An Effective Hybrid Genetic Algorithm for Hybrid Flow Shops with Sequence Dependent Setup Times and Processor Blocking

Mostafa Zandieh a,*, Eghbal Rashidi b

aDepartment of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, G.C., Tehran, Iran
bDepartment of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

Received 15 Jul., 2009; Revised 5 Aug., 2009; Accepted 29 Sep., 2009

Abstract

Hybrid flow-shop or flexible flow shop problems have remained subject of intensive research over several years. Hybrid flow-shop problems overcome one of the limitations of the classical flow-shop model by allowing parallel processors at each stage of task processing. In many papers the assumptions are generally made that there is unlimited storage available between stages and the setup times are neglected or considered independent from sequences of jobs. In this paper we study the hybrid flow shop problems with sequence dependent setup times and processor blocking. We present an effective hybrid genetic algorithm with some state-of-the-art procedures for these NP-hard problems to minimize total completion time or makespan. We established a benchmark to draw an analogy between the performance of our algorithm and RKGA. The obtaining results clearly show the superiority performance of our algorithm.

Keywords: Hybrid flow shop; sequence dependent setup times; Processor blocking; Genetic algorithm; Local search.

1. Introduction

Scheduling is an important process widely used in manufacturing, production, management, computer science, and so on. In simple flow-shop problems, each machine center (stage) has just one machine. If at least one stage has more than one machine, the problem is called a flexible flow-shop or hybrid flow shop problem. Hybrid Flow Shops (HFS) are manufacturing environments in which a set of jobs must be processed in a series of stages with multiple parallel machines [39]. Given the NP-hard nature, [7], [10], of this problem, to be effective, generic methods rely on the information provided by specialized ones. Often, this information is decided by the objective being optimized. Consequently, the resulting hybrids are not as effective on problems with other objectives. This is a serious shortcoming since, in practice; it is desirable to have solution tools that are reliable on problems with different objectives. Successful algorithms for HFS use Meta-heuristics to schedule the first stage of the shop and a simple constructive procedure to schedule the rest, [40], [43], [46], [31], [25]. Genetic Algorithms (GA) have been used successfully in exploiting this idea. In [14] and [15], for instance, a Random Keys Genetic Algorithm (RKGA) performed well against many specialized heuristics and meta-heuristics such as the problem space-based search method, [16], on HFS problems with sequence dependent setup times. In [31], a GA with a permutation representation of the individuals, and many variants of the crossover operator, also performed well against several heuristics such as ant colonies, Tabu Search, Simulated Annealing, other GA’s and deterministic methods on HFS with sequence dependent setup times and machine eligibility. Note that, most of these methods and the ones reviewed in [45] and [19] consider problems with makespan as the optimization criterion, mainly.

A single machine sequence-dependent setup scheduling problem is equivalent to a traveling-salesman problem and is NP-hard [20]. Even for a small system, the complexity of this problem is beyond the exact theories [32]. Hybrid flow shop problems are significantly more complex than the regular single machine scheduling. On the other hand, Gupta [8] shows the flow shop with multiple processors (FSMP) problem with only two stages ($t=2$) to be NP-hard even when one
of the two stages contains a single machine. Since the FSMP problem can be considered as a specific case of the hybrid flow shop, then we can conclude that this latter problem is also NP-hard [32]. Since these types of problems belong to NP-hard class, no exact method has so far been introduced to be able to tackle these problems within a reasonable amount of time. Hence, in this paper we aim to introduce an effective hybrid genetic algorithm for the problem considered. The researchers have studied various objective functions in production scheduling, ranging from minimizing makespan, maximum tardiness, total completion time, total tardiness, early and tardy penalties [44], and job waiting variance [18]. In this paper, we consider total completion time or makespan.

To the best of our knowledge there isn’t any study in the literature that applies heuristic or meta-heuristic algorithms to solve the HFLB with SDST constraint. Kurz and Askin [14] developed a random-keys genetic algorithm to solve the problem of flexible flow lines with sequence dependent setup times by minimizing the makespan. Zandieh et al. [48] introduce an immune algorithm for HFS with SDST constraint which outperforms the RKGA of Kurz and Askin. Tavakkoli-Moghaddam et al. [41] proposed a genetic algorithm (GA) with a novel, GA representation and operators to solve the FFLB by minimizing the makespan. Torabi et al. [42] proposed a hybrid genetic algorithm (HGA) to solve a lot-size and delivery scheduling problem in a simple supply chain, where a single supplier produces multiple components on an FFL and delivers them directly to an assembly facility (AF). Jenabi et al. [11] proposed two meta-heuristic algorithms, including the HGA and simulated annealing (SA), to solve a new 0–1 mixed-nonlinear mathematical model of the economic lot-sizing and scheduling problem in flexible flow lines with unrelated parallel processors over a finite planning horizon. The objective determines a cyclic schedule by minimizing the sum of setup and inventory holding costs per unit time without any stock-out. Akrani et al. [1] developed two heuristic approaches, including GA and an optimal enumeration method (OEM), to solve a new model of common cycle multi-product lot-sizing and scheduling problem in deterministic flexible flow shops with a finite planning horizon and limited intermediate buffers. The objective minimizes the sum of setup cost, inventory holding costs, and number of cycles. Kaczmarczyk et al. [12] proposed an improvement heuristic approach for scheduling of printed wiring board assembly in SMT lines. They considered the processor blocking and limited processor availability due to the scheduled downtimes. The heuristic approach, which is a combination of Tabu Search (TS) and set of dispatching rules, has a hierarchical structure based on the decomposition of the scheduling problem into two sub-problems: sequencing and assignment/timing solved sequentially.

So far, many literatures are available for scheduling problems with limited buffers. Hall and Sriskandarajah [9] provided a survey for scheduling problems with blocking and no-wait in process. Salvador [34] first considered hybrid flow shop with no buffers between stages. He applied branch-and-bound techniques to minimize makespan. Rajendran and Chaudhuri [28] utilized branch-and-bound but they also focus only on permutation schedules. Brah and Hunsucker [4] used branch-and-bound in the hybrid flow shop with an arbitrary number of stages and intermediate buffers. Moreover, they provide a means by which non-permutation schedules or schedules with inserted idle time can be created. Sawik [37] presented a mixed integer programming formulation for scheduling flexible flow line with limited buffers and also a FFS with limited intermediate buffers [35] and with no-process buffers [36]. In addition, batch scheduling in buffer constrained flow shop and flexible flow shop has also received academic attention [38, 27]. Norman [23] applied Tabu Search to schedule problems containing both sequence-dependent setup times and finite buffers but in flow shop environment. Wardono and Fathi [46] developed a Tabu Search for multi-stage parallel machine problem with limited buffers.

The rest of the paper is organized as follows: Section 2 is dedicated to describe the problem in detail. In Section 3 we elaborate our proposed algorithm. Section 4 discuss about robust calibration process by means of Taguchi method. Section 5 contains description of generation test data and computational results. At last Section 6 concludes the research and suggests some guidelines for future studies.

2. Problem description

Hybrid flow shops are generalization of simple flow shops. The line produces several different product types, and each product must be processed by at least one processor in each stage. A product, once its processing is completed on a processor in some stage, is transferred directly to either an available processor in the next stage (or another downstream stage depending on the product processing route), or a buffer ahead of that stage, when such an intermediate buffer is available [26]. When an intermediate buffer is unavailable, the product remains blocking the processor until a downstream processor becomes available. However, this blocking prevents processing of other products on the blocked processor.
This type of problem is referred to as a hybrid flow shop problem with processor blocking (HFLB). We also consider another assumption in this paper: the setup times are dependent on sequences of jobs. After finishing processing of one job and before starting processing of the next job, some sort of setup such as cleaning up or changing tools must be carried out. In SDST, the magnitude of this time depends on both the immediately preceding and current jobs to be performed on the same machine [47, 2, 5]. The major reason for this consideration is to tackle scheduling problems in a real manner and also because of the remarkable savings when setup times are explicitly consolidated in the scheduling decisions. Additionally, we consider that setup is non-anticipatory meaning that the setup can be started as soon as the machine and the job are available. With respect to the corresponding explanation, we take into consideration sequence-dependent setup times to further actualize our problem.

3. Proposed hybrid genetic algorithm

We want to use of fundamental concepts of RKGA. First Bean [3] has introduced a novel encoding representation using random numbers called random keys genetic algorithm. This type of GA differs in the solution representation from usual type. Randomness is the main characteristic of RKGA. This representation is widely used in the literature e.g. [14, 48, 6, 24]. According to Kurz and Askin [14] RKGA has been applied to resource allocation problems, quadratic assignment problems, multiple machine tardiness scheduling problems, job-shop makespan minimization problems and the generalized traveling salesman problem [23, 24, 13]. Kurz and Askin [14] reports good performance of RKGA in FFL with sequence-dependent setup times. Here, we present an effective genetic algorithm hybridized with a stochastic local search called HGA. Besides the local search procedure that we added to HGA, our HGA also enjoys a helpful procedure called Restart, which the RKGA did not have. We also use Taguchi method to tune the algorithm, which again differs from the experimental process that use in the RKGA and many other algorithms in the literature. However we use the same representation as used in the RKGA.

3.1. Genetic operators

3.1.1. Crossover

The well-known traveling salesman problems (TSPs) are frequently used to model and solve manufacturing scheduling [13]. Michalewicz [21] showed that crossover operators such as the partially mapped crossover (PMX), the order crossover (OX) and the cycle crossover (CX), can be used to handle TSPs. One of the most applied crossovers in the literature for random keys representation is parameterized uniform crossover (PMX). We evaluate three crossovers in the calibration process to pick the best one for our algorithm.

- Parameterized uniform crossover (PUX): This type of crossover uses of random numbers to determine which parent is selected to make current cell of children. For each job, a random number is generated. If the value is less than 0.7 the value from the “first” chromosome is copied to the new chromosome, otherwise the value from the “second” chromosome is selected.
- One point crossover (OPX): Two parent chromosomes are joined at a randomly selected crossover point somewhere along the length of the chromosome and the sections on either side are swapped.
- Two points crossover (TPX): Two positions are chosen at random in each parent and the segments between them are exchanged.

3.1.2. Mutation operator or random generation engine

- Swap mutation (SWAP): Two randomly selected positions are chosen and their contents swapped.
- Random generation (RG): It isn’t a mutation operator. Instead of mutating, chromosomes are generated randomly.
- Shift mutation (SHIFT): A randomly picked position in the sequence is relocated at another random position and the jobs between these two positions move along. For example, if we remove job at position 3 of the sequence and we want to reinsert it at position 7, jobs at positions from 4 to 7 slide and occupy positions from 3 to 6 [30].

3.2. Restart procedure

To avoid premature convergence in the population we have implemented a modified restart method. The restart scheme is based on [17] and is similar to the one implemented in [33]. According to Ruiz and Maroto [30] this method works as follows: At each generation we store the minimum makespan. If in the following generation the minimum makespan has not changed, we increment a counter. When this counter becomes higher than a control parameter called $G_r$, the following procedure is applied:

- Create a sorted list of Pop_size (size of population) elements with the $C_{\text{max}}$ of the chromosomes in ascending order.
3.3. Local search

The idea of adding an improvement phase to a GA has been widely exploited before. For example, Murata et al. [22] suggest an improvement phase by applying a local search step before selection and crossover in a GA. The drawback of this approach is that applying local search to all individuals in every generation results in a very slow GA. Our proposal is to apply a local search after selection, crossover and mutation, but not to all individuals in the population. We define an “enhancement probability” or $P_{enh}$ as follows: We draw a random uniformly distributed number between 0 and 1 and if this number is less than or equal to $P_{enh}$, the individual will undergo local search. For the improvement phase we make use of a simple method: for the chromosomes which have been determined to go under improvement phase, one machine number at stage one is chosen randomly; for this machine we select a random job which is on the machine considered and by using Swap mutation we try to find better position for the job. This method is simple and efficient as we can see from the section 5. A pseudo code is shown in Fig. 1. for better understanding of our HGA.

(1)  $\text{Program HGA:}$
(2)  \begin{verbatim}
Begin
(3)  Initialization ();  // Produce initial solutions
(4)  Iter := 1;  // Counter for number of iteration
(5)  Counter := 0;  // Counter used in Restart procedure
(6)  While stopping criterion is not met yet do
(7)  Begin
(8)    Evaluation ();
(9)    Elitism ();
(10)   Crossover ();
(11)   Mutation ();
(12)   For I := 1 to Pop_size do
(13)     Begin
(14)       Random (i) := a random number between 0 and 1;
(15)       If (Random(i) < = Penh) then
(16)         Local search (i);  // perform Local search for
(17)         individual number i
(18)       End;
(19)       Iter := Iter + 1;
(20)       Evaluate Min_Makespan ();
(21)       If (Min_Makespan (Iter) := Min_Makespan (Iter -1))
(22)         then
(23)           Counter := Counter +1
(24)         Else Counter := 0;
(25)       End;  // End while
(26)     End;
(27)     Evaluate Min_Makespan ();
(28)     Restart ();
(29)     End;  // End while
(30) End;
\end{verbatim}

(31)     Fig. 1. General outline of our proposed HGA (Pseudo code).
After you determine which factors affect variation, you can try to find settings for controllable factors that will either reduce the variation, make the product insensitive to changes in uncontrollable (noise) factors, or both.

In this study, the GA control factors are:

- Crossover type: 3 levels (PUX, OPX, TPX)
- Mutation type: 3 levels (RG, Shift, Swap)
- (Crossover rate, Mutation rate): 6 levels ((70, 10), (70, 15), (70, 20), (80, 5), (80, 10), (80, 15))
  (Elitism strategy is used for remain individuals of population)
- Population size (Pop_size): 3 levels (200, 250, 300)
- Enhancement probability: 3 levels (0.001, 0.01, 0.1)

Table 1
Factors and their levels

<table>
<thead>
<tr>
<th>Factor</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Crossover rate, Mutation rate)</td>
<td>(70, 10), (70, 15), (70, 20), (80, 5), (80, 10), (80, 15)</td>
</tr>
<tr>
<td>Mutation</td>
<td>RG, SHIFT, SWAP</td>
</tr>
<tr>
<td>Enhancement probability</td>
<td>0.001, 0.01, 0.1</td>
</tr>
<tr>
<td>Pop_size</td>
<td>200, 250, 300</td>
</tr>
<tr>
<td>Crossover</td>
<td>OPX, TPX, PUX</td>
</tr>
</tbody>
</table>

Different levels of the abovementioned factors are shown in Table 1. The L_{18} is selected as the fittest orthogonal array design. A set of 18 instances was generated as such: the set of instances comprises 9 combinations of \( n \) and \( g \), being \( n = \{6, 30, 100\} \) and \( g = \{2, 4, 8\} \) with the processing times uniformly distributed on \((1, 99)\) and setup times are uniformly distributed on \((1, 49)\) and \((1, 99)\). Number of machines is uniformly distributed from one to four. To yield more reliable information, we tackled each instance five times to finally average the results. Therefore, we have 18 results for each trial to do the statistical analyses. All experiments were performed in Delphi 2007 and run on a PC with 2.0 GHz Intel Core 2 Duo and 1 GB of RAM memory. The stopping criterion for calibration and comparative analogy is set to a CPU time limit fixed to \( n \times g \times 100 \) m. sec.

The response variable is based on the following performance measure:

Relative Percentage

\[
\text{Deviation (RDP)} = \frac{\text{Alg}_{sol} - \text{Best}_{sol}}{\text{Best}_{sol}} \times 100
\]

Where \( \text{Alg}_{sol} \) is the makespan obtained by a given algorithm alternative on a given instance and in this case \( \text{Best}_{sol} \) is the best makespan obtained for each instance.

After obtaining the results of the Taguchi experiment, they are transformed into S/N ratio. Fig. 2 shows the S/N ratio obtained at each level. As indicated in Fig. 2, better robustness of the algorithm is achieved when the parameters are set as shown indicated in Table 2.
<table>
<thead>
<tr>
<th>Factor</th>
<th>Optimal level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Crossover rate, Mutation rate)</td>
<td>(70, 10)</td>
</tr>
<tr>
<td>Mutation</td>
<td>SWAP</td>
</tr>
<tr>
<td>Enhancement probability</td>
<td>0.01</td>
</tr>
<tr>
<td>Pop_size</td>
<td>300</td>
</tr>
<tr>
<td>Crossover</td>
<td>OPX</td>
</tr>
</tbody>
</table>

### 5. Computational evaluation

In this section we aim to test efficiency and effectiveness of our proposed HGA against popular RKGA. To test we need to generate required data. Data required for a problem consist of the number of jobs, range of processing times, range of setup times, number of stages and number of machines per stage. A set of 36 instances was generated as such: the set of instances comprises 9 combinations of n and g, being n = {6, 30, 100} and g = {2, 4, 8} with the processing times uniformly distributed on (1, 99) and setup times uniformly distributed on (1, 24), (1, 49), (1, 99) and (1,124). The duration of sequence-dependent setup times is defined as 25%, 50%, 100%, and 125% percent of the processing time [33]. A random uniformly distributed number of machines ranging from one to four machines per stage are considered. For each instance we have generated 5 different problems. The algorithms are stochastic in nature and we have conducted five different replicates of each experiment to finally average the results. For the tests we use the same computing platform as in Section 5.

After obtaining results, ANOVA test is invoked. As regards the ANOVA’s models adequacy to the data we can say that all three hypotheses (normality, homogeneity of variance and independence of the residuals) were accepted at 95% confidence level. The means plot and LSD intervals for our HGA against RKGA are shown in Fig. 3. As it could be seen, the performances of our proposed HGA statistically supersede the RKGA. Our proposed algorithm, as we said and described before, uses an effective Local Search procedure which is one of the most important factors that makes our HGA to be successful against the RKGA. We also went through a logical and scientific way to fine tune the algorithm (Taguchi method). And we also examined more than one types and rates of crossovers and mutations and finally selected the best ones. The Restart procedure also helps our HGA to be more effective against the RKGA.

### 6. Conclusion and future work

In this paper we consider the hybrid flow shops with additional assumptions: sequence dependent setup times and processor blocking without intermediate buffer. The problem has proven to be NP-hard in strong sense; hence we introduce a genetic algorithm combined with simple stochastic local search developing an effective HGA. For the purpose of having a robust parameter tuning, we made use of Taguchi method. The obtained results demonstrated the superiority of our proposed HGA with respect to RKGA. As a direction for future research, it would be interesting to work on other metaheuristics, such as Particle Swarm Optimization, Tabu Search and Simulated Annealing, and compare them with our HGA, or to examine the performance of our algorithm in other complex scheduling problems, such as the flexible job shop and an open shop, to see whether the high performance of our HGA is transferable to other scheduling problems. Another direction for future research is to consider other realistic assumptions, such as machine availability constraints and transportation times between stages. Another opportunity for research is to consider the problems with other optimization objectives, such as total weighted tardiness or total completion time, or even multi-objective cases.
References


