9

A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem

Jen-Shiang Chen[1], Jason Chao-Hsien Pan[2] and Chien-Min Lin[2]
[1]*Far East University,* [2]*National Taiwan University of Science and Technology*
Taiwan, R.O.C.

1. Introduction

In many manufacturing and assembly facilities, a number of operations have to be done on every job. Often these operations have to be done on all jobs in the same order implying that the jobs follow the same route. These machines are assumed to be set up in series, and the environment is referred to as a flow-shop. The assumption of classical flow-shop scheduling problems that each job visits each machine only once (Baker, 1974) is sometimes violated in practice. A new type of manufacturing shop, the re-entrant shop has recently attracted attention. The basic characteristic of a re-entrant shop is that a job visits certain machines more than once. The re-entrant flow-shop (RFS) means that there are n jobs to be processed on m machines in the shop and every job must be processed on machines in the order of M_1, M_2, …, M_m, M_1, M_2, …, M_m, …, and M_1, M_2, …, M_m. For example, in semiconductor manufacturing, consequently, each wafer re-visits the same machines for multiple processing steps (Vargas-Villamil & Rivera, 2001). The wafer traverses flow lines several times to produce different layer on each circuit (Bispo & Tayur, 2001).

Finding an optimal schedule to minimize the makespan in RFS is never an easy task. In fact, a flow-shop scheduling, the sequencing problem in which n jobs have to be processed on m machines, is known to be NP-hard (Kubiak et al., 1996; Pinedo, 2002; Wang et al., 1997); except when the number of machines is smaller than or equal to two. Because of their intractability, this study presents the genetic algorithm (GA) to solve the RFS scheduling problems. GA has been widely used to solve classical flow-shop problems and has performed well. In addition, hybrid genetic algorithms (HGA) are proposed to enhance the performance of pure GA. The HGA is compared to the optimal solutions generated by the integer programming technique, and to the near optimal solutions generated by pure GA and the non-delay schedule generation procedure. Computational experiments are performed to illustrate the effectiveness and efficiency of the proposed HGA algorithm.

2. Literature review

Flow-shop scheduling problem is one of the most well known problems in the area of scheduling. It is a production planning problem in which n jobs have to be processed in the same sequence on m machines. Most of these problems concern the objective of minimizing makespan. The time between the beginning of the execution of the first job on the first

Source: Multiprocessor Scheduling: Theory and Applications, Book edited by Eugene Levner,
ISBN 978-3-902613-02-8, pp.436, December 2007, Itech Education and Publishing, Vienna, Austria

machine and the completion of the execution of the last job on the last machine is called makespan. To minimize the makespan is equivalent to maximize the utilization of the machines.

Johnson (1954) is the pioneer in the research of flow-shop problems. He proposed an "easy" algorithm to the two-machine flow-shop problem with makespan as the criterion. Since then, several researchers have focused on solving m-machine ($m > 2$) flow-shop problems with the same criterion. However, these fall in the class of NP-hard (Rinnooy Kan, 1976; Garey et al., 1976), complete enumeration techniques must be used to solve these problems. As the problem size increases, this approach is not computationally practical. For this reason, researchers have constantly focused on developing heuristics for the hard problem.

In today's competitive, global markets, effective production scheduling systems which manage the movement of material through production facilities provide firms with significant competitive advantages such as utilization of production capacity. These systems are particularly important in complex manufacturing environments such as semiconductor manufacturing where each wafer re-visits the same machines for multiple processing steps (Vargas-Villamil & Rivera, 2001). A wafer traverses flow lines several times to produce different layers on each circuit. This environment is one of the RFS scheduling problems.

In a RFS problem, these processes cannot be treated as a simple flow-shop problem. The repetitive use of the same machines by the same job means that there may be conflicts among jobs, at some machines, at different levels in the process. Later operations to be done on a particular job by some machine may interfere with earlier operations to be done at the same machine on a job that started later. This re-entrant or returning characteristic makes the process look more like a job-shop on first examination. Jobs arrive at a machine from several different sources or predecessor facilities and may go to several successor machines.

A number of researchers have studied the RFS scheduling problems. Graves et al. (1983) modeled a wafer fab as a RFS, where the objective is to minimize average throughput time subject to meeting a given production rate. Kubiak et al. (1996) examined the scheduling of re-entrant shops to minimize total completion time. Some researchers examined dispatching rules and order release policies for RFS. Hwang and Sun (1998) addressed a two-machine flow-shop problem with re-entrant work flows and sequence dependent setup times to minimize makespan. Demirkol and Uzsoy (2000) proposed a decomposition method to minimize maximum lateness for the RFS with sequence-dependent setup times.

Pan and Chen (2004) studied the RFS with the objective of minimizing the makespan and mean flow time of jobs by proposing optimization models based on integer programming technique and heuristic procedures based on active and non-delay schedules. In addition, they presented new priority rules to accommodate the reentry feature. Both the new rules and some selected rules of earlier research were incorporated in the schedule generation algorithm of active (ACT) and non-delay (NDY) schedules, and that of the priority rules in finding heuristic solutions for the problems. They compared ACT and NDY procedures and tested the combinations of 12 priority rules with ACT and NDY. Their simulation results showed that for RFS the best combinations were (NDY, SPT/TWKR) for minimizing makespan, where SPT means shortest processing time and TWKR means total work remaining.

3. Problem statement and optimization model

3.1 Problem description

Assumed that there are n jobs, J_1, J_2, …, J_n, and m machines, M_1, M_2, …, M_m, to be processed through a given machine sequence. Every job in a re-entrant shop must be processed on machines in the order of M_1, M_2, …, M_m, M_1, M_2, …, M_m, …, and M_1, M_2, …, M_m. In this case, every job can be decomposed into several levels such that each level starts on M_1 and finishes on M_m. Every job visits certain machines more than once. The processing of a job on a machine is called an operation and requires a duration called the processing time. The objective is to minimize the makespan. A minimum makespan usually implies a high utilization of the machine(s).

The assumptions made for the RFS scheduling problems are summarized here. Every job may visit certain machines more than once. Any two consecutive operations of a job must be processed on different machines. The processing times are independent of the sequence. There is no randomness; all the data are known and fixed. All jobs are ready for processing at time zero at which the machines are idle and immediately available for work. No pre-emption is allowed; i.e., once an operation is started, it must be completed before another one can be started on that machine. Machines never break down and are available throughout the scheduling period. The technological constraints are known in advance and immutable. There is only one of each type of machine. There is an unlimited waiting space for jobs waiting to be processed.

3.2 Optimization model

General symbol definition

J_i = job number i;

M_k = machine number k;

O^i_{lk} = the operation of J_i on M_k at layer l;

Problem parameters

m = number of machines in the shop;

n = number of jobs for processing at time zero;

M = a very large positive number;

L = number of layers for every job;

p^i_{lk} = the processing time of O^i_{lk};

Decision variables

$C_{\max}$ = maximum completion time or makespan;

s^i_{lk} = the starting time of O^i_{lk};

$Z^{ii'}_{ll'k}$ = 1 if O^i_{lk} precedes $O^{i'}_{l'k}$ (not necessarily immediately); 0 otherwise;

Pan and Chen (2004) were the first authors to present the integer programming model for solving the reentrant flow-shop problem. The model is as follows.

Minimize $$C_{\max} \tag{1}$$

Subject to $$s^i_{lk} + p^i_{lk} \le s^i_{l,k+1} \quad i = 1, 2,..., n;\ l = 1, 2,..., L;\ k = 1, 2, ..., m-1 \tag{2}$$

$$s^i_{lm} + p^i_{lm} \le s^i_{l+1,1} \quad i = 1, 2,..., n;\ l = 1, 2,..., L-1 \tag{3}$$

$$M(1 - Z^{ii'}_{ll'k}) + (s^{i'}_{l'k} - s^i_{lk}) \ge p^i_{lk} \quad 1 \le i < i' \le n;\ l, l' = 1, 2, ..., L;\ k = 1, 2, ..., m \tag{4}$$

$$M Z^{ii'}_{ll'k} + (s^{i}_{lk} - s^{i'}_{l'k}) \geq p^{i'}_{l'k} \quad 1 \leq i < i' \leq n;\ l, l' = 1, 2, ..., L;\ k = 1, 2, ..., m \tag{5}$$

$$s^{i}_{L,m} + p^{i}_{L,m} \leq C_{\max} \quad i = 1, 2, ..., n \tag{6}$$

$$C_{\max} \geq 0,\ s^{i}_{lk} \geq 0 \quad i = 1, 2, ..., n;\ l = 1, 2, ..., L;\ k = 1, 2, ..., m$$

$$Z^{ii'}_{ll'k} = 0 \text{ or } 1 \quad 1 \leq i < i' \leq n;\ l, l' = 1, 2, ..., L;\ k = 1, 2, ..., m \tag{7}$$

Constraint set (2) ensures that M_k begins to work on $O^{i}_{i+1,k}$ only after it finishes O^{i}_{lk}. Constraint set (3) ensures that the starting time of $O^{i}_{i+1,1}$ is no earlier than the finish time of O^{i}_{lm}. Constraint sets (2) and (3) together specify the technological constraints. Constraint sets (4) and (5) satisfy the requirement that only one job may be processed on a machine at any instant of time. Constraint set (6) defines $C_{\max}$ to be minimized in the objective function (1). The non-negativity and binary restrictions for s^{i}_{lk} and $Z^{ii'}_{ll'k}$, respectively, are described in (7).

4. A hybrid genetic algorithm for re-entrant flow-shop

4.1 Basic genetic algorithm structure

GA is one of the meta-heuristic searches. Holland (1975) first presented it in his book, *Adaptation in Natural and Artificial Systems*. It originates from Darwin's "survival of the fittest" concept, which means a good parent produce better offspring. GA searches a problem space with a population of chromosomes and selects chromosomes for a continued search based on their performance. Each chromosome is decoded to form a solution in the problem space in the context of optimization problems. Genetic operators are applied to high performance structures (parents) in order to generate potentially fitter new structures (offspring). Therefore, good performers propagate through the population from one generation to the next (Chang et al., 2005). Holland (1975) presented a basic GA called "Simple Genetic Algorithm" in his studies that is described as follows:

```
Simple genetic algorithm ()
  {
      Generate initial population randomly
      Calculate the fitness value of chromosomes
      While termination condition not satisfied
      {
          Process crossover and mutation at chromosomes
          Calculate the fitness value of chromosomes
          Select the offspring to next generation
      }
  }
```

A GA contains the following major ingredients: parameter setting, representation of a chromosome, initial population and population size, selection of parents, genetic operation, and a termination criterion.

4.2 Hybrid genetic algorithm

The role of local search in the context of the genetic algorithm has been receiving serious consideration and many successful applications are strongly in favor of such a hybrid

approach. Because of the complementary properties of GA and conventional heuristics, a hybrid approach often outperforms either method operation along. The hybridization can be done in a variety of ways (Cheng et al., 1999), including:

1. Incorporation of heuristics into initialization to generate well-adapted initial population. In this way, a hybrid genetic algorithm (HGA) with elitism can guarantee to do no worse than the conventional heuristic does.
2. Incorporation of heuristics into evaluation function to decode chromosomes to schedules.
3. Incorporation of local search heuristic as an add-on extra to the basic loop of GA, working together with mutation and crossover operations, to perform quick and localized optimization in order to improve offspring before returning it to be evaluated.

One of the most common HGA forms is incorporating local search techniques as an add-on to the main GA's recombination and selection loop. In the hybrid approach, GAs are used to perform global exploration in the population, while heuristic methods are used to perform local exploitation of chromosomes. HGA structure is illustrated in Fig. 1.

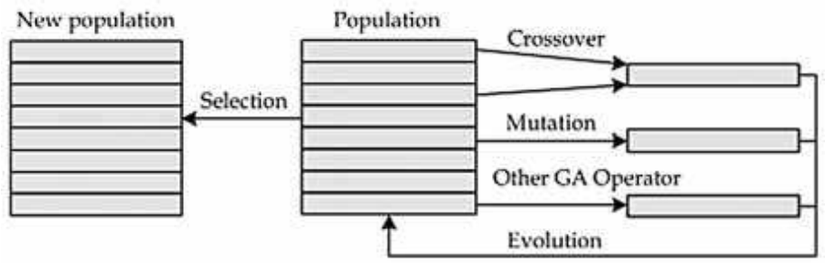


Figure 1. The hybrid genetic algorithm structure

4.3 The proposed hybrid genetic algorithms for re-entrant flow-shop

In this study, we propose an HGA for RFS with makespan as the criterion. The flowchart of the hybrid approach is illustrated in Fig. 2.

4.3.1 Parameters setting

The parameters in GA comprise population size, number of generations, crossover probability, mutation probability, and the probability of processing other GA operators.

4.3.2 Encoding

In GA, each solution is usually encoded as a bit string. That is, binary representation is usually used for the coding of each solution. However, this is not suitable for scheduling problems. During the past years, many encoding methods have been proposed for scheduling problem (Cheng et al., 1996). Among various kinds of encoding methods, job-based encoding, machine-based encoding and operation-based encoding methods are most often used for scheduling problem. This study adopts operation-based encoding method.

For example, we have a three-job, three-machine, two-level problem. Suppose a chromosome to be (1, 1, 2, 3, 1, 2, 3, 1, 3, 2, 1, 2, 3, 1, 2, 2, 3, 3), which means each job has six operations, it occurs exactly six times in the chromosome. If one of the alleles is generated more than six times or less than six times by GA operators such as crossover or mutation,

this chromosome is not a feasible solution of the RFS problem and it should be repaired to form a feasible one. Each gene uniquely indicates an operation and can be determined according to its order of occurrence in the sequence. Let O_{ijk} denote the jth operation of job i on machine k. The chromosome can be translated into a unique list of ordered operations of (O_{111}, O_{122}, O_{211}, O_{311}, O_{133}, O_{222}, O_{322}, O_{141}, O_{333}, O_{233}, O_{152}, O_{241}, O_{341}, O_{163}, O_{252}, O_{263}, O_{352}, O_{363}). Operation O_{111} has the highest priority and is scheduled first, then O_{122}, and so on. Hence there are $(n\times m\times l)!/[(m\times l)!]^n$ schedules for an n-job, m-machine, l-level RFS problems.

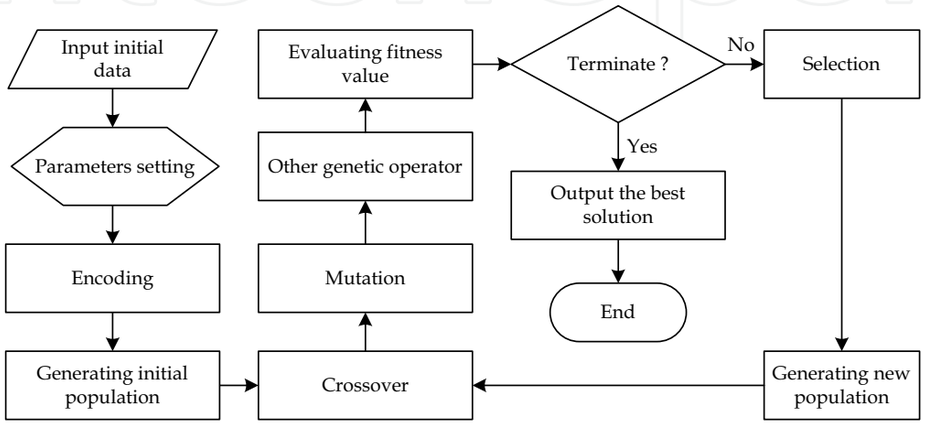


Figure 2. The flow chart of the proposed hybrid approach

4.3.3 Generation of initial population

The initial population sets are generated by two heuristic methods; one is (NDY, SPT/TWKR), the best heuristic for RFS problems proposed by Pan and Chen (2004). The other is NEH heuristic (Pan & Chen, 2003), the best heuristic for re-entrant permutation flow-shop (RPFS) problems. The RFS scheduling problem where no passing is allowed is called the RPFS (Pan & Chen, 2003).

The population is separated into two parts and each part contains a number of 1/2 population size of individuals. The first schedule of the first part was generated by (NDY, SPT/TWKR), the rest of the first part were generated by selecting two locations in the first schedule and swapping the operations in them. The first schedule of the second part was generated by NEH heuristic (Pan & Chen, 2003) and the remaining individuals of this part were produced by interchanging two randomly chosen positions of it. Because the NEH heuristic (Pan & Chen, 2003) is based on job number, it is needed to re-encode those individuals of the second part based on operations.

4.3.4 Crossover

Crossover is an operation to generate a new string (i.e., child) from two parent strings. It is the main operator of GA. During the past years, various crossover operators had been proposed (Murata et al., 1996). Murata et al. (1996) showed that the two-point crossover is effective for flow-shop problems. Hence the two-point crossover method is used in this study.

Two-point crossover is illustrated in Fig. 3. The set of jobs between two randomly selected points are always inherited from one parent to the child, and the other jobs are placed in the order of their appearance in the other parent.

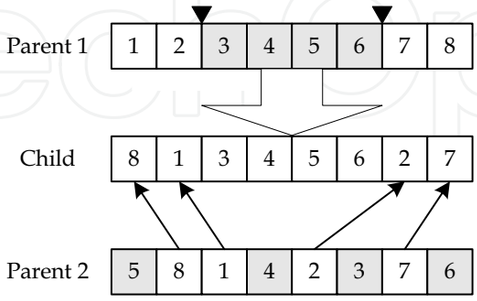


Figure 3. A two-point crossover

4.3.5 Mutation

Mutation is another usually used operator of GA. Such an operation can be viewed as a transition from a current solution to its neighborhood solution in a local search algorithm. It is used to prevent premature and fall into local optimum. In RFS, neighborhood search-based method is used to replace mutation as discussed next.

4.3.6 Other genetic operators

In traditional genetic approach, mutation is a basic operator just used to produce small variations on chromosomes in order to maintain the diversity of population. Tsujimura and Gen (1999) proposed a mutation inspired by neighbor search technique which is not a basic operator and is used to perform intensive search in order to find an improved offspring. Hence, we use neighborhood search-based method to replace mutation.

Parent

4	1	3	1	2	3	2	4	3	1	3	2	4	1	4	2

Neighbor chromosome

4	1	3	3	2	1	2	4	3	1	3	2	4	1	4	2
4	1	3	4	2	3	2	1	3	1	3	2	4	1	4	2
4	1	3	1	2	4	2	3	3	1	3	2	4	1	4	2
4	1	3	3	2	4	2	1	3	1	3	2	4	1	4	2
4	1	3	4	2	1	2	3	3	1	3	2	4	1	4	2
4	1	3	1	2	3	2	4	3	1	3	2	4	1	4	2

Figure 4. A local search mutation

For operation-based encoding, the neighborhood for a given chromosome can be considered as the set of chromosomes transformable from a given chromosome by exchanging the position of k genes (randomly selected and non-identical genes). A chromosome is said to be k-optimum, if it is better than any others in the neighborhood according to their fitness value. Consider the following example. Suppose genes on position 4, 6, and 8 are randomly selected. They are (1, 3, 4) and their possible permutations are (3, 1, 4), (4, 3, 1), (1, 4, 3), (3, 4, 1) and (4, 1, 3). The permutations of the genes together with remaining genes of the chromosome from the neighbor chromosomes are shown in Fig. 4. Then all neighbor chromosomes are evaluated and the chromosome with the best fitness value is used as the offspring.

4.3.7 Fitness function

Fitness value is used to determine the selection probability for each chromosome. In proportional selection procedure, the selection probability of a chromosome is proportional to its fitness value. Hence, fitter chromosomes have higher probabilities of being selected to next generation. To determine the fitness function, first calculate the makespan for all the chromosomes in a population, find the largest makespan over all chromosomes in current population and denote it as $V_{\max}$. The difference between each individual's makespan and $V_{\max}$ to the 1.005 power is the fitness value of that particular individual. The power law scaling (α) was proposed by Gillies (1985), which powers the raw fitness to a specific value. In general, the value is problem-dependent. Gillies (1985) reported a value of 1.005. The fitness function denote by $F_i = (V_{\max} - V_i)^{\alpha}$. This is done to ensure that the probability of selection for a schedule with lower makespan is high.

4.3.8 Termination

GA continues to process the above procedure until achieving the stop criterion set by user. The commonly used criterions are: (1) The number of executed generation; (2) A particular object; and (3) The homogeneity of population. This study uses a fixed number of generations to serve as the termination condition.

4.3.9 Selection

Selection is another important factor to consider in implementing GA. It is a procedure to select offspring from parents to the next generation. According to the general definition, the selection probability of a chromosome should show the performance measure of the chromosome in the population. Hence a parent with a higher performance has higher probabilities of being selected to next generation. In this study, the process for selecting parents is implementing via the common roulette wheel selection procedure presented by Goldberg (1989). The procedure is described below.

Step 1: Calculate the total fitness value for each chromosome in the population.

Step 2: Calculate the selection probability of each chromosome. This is equal to the chromosome's fitness value divided by the sum of each chromosome's fitness value in the population.

Step 3: Calculate the cumulative probability of each chromosome.

Step 4: Generate a probability $\boldsymbol{P}$ randomly where $\boldsymbol{P}$~[0, total cumulative probability], if $P(n) \le \boldsymbol{P} \le P(n+1)$, after that select the $(n+1)$ chromosome of population to next generation, where $P(n)$ is the cumulative probability of the nth chromosome.

In this way, the fitter chromosomes have a higher number of offspring in the next generation. However, this method is not guaranteed that every good chromosome can be selected to the offspring to next generation. Hence one chromosome is randomly selected to be replaced by the best chromosome found until now.

5. Analysis of experiment results and conclusions

5.1 Experiment design

We describe types of problems, comparison of exact and heuristic algorithms, experimental environment, and facility in this section.

5.1.1 Types of problems

The instance size is denoted by $n \times m \times L$, where n is the number of jobs, m is the number of machines, and L represents the number of levels. The test instances are classified into three categories: small, medium, and large problems. Small problems include 3×3×3, 3×3×4, 3×4×2, 4×3×3, 4×4×3, 4×5×3, 4×4×4, and 4×5×4. Medium problems include 6×6×2, 6×8×5, 6×9×3, 7×7×5, 7×8×4, 8×8×3, 9×9×2, and 10×10×2. Large problems include 12×12×10, 15×15×5, 20×20×4, 25×25×8, and 30×30×5. The processing time of each operation for each type of problem is a random integer number generated from [1, 100], since the processing times of most library benchmark problems are generated in this range (Beasly 1990).

5.1.2 Performance of exact and heuristic algorithms

For small problems, the performances of HGA are compared with optimal solution, NEH, and (NDY, SPT/TWKR). For medium and large problems, the performances of HGA are compared with that of (NDY, SPT/TWKR), and non-hybrid version of GA, i.e., pure GA.

5.1.3 Experimental environment and facility

Hybrid GA, pure GA, NEH, and (NDY, SPT/TWKR) are implemented in Visual C++ while optimal solutions are solved by ILOG CPLEX. These programs are executed on a PC with Pentium IV 1.7GHz.

5.2 Analysis of RFS experiment results

The analysis of RFS experiment results are described in this section. The test instances are classified into three categories: small, medium, and large problems.

5.2.1 Small problems

The HGA parameters setting are as follows: the population size is 50, the crossover probability is 0.8, the mutation probability is 0.1, the hybrid operator probability is 0.5, and the maximum number of generations allowed is 100.

For small size problems, there are 8 types of problems with 10 instances in each type, i.e., 80 instances are tested. The optimal solution is obtained by using integer programming technique (Pan & Chen, 2004). Because GA is a stochastic searching heuristic, the result of every test instance is unlikely to be the same. In order to compare the average performance, 10 instances were solved in each test and the average makespan (denoted by Avg. C_{max}) and the minimum of these makespans (denoted by Min. C_{max}) are recorded.

The decoding scheme in this study is based on NDY schedule generation method, i.e., the schedules are always non-delay. Though sometimes the HGA cannot find optimal solutions because optimal solutions are not necessarily non-delay, Pan and Chen (2004) reported that for RFS problems, the solution quality of non-delay schedules is obviously superior to that of the active schedules; therefore, the makespan is calculated by non-delay schedule in this study.
The experimental results for small size problems of integer programming (IP), HGA, NEH and (NDY, SPT/TWKR) are listed in Table 1. The deviation is defined as follows.

$$\text{Deviation} = \frac{C_{\max}(\text{H}) - C_{\max}(\text{IP})}{C_{\max}(\text{IP})} \times 100\%$$

where C_{max}(H) denotes the makespan obtained by heuristic H. Heuristic H includes pure GA, HGA, NEH, and (NDY, SPT/TWKR). C_{max}(IP) denotes the optimal makespan and that is obtained by using integer programming technique (Pan & Chen, 2004).
The improvement rate of method A over method B is defined as follows.

$$\text{Improvement rate} = \frac{C_{\max}(\text{H\_B}) - C_{\max}(\text{H\_A})}{C_{\max}(\text{H\_B})} \times 100\%$$

where C_{max}(H_A) and C_{max}(H_B) denote the makespan obtained by heuristics H_A and H_B, respectively.
The experimental results of IP, HGA, NEH and (NDY, SPT/TWKR) for small size problems are listed in Table 1. From Table 1, HGA performs quite well. The objective function values it obtained are about 0.3% above the optimal values. While compared to NEH and (NDY, SPT/TWKR), HGA performs better than both of them by having improvement rate of 2.68% and 5.28%, respectively. The number of times that HGA finds optimal solutions is obviously more than those of NEH and (NDY, SPT/TWKR). This result is similar to that of small size problems, and it is found that the range of processing time does not affect the solution quality of the proposed GA.

5.2.2 Medium problems

The parameters are the same as those in small problems, except that generation is 200. There are 8 types of problems with 10 instances in each type. The performances are compared with (NDY, SPT/TWKR).
Table 2 shows the comparison results of pure GA, HGA, and (NDY, SPT/TWKR). The column (C_{max}(HGA) < C_{max}(GA)) is the number of times that the Min. C_{max} of HGA is better than that of pure GA in each instances. In medium size problems, the improvement rate of HGA over (NDY, SPT/TWKR) is nearly 6.93%. Table 2 also shows that although the improvement rate does not enhance obviously, the solution of HGA are consistent better than that of pure GA.

5.2.3 Large problems

The parameters are the same as those in small problems, except that generation is 400. There are 5 types of problems with 10 instances in each type. Table 3 reports the performances of pure GA, HGA, and (NDY, SPT/TWKR) in large problems.

The experimental results show that even when dealing with large size problems, HGA still has good performance. The average improvement rate of HGA over (NDY, SPT/TWKR) is 5.25% and average improvement of HGA over pure GA is 1.36%.

Problems*	Number of optimal solution found			CPU time(s)		The improvement rate of HGA over		Avg. deviation of HGA
	HGA	NEH	(NDY, SPT/TWKR)	IP	HGA	NEH	(NDY, SPT/TWKR)	
3×3×3	10	6	2	0.31	7.05	1.32%	3.69%	0.06%
3×3×4	10	3	2	0.80	6.73	2.50%	4.04%	0.00%
3×4×2	10	5	4	0.09	4.86	1.10%	4.22%	0.00%
4×3×3	6	0	0	7.38	5.33	4.46%	5.34%	0.42%
4×4×3	7	0	0	6.65	4.04	2.13%	4.50%	0.59%
4×5×3	8	1	0	6.75	16.25	2.66%	5.50%	0.29%
4×4×4	5	0	0	209.44	12.29	4.41%	9.02%	0.50%
4×5×4	8	0	0	32.76	17.85	2.87%	5.95%	0.28%

*Specified by n jobs × m machines × L levels.

Table 1. Comparison of all small problems

Problems*	CPU time(s)			HGA versus GA		HGA versus NDY(SPT/TWKR)	
	GA	HGA	(NDY, SPT/TWKR)	The improvement rate of HGA over GA	C_{max}(HGA) < C_{max}(GA)	The improvement rate of HGA over (NDY, SPT/TWKR)	C_{max}(HGA) < C_{max}(NDY, SPT/TWKR)
6×6×2	5.56	23.88	<0.1	1.82%	10	6.42%	10
6×8×5	8.04	23.88	<0.1	2.31%	10	7.27%	10
6×9×3	8.43	19.37	<0.1	1.74%	10	8.86%	10
7×7×5	13.13	26.55	<0.1	2.73%	10	6.87%	10
7×8×4	9.83	26.73	<0.1	1.76%	9	5.54%	10
8×8×3	5.27	32.40	<0.1	1.43%	9	4.22%	10
9×9×2	5.02	31.89	<0.1	1.29%	10	7.24%	10
10×10×2	5.90	38.28	<0.1	1.44%	10	8.97%	10

*Specified by n jobs × m machines × L levels.

Table 2. Comparison of all medium problems

Problems*	CPU time(s)			HGA versus GA		HGA versus NDY(SPT/TWKR)	
	GA	HGA	(NDY, SPT/TWKR)	The improvement rate of HGA over GA	C_{max}(HGA) < C_{max}(GA)	The improvement rate of HGA over (NDY, SPT/TWKR)	C_{max}(HGA) < C_{max}(NDY, SPT/TWKR)
12×12×10	121.61	368.28	0.12	1.38%	10	4.83%	10
15×15×5	107.77	366.26	0.13	1.39%	10	4.76%	10
20×20×4	161.29	695.76	0.13	1.27%	10	5.56%	10
25×25×8	241.36	965.44	0.17	1.44%	10	5.73%	10
30×30×5	188.70	634.80	0.15	1.31%	10	5.37%	10

*Specified by *n* jobs × *m* machines × *L* levels.

Table 3. Comparison of all large problems

6. Conclusions and suggestions

This study developed a hybrid genetic algorithm (HGA) for the RFS problems with makespan as the criterion. The computational experiments have shown that the HGA can favorably improve the results obtained by (NDY, SPT/TWKR) and NEH in RFS problems. GA is inspired by nature phenomena. If it mimics exactly the way nature works, an unexpected long computational time must take. Hence the effect of parameters must be studied thoroughly in order to obtain good solution in a reasonable time. The probability to obtain near-optimal solution increases in the cost of longer computational time when the number of generations or population size enlarges. When dealing with large size problems or large re-entrant times, the probability to obtain near optimal solution increases by setting larger population size or generations. In conclusion, GA provides a variety of options and parameter settings which still have to be fully investigated. This study has demonstrated the potential for solving RFS problems by means of a GA, and it clearly suggests that such procedures are well worth exploring in the context of solving large and difficult combinatorial problems.

The most challenging problem in the test of RFS is to prevent early convergence of the genetic algorithm. The convergence speeds up when the number of operations enlarges. In future study, a thorough investigation may be done on this issue. The parameter setting in GA affects computational efficiency and quality of solution greatly. Not only job numbers and machine numbers have impacts on parameter setting, but also the number of levels contributed a lot. It is an important future study issue to determine the best parameter setting for GA in different environment. In future study, the GA can be combined with other heuristics or algorithms to obtain the more efficiency and the better quality solution.

7. References

Baker, K. R., (1974). *Introduction to sequencing and scheduling*, John Wiley & Sons, ISBN: 0-471-04555-1, New York.

Beasly, J. E. (1990). OR-library: distribution test problems by electronic mail. *Journal of the Operational Research Society*, Vol. 41, No. 11, 1069-1072, ISSN: 0160-5682.

Bispo, C. F. & Tayur, S. (2001). Managing simple re-entrant flow lines: theoretical foundation and experimental results. *IIE Transactions*, Vol. 33, No. 8, 609-623, ISSN: 0740-817X.

Chang, P. C.; Chen, S. H. & Lin, K. L. (2005). Two-phase sub population genetic algorithm for parallel machine-scheduling problem. *Expert Systems with Applications*, Vol. 29, 705-712, ISSN: 0957-4174.

Cheng, R.; Gen, M. & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms – I. Representation. *Computers & Industrial Engineering*, Vol. 30, No. 4, 983-997, ISSN: 0360-8352.

Cheng, R.; Gen, M. & Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, Part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, Vol. 36, No. 2, 343-363, ISSN: 0360-8352.

Demirkol, E. & Uzsoy, R. (2000). Decomposition methods for re-entrant flow shops with sequence-dependent setup times. *Journal of Scheduling*, Vol. 3, No. 3, 155-177, ISSN: 1094-6136.

Garey, M. R.; Johnson, D. S. & Sethi, R. (1976). The complexity of flow-shop and job-shop scheduling. *Mathematics of Operations Research*, Vol. 1, No. 2, 117-129, ISSN: 0364-765X.

Gillies, A. (1985). *Machine learning procedures for generating image domain feature detectors,* University of Michigan Press, Ann Arbor.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine Learning*, Addison-Wesley, ISBN: 0-201-15767-5, Reading, MA.

Graves, S. C.; Meal, H. C.; Stefek, D. & Zeghmi, A. H. (1983). Scheduling of re-entrant flow shops. *Journal of Operations Management*, Vol. 3, No. 4, 197-207, ISSN: 0272-6963.

Holland, J. (1975). *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor.

Hwang, H. and Sun, J. U. (1998). Production sequencing problem with re-entrant work flows and sequence dependent setup times. *International Journal of Production Research,* Vol. 36, No. 9, 2435-2450, ISSN: 0020-7543.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with set up times included, *Naval Research Logistics Quarterly*, Vol. 1, No. 1, 61-68.

Kubiak, W.; Lou, S. X. C. & Wang, Y. (1996). Mean flow time minimization in re-entrant job-shops with a hub. *Operations Research*, Vol. 44, No. 5, 764-776, ISSN: 0030-364X.

Murata, T.; Ishibuchi, H. & Tanaka, H. (1996). Genetic algorithms for flows shop scheduling problems. *Computers & Industrial Engineering*, Vol. 30, No. 4, 1061-1071, ISSN: 0360-8352.

Pan, J. C. H. & Chen, J. S. (2003). Minimizing makespan in re-entrant permutation flow-shops. *Journal of the Operational Research Society*, Vol. 54, No. 6, 642-653, ISSN: 0160-5682.

Pan, J. C. H. & Chen, J. S. (2004). A comparative study of schedule-generation procedures for the re-entrant shops. *International Journal of Industrial Engineering– Theory, Applications and Practice*, Vol. 11, No. 4, 313-321, ISSN: 1072-4761.

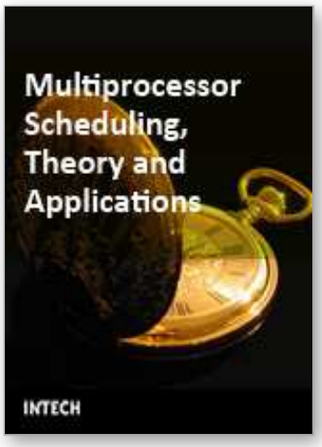
Pinedo, M. (2002). *Scheduling: theory, algorithms, and systems*, Prentice-Hall, New Jersey.

Rinnooy Kan, A. H. G. (1976). *Machine scheduling problems: classification, complexity and computations*, Martinus Nijhoff, ISBN: 90.247.1848.1, The Hague, Holland.

Tsujimura, Y. & Gen, M. (1999). Parts loading scheduling in a flexible forging machine using an advanced genetic algorithm. *Journal of Intelligent Manufacturing*, Vol. 10, No. 2, 149-159, ISSN: 0956-5515.

Vargas-Villamil, F. D. & Rivera, D. E. (2001). A model predictive control approach for real-time optimization of re-entrant manufacturing lines. *Computers in Industry*, Vol. 45, No. 1, 45-57, ISSN: 0166-3615.

Wang, M. Y.; Sethi, S. P. & Van De Velde, S. L. (1997). Minimizing makespan in a class of re-entrant shops. *Operations Research*, Vol. 45, No. 5, 702-712, ISSN: 0030-364X.



Multiprocessor Scheduling, Theory and Applications

Edited by Eugene Levner

ISBN 978-3-902613-02-8

Hard cover, 436 pages

Publisher I-Tech Education and Publishing

Published online 01, December, 2007

Published in print edition December, 2007

A major goal of the book is to continue a good tradition - to bring together reputable researchers from different countries in order to provide a comprehensive coverage of advanced and modern topics in scheduling not yet reflected by other books. The virtual consortium of the authors has been created by using electronic exchanges; it comprises 50 authors from 18 different countries who have submitted 23 contributions to this collective product. In this sense, the volume can be added to a bookshelf with similar collective publications in scheduling, started by Coffman (1976) and successfully continued by Chretienne et al. (1995), Gutin and Punnen (2002), and Leung (2004). This volume contains four major parts that cover the following directions: the state of the art in theory and algorithms for classical and non-standard scheduling problems; new exact optimization algorithms, approximation algorithms with performance guarantees, heuristics and metaheuristics; novel models and approaches to scheduling; and, last but least, several real-life applications and case studies.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen