Applying Metaheuristic Algorithms on a Two Stage Hybrid Flowshop Scheduling Problem with Serial Batching

E. Ghafari, R. Sahraeian*

Department of Industrial Engineering, College of Engineering, Shahed University, Tehran, Iran

ABSTRACT

In this paper the problem of serial batch scheduling in a two-stage hybrid flow shop environment with minimizing Makespan is studied. In serial batching, it is assumed that jobs in a batch are processed serially, and their completion time is defined to be equal to the finishing time of the last job in the batch. The analysis and implementation of the prohibited transference of jobs among machines of stage one in serial batch is the main contribution of this study. Machine set-up and ready time for all jobs are assumed to be zero and no Preemption is allowed. Machines may not breakdown, but at times they may be idle. As the problem is NP-hard, a simulated annealing and genetic algorithm are proposed to provide near-optimal solutions. Since this problem has not been studied previously, therefore, a lower bound is developed for evaluating the performance of the proposed SA and GA solutions. Many test problems have been solved using SA and GA; results show both solving procedures provide near-optimum solutions regarding the lower bound solution. In the case of large scale problems, solutions provided by GA overcome those from SA algorithm.

1. INTRODUCTION

As industries are facing increasingly competitive situations, many classical manufacturing systems shift to novel environments such as hybrid flow shop in which a combination of flow shop and parallel machines operates together. A hybrid flow shop environment is similar to flow shop, but in at least one stage the number of machines is more than one. There are two types of batch productions, namely, serial batches and parallel batches. In serial batches, jobs within the same batch are processed sequentially, while in parallel batches a group of jobs precede through a machine and are processed simultaneously. Ribas et al. [1]. Implementations of hybrid flow shop can be found in various industries including automotive, chemical, and metallurgical industries, and iron manufacturing. In this paper, hybrid flow shop with serial batching has been considered. It is assumed that jobs in a batch are processed serially, and their completion time is defined to be equal to the finishing time of the last job in the batch. The processing time of a batch equals to the sum of the processing time of all the jobs in the batch. The literature on this subject consists of two sections; the first section considers hybrid flow shop scheduling and the second one considers batch scheduling. A survey of scheduling literature in hybrid flow shop environment has been conducted by Ribas et. al [1]. They considered previous research works in three different points of view, including processing complexity, scheduling criteria and approaches to hybrid flow shop (HFS) scheduling. Researches show that HFS problems in processing complexity situations are usually grouped into three categories: (1) two-stage HFS, (2) three-stage HFS, and (3) k-stage HFS. Research in scheduling criteria shows two types of categories: (1) based on flow time and (2) based on due dates. Research in terms of approaches to the hybrid flow shop scheduling is grouped into three categories: (1) Exact algorithms, (2) Heuristics and (3) Metaheuristics approaches. In the recent decade, most of researches have been dedicated to HFS; they impose various constraints on the problem to get closer to the real world problems. Botta-Genoulaz [2] has considered scheduling in hybrid flow shop environment with precedence constraints and time lags to minimize maximum lateness. He presented six heuristics to solve this problem). Sawik [3] has used
mixed integer programming for scheduling flexible flow lines with Limited intermediate buffers to minimize Cmax. Riane et al. [4] have presented an integrated production planning and scheduling system for hybrid flowshop production lines. Gupta et al. [5] have considered heuristics for hybrid flow shop scheduling with controllable processing times and assignable due dates. They proposed constructive algorithms using job insertion techniques and iterative algorithms based on local search. Oguz et al. [6] have proposed heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flowshop. Engin and Doyen [7] proposed an artificial immune system approach, for solving the hybrid flowshop scheduling problem with minimizing maximum completion times. Oguz et al. [8] considered hybrid flowshop scheduling problem with multiprocessor task system and minimizing maximum completion time. Problems with multi-processor task systems relax the limitation of the classical parallel model by permitting tasks that require more than one processor simultaneously. Morita and Shio [9] have used hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem.

Tang et al. [10] have used heuristic combined artificial neural networks to schedule hybrid flow shop with sequence dependent setup times. Ruiz and Maroto [11] applied a genetic algorithm on hybrid flowshops with sequence dependent setup times and machine eligibility (2006). Tang et al. [12] have considered a new lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. Zandieh et al. [13] have used an immune algorithm approach to hybrid flow shops scheduling with sequence dependent setup times. Voss and Witt [14] studied hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements by minimizing the weighted tardiness. Their mathematical model was based on the well-known resource constrained project scheduling problem. Chen and Chuen [15] considered Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. Figielska [16] has used a genetic and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources. Naderi et al. [17] have considered an improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. Jabbarizadeh et al. [18] studied hybrid flexible flowshops with sequence-dependent setup times and machine availability constraints. They proposed 3 heuristics and 2 metaheuristics based on genetic algorithm and simulated annealing. Behnamian and Fatemi Ghomi [19] proposed hybrid flowshop scheduling with machine and resource-dependent processing times to minimize makespan and total resource allocation costs.

The solution methodology, which can be seen from the literature, for the hybrid flow shop problems can be classified in three groups; (1) the exact method such as branch and bound technique and mathematical programming, (2) the metaheuristic approach such as tabu search, simulated annealing and genetic algorithm, and (3) the heuristic algorithms. There are many researches in the literature considering batch scheduling. Sawik [20] proposed a mixed integer programming formulation for serial batch scheduling in flexible flow lines with limited intermediate buffers. Yuan et al. [21] have studied the unbounded single machine parallel batch scheduling problem with family jobs and release dates to minimize Cmax. They proposed a dynamic programming model to solve it. Li and Yuan [22] have considered a dynamic programming formulation in order to minimize the Cmax, machine occupation time and stocking cost in the single machine parallel batch scheduling problem. Ruiz and Maroto [11] considered a genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. Husseinzadeh Kashan et al. [23] considered a hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. They proposed a hybrid genetic heuristic (HGH) to minimize makespan objective. Nong et al. [24] have studied online scheduling in the single-machine environment considering parallel-batching of jobs. Bellanger and Oulamara [25] have considered Scheduling hybrid flowshop term with parallel batching machines and compatibilities, where two stages with the second one containing batching machines is solved using tailored heuristics.

The remainder of the paper is organized as follows: Section 2 describes the problem in detail and presents a lower bound. Section 3 explains the proposed simulated annealing. Calibration of the proposed algorithm is presented in section 4. In section 5, experimental results and comparison of the proposed algorithm with lower bound is reported. Section 6 concludes the paper and provides some directions for future studies.

2. PROBLEM STATEMENTS

2.1 Problem Description In this paper, a hybrid flowshop serial batch (HFSB) is put forth for study. It is assumed that the completion time of all jobs in a batch is defined as the finishing time of the last job in the batch, i.e., the processing time of a batch equals to the sum of the processing times of all jobs in the batch. It is supposed that there are n jobs brought into the batches which should be processed on k stages. In stage j we have a number of machines; all the jobs in batches have the same production route. Each job should undergo all stages and be processed in each stage by only one
machine. Machine set-up and ready time for all jobs are assumed to be zero and no preemption is allowed. Machines may not breakdown, but at times they may be idle. Furthermore, it is also assumed that machines in each stage are identical and jobs arrive in batches with a different batch size. To the best of our knowledge, no general method has been proposed for the HFS with serial batch. In this study, it is considered two stages in HFSB environment which consists of m identical parallel machines in each stage. The first aim is to schedule the batches on the machines of stage one and second one is to schedule jobs which released from stage one to the machines of stage two such that the makespan is minimized.

Here, a two-stage HFSB system is considered which encompasses restriction of jobs transportation among machines in stage one, e.g. obstruction of jobs transportation among machines in x-ray stage due to thick insulation walls. Therefore, in stage one all jobs of the selected batch should be performed on only one of the machines. Consequently, in stage two jobs can be done on each of machines. This problem has been shown in Figure 1. The processing times and the batches size are known and non-identical. Considering the well-known three field notation $\alpha/\beta/\gamma$ for scheduling problems and the extension for hybrid flow shops proposed by Ribas et al. [1], the real production problem considered here can be noted as $HF(\{P_{m1}^1, P_{m1}^2\}; S - \text{batch} | C_{\text{max}})$. The proposed notation shows serial batch arrival in a hybrid flowshop environment with two stages each with m identical parallel machines. The aim of this paper is not to propose a mathematical formulation; but, a novel problem scheme is proposed and a SA is developed to solve the proposed problem. The proposed algorithm is intended to determine batches and job sequences in a two stage hybrid flowshop. Evolutionary search approaches have been successfully applied to a number of combinatorial optimization problems [18, 23]. An evolutionary search approach based on an SA can generate a good solution to the model in a reasonable computational time.

2. 2. NP-hardness of the Problem

Gupta [26], and Hoogeveen et al. [27], proved that the two stage hybrid flow shop scheduling problem is NP-hard when the objective is to minimize the makespan even if there is only one machine on the first stage and there are two machines on the second stage. On the other hand, if there is only one stage in which there are identical parallel machines, the problem under study HFSB reduces to $P || C_{\text{max}}$, which itself is NP-hard (Garey and Johnson [28]). Therefore, it can be concluded that the problem is NP-hard.

2. 3. A Lower Bound on the Optimal Makespan

In this section a lower bound (LB) is proposed for the $HF(\{P_{m1}^1, P_{m1}^2\}; S - \text{batch} | C_{\text{max}})$ problem based on lower bounds of $P_{m1}||C_{\text{max}}$. The lower bound is utilized to evaluate performance of results obtained from proposed SA and GA algorithms. A general lower bound for $P_{m1}||C_{\text{max}}$ is presented by Haouari et al. [29] as follows:

$$LB = \left\lceil \frac{1}{m} \sum_{i} p_i \right\rceil$$ (1)

Here, $p_i$ shows the processing times of jobs. Based on general LB in Equation (1), a lower bound for $HF(\{P_{m1}^1, P_{m1}^2\}; S - \text{batch} | C_{\text{max}})$ can be derived by Equation (2):

$$LB = \left\lceil \frac{1}{m} \sum_{i} p_i \right\rceil + \min(p_{i1})$$ (2)

where $p_{i1}$ denotes processing time of job i on stage 1, $p_{i2}$ processing time of job i on stage 2, m number of machines on stage 1, and n number of all jobs.

Example 1.2 illustrates the LB considering a $HF(\{P_{m1}^1, P_{m1}^2\}; S - \text{batch} | C_{\text{max}})$ problem with 5 batches, the processing time of each batch in stages, and batch sizes are given in Table 1 and LB computed as follows:

This lower bound is obtained from Equation (1) by adding the minimum processing time of jobs in stage 2.

### Table 1. Processing times of jobs for example 1.2

| Batch number | Batch size | Job number(i) | $P_{i1}$ | $P_{i2}$ |
|--------------|------------|---------------|---------|---------|
| 1            | 5          | 1-2-3-4-5     | 7       | 4       |
| 2            | 6          | 6-7-8-9-10-11 | 9       | 7       |
| 3            | 4          | 12-13-14-15   | 8       | 5       |
| 4            | 7          | 16-17-18-19-20-21-22 | 10 | 6       |
| 5            | 3          | 23-24-25      | 5       | 8       |

Example 1.2 illustrates the LB considering a $HF(\{P_{m1}^1, P_{m1}^2\}; S - \text{batch} | C_{\text{max}})$ problem with 5 batches, the processing time of each batch in stages, and batch sizes are given in Table 1 and LB computed as follows:

$$LB = \left\lceil \frac{1}{3} \sum_{i} p_i \right\rceil + \min(p_{i1}) = \left\lceil \frac{1}{3} \times \left[(5 \times 7) + (6 \times 9) + (4 \times 8) + (7 \times 10) + (3 \times 5)\right]\right\rceil + 4 = 73$$
3. SIMULATED ANNEALING

SA is a smart random-search technique exploiting an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure and search for a minimum in a more general system [30]. SA is a probabilistic algorithmic search procedure which has exhibited some promise when applied to combinatorial NP-hard problems. SA enjoys explicit mechanism, called acceptance criterion, which enables it to partially avoid getting trapped in local optima. The acceptance criterion decides probabilistically whether to accept a new solution or to reject it. The algorithm employs a random search which not only accepts superior changes ameliorating current solution, but also might accept inferior changes deteriorating the current solution. The probability of inferior solution acceptance depends on the parameter called temperature or $T$. When $T$ is high, there is more probability for inferior solutions being accepted. SA starts from a high initial temperature $T_0$ and this temperature is gradually decreased by means of a mechanism called cooling system. One characteristic of this algorithm is that it proceeds sequentially and slowly toward the area around current search area by a certain probability mechanism. As execution continues and $T$ falls, fewer inferior solutions are tolerated (or those that are tolerated are of smaller magnitude). $T$ decreases until a freezing temperature is reached. Various parameters and operators contribute to augment the final performance of the SA such as: initial solution, encoding scheme, number of temperatures between initial and final temperatures, number of iteration of each neighborhood search structure, initial temperature, types of cooling schedule and neighborhood search structures. In the following sections we describe all parameters and operators used in the proposed SA.

3.1. Solution Encoding and Initialization

Encoding schemes are used to make a candidate solution recognizable for algorithms. A proper encoding scheme plays a key role in maintaining the search effectiveness of any algorithm. Two commonly used approaches in the literature are job-based [11] and random key [31] representations. In job-based representation, the permutation of jobs is determined, and then by a dispatching rule jobs are assigned to machines. In HFS problems without considering SDST, the first available machine results in the earliest completion time, but while taking into account SDST HFS, this approach is not effective [11]. If setup times are considered in HFS, the way in which we assign jobs to machines is modified accordingly, meaning that each job is assigned to the machine that accomplishes the job at earliest time in a given stage.

![Illustration of a candidate solution in random key representation.](image)

The second random key representation was proposed by Norman and Bean [32]. They used the following solution representation for an identical multiple machine problem. Each batch is assigned a real number whose integer part is the machine number to which the batch is assigned, and its fractional part is used to sort the batches assigned to each machine. This representation is used for the batches in the first stage. The assignment of jobs to machines in subsequent stages has to go through the machines according to the foregoing procedure. For example, consider a problem with four batches ($b = 4$), two stages ($s = 2$), two machines at stage one ($m (1) = 2$). For this problem we must generate four random numbers from uniform distribution $(1, 1 + m (1))$ in first stage of process (Figure 2). As shown in Figure 2, each of the blocks denotes a batch. In this example, batches 2 and 4 are assigned to machine 1; also batches 1 and 3 are assigned to machine 2. The order of batches to be scheduled on machine 1 is batch 2 followed by batch 4, and the order of batches to be scheduled on machine 2 is batch 3 followed by batch 1. In this study we used second mechanism means random key representation to show algorithm solution. Many researchers have emphasized on the influential initial solution on the quality of final results of algorithms [11]. What has been utilized so far by majority of authors to generate initial solution for their algorithms has been random generation of initial solution or making use of underperforming heuristic methods which have led them to poor quality solutions. Hence, meticulous consideration should be given to intelligently select the initialization procedure in order to acquire a satisfactory level of solution quality in such an NP-hard problem. To this end, we use NEH (first proposed by Nawaz et al. [33]) for completion time objective.

To explicitly clarify the initialization procedure, NEH is explained. The idea behind the NEH algorithm is that batches with high processing times on all machines should be scheduled as early as possible. NEH procedure consists of three steps:

1. The total processing times for all the batches on the $m$ machines are calculated.
2. The batches are sorted in descending order of $p_i$.
3. Take batch $b_i, i = 1, ..., b$ and find the best schedule by placing it in all the possible $i$ positions in the sequence of batches that are already scheduled. For example, if $i = 5$, the already built sequence would contain the first four batches of the sorted list.
calculated in step 2, then, the fifth batches could be placed either in the first, in the second, in the third, in the fourth or in the last position of the sequence. The best sequence of the five would be selected for the next iteration.

3.2. Neighborhood Search Structure Neighborhood search structure generates a new solution from current candidate solution by making a slight change in it. Many different NSSs have been applied to scheduling problems. These NSSs must work in a way that they avoid generating infeasible solutions. In this paper, four different NSSs are considered. The first one is SWAP operator in which the RKs of two randomly selected operations are swapped. The second one is a SHIFT operator in which the RK of one randomly picked operation is randomly regenerated. The third one is an INVERSION operator in which the RKs between two randomly selected cut points are reversed. In fourth NSS, we aim at working on the precise determination of the neighborhood size which strongly influences the success and failure of SA. If the neighborhood size is too small, then the resultant process will not be able to move around the search space quickly enough to reach the optimum within a reasonable amount of time. On the other hand; if the neighborhood size is too large, then the process essentially performs a random search with the next possible state being chosen practically uniformly over the search space. Intuitively, it turned out that the neighborhood search structure that strikes a compromise between these extremes seems appropriate. According to the corresponding research findings, and also in order to diversify the search space to avoid getting trapped in local optima, we have been motivated to propose a novel NSS to assure the effectiveness of SA. Doing so, we have tested several combinations of small and large neighbor sizes. Finally, it appears that the following NSS provides the most convincing results: In this NSS, for generating a new neighbor in each temperature i, x best is improved and the counter proceeds. There exist three types of cooling schedules in the literature (more details could be found in [35]):

1. Linear cooling rate:
   \[ T_i = T_0 - \frac{T_f - T_0}{N}; i = 1, 2, ..., N \]

2. Exponential cooling rate:
   \[ T_i = \frac{A}{L+1} + B; A = \frac{(T_0 - T_f)(N+1)}{N}; B = T_0 - \lambda i; i = 1, 2, ..., N \]

3. Hyperbolic cooling rate:
   \[ T_i = \frac{1}{2}(T_0 - T_f)(1 - tgh(\frac{10N}{N} - 5)) + T_f; i = 1, 2, ..., N \]

where \( T_0 \) and \( T_f \) are initial temperature and stopping temperature and temperature of iteration l, respectively. \( N \) and \( tgh \) are desired number of temperature between \( T_0 \) and \( T_f \) and tangent hyperbolic, respectively. Figure 4 shows how to decrease the temperature by each cooling schedule.
Despite the fact that the exponential cooling rate has been recognized as an appropriate cooling schedule for the SA [34], we again explore the optimal type of cooling schedule in our parameter tuning subsection to make sure we have selected all the influential factors on the performance of the SA optimally. Initial experiment demonstrates that the temperature over the range 10–20 is appropriate for our problem, and the stopping temperature is fixed at 1.

**5. PARAMETER TUNING**

In this section, we aim at analyzing the behavior of the proposed SA and GA considering the above mentioned operators and parameters. Doing so, there exist various approaches to statistically design an experimental investigation. Each of these approaches is effective depending on the situation of experiment. Although the most widely used approach is a full factorial design, this approach is not always efficient because it becomes increasingly difficult to perform investigation when the number of factors is significantly high. To reduce the number of required tests, fractional factorial experiment (FFE) was developed [37]. FFE allow only a portion of the total possible combinations to estimate the main effect of factors and some of their interactions. Taguchi [38] developed a family of FFE matrices which finally reduce the number of experiments, but still provide sufficient information. In Taguchi method, orthogonal arrays are used to study a large number of decision variables with a small number of experiments. Taguchi separates the factors into two main groups: controllable and noise factors. Noise factors are those over which we have no direct control. Since elimination of the noise factors is impossible, the Taguchi method look for minimizing the effect of noise and to determine optimal level of important controllable factors based on the concept of robustness [39]. Besides determining the optimal levels, Taguchi identifies the relative significance of individual factors in terms of their main effects on the objective function. Taguchi has created a transformation of the repetition data to another value which is the measure of variation. The transformation is the signal-to-noise (S/N) ratio which explains why this type of parameter design is called robust design [39, 40]. Here, the term “signal” denotes the admirable value (mean response variable) and “noise” denotes the undesirable value (standard deviation). So, the S/N ratio indicates the amount of variation present in the response variable. The aim is to maximize the signal-to-noise ratio. Taguchi classifies objective functions into three selection. A genetic algorithm allows a population composed of many individuals to evolve under specified selection rules to a state that minimizes the cost function. The basic elements of a genetic algorithm that must be specified for any given implementation are representation, selection, crossover and mutation [36]. Figures 4a and 4b show the procedure of genetic algorithm, where POP is population size, $n_{max}$ is number of iterations or generation, $p_{cros}$ is crossover probability and $p_{mut}$ is mutation probability. According to the crossover probability, the crossover mechanism for two selected parents is applied if a random value is smaller than the crossover probability, otherwise the first parent is chosen without any changes.
categories: the smaller-the-better type, the larger-the-better type, and nominal-is-best type. Since almost all objective functions in scheduling are classified in the smaller the-better type, their corresponding S/N ratio [39] is:

\[
S/N \text{ ratio} = -10 \log_{10} (\text{objective function})^2
\]

As explained earlier, in this study, the SA factors are: (A) number of iteration between initial temperature and stopping temperature, (B) number of neighborhood searches in each temperature, (C) initial temperature and (D) cooling schedule type. Different levels of above mentioned factors are shown in Table 2. The GA factors are: (A) number of iteration, (B) population, (C) crossover probability and (D) mutation probability. Different levels of the above mentioned factors are shown in Table 3.

**TABLE 2. Factors and their levels of the SA**

| Factor                              | Symbol | Level | Type      |
|-------------------------------------|--------|-------|-----------|
| Number of temperatures between \(T_0\) and \(T_f\) | A      | 3     | A(1)-150  |
|                                     |        |       | A(2)-200  |
|                                     |        |       | A(3)-250  |
| Number of neighborhood searches in each temperature | B      | 3     | B(1)-30   |
|                                     |        |       | B(2)-60   |
|                                     |        |       | B(3)-100  |
| Initial temperature                 | C      | 3     | C(1)-10   |
|                                     |        |       | C(2)-15   |
|                                     |        |       | C(3)-20   |
| Cooling schedule type               | D      | 3     | D(1)-Linear |
|                                     |        |       | D(2)-Exponential |
|                                     |        |       | D(3)-Hyperbolic |

From standard table of orthogonal arrays, the \(L_9\) is selected as the fittest orthogonal array design which carries out all our minimum requirements. Table 4 shows four parameters characterizing the test problems. These parameters include the number of jobs \(n\), the number of machines on stages \(m\), the processing times \(p_{ij}\) and the batch size \(s_j\). To conduct the experiments, we have 18 \((3\times1\times3\times2)\) combinations of these parameters. Optimal levels of SA parameters for all of the 18 combinations are shown in Table 4, which we used 7 replications for each experiment. In Table 3, Column 1 represents the run code for the instances. For example, J1m1s1p1 implies that the test instance belongs to 100 jobs with number of machines, batch sizes and processing times generated at level 1. All the experiments were coded in Matlab 7.1 and were run on a computer with a 2.2 GHz Pentium(R) Dual-Core CPU with 2.00GB of RAM.

**TABLE 4. Different Parameters for test problems.**

| Parameters            | Levels                          |
|-----------------------|---------------------------------|
| Number of jobs        | 100, 300, 500                   |
| \(p_{ij}\)            | Uniform [1, 15]                 |
| Batch size            | Uniform [2, 10], Uniform [10, 15] and Uniform [15, 30] |
| Number of machines    | Uniform [2, 5] and Uniform [5, 10] |

**TABLE 5. Optimal (Tuned) level of SA parameters**

| Name                  | A    | B    | C    | D    |
|-----------------------|------|------|------|------|
| J1m1s1p1              | 150  | 30   | 10   | Linear |
| J1m2s2p1              | 150  | 30   | 10   | Linear |
| J1m1s3p1              | 150  | 60   | 15   | Exponential |
| J1m2s1p1              | 150  | 30   | 10   | Linear |
| J1m1s2p1              | 200  | 30   | 10   | Exponential |
| J1m2s3p1              | 150  | 30   | 15   | Exponential |
| J2m1s1p1              | 250  | 60   | 15   | Hyperbolic |
| J2m2s2p1              | 200  | 60   | 10   | Exponential |
| J2m1s3p1              | 200  | 100  | 20   | Exponential |
| J2m2s1p1              | 200  | 100  | 20   | Linear |
| J2m1s2p1              | 200  | 60   | 15   | Hyperbolic |
| J2m2s3p1              | 150  | 100  | 20   | Exponential |
| J3m1s1p1              | 250  | 60   | 15   | Hyperbolic |
| J3m2s2p1              | 250  | 60   | 10   | Hyperbolic |
| J3m1s3p1              | 200  | 100  | 20   | Exponential |
| J3m2s1p1              | 250  | 100  | 20   | Hyperbolic |
| J3m1s2p1              | 250  | 100  | 20   | Hyperbolic |
| J3m2s3p1              | 250  | 100  | 20   | Hyperbolic |
### TABLE 6. Optimal (Tuned) level of GA parameters

| Name       | A   | B   | C   | D   |
|------------|-----|-----|-----|-----|
| J1m1s1p1  | 50  | 10  | 0.90| 0.50|
| J1m2s2p1  | 50  | 10  | 0.90| 0.80|
| J1m1s3p1  | 50  | 25  | 1.00| 0.50|
| J1m2s1p1  | 50  | 25  | 1.00| 0.50|
| J1m1s2p1  | 100 | 25  | 0.90| 0.50|
| J1m2s3p1  | 50  | 10  | 0.90| 0.80|
| J2m1s1p1  | 50  | 10  | 0.90| 0.50|
| J2m2s2p1  | 50  | 25  | 0.90| 0.80|
| J2m1s3p1  | 100 | 25  | 1.00| 0.50|
| J2m2s1p1  | 100 | 40  | 1.00| 0.80|
| J2m1s2p1  | 100 | 25  | 1.00| 0.50|
| J3m1s1p1  | 100 | 40  | 0.90| 0.50|
| J3m2s2p1  | 100 | 40  | 0.90| 0.80|
| J3m1s3p1  | 100 | 40  | 1.00| 0.80|
| J3m2s3p1  | 150 | 40  | 0.90| 0.80|

### 6. COMPUTATIONAL EXPERIMENTS

In this section the performance of the proposed algorithm would be evaluated and compared by conducting some experiments. Several test problems with considering some parameters were generated and the results analyzed. The following subsections describe the details of the experiments.

#### 6.1. Generating Data

To evaluate the performance of the algorithm in varied situations, four parameters were characterized in generating the test problems; these parameters are the same as what we explained in Section 4 in Table 2. For example, for parameter job, three levels of low, medium, and high, with 100, 300, and 500 jobs were considered, respectively. Processing times, sizes of the batches and number of machines in two stages were generated from the discrete uniform distribution. There are 18 types of problems \((3 \times 3 \times 2 \times 1 = 18)\) generated when combining different values given for these 4 parameters, and 40 data sets are generated randomly for each type, creating 720 problems all together. Tables 5 and 6 present the results obtained from the SA and the GA, respectively. Columns 1 and 2 represent the run code for the instances same as what we explained in section 4 respectively for the SA and the GA. Columns 3 and 4 report the average of makespans and average of relative deviation from lower bound (DEVLB), respectively for the SA and the GA.

Relative deviation =

\[
\frac{\text{Obtained makespan from the proposed algorithm} - \text{Lower bound}}{\text{Lower Bound}}
\]

Tables 7, 8 and 9 present the results for the 100, 300 and 500 jobs instance, respectively.

#### TABLE 7. Results for 100 job instances

| Run code | Average of Makespans (SA) | Average of DEVLB (SA) | Average of Makespans (GA) | Average of DEVLB (GA) |
|----------|---------------------------|-----------------------|---------------------------|-----------------------|
| J1m1s1p1 | 275                       | 0.072                 | 275                       | 0.072                 |
| J1m2s2p1 | 169                       | 0.053                 | 169                       | 0.053                 |
| J1m1s3p1 | 243                       | 0.081                 | 243                       | 0.081                 |
| J1m2s3p1 | 141                       | 0.026                 | 141                       | 0.026                 |
| J2m1s2p1 | 458.14                    | 0.098                 | 457.68                    | 0.098                 |
| J2m2s3p1 | 148                       | 0.058                 | 148                       | 0.058                 |

#### TABLE 8. Results for 300 job instances

| Run code | Average of Makespans (SA) | Average of DEVLB (SA) | Average of Makespans (GA) | Average of DEVLB (GA) |
|----------|---------------------------|-----------------------|---------------------------|-----------------------|
| J2m1s1p1 | 667                       | 0.040                 | 666.87                    | 0.039                 |
| J2m2s2p1 | 804.27                    | 0.020                 | 804.02                    | 0.020                 |
| J2m1s3p1 | 756.36                    | 0.070                 | 756.11                    | 0.070                 |
| J2m2s1p1 | 578.43                    | 0.110                 | 578.61                    | 0.110                 |
| J2m1s2p1 | 460.35                    | 0.120                 | 459.57                    | 0.119                 |
| J2m2s3p1 | 458.14                    | 0.100                 | 457.68                    | 0.098                 |

#### TABLE 9. Results for 500 job instances

| Run code | Average of Makespans (SA) | Average of DEVLB (SA) | Average of Makespans (GA) | Average of DEVLB (GA) |
|----------|---------------------------|-----------------------|---------------------------|-----------------------|
| J3m1s1p1 | 1414.21                   | 0.063                 | 1406.83                   | 0.061                 |
| J3m2s2p1 | 1367.19                   | 0.013                 | 1362.41                   | 0.011                 |
| J3m1s3p1 | 994.27                    | 0.027                 | 990.75                    | 0.024                 |
| J3m2s1p1 | 994.54                    | 0.096                 | 988.94                    | 0.093                 |
| J3m1s2p1 | 689.34                    | 0.125                 | 683.02                    | 0.121                 |
| J3m2s3p1 | 513                       | 0.085                 | 511.66                    | 0.084                 |
The objective function of minimizing makespan has been two-stage hybrid flow shop environment with the concentration on an exact value that indicates its good quality of convergence. In this paper, a serial batch scheduling problem in a 7. CONCLUSIONS AND FUTURE RESEARCH

In this paper, a serial batch scheduling problem in a two-stage hybrid flow shop environment with the objective function of minimizing makespan has been proposed. No previous work in the literature of scheduling has dealt with the serial batching problem of this kind (to the best of our knowledge). Since it is a generalized form of $P||C_{\text{max}}$ then the problem is NP-hard. A lower bound for this problem has been proposed based on other researches and our heuristic. The simulated annealing and the genetic algorithm have been used to solve the problem. In order to evaluate the performance of the SA and the GA, a large number of randomly problems generated and results compared with lower bound. Results showed that SA and GA have obtained a near optimal solution in reasonable time. Based on the computational results, GA outperforms SA, especially in the large-sized problems.

In the future research, other scheduling objectives such as minimizing the sum of earliness/tardiness and maximum lateness can be tested. The restriction of machine eligibility is quite common in practice. As a consequence, the development of a heuristic for problems with machine eligibility is a practical area of research. Finally, the concept of batch arrivals can be extended to batch delivery [41], which is also encountered quite often in the real world.

8. REFERENCES

1. Ribas, I., Leisten, R. and Framiñan, J.M., “Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective”, Computers & Operations Research, Vol. 37, No. 8, (2010), 1439-1454.
2. Botta-Genoulaz, V., “Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness”, International Journal of Production Economics, Vol. 64, No. 1, (2000), 101-111.
3. Sawik, T., “Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers”, Mathematical and Computer Modelling, Vol. 31, No. 13, (2000), 39-52.
4. Riane, F., Artiba, A. and Iassinovski, S., “An integrated production planning and scheduling system for hybrid flowshop organizations”, International Journal of Production Economics, Vol. 74, No. 1, (2001), 33-48.
5. Gupta, J.N., Krüger, K., Lauff, V., Werner, F. and Sotskov, Y.N., “Heuristics for hybrid flow shops with controllable processing times and assignable due dates”, Computers & Operations Research, Vol. 29, No. 10, (2002), 1417-1439.
6. Öğuz, C., Fikret Ercan, M., Edwin Cheng, T. and Fung, Y.-F., “Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop”, European Journal of Operational Research, Vol. 149, No. 2, (2003), 390-403.
7. Engin, O. and Döyen, A., “A new approach to solve hybrid flow shop scheduling problems by artificial immune system”, Future Generation Computer Systems, Vol. 20, No. 6, (2004), 1083-1095.
8. Öğuz, C., Zinder, Y., Ha Do, V., Janiak, A. and Lichtenstein, M., “Hybrid flow-shop scheduling problems with multiprocessor task systems”, European Journal of Operational Research, Vol. 152, No. 1, (2004), 115-131.
9. Morita, H. and Shio, N., “Hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem”,

Table 10 represents the best and the worst makespans of both algorithms for all instances. As a performance criterion, it is desirable to take into consideration the amount of variation between the worst and the best performance of each algorithm.

6. 2. Results Analysis

Computational analysis shows that in all test problems, the GA performs better than the SA. Especially, its superiority over SA has been proved in large sized problems. This can be related to the high rate of convergence in the GA because of its effective components and the number of neighboring generation at same time compared to the SA. For each problem instance, the SA reports wide range of variation (due to big difference between the worst and the best cases) where the results for the GA show its concentration on an exact value that indicates its good quality of convergence.

| Run code | Best of Makespans SA | Worst of Makespans GA |
|----------|-----------------------|-----------------------|
| J1m1s1p1 | 275                   | 275                   |
| J1m1s2p1 | 169                   | 169                   |
| J1m1s3p1 | 243                   | 243                   |
| J1m2s1p1 | 141                   | 141                   |
| J1m2s2p1 | 117                   | 117                   |
| J1m2s3p1 | 148                   | 148                   |
| J1m3s2p1 | 801                   | 809                   |
| J2m1s1p1 | 659                   | 671                   |
| J2m1s2p1 | 574                   | 583                   |
| J2m1s3p1 | 455                   | 467                   |
| J2m2s1p1 | 1410                  | 1421                  |
| J2m2s2p1 | 1345                  | 1372                  |
| J3m1s1p1 | 991                   | 1019                  |
| J3m1s2p1 | 987                   | 1002                  |
| J3m1s3p1 | 685                   | 698                   |

Table 10. Best and Worst results of all instances
Appling Metaheuristic Algorithms on a Two Stage Hybrid Flowshop Scheduling Problem with Serial Batching

E. Ghafari, R. Sahraeian

Department of Industrial Engineering, College of Engineering, Shahed University, Tehran, Iran

Paper history:
Received 30 April 2013
Received in revised form 15 September 2013
Accepted 12 December 2013

Keywords:
Scheduling
Hybrid Flowshop
Serial Batching
Simulated Annealing
Genetic algorithm
Taguchi Method

doi: 10.5829/idosi.ije.2014.27.06c.08
