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

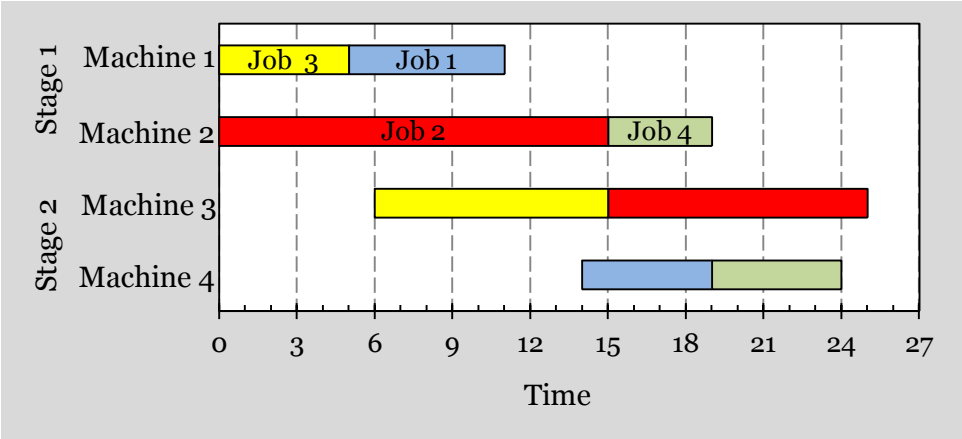

Fig. 1. Gantt chart of a feasible solution for the example

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.

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

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.

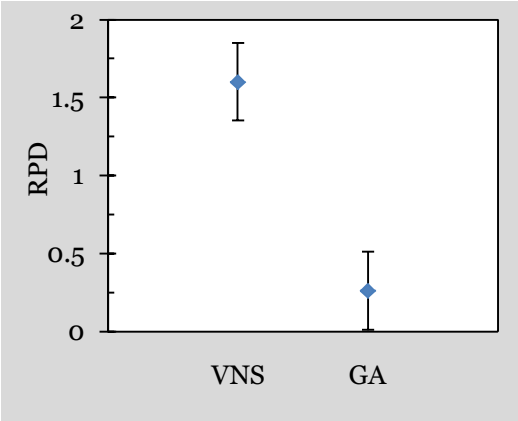

Fig. 5. The average RPD and LSD interval of the algorithms

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.

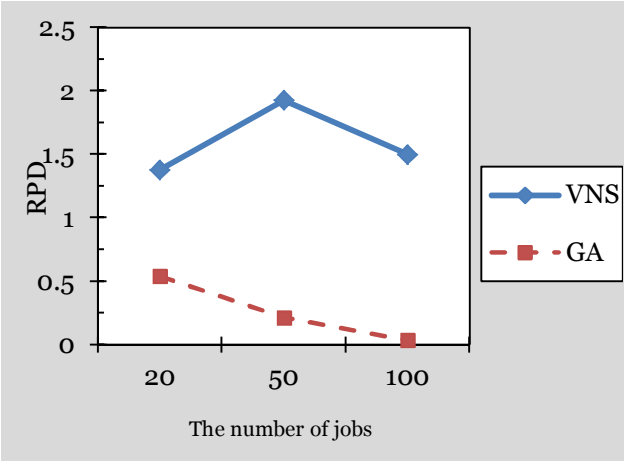

Fig. 6. The average RPD of the algorithms versus the problem size

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.