Journal of Optimization in Industrial Engineering 21 (2017) 59-66

Modeling and Scheduling No-idle Hybrid Flow Shop Problems

Mehdi Yazdani[a,*], Bahman Naderi[b]

[a] *Assistant Professor, Department of Industrial Engineering, Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

[b] *Assistant Professor, Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran*

Received 29 December 2015; Revised 03 November 2016; Accepted 08 November 2016

Abstract

Although several papers have studied no-idle scheduling problems, they all focused on flow shops, assuming one processor at each working stage. But, companies commonly extend to hybrid flow shops by duplicating machines in parallel in stages. This paper considers the problem of scheduling no-idle hybrid flow shops. A mixed integer linear programming model is first developed to mathematically formulate the problem. Using commercial software, the model can solve small instances to optimality. Then, two metaheuristics, based on variable neighborhood search and genetic algorithms, are developed to solve larger instances. Using numerical experiments, the performance of the model and algorithms are evaluated.

Keywords: Scheduling, No-idle hybrid flow shops, Mixed integer linear programming, Variable neighborhood search, Genetic algorithm.

1. Introduction

In flow shop scheduling problems, we have a set of jobs and a set of working stage. To complete each job, some operations have to be performed. Each operation is carried out at one working stage, where in each stage, there is only one processor, say machine. The processing routes of all jobs are the same, starting from stage 1 to stage *m*. The objective is to sequence jobs so as to complete all jobs as soon as possible.

One criticism to scheduling problems is the gap between academic and practical problems. To bridge this gap, it is always interesting to extend scheduling problems by assumptions taken form realistic industries. One restrictive assumption in flow shop is to consider one machine at each working stage, while companies employ more than one machine at stages with more workload. In this case, they can reduce the impact of bottleneck stages, or even balance their production capacity more. The problem with more than one machine at each stage is called hybrid flow shops.

This extension of flow shop is a very active field of research. Ebrahimi et al. (2014) studied hybrid flow shop scheduling with sequence-dependent family setup time and uncertain due dates. Fattahi et al. (2014) considered hybrid flow shop scheduling problem with setup time and assembly operations. Lahimer et al. (2013) investigated hybrid flow shop scheduling with multiprocessor tasks. Luo et al. (2013) studied hybrid flow shop scheduling with machine electricity consumption cost. Elmi and Topaloglu, (2013) considered blocking hybrid flow shop robotic cells with multiple robots.

In some productions, it is completely uneconomical to maintain such machines idle, especially industries with highly expensive machines. Moreover, in industries with less expensive machines, it might not be desirable to stop machines between jobs. An example of such an industry is the furnace of the fiberglass industry. Heating the furnace up to the necessary temperature is both time-consuming and expensive. Thus, it is always kept on when it starts working. These practical situations raise a scheduling environment, called no-idle scheduling. In a no-idle scheduling, idle time on a machine is not allowed. In other words, each machine must continuously process jobs from the start of processing the first job to the end of the last job. To fulfill this restriction, the start of processing the first job on a given machine might be delayed. Other applications of no-idle scheduling (i.e., as it might be technically infeasible or uneconomical to stop a machine in between jobs) are foundries, production of integrated circuits, and the steel-making industry (Pan and Ruiz, 2014).

Although there are several papers considering no-idle scheduling, they all study flow shop problems. Kalczynski and Kamburowski (2005) considered no-idle flow shops and proposed a heuristic for this problem. Deng and Gu (2012) developed a hybrid discrete

* Corresponding author Email address: mehdi_yazdani2007@yahoo.com

differential evolution algorithm for the no-idle permutation flowshop scheduling problem to minimize make-span. Tasgetiren et al. (2013a) investigated the no-idle permutation flowshop scheduling problem to minimize the total tardiness. They developed a discrete artificial bee colony algorithm for this problem. Tasgetiren et al. (2013b) investigated the same problem with Tasgetiren et al., (2013a) and proposed a variable iterated greedy algorithm with differential evolution. Moreover, the classical iterated greedy and a variable iterated greedy from the literature are reimplemented. Pan and Wang (2008) studied the no-idle permutation flow shop scheduling problems to minimize makespan. They developed two straightforward methods to calculate the makespan: a speed-up method and a discrete particle swarm optimization algorithm. This algorithm outperformed two heuristics of Tasgetiren et al. (2013a) and Kalczynski and Kamburowski (2005).

Saadani et al. (2005) studied no-idle flowshop problems to minimize the makespan. Since this problem can be modeled as a travelling salesman problem, an adaptation of the well-known nearest insertion rule is proposed to solve the problem. Baraz and Mosheiov (2008) developed a greedy algorithm for no idle flowshops to minimize makespan. Goncharov and Sevastyanov (2009) proposed several polynomial time heuristics based on a geometrical approach for the general case and for special cases of 3 and 4 machines. They also proposed a complete survey of relevant works. Zhou et al. (2014) also proposed an invasive weed optimization algorithm. Lu (2016) studied no-idle flow shop with time-dependent learning effect and deteriorating jobs. Pan and Ruiz (2014) investigated mixed no-idle flow shops, where not all machines require no-idle restriction. They first mathematically formulated the problem and proposed an iterated greedy algorithm enhanced by a speed-up feature. After reviewing the literature of no-idle scheduling, we can conclude that all the papers studied the flow shop problem.

In this paper, we generalize no-idle flow shop problems to no-idle hybrid flow shops. Using Grahams' notation, the problem is $FFc/no-idle/C_{max}$. We first formulate the problem by a mixed integer linear programming model. Using the model, the small instances of the problem are solved to optimality. Since the problem under consideration is NP-hard, the best solution method is metaheuristic. There are two different metaheuristic types: single-individual and multi-individual (population-based) ones. To find out what type of metaheuristic performs well for this problem, we have decided to develop one single-individual and one multi-individual metaheuristics. Among different single-individual alternatives, variable neighborhood search has shown high performance in different scheduling problems (Xiao et al., 2014; Kocatürk and Özpeynirci, 2014). Also, among different multi-individual alternatives, genetic algorithm seems the best one regarding the literature (Dai et al., 2013). Thus, two metaheuristics, based on variable neighborhood search and genetic algorithm, are also developed to solve large instances. Two numerical experiments are conducted to evaluate and compare the models and algorithms.

The rest of the paper is organized as follows. Section 2 formulates the problem by three different mathematical models. Section 3 develops two metaheuristics. Section 4 evaluates both the models and metaheuristics. Section 5 concludes the paper.

2. Problem Definition and Formulation

This section first describes the problem, then mathematically formulates the problem. The problem of scheduling no-idle hybrid flow shops can be defined as follows. There are n jobs and m working stages. Each stage has a number of m_i identical machines. All jobs visit all stages from the first to the last stage. They are processed by one machine at each stage. Each job can be processed by at most one stage at a time and each machine can process at most one job at a time. The machines are continuously available. All jobs are independent and available at time zero. The objective is to assign jobs to machines at each stage and sequence jobs at each stage.

To illustrate the problem, we first present a numerical example. Consider a problem with 4 jobs and 2 stages. There are two machines at each of the two stages. Table 1 shows the processing times of jobs at different stages. One solution for this problem is to assign jobs 1 and 3 to machine 1 (job 3 followed by job 1), and jobs 2 and 4 to machine 2 at stage 1 (job 2 followed job 4). At stage 2, jobs 2 and 3 are assigned to machine 1, and jobs 1 and 4 to machine 2. Makespan of this solution becomes 25.

Table 1
Processing times of the example

Jobs	Stages	
	1	2
1	6	5
2	15	10
3	5	9
4	4	5

Scheduling problems are commonly formulated as mixed integer linear programming models. The problem under consideration is modeled by a mathematical model. Before presenting the models, we establish the following parameters and sets.

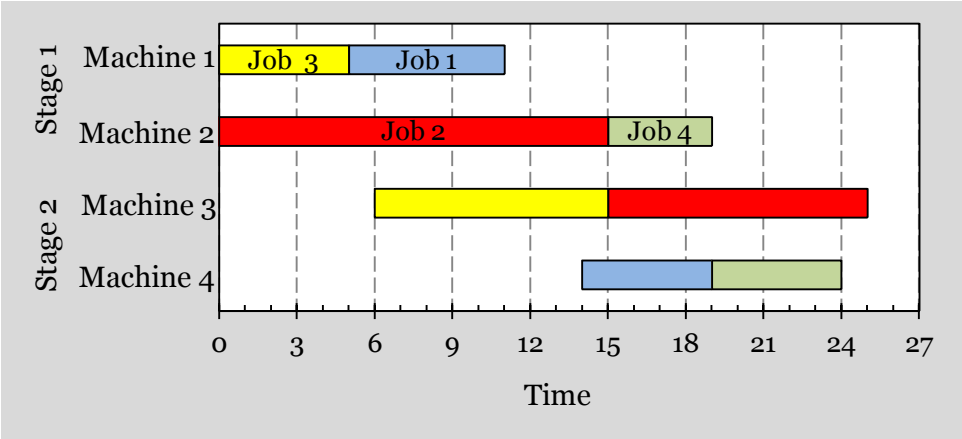

Fig. 1. Gantt chart of a feasible solution for the example

Parameters:

n	The number of jobs
j, k	Indices for jobs where $\{1, 2, \dots, n\}$
m	Number of stages
i	Indices for stages where $\{1, 2, \dots, m\}$
m_i	Number of machines in stage i
l	Indices for machines at stage i where $\{1, 2, \dots, m_i\}$
$p_{j,i}$	Processing time of job j at stage i
M	A large positive number

Decision variables:

$X_{j,i,k}$	Binary variable taking value 1 if job j is processed after job k at stage i, and 0 otherwise $k > j$.
$Y_{j,i,l}$	Binary variable taking value 1 if job j is processed at stage i on machine l, and 0 otherwise.
$C_{j,i}$	Continuous variable for the completion time of job j at stage i
$S_{i,l}$	Continuous variable for the starting time of machine l at stage i
$F_{i,l}$	Continuous variable for the finishing time of machine l at stage i

The model is as follows.

Min C_{max}
Subject to:

$$\sum_{l=1}^{m_i} Y_{j,i,l} = 1 \qquad \forall_{j,i} \qquad (1)$$

$$C_{j,i} \geq C_{j,i-1} + p_{j,i} \qquad \forall_{j,i>1} \qquad (2)$$

$$C_{j,i} \geq C_{k,i} + p_{j,i} - M \cdot (3 - X_{j,i,k} - Y_{j,i,l} - Y_{k,i,l}) \qquad \forall_{j<n,k>j,i,l} \qquad (3)$$

$$C_{k,i} \geq C_{j,i} + p_{k,i} - M \cdot \begin{pmatrix} 2 + X_{j,i,k} \\ -Y_{j,i,l} - Y_{k,i,l} \end{pmatrix} \qquad \forall_{j<n,k>j,i,l} \qquad (4)$$

$$C_{j,i} \geq S_{i,l} + p_{j,i} - M \cdot (1 - Y_{j,i,l}) \qquad \forall_{j,i,l} \qquad (5)$$

$$F_{i,l} \geq C_{j,i} - M \cdot (1 - Y_{j,i,l}) \qquad \forall_{j,i,l} \qquad (6)$$

$$F_{i,l} = S_{i,l} + \sum_{j=1}^{n} Y_{j,i,l} \cdot p_{j,i} \qquad \forall_{i,l} \qquad (7)$$

$$C_{max} \geq C_{j,m} \qquad \forall_{j} \qquad (8)$$

$$C_{j,i} \geq 0 \qquad \forall_{j,i} \qquad (9)$$

$$S_{i,l} \geq 0 \qquad \forall_{i,l} \qquad (10)$$

$$F_{i,l} \geq 0 \qquad \forall_{i,l} \qquad (11)$$

$$X_{j,i,k} \in \{0, 1\} \qquad \forall_{j,i,k>j} \qquad (12)$$

$$Y_{j,i,l} \in \{0, 1\} \qquad \forall_{j,i,l} \qquad (13)$$

Constraint set (1) assigns each operation to one of machines of its corresponding stage. Constraint set (2) ensures that each job can be processed by at most one machine at a time. Constraint sets (3) and (4) are the pair of disjunctive constraints (one of them holds at most depending on which job proceeds the other one) that show each machine can process at most one job at a time. Constraint sets (5), (6), and (7) ensure that no-idle restrictions are met. Constraint set (8) calculates makespan. Finally, Constraint sets (9) and (13) define the decision variables.

3. Developed Metaheuristics

In this paper, we develop two metaheuristics of variable neighborhood search and genetic algorithm to solve large instances of the problem. Later on, the variable neighborhood search and genetic algorithm are described in detail.

3.1. Variable neighborhood search

The general timetabling problem is known to be complex and difficult. In this context, exact solutions would be only possible for problems of limited sizes. Instead, solution algorithms based on metaheuristics have shown to be highly effective. Examples of these algorithms include genetic algorithm (Wang, 2002; Wang, 2003), Tabu Search (Aladag et al., 2009), simulated annealing (Zhang et al., 2010), variable neighborhood search (Burke et al., 2010), and etc.

Variable neighborhood search (VNS) is a simple but effective local search-based metaheuristic proposed by Mladenovic and Hansen (1997). Local search-based methods have been applied in the optimization literature with very good results, like simulated annealing (SA), tabu search (TS), and the iterated local search (ILS). However, all these methods are based on the exploration of a single neighborhood structure. Hence, there exists high probability for them to get trapped in local optima after a certain number of iterations and the move required to separate the algorithms from the local optima cannot be performed. Therefore, they need mechanisms to have sufficient potentiality to escape from local optima.

Instead of iterating over one constant type of neighborhood structure and relying on mechanisms such as random perturbations of ILS or memory structures of TS or metropolis mechanism of SA, VNS proceeds in this case by using a different type of neighborhood structure, which might contain the required improving moves. The term "VNS" is referred to all local search-based approaches that are centered on the principle of systematically exploring more than one type of neighborhood structure during the search. VNS is based on two important facts: (1) a local optimum, with respect to one type of neighborhood, is not necessarily so with respect to another type; (2) a global optimum is a local optimum with respect to all types of neighborhoods (Hansen and Mladenovic, 2001).

The reasons why VNS has obtained its acceptability and popularity among researcher are due to the utilization of several neighborhood structures, easy to implement, and high flexibility, and brilliant adaptability of VNS to different problems. VNS has been applied with success to other problems including (Flesza and Hindi 2004; Liao and Cheng 2007). In the following sections, we provide the proposed VNS methods with further details.

3.1.1. Solution representation

The proposed algorithms are based on permutation encoding scheme. The permutation scheme is a sorted list of all the jobs being processed. By considering the permutation from left to right, the relative sequence of jobs is determined. For the sake of simplicity, let us describe the permutation scheme by an example. Suppose that we have a problem with $n = 5$. In the case of permutation list, $\{2,5,1,4,3\}$ is one possible permutation. This permutation merely shows the relative order of jobs at the first stage. Jobs are taken one by one from the list and assigned to the first available machine. The sequence of jobs at the subsequent stages is determined by the earliest completion time of jobs at the previous stage. The first available machine rule is also used for the assignment.

While decoding the encoded solution, no-idle restriction has to be fulfilled. Note that no-idle restriction in the first stage is always held. To meet this restriction for the rest, the following two-phase decoding scheme is used. In the first phase, no-idle restriction is ignored; jobs are sequenced and assigned according to "earliest completion time" and "first available machine" rules, respectively. In the second phase, the last job of each machine is fixed; the preceding jobs are moved left. That is, the processing of a job is postponed so as to make sure that the job is completed when the next job can be started.

3.1.2. General structure of the proposed VNS and its neighborhoods

Generally, VNS iterates over some neighborhood structures until some stopping criterion is met. Our proposed VNS algorithm incorporates two different local search types: one local search for small changes and another for larger changes. The stopping criterion is set at a limit CPU time fixed to nm seconds. This stopping criterion allows for more time as the number of jobs and stages increases.

In the first local search, one job is randomly selected and reinserted into another randomly selected position. For example, consider a problem with $n = 5$. Suppose that the current solution is $\{2,5,1,4,3\}$. The randomly selected job is job 5 and randomly selected position is 4. In this case, new solution becomes $\{2,1,4,5,3\}$. This local search repeats for ρ_1 times and each time, a new solution is generated.

After generating each new solution, it can be either accepted or rejected by another mechanism. If this new solution improves the current solution, it is accepted. If new solution deteriorates the current solution by more than 10% (called i), it is rejected. Otherwise, it is probably accepted. The probability of acceptance depends on the inferiority gap size. The larger the inferiority gap size is, the lower its chance of being accepted becomes. A new solution with $g\%$ inferiority gap is accepted with

probability of 10-g%. Figure 2 shows the outline of the proposed VNS.

Procedure: *The_local_search_type* 1

For i = 1 **to** ρ_1 **do**
Reinsert a job into a new position
Accept/reject the new solution
Endfor

Fig. 2. General outline of local search type 1

If the local search type one is implemented and the best solution is improved, it is re-implemented. Otherwise, the local search type two is performed. In the local search type two, two jobs are randomly selected and they are reinserted into new randomly selected positions. This local search also repeats for $\rho_2$times. Each time, a new solution is generated. To accept or reject a new solution, the mentioned acceptance mechanism of the first local search is used. Figure 3 shows the outline of the proposed VNS.

Procedure: *The_proposed_VNS*

Step 1: Generate initial solution, say θ.
Step 2: If the stopping criterion is not met, go to step 3.
Step 3: Apply local search 1.
Step 4: If θ is improved; go to step 3; otherwise, go to step 5.
Step 5: Apply local search 2.
Step 6: If θ is improved, go to step 5; otherwise, go to step 2.

Fig. 3. General outline of the proposed VNS

3.2. Genetic algorithm

Genetic algorithm (GA) is designed to deal with some problems of industry that were difficult to solve by conventional methods. Todays, GA is a well-known population-based evolutionary algorithm tackling both discrete and continuous optimization problems. The idea behind GA comes from Darwin's ''survival of the fittest'' concept, meaning that good parents produce better offsprings. Many hard optimization problems have been successfully solved by GA (Wang, 2002; Toledo et al., 2013; Balakrishnan et al., 2003). Wang (2002) solved teacher assignment problems by GA. Toledo et al. (2013) developed a GA to tackle lot-sizing problems. Balakrishnan et al. (2003) also solved dynamic layout problem by GA.

3.2.1. General structure

GA searches for a solution space with a population of chromosomes, each of which represents an encoded solution. A fitness value is assigned to each chromosome according to its performance. The better the chromosome is, the higher this value becomes. The population evolves by a set of operators until some stopping criteria are visited. A typical iteration of a GA, generation proceeds as follows. The best chromosomes of current population are directly copied to next generation (reproduction). A selection mechanism chooses chromosomes of the current population so as to give higher chance to chromosomes with the higher fitness value. The selected chromosomes are crossed to generate new offspring. After crossing process, each offspring might mutate by another mechanism called mutation. Afterwards, the new population is evaluated again and the whole process is repeated. The outline of the proposed GA is shown in Figure 4.

The procedure: the proposed GA
Initialization mechanism
While the stopping criterion is not met, **do**
Selection mechanism
Crossover mechanism
Mutation mechanism
Endwhile

Fig. 4. The outline of the proposed GA

3.2.2. *Initialization and selection mechanisms*

GA starts with a number of chromosomes, each of which represents a possible solution. The number of chromosomes is the population size indicated by *pop*, set to 50. The initial chromosomes are randomly generated from the feasible solutions. After initializing the algorithms, each chromosome is evaluated and its fitness (i.e., objective function) is determined. The chance of chromosome k to be selected for crossover mechanism is as follows.

$$p_k = \frac{fit(k)}{\sum_{h=1}^{pop} fit(h)}$$

where $fit(k)$ is the fitness of chromosome k.

3.2.3. Crossover and mutation mechanisms

New solutions are produced by crossing two other solutions already selected by selection mechanism. These two solutions are called parents. The operators of combining parent are called crossover. The purpose of this combining is to generate better offsprings. To move the search towards better areas, we define a new solution that inherits from both parents. In fact, we combine two parents to form a new solution. In this research, this is done through an operator with the following steps.

Two randomly cut points are selected. Then, the jobs between these cut points from Parent 1 are copied to offspring in the same positions. The remaining jobs are put into the empty positions of the offspring from Parent 2. The order of the remaining jobs is determined by their relative order in Parent 2. For example consider a problem with n = 6. Suppose that two parents are:

Parent 1: {2,1, 5,4,6,3}
Parent 2: {6,1, 3,2,4,5}

Suppose that the two randomly selected cut points are 2 and 4. In this case, the operations from position 2 to

position 4 are copied into the same position in the offspring.

$$\text{offespring: } \{-,1,5,4,-,-\}$$

The remaining jobs are 2, 6, and 3. These jobs are copied into offspring according to Parent 2. Thus, the complete offspring becomes:

$$\text{offespring: } \{6,1,5,4,3,2\}$$

After crossover, each solution is changed by the mutation operator. The main purpose of applying mutation is to avoid convergence to a local optimum and diversify the population. We use the swap mutation operator which works as follows. Two positions are randomly selected and the jobs of these two positions are swapped. For example, consider a problem with $n = 6$. Suppose that the encoded solution is:

$$\{6,1,5,4,3,2\}$$

The two randomly selected positions are 3 and 6. By swapping the corresponding operations, we have:

$$\{6,1,2,4,3,5\}$$

4. Computational Evaluation

This section numerically evaluates the performance of the model and algorithms. To do that, two sets of instances are generated: one including small instances and one with larger ones. First, the proposed MILP model is assessed by a computational experiment with small-sized instances. Then, the general performance of the proposed metaheuristics (i.e., GA and VNS) is evaluated against the optimal solutions obtained by the model. We use a performance measure named relative percentage deviation (RPD) obtained by the following formula:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \cdot 100$$

where Min_{sol} and Alg_{sol} are the lowest C_{max} for a given instance obtained by any of algorithms and the solution obtained by a given algorithm. We implement the MILP models in CPLEX 12 and the other algorithms in MATLAB and run on a PC with 2.0 GHz Intel Core 2 Duo and 2 GB of RAM memory. The stopping criterion used when testing all instances with the metaheuristics is set to a computational time limit fixed to $n \times m \times 0.5$ seconds. This stopping criterion permits for more time as the number of jobs or machines increases.

4.1. Evaluation on small-sized instances

This subsection first evaluates the efficiency of the MILP model to solve the problem under consideration. We generate a set of different instances as follows. We have 6 problem sizes

$$n = \{4,6,8\} \text{ and } m = \{2,4\}$$

The processing times are randomly distributed over (1, 99). For each problem size, we generate 2 instances. Therefore, it sums up to 12 instances. The MILP model is allowed a maximum of 1000 seconds of computational time. Table 2 shows the results obtained by the model. The model can optimally solve instances up to 6 jobs and 2 stages. The required computational time is less than 6 seconds. Yet, for larger instances than 6 jobs and 2 stages, it yields average optimality gap of 9%.

Table 2
Model's results (computational time in seconds)

$n \times m$	Model		
	Cmax	Time	Optimality Gap
4×2	161	0.06	0
4×2	153	0.06	0
4×4	274	3.84	0
4×4	288	1.34	0
6×2	196	2.28	0
6×2	236	5.03	0
6×4	348	1000	7%
6×4	346	1000	11%
8×2	281	1000	8%
8×2	258	1000	6%
8×4	316	1000	12%
8×4	335	1000	6%

We are going to evaluate the algorithms (i.e., GA and VNS) against the optimal solutions obtained by the models in the previous small instances. Table 3 shows the results. GA and VNS perform the same by optimally solving 5 instances out of 6 instances. The average optimality gap becomes 0.63%.

Table 3
Algorithm' results on small instances

$n \times m$	Model	Algorithm	
		GA	VNS
4×2	161	161	161
4×2	153	153	153
4×4	274	275	275
4×4	288	298	298
6×2	196	196	196
6×2	236	236	236
Average		0.63%	0.63%

4.2. Evaluation on large-sized instances

After having investigated the general performance of the metaheuristics, we intend to further compare the proposed algorithms against a set of large instances. We consider the following 16 combinations of n, m, and m_i.

$$n = \{20,50,100\}, m = \{5,10\}, m_i = \{2, U[1,3]\}$$

For each combination, we generate 5 random instances by producing random processing times from a uniform distribution over (1, 99). In this case, we have 60 instances. We use the RPD measure to compare the algorithms.

Table 4 summarizes the results of the experiments averaged for each combination of n and m. GA is still the best performing algorithm with RPD of 0.26%. VNS yields average RPD of 1.60%.

Table 4
Algorithm' results on small instances

$n \times m$	Algorithm (gap)	
	VNS	GA
20×5	1.6503	0.4487
20×10	1.1072	0.6285
50×5	2.0444	0.2082
50×10	1.8091	0.217
100×5	0.9875	0.0509
100×10	2.0134	0.0168
Average	1.602	0.2617

To further statistically analyze the results, we carry out an analysis of variance test or ANOVA. The results demonstrate that there are significant differences between the algorithms with p-value very close to 0. Figure 5 shows the mean of plot and least significant difference or LSD intervals at 95% confidence level for the different algorithms.

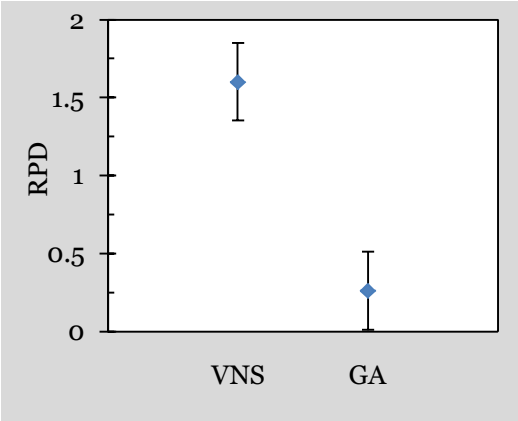

Fig. 5. The average RPD and LSD interval of the algorithms

It is also interesting to plot the performance of the algorithms versus the problem size. Figure 6 shows the mean obtained by the algorithms in the different problem sizes. In all the three problem sizes, GA outperforms VNS. There is a clear trend that GA works better in larger sizes.

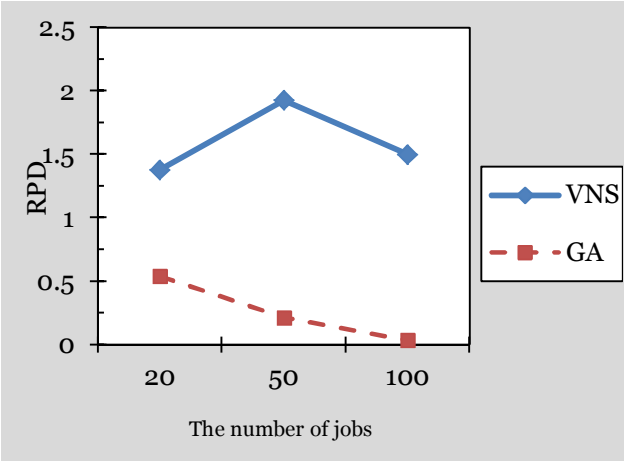

Fig. 6. The average RPD of the algorithms versus the problem size

5. Conclusion

This paper considered a practical extension of hybrid flow shops, called no-idle scheduling. In this type of scheduling, it is assumed that a machine has to continuously process jobs. That is, no idle time on machine is allowed. Although no-idle scheduling is an active field of research in the literature, all the papers in this area focus on flow shop problems and there is no paper studying no-idle hybrid flow shops. For the very first time, this problem is mathematically formulated by a mixed integer linear programming model. Using this model and CPLEX software, the instances up to 6 jobs can be solved to optimality. Yet, larger instances cannot be optimally solved due to hardness of the problem. Therefore, two metaheuristics in form of variable neighborhood search and genetic algorithm were developed.

To evaluate the performances of model and metaheuristics, two computational experiments were done. In the first experiment, small instances were used to assess the model' computational time to solve the instances (Table 2). The experiment was implemented in CPLEX software. Moreover, the general performance of the proposed metaheuristics was evaluated (Table 3). The results of numerical experiment showed that the proposed metaheuristics effectively solve the problem. In the second experiment, using large instances, the proposed metaheuristics were compared (Table 4). The results showed that genetic algorithm outperformed variable neighborhood search.

References

Aladag, C.H., Hocaoglu G., Basaran M.A. (2009). The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem, *Expert Systems with Applications*, Vol. 36, 12349–12356.

Baraz, D., Mosheiov, G., (2008). A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time. *European Journal of Operational Research*, Vol. 184, 810–813.

Balakrishnan, J., Cheng, C.H., Conway, D.G., Lau, C.M., (2003). A hybrid genetic algorithm for the dynamic plant layout problem. *International Journal of Production Economics*, Vol. 86, 107–20.

Burke, E.K., Eckersley A.J., McCollum B., Petrovic S., Qu R. (2010). Hybrid variable neighbourhood approaches to university exam timetabling, *European Journal of Operational Research*, Vol. 206, 46–53.

Dai, M., Tang, D., Giret, A., Salido, M.A., Lid, W.D., (2013). Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, Vol. 29(5), 418-429.

Deng, G., Gu, X., (2012). A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Computers and Operations Research*, Vol. 39, 2152–2160.

Ebrahimi, M., Fatemi Ghomi, S.M.T., Karimi, B., (2014), Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates, *Applied Mathematical Modelling*, Vol. 38, 2490-2504.

Elmi, A., Topaloglu, S., (2013). A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots, *Computers and Operations Research*, Vol. 40, 2543-2555.

El Houda Saadani, N., Guinet, A., Moall, M., (2005). A travelling salesman approach to solve the F/no-idle/Cmax problem. *European Journal of Operational Research*, Vol. 161, 11–20.

Fatih Tasgetiren, M., Pan, Q.K., Suganthan, P.N., Oner, A., (2013a). A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Applied Mathematical Modelling*, Vol. 37, 6758–6779.

Fatih Tasgetiren, M., Pan, Q.K., Suganthan, P.N., Buyukdagli, O., (2013b). A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Computers and Operations Research*, Vol. 40, 1729-1743.

Fattahi, P., Hosseini, S.M.H., F. Jolai, Tavakkoli-Moghaddam, R., (2014). A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations, *Applied Mathematical Modelling*, Vol. 38, 119-134.

Flesza, K., Hindi K.S. (2004). Solving the resource-constrained project scheduling problem by a variable neighborhood search, *European Journal of Operational Research*, Vol. 155, 402–413.

Hansen, P., Mladenovic N. (2001). Variable neighborhood search: principles and applications, European Journal of Operational Research, Vol. 130(3), 449–467.

Goncharov, Y., Sevastyanov, S., (2009). The flow shop problem with no-idle constraints: A review and approximation. *European Journal of Operational Research*, Vol. 196, 450–456.

Lahimer, A., Lopez, P., Haouari, M., (2013). Improved bounds for hybrid flow shop scheduling with multiprocessor tasks, *Computers and Industrial Engineering*, Vol. 66, 1106-1114.

Liao, C.J., Cheng, C.C. (2007). A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date, *Computers and Industrial Engineering*, Vol. 52, 404–413.

Lu, Y.Y., (2016). Research on no-idle permutation flowshop scheduling with time-dependent learning effect and deteriorating jobs, *Applied Mathematical Modelling*, Vol. 40, 3447–3450.

Luo, H., Du, B., Huang, G.Q., Chen, H., Li, X. (2013). Hybrid flow shop scheduling considering machine electricity consumption cost, *International Journal of Production Economics*, Vol. 146, 423-439.

Kalczynski, P.J., Kamburowski, J., (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers and Industrial Engineering*, Vol. 49, 146–154.

Kocatürk, F., Özpeynirci, Ö., (2014). Variable neighborhood search for the pharmacy duty scheduling problem, *Computers and Operations Research*, Vol. 51, 218-226.

Mladenovic, N., Hansen P. (1997). Variable neighborhood search, Computers and Operations Research, Vol. 24(11), 1097–1100.

Moslehi, G., Khorasanian, D., (2013). A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion, *Computers and Operations Research*, DOI: 10.1016/j.cor.2013.09.014.

Pan, Q.K., Ruiz, R., (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, Vol. 44, 41-50.

Pan, Q.K., Wang, L., (2008). No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *International Journal of Advanced Manufacturing Technology*, Vol. 39, 796–807.

Toledo, C.F.M., Oliveira, R.R.R., Franca, P.M., (2013). A hybrid multi-population genetic algorithm applied to solve the multi-level capacitated lot sizing problem with backlogging. *Computers and Operations Research*, Vol. 40, 910-919.

Wang, Y.Z., (2002). An application of genetic algorithm methods for teacher assignment problems. *Expert Systems with Applications*, Vol. 22, pp. 295–302.

Wang Y.Z. (2003). Using genetic algorithm methods to solve course scheduling problems, *Expert Systems with Applications*, Vol. 25, 39-50.

Zhang, D., Liu, Y., M'Hallah, R., Leung, S.C.H., (2010). A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, Vol. 203, 550–558.

Zhou, Y., Chena, H., Zhouc, G., (2014). Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem, Neurocomputing, Vol. 137, 285–292.

Xiao, Y., Zhang, R., Zhao, Q., Kaku, I., Xu, Y., (2014). A variable neighborhood search with an effective local search for uncapacitated multilevel lot-sizing problems, *European Journal of Operational Research*, Vol. 235, 102-114.

This article can be cited: Yazdani, M. & Naderi, B. (2017). Modeling and Scheduling No-idle Hybrid Flow Shop Problems. *Journal of Optimization in Industrial Engineering*.10(21), 59-66.

URL: http://qjie.ir/article_261_37.html