begins again at the first iteration of the internal loop. Otherwise, one unit is added to the iteration counter. On the condition that all iterations of internal loop are considered, one iteration of PVNS algorithm is accomplished. In that case, a stopping condition is checked by the external loop. If the stopping condition is not met, the next iteration of algorithm begins. The PVNS algorithm continues until the stopping condition is satisfied. When the process of PVNS algorithm stops, the final current solution is used as the best solution, y^*.

3-3. Hybrid genetic algorithm and parallel variable neighborhood search

Considering exploration ability of genetic algorithms and exploitation capability of parallel variable neighborhood search, a new hybrid algorithm is presented in this subsection. HGAPVNS runs the genetic algorithm as the main algorithm and uses the PVNS procedure for improving individuals in the population. Each individual in the population is used to generate new offspring by applying the appropriate genetic operators such as selection, crossover, and

mutation. The fitness function of the individuals will be computed; finally, the best solution obtained by genetic algorithm will be improved by PVNS. The HGAPVNS is also a population-based algorithm. The initial population is generated based on a dispatching rule.

3-3-1. Solution representation

In this paper, operation-based representation (OB) provided by Gen et al. [25] is used to represent the proposed algorithm solutions. This representation uses a single string of genes, where each job is represented by a number of genes equal to the number of operations it constrains. This kind of representation always generates feasible schedules. In this paper, another string of genes is added below the string of operations in order to show the number of products. Figure 2 shows an operation-based representation. The first row belongs to the jobs, and the second row shows the products. For example, Job 1 of Product 1 has two operations that are in the first and third priorities of the process.

1	2	1	1	2	3	1
1	1	1	2	2	1	2

Fig. 2. Operation based representation

3-3-2. Initial solution

In general, a random solution is used as the initial solution. In this paper, a famous rule, namely shortest processing time (SPT), is used to make initials. According to this rule, operations with shortest processing times are scheduled first.

3-3-3. Selection operator

In the current study, a roulette wheel selection is used as a selection operator. Based on this selection, each member of the population is assigned to a segment of the wheel with size proportional to its fitness. Then, individuals are selected randomly from the wheel. Therefore, the fitter individuals are more likely to be selected.

3-3-4. Crossover operator

Spear and De Jong (1991) investigated different crossovers, and indicated that uniform crossover produced the best results. Unlike the one- and two-point crossovers, the uniform crossover can develop offspring in each point of space. Based on their study, the uniform crossover has more exploratory power than the n-point crossover [26]. Therefore, uniform crossover is used in this paper. In the uniform crossover, a random vector containing numbers between [0,1] is generated. Offspring 1 is produced by taking the bit from Parent 1 if the corresponding mask bit is 1 or the bit from Parent 2 if the corresponding mask bit is 0; Offspring 2 takes the bit from Parent 1 if the corresponding mask bit is 0.

3-3-5. Mutation operator

In genetic algorithm, mutation is a random deformation of genes with a certain probability [27]. The mutation operator preserves diversification in the search [28]. In this study, two random positions of the string are chosen, and the bits corresponding to those positions are interchanged.

3-3-6. Neighborhood structures

The technique of moving from one solution to its neighboring solution is delineated by a key factor known as neighborhood structure. In this subsection, four types of neighborhood structures employed in the proposed algorithm are presented.

The first neighborhood structure produces a number randomly between 2 and number of operations. Then, the value of the selected operation, i.e., k, will be replaced with the $k-1_{th}$ operation if the values of these two operations be different at least in one row. Otherwise, it will be replaced with the $k+1_{th}$ operation.

In the second neighborhood structure, two operations are randomly chosen; then, all genes with the same value of these two operations will be selected; finally, they will be interchanged.

In the third type of neighborhood structure, two operations will be chosen randomly whose values will be replaced.

The last neighborhood structure acts the same as previous neighborhood structure; in addition, it will interchange value of the genes between these two selected operations.

4. Computational Experiments

In this section, the performance and effectiveness of the proposed algorithms to solve the problem are investigated. The mathematical model is solved by GAMS, and the proposed algorithms are coded by MATLAB R2011a. Due to the lack of information on solved problems in similar papers, in order to validate the proposed model, some random samples are used. The samples are classified into three groups (small-, medium-, and large-sized instances) based on the number of processed operations. To compare the solving

techniques, Relative Percentage Deviation (RPD) factor is used. It can be formulated as follows:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} * 100 \quad (19)$$

where $\mathrm{Alg_{sol}}$ is the makespan of a specific individual obtained by considering algorithm, and $\mathrm{Min_{sol}}$ is the minimum makespan of that specific individual obtained by the algorithms. In addition, IMP% is used to measure the percentage of solution improvements.

$$Imp = \frac{Alg_{initial\,sol} - Alg_{final\,sol}}{Alg_{final\,sol}} * 100 \quad (20)$$

Moreover, the mean deviation of the best solution (D_{f^*}) is evaluated. This factor is computed according to the following equation.

$$D_{f^*} = \frac{\sum_{i=1}^{n}(f_i - f^*)}{n.f^*} \quad (21)$$

where f^* is the solution obtained by the mathematical model for small problems. Further, for medium- and large-sized problems, it is the best solution obtained by the algorithms.

Since the parameters of the meta-heuristic algorithms have a great influence on their effectiveness and the purpose of the parameter design is to bring quality into the process by determining the relevant parameters, in this paper, Taguchi's method has been used to determine the appropriate values for key parameters of the proposed algorithms. The results of Taguchi's method are shown in Table 2.

Tab. 2. Parameters value

algorithm	parameter	value
HGAPVNS	Npop	60
	Crossover Rate	0.5
	Mutation Rate	0.3
	Nmax	5
	Kmax: maximum number of internal loops	4
	Npr: number of processors	3

Then, the optimal solutions for small-sized problems are obtained by GAMS and reported in Table 3. The algorithms achieved the same results, yet the computational time of the algorithms increasingly is considerably lower than the time obtained by GAMS. Figure 5 illustrates the IMP values. It is noticeable that high values of improvement demonstrate the efficiency of the algorithms. Thus, according to Figure 4, for small-sized problems, the HGAPVNS algorithm outperforms GA.

In medium-sized problems, GAMS was not able to obtain the optimal solution in an acceptable amount of time. Due to the RPD values with a 95% confidence interval, in Table 4 and Figure 3, the HGAPVNS performs better than GA, statistically. Moreover, according to Table 4 and Figure 5, the GA has made greater improvements in solutions.

In large-sized problems, according to the RPD values in Table 5 and Figure 4, the HGAPVNS outperforms GA. Moreover, by considering the IMP values, the HGAPVNS is superior to GA.

As mentioned above, Figure 5 shows the performance of the algorithms in solution improvements. Generally, the IMP values have an ascending trend by increasing the size of problems. In small- and medium-sized problems, the HGAPVNS outperforms the GA. However, for large problems, the GA performs better. No improvements in small problems indicate that the algorithms can meet the optimal solution in the first iteration.

For both of the algorithms, the summarized averages of D_{f^*} are shown in Figure 7. It is concluded that an increase in the size of problems leads to an increase in D_{f^*}. According to Figure 7, the HGAPVNS algorithm performs better than GA in all small-, medium-, and large-sized instances. On the other hand, the convergence of HGAPVNS and GA is shown in Figure 8. Based on the Figure, the best objective value found by the algorithms was compared across generations. The results revealed that the convergence of HGAPVNS was relatively better than the other algorithm.

5. Conclusion

In this paper, a jobshop scheduling problem with an assembly stage was considered. A linear model was presented that attempted to minimize the makespan time. According to the complexity of this problem, the meta-heuristic algorithms were used to solve the problem for medium- to large-sized instances. This algorithm is based on a GA and a parallel variable neighborhood search. Furthermore, to evaluate the performance of the proposed algorithm, it was compared to GA. The results revealed that, for small-sized problems, both algorithms had approximately the same performance with each other. For medium- and large-sized problems, the HGAPVNS

outperformed the GA; however, the HGAPVNS in most instances needed more computational time. Several topics can be proposed for future research. For example, setup times, intermediate buffer, and uncertainty for the parameters of the problem are considered. In addition, in order to improve the quality of HGAPVNS solutions, it is recommended saving and improving half of the best solutions obtained by GA with PVNS instead of improving only the best solution.

Tab. 3. Computational results (small problems)

Instance		GAMS			HGAPVNS			GA		
		Optimal solution	CPU time	RPD	CPU time	IMP	RPD	CPU time	IMP	RPD
1	2*2*2	360	7	0	6.82	0	0	4.87	0	0
2	2*2*3	480	9.6	0	11.01	0	0	5.64	0	0
3	2*3*3	500	12	0	13.92	0	0	7.06	0	0
4	3*2*2	380	14.4	0	11.44	2	0	7.78	0	0
5	3*3*2	560	15.5	0	15.68	1.84	0	8.56	0	0
6	3*2*3	680	31	0	18.25	0	0	11.7	0	0
7	3*3*3	700	55.5	0	16.52	2.5	0	9.4	0	0
8	4*2*3	880	639	0	17.64	4	0	15.14	0	0

Tab. 4. Computational results (medium problems)

Instance		HGAPVNS			GA		
		CPU time	IMP	RPD	CPU time	IMP	RPD
1	5*3*2	20.19	0	0	11.61	0	0
2	5*4*3	55.58	3	0	20.32	2.56	5.5
3	4*3*3	26.52	1.8	0	17.02	0	2.5
4	4*4*3	30.62	8.5	0	19.1	9.37	3.57
5	4*4*4	45.15	2	0	26.9	0	5
6	5*5*6	121.1	18.8	0	82.07	14.4	0.48
7	5*6*6	135.9	9.11	0	98.6	9.61	3.29
8	5*6*7	180.11	10	0	123.04	8.57	2.12
9	4*4*6	112.25	11.3	0	110.11	0	2.98

Tab. 5. Computational results (large problems)

Instance		HGAPVNS			GA		
		CPU time	IMP	RPD	CPU time	IMP	RPD
1	7*5*6	177.9	15.5	0	135.7	4.76	0.84
2	6*5*8	225	3	0	161.5	2.38	0.81
3	7*6*6	237.6	1.18	0	170.37	6.97	2.54
4	4*6*6	119	5.6	0	64.6	12.9	0.75
5	7*6*7	339	8.8	0	224.1	14.2	4.79
6	9*5*7	419	3.2	0	261.9	4.65	0.61
7	8*5*8	545	0.84	0	281.4	4.14	1.64
8	10*5*8	567	2.2	0	434	1.25	0.42

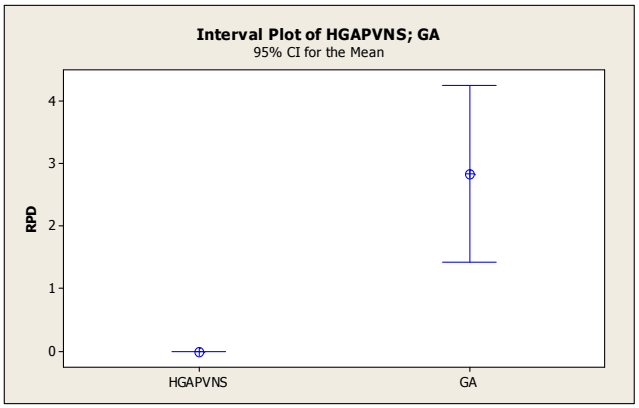

Fig. 3. Interval plot of RPD value for medium problems

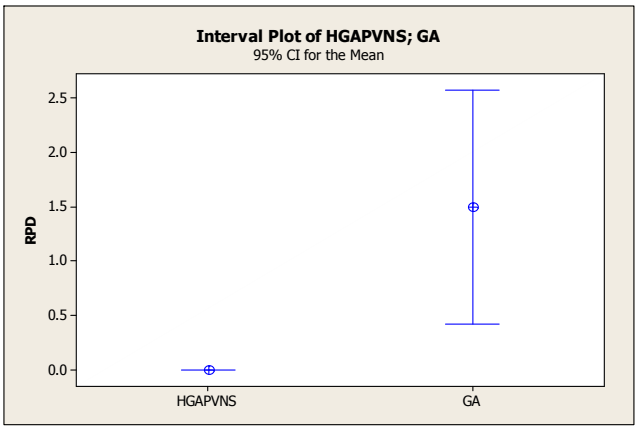

Fig. 4. Interval plot of RPD value for large problems

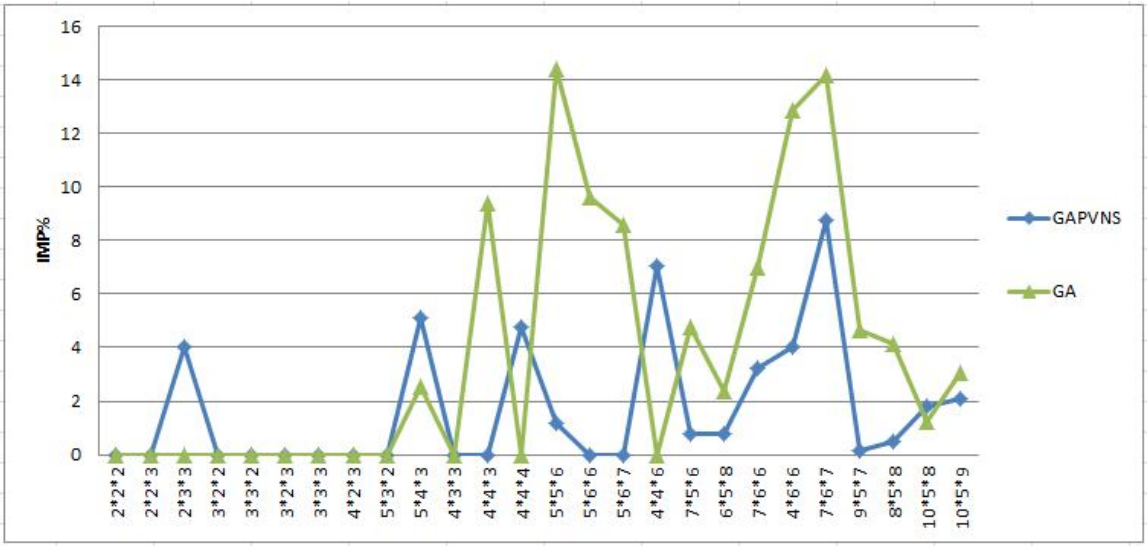

Fig. 5. Improvements in initial solution

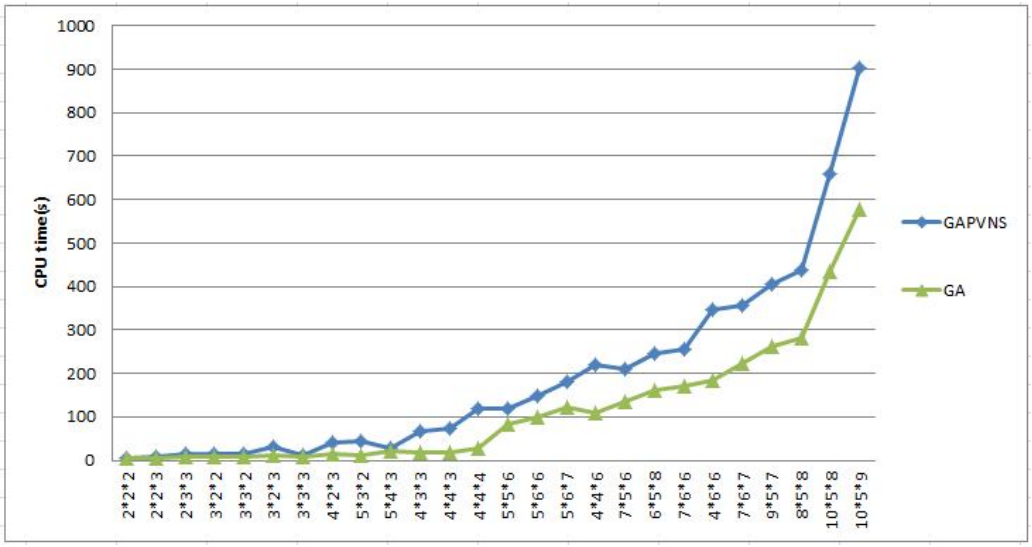

Fig. 6. Computing time of algorithms

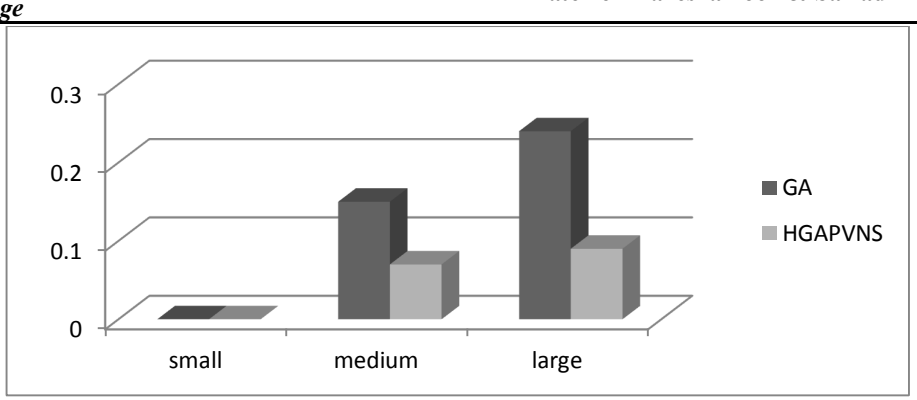

Fig. 7. Averages of D_{f^*} for the problems

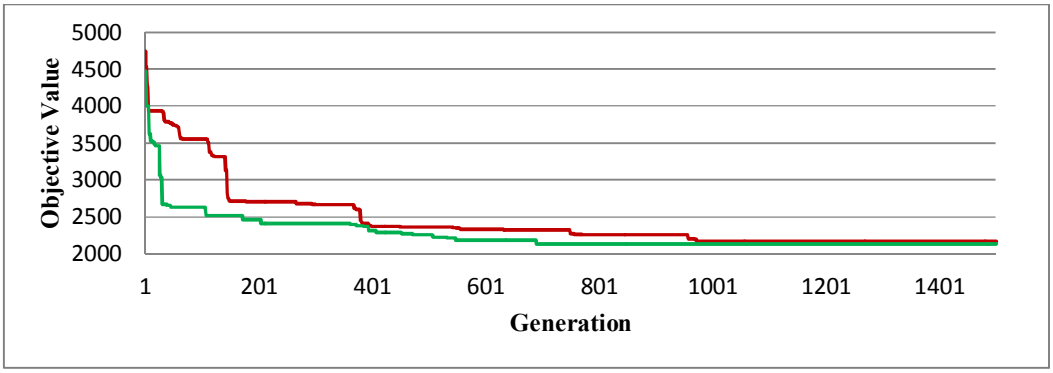

Fig. 8. Convergence of the proposed algorithms

References

[1] Cummings, D.H., Egbelu P.J., "Minimizing production flow time in a process and assembly jobshop," International Journal of Production Research, Vol. 36, No. 3, (1998), pp. 2315–2332.

[2] Sangsawang, C., Sethanan, K.., Gen, M., Fujimoto T., "Metaheuristics optimization approaches for two-stage reentrant flexible flow shop with blocking constraint ," Computer & Operation Research, Vol. 42, (2015), pp. 2395-2410.

[3] Zhang, R., Cheng, W., "A simulated annealing algorithm based on blocking properties for the jobshop scheduling problem with total weighted tardiness objective," computer and operation research, Vol. 38, (2011), pp. 854-867.

[4] akano, R., Yamada, T., "Conventional genetic algorithm for job-shop problem," In Proc. of 4th ICGA, (1991), pp. 477–479.

[5] Gen, M., Tsujimura, Y., Kubota, E., "Solving job-shop scheduling problems by genetic algorithm," Proceeding of the IEEE international conference on systems, (1994), pp. 1577-1582.

[6] Dorndorf, U., Pesch, E., "Evolution based learning in a job shop scheduling environment," European Journal of Operational Research, Vol. 22, (1995), pp. 25-40.

[7] Moonen, M., Janssesens, G., "A Giffler-Thompson focused genetic algorithm for the static jobshop scheduling problem," (2000).

[8] Sha, D.Y., Hsu, C.Y., "A hybrid particle swarm optimization for job shop scheduling problem," Computer & Industrial Engineering, Vol. 51, (2006), pp. 791-808.

[9] Roshanaei, V., Naderi, B., Jolai, F., Khalili, M., "A variable neighborhood

search for job shop scheduling with set-up times to minimize makespan," Future Generation Computer Systems, Vol. 25, (2009), pp. 654-661.

[10] Fakhrzad, M.B., Sadeghieh, A., Emami, L., "A new multi-objective job shop scheduling with setup times using a hybrid genetic algorithm," International Jounal of Engineering, Vol. 26, (2007), pp. 207-218.

[11] ee, C.Y., Cheng, T.C.E., Lin, B.M.T., "Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem," Management Science, Vol. 39, (1993), pp. 616-625.

[12] Potts, C.N., Sevast'Janov, S.V., Strusevich, V. A., Van Wassenhove, L.N., Zwaneveld, C.M., " The two-stage assembly scheduling problem Complexity and approximation," Operations Research, Vol. 43, (1995), pp. 346-355.

[13] Jabbari, M., "A meta-heuristic algorithm for flowshop scheduling problem with a parallel assembly stage," Master of Science Thesis, Buali-Sina University, (2013).

[14] Wong, T.C., Ngan, S.C., "A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop," Applied Soft Computing, Vol. 13, (2013), PP. 1391–1399.

[15] Cheng, T.C.E., "Analysis of material flow in a job shop with assembly operations," INT. J. PROD. RES, Vol. 28, 1990, pp. 1369-1383.

[16] Daneshamooz, F., "A new model for jobshop scheduling problem with a parallel assembly stage," Master of Science Thesis, Buali-Sina University, (2013).

[17] Sculli, D., "Priority Dispatching Rules in Job Shops with Assembly Operations and Random Delays," OMEG;4 The Int. JI of Mgmt Sei, Vol. 8, (1979), pp. 227-234.

[18] Zhang, R., "A Simulation-based Genetic Algorithm for Job Shop Scheduling with Assembly Operations," International Journal of Advancements in Computing Technology, (2011).

[19] Dimyati, T., "Minimizing production flow time in a process and assembly job shop," Proceedings of the International Seminar on Industrial engineering and Management, Jakarta, (2007), pp. 68-73.

[20] Melanie, M., "An Introduction to Genetic Algorithms," MIT Press, (1996).

[21] Omar, M., Baharum, A., AbuHasan, Y., "A job-shop scheduling problem (jssp) using genetic algorithm," The 2nd IMT-GT Regional Conference on Mathematics, Statistics and Applications, (2006), pp. 13-15.

[22] Abu-Srhan, A., Al daoud, E., "A hybrid algorithm using genetic algorithm and cuckoo search algorithm to solve the traveling salesman problem and its application to multiple sequence alignment," International Journal Of Advanced Science and technology, Vol. 61, pp. 29-38.

[23] Manar, H., Shameem, F., "A Survey of Genetic Algorithms for the University Timetabling Problem," International Conference on Future Information Technology IPCSIT, Vol. 13, (2011).

[24] Mladenovi´c, N., Hansen, P., "Variable neighborhood search," Computers and Operations Research, Vol. 11, (1997), pp. 1097–110.

[25] Gen, M., Tsujimura, Y., Kubota, E., "Solving job-shop scheduling problems by genetic algorithm," Proceeding of the IEEE international conference on systems, (1994), pp .1577-1582.

[26] Eschelman, L., Caruana, R., Schaffer, D., "Biases in the crossover landscape," Proc. Third international conference on genetic

algorithms, Morgan Kaufman Publishing, (1989), pp. 21-29.

[27] Sharapov, R.R., "Genetic algorithms: Basic ideas, variants and analysis, Vision system: Segmentation and pattern recognition," Goro Obtained and Ashish Dutta (Ed), (2007), pp. 407-422.

[28] Magalhaes-Mendes, J., "A comparative study of crossover operators for genetic algorithms to solve the jobshop scheduling problem," Wseas Transaction on computers, Vol. 16, (2013).

Follow This Article at The Following Site:

Fattahi P., Keneshloo S., Daneshamooz F., A Hybrid Genetic Algorithm and Parallel Variable Neighborhood Search for Jobshop Scheduling With an Assembly Stage. IJIEPR. 2019; 30 (1) :25-37
URL: http://ijiepr.iust.ac.ir/article-1-752-en.html