Scientia Iranica E (2013) 20(6), 2275–2283

Sharif University of Technology
Scientia Iranica
Transactions E: Industrial Engineering
www.scientiairanica.com

Three meta-heuristics to solve the no-wait two-stage assembly flow-shop scheduling problem

A. Mozdgir[a], S.M.T. Fatemi Ghomi[b,*], F. Jolai[c] and J. Navaei[b,d]

a. *Payame Noor University, Tehran, 19395-4697, Iran.*
b. *Department of Industrial Engineering, Amirkabir University of Technology, Tehran, 424, Iran.*
c. *Department of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran.*
d. *Intelligent Manufacturing Systems (IMS) Centre, University of Windsor, Windsor, Ontario, Canada.*

Received 21 November 2012; accepted 11 May 2013.

KEYWORDS
No-wait assembly flow-shop;
Genetic algorithm;
Differential evolution algorithm;
Population-based variable neighborhood search.

Abstract. This paper addresses the No-Wait Two-Stage Assembly Flow-shop Scheduling Problem (NWTSAFSP) with the objective of makespan minimization. The problem is a generalization of previously proposed general problem in the Two-Stage Assembly Flow Shop Scheduling Problem (TSAFSP). The TSAFSP is NP-hard, thus the NWTSAFSP is NP-hard too, and three meta-heuristic algorithms, namely, Genetic Algorithm (GA), Differential Evolution Algorithm (DEA) and Population-based Variable Neighborhood Search (PVNS) are proposed in this article to solve this problem. Computational results reveal that PVNS outperforms other algorithms in terms of average error and average Coefficient of Variation (CV). Nevertheless, GA has the least run time among the proposed algorithms.

1. Introduction

With the advent of global markets and product ordering systems, manufacturing firms are more and more being dragged into a contest in which cutting cost and reducing production span are vital traits of dominant companies. For this reason, production planning and scheduling are of a foremost importance for such companies. By making wise plans and schedules, manufacturing firms can reduce their redundancy times and thus reduce their current costs. In pursuing this goal, abundant researches have addressed flow-shop, job-shop and other kinds of scheduling problems. One of these problems which most resembles the manufacturing processes in reality is the Two-Stage Assembly Flow-shop Scheduling Problem (TSAFSP) in which in the first stage parts are processed in parallel, and then in the second stage, the final product is assembled.

Since its introduction by Lee et al. [1] and Potts et al. [2], TSAFSP has been widely studied by researchers. This problem is recognizable in many real world manufacturing problems where a set of parts are processed on different machines, and then have to be assembled to form the final product. Moreover, the problem has gained application in areas other than production scheduling, such as in database distribution [3], supply chain management and batch production [4] and invoice printing system [5]. On the other hand, there are some industries such as chemical, pharmaceutical, food, plastic, and textile goods in which we cannot have any delay between stages [6], because these kinds of products are exposed to become decayed. Thus, the beginning of a given job must be postponed as if the completion of the operation concurs with the start of the operation at the next stage; however, the machine that processes it is idle. This is known as "no-wait assembly flow-

*. *Corresponding author. Tel.: +98 21 66545381; Fax: +98 21 66954569*
E-mail address: fatemi@aut.ac.ir (S.M.T. Fatemi Ghomi)

shop" problem. Gupta et al. [7], who addressed it for the first time, extended two models of the problem and developed a comprehensive complexity classification of the associated two-stage no-wait scheduling problems. Some researchers have attempted in the field of no-wait flow-shop problems [6-10] according to which the most recent work is corresponded to Ruiz and Allahverdi [10]. They have considered the m-machine no-wait flow-shop problem with independent setup times, and proposed four genetic algorithms as solution approaches.

To the best of our knowledge, no one has considered the no-wait two-stage assembly flow-shop. Lee et al. [1] and Potts et al. [2] proved that TSAFSP with only two machines in the first stage and a single assembly machine in second stage is strongly NP-hard. We can effortlessly assume that this holds true for problems with more than two processing machines and no-wait consideration. For this reason, implementation of heuristics are more popular than exact models among researchers; heuristics have been widely used in the literature, and mathematical models are less taken note of. One of the few mathematical models in this area is carried out by Zhang et al. [5] in which an invoice printing system, a special assembly flow-shop problem, is modeled into Mixed-Integer Linear Programming (MILP). However, the model is only able to solve small-sized problems, and so different heuristics and meta-heuristics have been used in addressing the TSAFSP. Nepalli et al. [11] used genetic algorithm in a bi-criteria problem. Allahverdi and Al-Anzi [3] proposed three heuristics; simple Earliest Due Date (EDD), Particle Swarm Optimization (PSO) and tabu search heuristics in which both PSO and tabu search proved to yield good results. In another study, Al-Anzi and Allahverdi [12] utilized tabu search, PSO and Self-adaptive Deferential Evolution (SDE) heuristics in addressing the problem with bi-criteria of maximum lateness and makespan minimization in which PSO was again shown to be the best heuristic. Gupta et al. [13] extensively experimented several variants of simulated annealing, threshold accepting and tabu search, and analyzed the parameter settings of these heuristics. Allouhi and Artiba [14] developed two algorithms and a heuristic in addressing TSAFSP with availability constraints. Koulamas and Kyparisis [15], Lin et al. [16] and others have also proposed heuristics of their own. Javadian et al. [17] however developed a Variable Neighborhood Search (VNS) heuristic for TSAFSP, and concluded that VNS outperforms SA in terms of accuracy and consistency.

This paper proposes three meta-heuristic algorithms; GA, DEA and PVNS. GA is a well-known algorithm, which has shown it is germane for the scheduling problems [18-20]. The most important benefits of using a DEA is its simple but effective mutation process [21]. On the other hand, the PVNS is a novel metaheuristic algorithm. Since almost all the meta-heuristics use one type of neighborhood mechanism, there exists high probability to become trapped in a local optimum, while the use of multiple neighborhoods in PVNS overcomes this drawback successfully.

The rest of this paper is organized as follows: Section 2 states the problem. Section 3 proposes three meta-heuristic algorithms for the problem. Experimental results are presented in Section 4, and finally Section 5 concludes the paper and recommends some areas for future studies.

2. Problem statement

In the No-Wait Two-Stage Assembly Flow-shop Scheduling Problem (NWTSAFSP) there are n jobs available at time zero. Each job consists of $m + 1$ operations; the first m operations are performed at the first stage on m parallel machines, so that each machine is capable of performing one job at a time with no preemption, i.e. once a job has begun, it will continue without interruption till the job is finished in the first stage. Then the output is assembled by an assembly machine at the second stage. Because of the no-wait limitation, when a job begins to be processed, it must be continued without any interruptions between stages. In other words, the beginning of a part of a given job in a given machine at the first stage should be delayed, so that the completion of the m operations can coincide with the start of assembly process at the second stage. We aim to find the best sequence of jobs to minimize the makespan.

As stated in previous section, makespan is the dominating objective function in the flow-shop problems. This is because the overall time of finishing the jobs most particularly affects the total costs of the firm. Makespan is thus chosen as the objective function of our study.

Let:

$t_{[jk]}$	Process time of the job in position j on machine k at the first stage, $k = 1, ..., m$.
$P_{[j]}$	Assembly time of job in position j on assembly machine at the second stage, $j = 1, 2, ..., n$.
$R_{[j]}$	Finishing time of job in position j at the first stage, $j = 1, 2, ..., n$.
$C_{[j]}$	Completion time of job in position j, $j = 1, 2, ..., n$.

$$C_1 = \max_{k=1}^{m}\{t_{1k}\} + P_1, \tag{1}$$

$$C_2 = R_2 + P_2, \tag{2}$$

$$R_2 = \max\{\max_{k=1}^{m}\{t_{1k}\} + \max_{k=1}^{m}\{t_{2k}\}, C_1\}, \quad (3)$$

$$C_3 = R_3 + P_3, \quad (4)$$

$$R_3 = \max\{\max\{\max_{k=1}^{m}\{t_{1k}\} + \max_{k=1}^{m}\{t_{2k}\}, C_1\} + \max_{k=1}^{m}\{t_{3k}\}, C_2\} = \max\{R_2 + \max_{k=1}^{m}\{t_{3k}\}, C_2\}, \quad (5)$$

$$C_j = R_j + P_j, \quad (6)$$

$$R_j = \max\{R_{j-1} + \max_{k=1}^{m}\{t_{jk}\}, C_{j-1}\} \quad \forall j = 1, ..., n. \quad (7)$$

3. Meta-heuristic algorithms

As mentioned before, the NWTSAFSP is known as NP-hard, and therefore, heuristic or meta-heuristic algorithms should be used to solve the problem. In this paper, we propose GA, DEA and PVNS in Subsections 3-1, 3-2 and 3-4, respectively.

3.1. Genetic algorithm

Genetic algorithm is a population-based heuristic which first was introduced by Holland [22]. GA is usually termed as a part of, and sometimes used as an equivalent of, Evolutionary Algorithms (EA) to solve optimization problems, using techniques naturally inspired evolution such as selection, inheritance, mutation and crossover. GA is extensively used in optimization and search problems, and is proved to be a powerful heuristic. The basic idea of GA is: At each iteration, the current population or genetic pool consists of different solutions called genomes or chromosomes.

The evolution begins with a population of randomly generated individuals, and later takes place in generations to come. In each generation, the fitness of every genome in the population is calculated. The fitness function, depending on the problem, is defined over the genetic representation, and measures the quality of solution.

Several genomes are then selected from the current population. In this selection, usually by the roulette wheel, the genomes which have the best fitness value are most likely to be selected, and this probability decreases for genomes with reduced fitness. When the candidate genomes are selected, the GA improves the solution through mutation and crossover operators, and creates new offspring and population. The new population is then used in the next iteration of the algorithm. Usually, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. In this paper, we have used a maximum number of generations.

While solutions are traditionally represented in binary 0 and 1 strings, other encodings are also possible. For instance, in our problem, genomes are strings of n successive numbers indicating the sequence of the jobs. The reproduction operations used in this paper are defined below.

1. *Crossover*

We have used a certain type of one-point cross over for ordered chromosomes in our algorithm. To do the crossover, two of the genomes are randomly chosen. As stated earlier, in choosing the two genomes, solutions are given a weight according to the quality of their fitness function, and so, better solutions have a higher chance to be chosen for the crossover operation. Then a random number k is generated between 1 and n, and the first k sequences of the two genomes are swapped. In each solution, there must be numbers 1 to n, so the algorithm then searches for duplicate numbers, and randomly swaps duplicated numbers of the two genomes with each other. The selection from two of the repetitive numbers is also random. Consider the following two genomes and their new offspring; random number is $k = 3$.

[**1 5 2** 3 4 6 7] → [3 5 1 **3** 4 6 7] → [3 5 1 2 4 6 7]

[**3 5 1** 7 4 6 2] → [1 5 **2** 7 4 6 2] → [1 5 3 7 4 6 2]

Note that the second 3 was selected randomly from the first genome, and was swapped by randomly selected first repetitive 2 in the second genome. If more than one duplicate number was seen, a random repetitive number from first genome was swapped with a random one from the second genome.

2. *Job mutation*

Two random jobs of the best solution are selected and swapped. Consider the following offspring from its parent:

[1 2 **3** 4 5 6 **7**] → [1 2 7 4 5 6 3]

3. *Sequence mutation*

A random job of the best solution is selected, and the place of former jobs and later jobs are swapped. Consider the following offspring from its parent:

[1 2 | 3 4 5 6 7] → [3 4 5 6 7 1 2]

By changing some genomes randomly, both job and sequence mutations prevent the algorithm to fall in local minima.

Provided that the rate of crossover, job mutation and sequence mutation are f, g, h, respectively, and the rate of last generation which goes to the next

generation directly is i, a pseudo code of GA will be as below:

1. Create the first generation (population) with N initial solutions randomly;
2. **Set** $k \leftarrow 1$;
3. **while** $k \leq k_{\max}$ **do**/// *maximum number of generations*;
4. Sort the solutions according to their fitness;
5. Calculate the selection probability of each solution, using roulette wheel rule;
6. Do crossover and job mutation and sequence mutation according to their rates (f, g, h) and the probability calculated for each current solution to create a part of next generation;
7. Complete the next generation, using the rate of i and the probability calculated for each current solution;
8. $k \leftarrow k + 1$;
9. **End while.**

3.2. Differential evolution algorithm

Differential Evolution Algorithm (DEA) is a kind of Evolutionary Algorithms (EA), first introduced by Storn and Price [23] for optimization over continuous spaces. Due to its invention, DEA has been extensively applied with high success on many numerical optimization problems outperforming other more popular population heuristics such as GAs [24,25]. Recently, some researchers successfully extended the application of DEA to complex combinatorial optimization problems with discrete decision variables, such as the machine layout problem [26] and the flow-shop scheduling problem [27]. The main features and stages of a classical DEA are as follows:

Step 1. DE utilizes Np, D-dimensional parameter vectors $x_{i,k}$, $i = 1, 2, ..., N_p$, as a population to search the feasible region Ω uniformly of a given problem. The index k denotes the iteration (or generation) number of the algorithm ($k = 0$ denotes initial population).

$$\phi = \{x_{1,0}, x_{2,0}, ..., x_{NP,0}\}. \tag{8}$$

Step 2. At each iteration, all vectors in φ are targeted for replacement. Therefore, Np competitions are held to determine the members of φ for the next iterations. This is achieved using three operators which are mutation, crossover and acceptance.

Step 3: Mutation phase.
For each target vector $x_{i,k}$, $i = 1, ..., Np$, a mutant vector $\hat{x}_{ik}$ is obtained by:

$$\hat{x}_{i,v} = r.x_{best,k} + (1 - r)(x_{\alpha,k} + Fs(x_{\beta,k} - x_{\gamma,k})), \tag{9}$$

where α, β, $\gamma \in \{I, ..., Np\}$ are mutually exclusive random indices, and are also different from the current target index i, and $r \in [0, 1]$ is a random number that is generated randomly for each mutant vector. It is a coefficient for the convex combination between the best element $x_{best,k}$ of φ, and a randomly combination of three random elements. The vector $x_{\alpha,k}$ is known as the base vector, and $Fs \in (0, 2]$ is a scaling parameter.

Step 4. Crossover phase.
The crossover operator is applied to obtain the trial vector $y_{i,k}$ from $\hat{x}_{ik}$ and $x_{i,k}$ by the following equation:

$$y^L_{i,k} = \begin{cases} \hat{x}^L_{i,k} & \text{if } R^L \leq CR \text{ or } L = I_i \\ x^L_{i,k} & \text{otherwise} \end{cases}. \tag{10}$$

In the above equation, CR shows the crossover rate, I_i is a random number in $[1, D]$ which guarantees that the difference between $x_{i,k}$ and $y_{i,k}$ is at least one gene and therefore, $x_{i,k}$ and $y_{i,k}$ are not the same no way.

Step 5. Acceptance phase.
Acceptance is applied after all Np trial vectors $y_{i,k}$ have been generated. In the acceptance phase, the fitness function of the trial vector, $F(y_{i,k})$, is compared with $F(x_{i,k})$; the value at the target vector, and the target vector is updated as below:

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } F(y_{i,k}) < F(x_{i,k}) \\ x_{i,k} & \text{otherwise} \end{cases}. \tag{11}$$

Step 6: Mutation, crossover and acceptance phases continue until some stopping conditions are met.

3.3. Population-based variable neighborhood search

The Variable Neighborhood Search (VNS) is a recent meta-heuristic technique introduced by Mladenovic and Hansen [28], and has quickly gained widespread popularity. This meta-heuristic was successfully applied to solve complex optimization problems such as routing problem [29], allocation problems [30] and production scheduling problems [31]. The basic idea in VNS is that it focuses on the mechanism of systematically exploring more than one kind of neighborhood structure during the searching process. The local optimum in each neighborhood is found iteratively and hopefully to reach the global optimum at the end. In its basic form, VNS explores a set of neighborhoods of the current solution, makes a local search from a neighbor solution to a local optimum, and moves to it only if there has been an improvement [32]. Since

almost all the meta-heuristics make use of one type of neighborhood mechanism, there exists high probability to become trapped in a local optimum, while the use of multiple neighborhoods in VNS overcomes this drawback successfully. The reasons why VNS has been used widely in researches are due to the utilization of several neighborhood structures, easy to implement and high flexibility and brilliant adaptability of VNS to different problems [31]. Despite the good potentials of VNS, it is necessary to devise strong systematic neighborhood structures and sufficient potential to escape from local optima. Therefore, in this paper we describe a population-based VNS algorithm (PVNS), which is an enhanced version of the basic VNS algorithm.

3.3.1. Implementation of PVNS

Though the variable neighborhood search has been successfully applied in many combinatorial optimization problems, the existing research shows a lack of application of PVNS to the NWTSAFSP. As stated before, in this subsection we present a population-based VNS algorithm (PVNS) for the NWTSAFSP, which is an enhanced version of the basic VNS. The idea of PVNS was first introduced by Wang and Tang [33]. Like basic VNS algorithm, PVNS uses two important functions, i.e. shake procedure and local search to enhance the solution quality. The shake function diversifies the search by switching the solution to another neighborhood structure, while the local search strives to find an improved solution within the current neighborhood structure. In this section, to further improve the performance of the basic VNS, we adapt a population k of solutions in which the solution quality is taken into account. Different from basic VNS, in the PVNS, multiple trial solutions are simultaneously generated to improve the search diversification and prevent local optimum solutions. A subset of population is randomly selected in each iteration of inner loop, and the epitome of group is established as the average property of the selected individuals as follows:

$$e = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{12}$$

where x_i represents the ith member of subset. Since the new chromosomes obtained as epitome of the group may represent infeasible solutions, some modifications are required. For production part, according to Largest Order Value (LOV) rule [34], we suggest to convert the individuals found by Eq. (12) into the feasible solutions. According to the LOV rule, $e = (e_1, e_2, ..., e_n)$ are firstly ranked by descending order to obtain a sequence $\beta = (\beta_1, \beta_2, ..., \beta_n)$. Then, the job permutation $y = (y_1, y_2, ..., y_n)$ is calculated by the formula $y_{\beta k} = k$.

After the shake procedure is implemented and the resultant solution x' is transformed to the local optimum x'', the current solution x^* and population P would be updated if x'' dominates x'. The procedure of adapted PVNS is illustrated below:

1 Create a set of neighborhood structures $N_k (k = 1, 2, ..., K_{\max})$;
2 Initialize the first population P with N_p solutions;
3 **For** $i = 1$ **to** $i_{\max}$ **do** /// *maximum number of iterations of outer loop;*
4 **Set** $k \leftarrow 1$;
5 Generate a random number q $(1 \leq q \leq N_p)$;
6 Select q solutions randomly from P and calculate the epitome of solutions x^*;
7 Rank elements of x^* in production part by descending order to reach sequence $\beta = (\beta_1, \beta_2 ..., \beta_n)$;
8 Calculate the feasible solution $y = (y_1, y_2, ..., y_n)$ by the formula $y_{\beta k} = k$, and replace the production part of x^* with y;
9 **set** $x \leftarrow x^*$;
10 **while** $k \leq k_{\max}$ **do** /// *maximum number of iterations of inner loop;*
11 For current solution x^*, execute shake procedure r times to find r trial solutions $x' \in N_k(x^*)$;
12 For each trial solution, execute local search on $N_k(x')$ to find the best local solution x'';
13 **if** fitness (x'') < fitness (x), **then**
14 $x^* \leftarrow x''$;
15 **set** $k \leftarrow 1$;
16 Update population P by replacing the worst solution in P with x^*;
17 **else;**
18 $k \leftarrow k + 1$;
19 **end if;**
20 **end while;**
21 $i = i + 1$;
22 **End for**.

3.3.2. Neighborhood search

The neighborhood structure deals with the systematic way of moving from one solution to its neighborhood such that infeasible solutions are prevented. A variety of neighborhood structures has been adapted to scheduling problems. As described previously, the main principle of VNS concerns with the use of several neighborhood structures and systematic switching from one structure to another, so as to improve the search

Table 1. Parameters tuned for the proposed GA.

Parameter	Range	Best value
N	10-50 with an increment of 10	20
$k_{\max}$	10-100000 multipled by 10 each time	1000
f, g, h, i	$f+g+h+i=1$	0.8,0.05,0.05,0.1

intensification as well as its diversification. Javadian et al. [17] developed a three-neighborhood search which is able to find excellent solutions to the TSAFS problem, and therefore, we use it in this paper. The proposed neighborhood structures are as follows:

Neighborhood 1. Two pairs of jobs are selected randomly from a given sequence and swapped independently.

Suppose that we have a sequence of jobs: [3 7 2 1 4 6 8 5]. Four numbers are generated randomly from "1" to "8". Consider that these random numbers are: 1, 2, 6, 8. So, the jobs in positions 1 and 2 and the jobs in positions 6 and 8 must be swapped independently:

[**3 7** 2 1 4 **6** 8 **5**] → [**7 3** 2 1 4 **5** 8 **6**].

Neighborhood 2. A random number ψ, in which $2 < \psi < n/2$, is generated and the first ψ jobs are swapped with the last ψ ones. For addressed sequence and a random $\psi = 3$, we have:

[**3 7 2** 1 4 **6 8 5**] → [**6 8 5** 1 4 **3 7 2**].

Neighborhood 3. A random number ψ is generated. Then, the first ψ jobs are inversed, and last $n-\psi$ jobs are inversed too. For the above sequence and a random $\psi = 5$, we have:

[**3 7 2 1 4** 6 8 5] → [**4 1 2 7 3** 5 8 6].

4. Computational results

Subsection 4.1 tunes parameters of three meta-heuristics, while Subsection 4.2 compares their performances.

4.1. Tuning parameters

Setting parameters is an important factor, whereby we can establish a trade-off between time and quality of heuristic solutions. Each meta-heuristic has some parameters which should be tuned. Theses parameters are set by doing some experiments and evaluating the results with respect to run time and quality of solutions. Tables 1 to 3 summarize the results for GA, DEA and PVNS, respectively. For each meta-heuristic, no significant improvement was observed beyond the value given in the tables. The parameters for GA have been acquired in Table 1. Parameters required for DEA consist of the Crossover Rate (CR), scaling parameters (Fs), rate (coefficient) of best function (r) and finally Number of Populations and Iterations (NP and NI). Table 2 shows these parameters. According to Subsection 3.3, parameters tuned for PVNS are shown in Table 3.

Table 2. Parameters tuned for the proposed DEA.

Parameter	Range	Best value
CR	(0.5,0.75,0.9)	0.75
Fs	(0.5,1,1.5)	1.5
r	(0,0.4,0.8)	0.4
(NP, NI)	$((n,3n),(1.75n,1.75n),(3n,n))$	$(n,3n)$

Table 3. Parameters tuned for the proposed PVNS.

Parameter	Range	Best value
$i_{\max}$	(3,4,5,6,7)	5
$k_{\max}$	(5,6,7,8,9,10,11,12)	8
Np	(15,20,25)	25

4.2. Evaluation of the meta-heuristics

We have used a PC with 8.2 GHz CPU processor under the Windows XP operating system with 2 GB of RAM, and the proposed meta-heuristics were implemented in Borland C++ compiler version 5.02.

The processing time at the first stage on each m machines and assembly time at the second stage were generated randomly from uniform distribution [1, 100]. The reason why a Uniform distribution with a wide range is used is that the variance of this distribution is large, and if a heuristic performs well with such a distribution, it is likely to perform well with other distributions [12].

In this paper, we consider 20, 40, 60 and 80 as different number of jobs (n) and 3, 5 and 7 as different number of machines at the first stage (m). These types of classes are very common among the researchers [3,8-10]. Each combination was solved 40 times by each meta-heuristic, and so a total of 1440($4\times3\times40\times3$) instances were evaluated.

In order to compare the performance of the

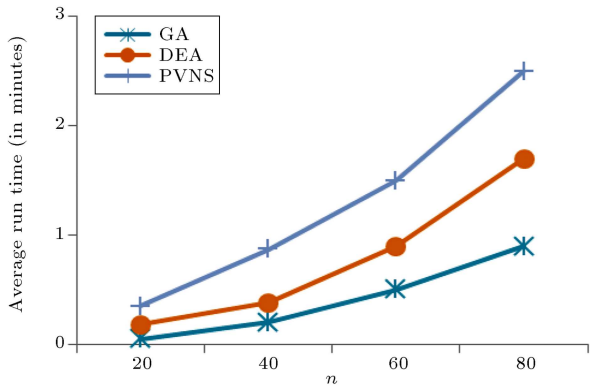

Figure 1. Average run time versus different number of jobs.

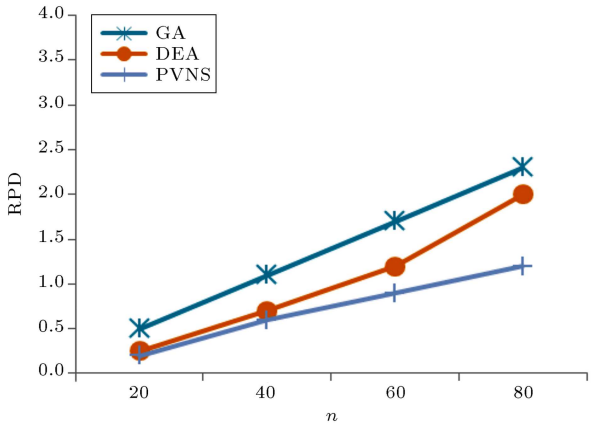

Figure 2. RPD versus different number of jobs.

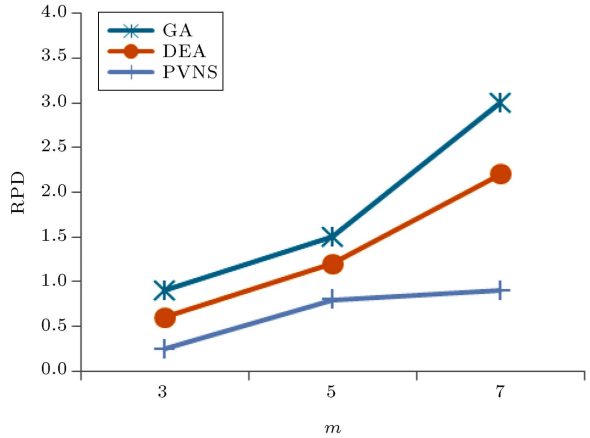

Figure 3. RPD versus different number of machines at the first stage.

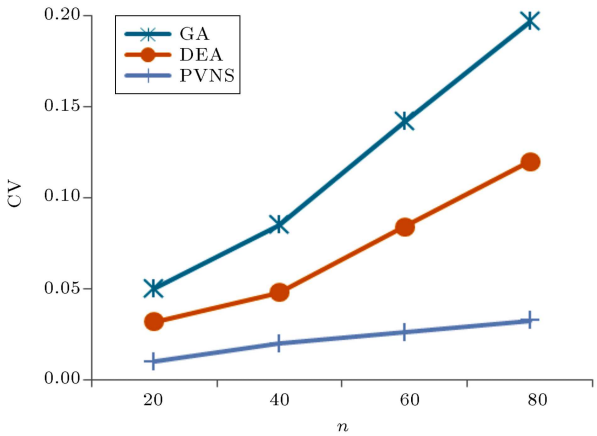

Figure 4. Average coefficient of variation versus different number of jobs.

algorithms, we use three measures: the first measure is average run time. Another measure is the Relative Percentage Deviation (RPD). RPD is a common measure frequently used in literature, and is obtained by the following manner. When the best solution yielded by three algorithms in each combination is found, the percentage error is calculated as follows: 100*(the solution found by the algorithm – the best solution)/ (the best solution). The last measure is the Coefficient of Variation (CV) which is defined as: CV= (standard deviations (STD) of the algorithm solutions)/ (average of the algorithm solutions). The reason why CV is used is that we cannot compare the STD of two combinations which have different means, and hence it is better to use CV instead of STD. For the sake of brevity, the results are shown in figures and are not tabulated.

Figure 1 illustrates run times of the meta-heuristics for different number of jobs. From Figure 1, it can be inferred that GA has the smallest CPU times and PVNS has the greatest one in comparison with other algorithms. In fact, according to the CPU times, GA outperforms other algorithms.

Figures 2 and 3 demonstrate the RPD measure, with respect to the number of jobs and number of machines at the first stage, respectively. Figure 2 reveals that GA has the worst performance among the algorithms. In contrast, PVNS has the lowest RPD and performs much better than GA and DEA. Figure 3 yields that PVNS outperforms other algorithms again, and GA has the worst performance. Figures 4 and 5 show CV versus different number of jobs and machines at stage one, respectively. Again, the figures reveal that GA and PVNS have the worst and best performances among the algorithms.

5. Conclusion and recommendation for future studies

In this paper, the no-wait two-stage assembly flow-shop scheduling problem was addressed. The makespan was selected as an objective function of the problem to be minimized. Three meta-heuristics were proposed: Genetic Algorithm (GA), Differential Evolution Algorithm (DEA) and Population base Variable Neighborhood Search (PVNS). Computational experiments have revealed that the performances of PVNS and GA are the best and worst, respectively, among the algorithms

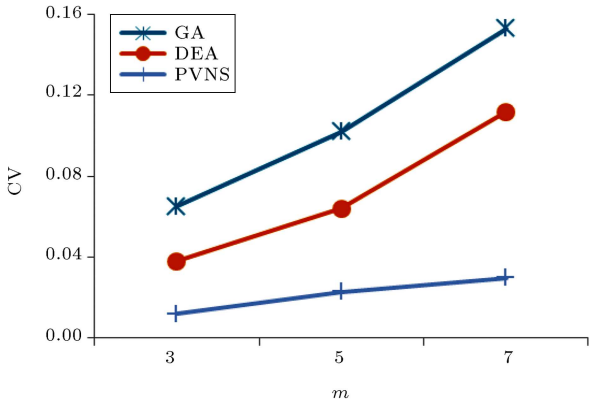

Figure 5. Average coefficient of variation versus different number of machines at the first stage.

in terms of RPD and CV. However, according to the CPU time, the result is different; GA and PVNS are the best and worst, respectively.

To have a more realistic situation, adding some constraint such as sequence-dependent setup time to the problem could be proposed. Furthermore, some other assumptions made in this paper could be also relaxed, such as priorities in jobs, having a set of similar jobs, probabilistic processing time, etc.

References

1. Lee, C.Y., Cheng, T.C.E. and Lin, B.M.T. "Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem", *Management Science*, **39**, pp. 616-625 (1993).

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

3. Allahverdi, A. and Al-Anzi, F.S. "A PSO and a tabu search heuristics for assembly scheduling problem of the two-stage distributed database application", *Computers & Operations Research*, **33**, pp. 1056-1080 (2006).

4. Lin, B.M.T, Cheng, T.C.E. and Chou, A.S.C. "Scheduling in an assembly-type production chain with batch transfer", *Omega*, **35**, pp. 143-151 (2007).

5. Zhang, Y., Zhou, Z. and Liu, J. "The production scheduling problem in a multi-page invoice printing system", *Computers & Operations Research*, **37**, pp. 1814-1821 (2010).

6. Aldowaisan, T. and Allahverdi, A. "Total flowtime in no-wait flowshops with separated setup times", *Computers & Operations Research*, **25**(9), pp. 757-765 (1998).

7. Gupta, J.N.D., Strusevich, V.A. and Zwaneveld, C.M. "Two-stage no-wait scheduling models with setup and removal times separated", *Computers & Operations Research*, **24**(11), pp. 1025-1031 (1997).

8. Allahverdi, A. and Aldowaisan, T. "No-wait and separate setup three-machine flowshop with total completion time criterion", *International Transactions in Operational Research*, **7**(3), pp. 245-264 (2000).

9. Allahverdi, A. and Aldowaisan, T. "Minimizing total completion time in a no-wait flowshop with sequence-dependent additive changeover times", *Journal of the Operational Research Society*, **52**(4), pp. 449-462 (2001).

10. Ruiz, R. and Allahverdi, A. "No-wait flowshop with separate setup times to minimize maximum lateness", *International Journal of Advanced Manufacturing Technology*, **35**, pp. 551-565 (2007).

11. Neppalli, V.R., Chen, C.L. and Gupta, J.N.D. "Genetic algorithms for the two-stage bicriteria flowshop problem", *European Journal of Operational Research*, **95**(2), pp. 356-373 (1996).

12. Al-Anzi, F.S. and Allahverdi, A. "Heuristics for a two-stage assembly flow shop with bicriteria of maximum lateness and makespan", *Computers & Operations Research*, **36**, pp. 2682-2689 (2009).

13. Gupta, J.N.D., Hennig, K. and Werner, F. "Local search heuristics for two-stage flow shop problems with secondary criterion", *Computers & Operations Research*, **29**, pp. 123-149 (2002).

14. Allaoui, H. and Artiba, A. "Scheduling two-stage hybrid flow shop with availability constraints", *Computers & Operations Research*, **33**, pp. 1399-1419 (2006).

15. Koulamas, C. and Kyparisis, G.J. "The three-stage assembly flowshop scheduling problem", *Computers & Operations Research*, **28**, pp. 689-704 (2001).

16. Lin, B.M.T., Cheng, T.C.E. and Chou, A.S.C. "Scheduling in an assembly-type production chain with batch transfer", *Omega*, **35**, pp. 143-151 (2007).

17. Javadian, N., Mozdgir, A., Gazani Koohi, E., Davallo Qajar, M.R. and Shiraqai, M.E. "Solving assembly flowshop scheduling problem with parallel machines using variable neighborhood search", *39th International Conference of Computers and Industrial Engineering*, University of Technology of Troyes, Paris, France, pp. 102-107 (2007).

18. Luo, H., Huang, G.Q., Zhang, Y., Dai, Q. and Chen, X. "Two-stage hybrid batching flowshop scheduling with blocking and machine availability constraints using genetic algorithm", *Robotics and Computer-Integrated Manufacturing*, **25**, pp. 962-971 (2009).

19. Chang, P.C., Chen, S.H., Fan, C.Y. and Chan, C.L. "Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems", *Expert Systems with Applications*, **36**, pp. 7135-7141 (2009).

20. Tseng, L.Y. and Lin, Y.T. "A hybrid genetic local search algorithm for the permutation flowshop scheduling problem", *European Journal of Operational Research*, **198**, pp. 84-92 (2009).

21. Mozdgir, A., Mahdavi, I., Seyedi, I. and Solimanpur, M. "Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing", *Mathematical and Computer Modelling*, **57**, pp. 137-151 (2011).

22. Holland, J., *Adaptation in Natural and Artificial Systems*, Ann Arbor, The University of Michigan Press, MI, USA (1975).

23. Storn, R. and Price, K. "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces", *Journal of Global Optimization*, **11**(4), pp. 341-354 (1997).

24. Ali, M.M. and Törn, A. "Population set-based global optimization algorithms: Some modifications and numerical studies", *Computers & Operations Research*, **31**, pp. 1703-1725 (2004).

25. Kaelo, P. and Ali, M.M. "A numerical study of some modified differential evolution algorithms", *European Journal of Operational Research,* **171**, pp. 674-692 (2006).

26. Nearchou, A.C. "Meta-heuristics from nature for the loop layout design problem", *International Journal of Production Economics,* **101**(2), pp. 312-328 (2006).

27. Onwubolu, G.O. and Davendra, D. "Scheduling flow shops using differential evolution", *European Journal of Operational Research,* **169**, pp. 1176-1184 (2006).

28. Mladenovic, N. and Hansen, P. "Variable neighborhood search", *Computers & Operations Research*, **24**, pp. 1097-1100 (1997).

29. Polacek, M., Hartl, R.F., Doerner, K. and Reimann, M. "A variable neighborhood search for the multi depot vehicle routing problem with time windows", *Journal of Heuristics,* **10**, pp. 613-627 (2004).

30. Yun-Chia, L. and Min-Hua, L. "Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm", *Journal of Heuristics*, **16**, pp. 511-535 (2010).

31. Naderi, B., Zandieh, M. and Fatemi Ghomi, S.M.T. "A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling", *Journal of Intelligent Manufacturing*, **20**, pp. 683-694 (2009).

32. Perez, J.A.M., Vega, J.M.M. and Martin, I.R. "Variable neighborhood tabu search and its application to the median cycle problem", *European Journal of Operational Research*, **151**, pp. 365-378 (2003).

33. Wang, X. and Tang, L. "A population-based variable neighborhood search for the single machine total weighted tardiness problem", *Computers & Operations Research*, **36**, pp. 2105-2110 (2009).

34. Bean, J.C. "Genetic algorithm and random keys for sequencing and optimization", *ORSA Journal on Computing*, **6**, pp. 154-160 (1994).

Biographies

Ashkan Mozdgir was born in Tehran, Iran, in 1984. He received his BS degree in Industrial Engineering from K. N. Toosi University of Technology, Tehran in 2006, and his MS degree in Industrial Engineering from Mazandaran University of Science and Technology, Mazandaran 2009. He is currently a PhD student in Industrial Engineering at K. N. Toosi University of Technology.

Seyyed Mohammad Taghi Fatemi Ghomi was born in Ghom, Iran, in 1952. He received his BS degree in Industrial Engineering from Sharif University, Tehran in 1973, and his PhD degree in Industrial Engineering from University of Bradford, England in 1980. He is currently a Professor in the Department of Industrial Engineering at the Amirkabir University of Technology in Tehran. His research and teaching interests are in stochastic activity networks, production planning and control, scheduling, queueing systems and statistical quality control.

Fariborz Jolai is currently Professor of Industrial Engineering at College of Engineering, University of Tehran, Tehran, Iran. He obtained his PhD degree in Industrial Engineering from INPG, Grenoble, France in 1998. His current research interests are scheduling and production planning, supply chain modeling and optimization problems under uncertainty condtions.

Javad Navaei was born in Tehran, Iran. He received his BS degree in Industrial Engineering from K. N. Toosi University of Technology, Tehran in 2009, and his MS degree in Industrial Engineering from Amirkabir University of Technology at Tehran, Iran, in 2011. He is currently a PhD student in Industrial Engineering at the Department of Industrial and Manufacturing Systems Engineering, University of Windsor, Windsor, Ontario, Canada.