International Journal of Industrial Engineering, 16(1), 23-31, 2009.

Solving the Reentrant Permutation Flow-Shop Scheduling Problem with a Hybrid Genetic Algorithm

Jen-Shiang Chen[1], Jason Chao-Hsien Pan[2], and Chien-Min Lin[3]

[1]Department of Business Administration
Far East University
Tainan, Taiwan (744), R.O.C.

[2]Department of Business Administration
Takming University of Science and Technology
Taipei, Taiwan (114), R.O.C.

[3]Department of Industrial Management
National Taiwan University of Science and Technology
Taipei, Taiwan (106), R.O.C.

Corresponding author's e-mail: {Jen-Shiang Chen, jschenc@ms25.hinet.net}

Most production scheduling-related research assumes that a job visits certain machines once at most, but this is often untrue in practical situations. A reentrant permutation flow-shop (RPFS) describes situations in which a job must be processed on machines in M_1, M_2, …, M_m, M_1, M_2, …, M_m, …, and M_1, M_2, …, M_m order and no job is allowed to pass a previous job. This study minimizes makespan by using the genetic algorithm to move from local optimal solutions to near-optimal solutions for RPFS scheduling problems. In addition, the hybrid genetic algorithm (HGA) improves the genetic algorithm's performance in solving RPFS.

Significance: This paper studies the reentrant permutation flow-shop scheduling problem with the objectives of minimizing the makespan of jobs. The genetic algorithm and hybrid genetic algorithm are presented for the proposed problem.

Keywords: Scheduling, genetic algorithm, hybrid genetic algorithm, reentrant permutation flow-shop.

(Received 15 June 2007; Accepted in revised form 30 December 2008)

1. INTRODUCTION

In production management, a scheduling problem is defined as some specific work time hypothesis regarding assignment of resources, including equipment and human resources (labor), in order to complete work in a certain amount of time. In the complex and dynamic world of manufacturing systems, scheduling is an extremely important issue. Scheduling deals with the allocation of scarce resources to tasks over time. In different machine environments, these problems can be categorized into single machine, parallel machines, flow-shop, job-shop and open-shop.

In manufacturing and assembly facilities, many operations must be completed for every job. Often, these operations must be done on all jobs in the same order, implying that the jobs must follow the same route. One assumes these machines are set up in series, and this environment is referred to as a flow-shop. Classical flow-shop scheduling problems assume that each job visits each machine only once (Baker 1974), but this assumption is sometimes violated in practice. A new type of manufacturing shop, the reentrant flow-shop, has recently attracted attention. The basic characteristic of a reentrant shop is that a job visits certain machines more than once. In a reentrant flow-shop (RFS), all jobs have the same route through the shop machines and the same sequence is executed several times (levels) to complete the jobs. For example, in semiconductor manufacturing, each wafer re-visits the same machines for multiple processing steps (Vargas-Villamil and Rivera 2001). The wafer traverses flow lines several times to produce a different layer in each circuit (Bispo and Tayur 2001).

The reentrant permutation flow-shop problem (RPFS) is a special case of the RFS problem. In a RFS, if job ordering is the same on any machine at each level, there is no passing since no job is allowed to pass a previous job (Pan and Chen 2003). Finding an optimal schedule that minimizes the makespan in RPFS is never easy. In fact, flow-shop scheduling, the sequencing problem in which n jobs must 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; the makespan can be minimized by Johnson's (1954) rule.

Because of their intractability, this study presents genetic algorithms (GA) to solve the RPFS 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 improve the GA performance and the heuristic methods proposed by Pan and Chen (2004) for solving RPFS.

2. LITERATURE REVIEW

The permutation flow-shop problem (PFS) is a special case of the flow-shop problem. A possible constraint in the flow-shop environment is that the queues for each machine operate according to the first-in-first-out discipline. This implies that the order (or permutation) in which the jobs go through the first machine is the same throughout the system. The most important property of PFS is deciding the job sequence on the first machine because once it is decided, all the jobs follow the same sequence on each machine throughout the system. Therefore, for a PFS with *n* jobs, there are *n*! solutions that are independent of machine numbers. The problem of *n*-jobs on *m*-sequential machines in a PFS with the criteria of minimizing makespan is proven to be NP-hard (Rinnooy Kan 1976) and can be solved exactly only for small size problems. Because of this intractability, many authors proposed various techniques to solve this problem. Palmer (1965), Campbell, Dudek and Smith (1970), Gupta (1971), Dannenbring (1977), Nawaz, Enscore and Ham (1983) have proposed existing heuristics for the *m*-machine *n*-job PFS problem with makespan as the criterion.

Palmer's (1965) heuristic first calculated a slope order for each job, and then sequenced the jobs according to the slope orders. This gives priority to the jobs with the strongest tendency to progress from short times to long times in the sequence of operations. Gupta (1971) presented a similar slope order index sequencing method, except he computed the slope differently. Campbell, Dudek and Smith (CDS) (1970) presented a heuristic that is an extension to Johnson's (1954) rule. This algorithm first generated a set of $(m-1)$ two-machine problems by aggregating the *m* machines into two groups systematically. Then it applied Johnson's (1954) two-machine algorithms to find the $(m-1)$ schedules and finally selected the best one. The best of these schedules was the solution to the original problem. Dannenbring's (1977) heuristic attempted to combine the advantages of the heuristics presented by Palmer and CDS. His method was called the rapid access (RA) procedure and its purpose was to provide a quick and successful solution by constructing an artificial two-machine problem in which the processing times were determined from a weighting scheme and then solved with Johnson's (1954) rule. The Nawaz, Enscore and Ham (NEH) (1983) heuristic algorithm was based on the assumption that a job with a high total processing time on all the machines should be given higher priority than a job with a low total processing time. The NEH algorithm did not transform the original *m*-machine problem into one artificial two-machine problem, but instead builds the final sequence in a constructive way, adding a new job at each step and finding the best partial solution.

The PFS scheduling problem can be modified to suit the RPFS scheduling problem by relaxing the assumption that each job visits each machine only once. This study considers the RPFS scheduling problems with the objective of minimizing makespan of jobs. The attachment of surface-mounting devices and the insertion of pin-through-hole devices of PCBs is a typical RPFS under first-come-first-served policy. It is popular to minimize makespans in industrial setting because this allows the machine to either increase its production capacity or reduce work in process. Minimizing makesapn in the RPFS is theoretically challenging, and such problems are NP-hard in the strong sense, even in the two-machine case (Kubiak *et al*. 1996, Pinedo 2002, Wang *et al*. 1997).

Pan and Chen (2003) presented the mixed binary integer programming technique for RPFS by extending the models of Wagner (1959), Wilson (1989), and Manne (1960), respectively. In addition, they proposed six extended heuristic algorithms to find sub-optimal solutions for RPFS. The experimental results showed that for a set of problems with known optimal solutions (small problems), the CDS-based heuristic was the best, followed by the NEH-based heuristic. The NEH-based heuristic outperformed the other heuristics in a set of problems with unknown optimal solutions (large problems), followed by the CDS-based heuristic.

3. PROBLEM STATEMENT AND HYBRID GENETIC ALGORITHM

3.1 Problem Description

The reentrant permutation flow-shop (RPFS) environment is described in this section. Assume that there are *n* jobs, J_1, J_2, …, and J_n, and *m* machines, M_1, M_2, …, and M_m, to be processed through a given machine sequence. Every job in a reentrant 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 so 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 its duration is called the processing time. The objective is to minimize the makespan.

The assumptions made for the RPFS scheduling problems are summarized as follows:

(1) The processing times are independent of the sequence.
(2) There is no randomness; all the data are known and fixed.
(3) All jobs are ready for processing at time zero, at which time the machines are idle and immediately available for work.
(4) No preemption is allowed; for example, once an operation is started, it must be completed before another one can be started on that machine.
(5) Machines never break down and are available throughout the scheduling period. The technological constraints are known in advance and immutable.
(6) There is only one of each type of machine.
(7) There is an unlimited waiting space for jobs waiting to be processed.
(8) Job ordering is the same on any machine at each level.

3.2 Basic Genetic Algorithm Structure

GA is a meta-heuristic search. John 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 produces 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 generations (Bowden 1992). Holland (1975) presented a basic GA called the "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 for the 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. In Holland's (1975) original GA, parents are replaced by their offspring soon after they give birth. This is called generational replacement. Since genetic operations are blind in nature, offspring may be worse than their parents. By replacing each parent with his offspring directly, some fitter chromosomes will be lost in the evolutionary process. To overcome this problem, several replacement strategies have been proposed. Holland (1975) suggested that when each offspring was born, it replaced a randomly chosen chromosome from the current population. This was called a reproductive plan. Since Grefenstette and Baker's (1989) work, selection is used to form the next generation, usually with a probabilistic mechanism. Michalewicz (1994) gave a detailed description of simple GAs where offspring in each generation replaced their parents soon after they were born and the next generation was formed by roulette wheel selection.

3.3 Hybrid Genetic Algorithm

Local search in the context of GA has been receiving serious consideration and many successful applications have strongly favored such a hybrid approach. As a result of the complementary properties of GAs and conventional heuristics, a hybrid approach often outperforms either method along. The hybridization can be done in a variety of ways (Cheng *et al*. 1999), including:

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

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 exploration of chromosomes. HGA structure is illustrated in Figure 1.

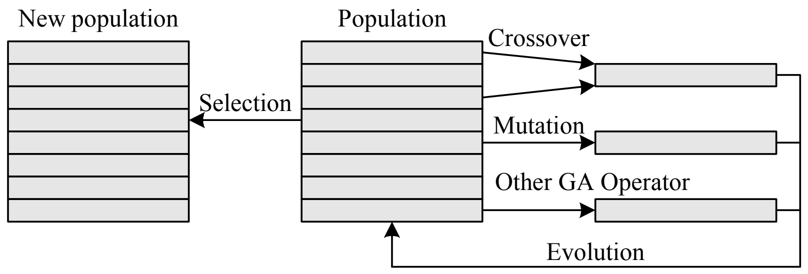

Figure 1. The HGA structure

3.4 The Proposed Hybrid Genetic Algorithms for Reentrant Permutation Flow-Shop

In this study, we propose an HGA for RPFS with makespan as the criterion. The hybrid approach procedure is illustrated as follows.

Step 1. Inputting job and machine data
Step 2. Parameter setting
Step 3. Encoding
Step 4. Generating initial population
Step 5. Crossover
Step 6. Mutation
Step 7. Other genetic operators
Step 8. Evaluating fitness value
Step 9. If termination conditions are satisfied, output the best solution and stop.
Step 10. Selection
Step 11. Generating new population
Step 12. Go to Step 5

3.4.1 Parameters Setting

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

3.4.2 Encoding

In GA, each solution is usually encoded as a bit string. In other words, binary representation is usually used for the coding of each solution. Unfortunately, this method is not suitable for scheduling problems and over the years, many encoding methods have been proposed (Cheng *et al*. 1996). Among the 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 the job-based encoding method.

For example, we have a three-job, three-machine, two-level problem (3×3×2). A chromosome can be easily described as a sequence of jobs. Therefore, there are $n!$ schedules. Suppose a chromosome is given as (1, 2, 3), where "Job 1" is processed first, "Job 2" is processed next, and so on. If the allele appears twice in the chromosome, the chromosome is not a feasible solution, so it should be fixed to form a feasible solution.

3.4.3 Generation of Initial Population

We generate initial population randomly since we found that if the chromosomes of a population were generated by heuristics, it was likely to fall into local optimum. The performance of randomly generated initial populations was better than that of heuristic-based populations.

3.4.4 Crossover

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

Two-point crossover. This crossover is illustrated in Figure 2. 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 order of their appearance in the other parent.

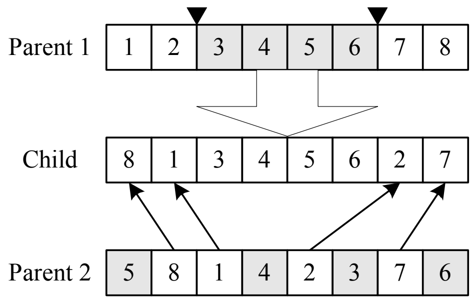

Figure 2. A two-point crossover

3.4.5 Mutation

Mutation is another common operator of GA. It 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 local optimum and fall into local optimum. The following four mutation operators for flow-shop scheduling problems are commonly used. Murata *et al*. (1996) showed that shift change mutation is effective for flow-shop problems. Therefore, the shift change mutation method is adopted for RPFS in this research since permutation flow-shop is a special case of RPFS.

Shift change. In this mutation operation, a job at one position is removed and put at another position as shown in Figure 3. The two positions are randomly selected. This mutation includes the adjacent two-job changes as a special case and has an intersection with the arbitrary three-job change.

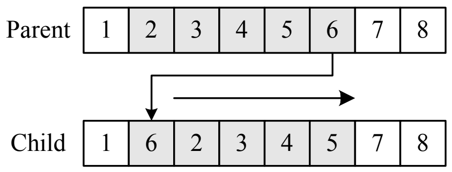

Figure 3. A shift change

3.4.6 Other Genetic Operators

The hybrid genetic operator proposed in RPFS works is similar to the decomposition method, which randomly cuts a sub-string from one chromosome and treats it as a sub-problem. This sub-problem is then solved by a NEH heuristic-based method presented by Pan and Chen (2004). This method sorts the jobs in order of decreasing total processing time requirements and builds the final sequence by adding a new job at each step and finding the best partial solution. The algorithmic steps are as follows:

Step 1. Calculate the sum of the processing times for each job. Sequence the jobs in a non-increasing order according to their total processing times on all the machines.

Step 2. Take the first two jobs (those with the largest total processing requirements) and schedule them in order to minimize the makespan, as if there were only two jobs.

Step 3. For i = 3 to r (the number of jobs in the cut string), insert the i-th job at the position that minimizes the partial makespan among the i possible positions.

Then, the new sequence is placed back into the chromosome to obtain a new chromosome. The procedure is shown in Figure 4.

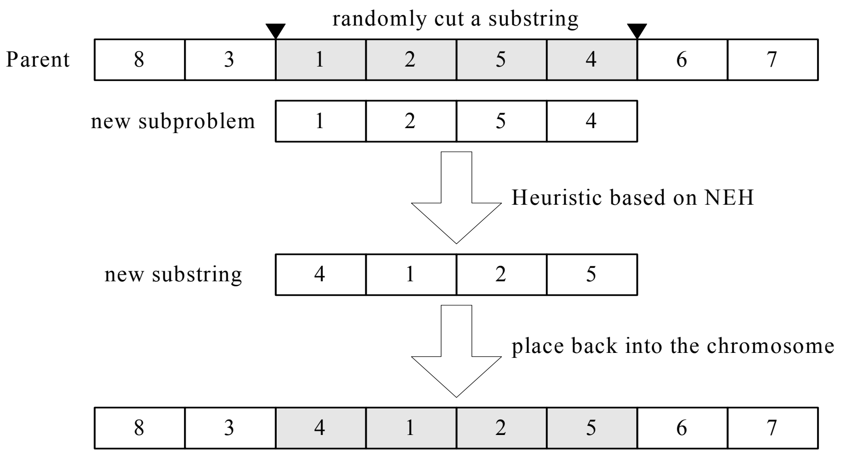

Figure 4. The hybrid operator in RPFS

3.4.7 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. Therefore, fitter chromosomes have higher probabilities of being selected for the next generation.

To determine the fitness function for the RPFS problems, first calculate the makespan for all the chromosomes in a population. Find the largest makespan of all chromosomes in the current population and label it $V_{\max}$. The difference between each individual chromosome's makespan and $V_{\max}$ to the 1.005th power is the fitness value of that particular chromosome. Gillies (1985) proposed power law scaling (α), which powers the raw fitness to a specific value. Generally, the value is problem-dependent. Gillies (1985) reported an α value of 1.005. The fitness function is as follows:

$F_i = (V_{\max} - V_i)^{\alpha}$

This is done to ensure that there is a high selection probability of a schedule with a lower makespan.

3.4.8 Termination

GA continues the above procedure until achieving the stop criterion set by the user. Commonly used criteria are:

(1) the number of executed generations,
(2) a particular object, and
(3) population homogeneity.

In this study, we use a fixed number of generations as our termination condition.

3.4.9 Selection

Selection is another important factor to consider in implementing GA. This procedure selects parents' offspring for 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. This means that a parent with high performance has a higher probability of being selected for the next generation.

In this study, the parent selection process is implemented via Goldberg's (1989) common roulette wheel selection procedure. The procedure is described below.

(1) Calculate the total fitness value for each chromosome in the population.
(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.
(3) Calculate the cumulative probability of each chromosome.
(4) Randomly generate a probability P where P~[0, total cumulative probability], if $P(n) \le P \le P(n+1)$. Then select the $(n+1)$ chromosome of the population for the next generation. $P(n)$ is the cumulative probability of the n-th chromosome.

In this way, the fitter chromosomes have a higher number of offspring in the next generation. This method, however, does not guarantee that every good chromosome will be selected for offspring in the next generation. Therefore, one chromosome is randomly selected to be replaced by the best chromosome found at that point.

4. ANALYSIS OF EXPERIMENT RESULTS

4.1 Experiment Design

In this section, we discuss types of problems, compare exact and heuristic algorithms, and describe the experimental environment and facility.

4.1.1 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, 4×4×4, 5×4×3, 5×5×4, 6×8×5, 7×8×4, 8×8×4, 9×7×4, 9×9×3, and 10×6×3. Medium problems include 11×17×5, 12×20×6, 13×19×7, 14×18×9, 15×17×6, 16×16×7, 17×5×8, 18×16×6, 19×12×10, and 20×15×3. Large problems include 20×20×10, 30×30×5, 40×40×5, 50×50×6, and 80×80×3. 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).

4.1.2 Performance of Exact and Heuristic Algorithms

For small problems, HGA performance is compared to optimal solution, NEH, and CDS. For medium and large problems, HGA performance is compared to pure GA, NEH, and CDS.

4.1.3 Experimental Environment and Facility

Pure GA, HGA, NEH, and CDS were implemented in Visual C++, while ILOG CPLEX solved optimal solutions. These programs were executed on a PC with Pentium IV 1.7GHz processor.

4.2 Analysis of RPFS Experiment Results

4.2.1 Small Problems

The HGA parameters setting were as follows: the population size was 50, the crossover probability was 0.8, the mutation probability was 0.3, the hybrid operator probability was 0.1, and the number of generations was 100. For small size problems, there were 10 types of problems with 10 instances in each type; 100 instances were tested.

The comparison results of HGA, NEH, and CDS in small problems are shown in Table 1. The experimental results show that HGA performance was very promising because all of the minimizing $C_{\max}$ in 100 instances reached optimal solutions, while NEH reached optimal solutions 30 times, and CDS reached optimal solutions 19 times. Although the

NEH and CDS heuristics were very efficient, it was difficult for them to reach optimal solution when job numbers were larger than 8. We found that the performance of NEH was better than that of CDS, and HGA was the best among these three methods. The experimental results for small size problems of integer programming, HGA are listed in following table. Table 1 shows that all of the average $C_{\max}$ from 10 types problems are very close to the mean value of optimal solutions (about 0.09% above optimal).

Table 1. Comparison Results in Small Problems

Types	No. of opt. found			CPU time(s)		The improvement rate of HGA over		Avg. deviation of HGA
	HGA	NEH	CDS	IP	HGA	NEH	CDS	
3x3x3	10	8	5	0.03	0.07	0.23%	0.86%	0.00%
4x4x4	10	8	5	0.11	0.11	0.61%	0.70%	0.00%
5x4x3	10	5	4	0.12	0.11	2.14%	1.60%	0.00%
5x5x4	10	2	2	0.34	0.14	1.96%	2.01%	0.00%
6x8x5	10	4	3	2.36	0.43	1.94%	1.65%	0.00%
7x8x4	10	2	0	6.52	0.58	2.17%	2.58%	0.09%
8x8x4	10	1	0	25.73	0.88	2.59%	3.19%	0.15%
9x7x4	10	0	0	61.39	1.20	3.46%	3.88%	0.18%
9x9x3	10	0	0	71.39	1.56	1.83%	3.37%	0.18%
10x6x3	10	0	0	20.59	1.31	3.02%	3.81%	0.16%
Average	10	3	1.9	18.86	0.64	2.00%	2.36%	0.09%

Table 2. Comparison Results in Medium Problems

Types	CPU time(s)				HGA versus GA		HGA versus NEH		HGA versus CDS	
	GA	HGA	NEH	CDS	The improvement rate of HGA over GA	C_{max}(HGA) < C_{max}(GA)	The improvement rate of HGA over NEH	C_{max}(HGA) < C_{max}(NEH)	The improvement rate of HGA over CDS	C_{max}(HGA) < C_{max}(CDS)
11x17x5	1.59	11.35	0.01	0.02	0.33%	9	1.76%	10	2.08%	10
12x20x6	3.84	12.60	0.01	0.01	0.34%	10	1.19%	10	2.40%	10
13x19x7	3.50	19.11	0.01	0.02	0.34%	9	1.86%	10	2.34%	10
14x18x9	8.78	24.82	0.02	0.02	0.63%	10	1.87%	10	2.73%	10
15x17x6	5.38	18.76	0.02	0.01	0.74%	10	2.79%	10	3.35%	10
16x16x7	5.65	17.00	0.02	0.02	0.61%	9	2.55%	10	3.48%	10
17x15x8	8.01	14.53	0.02	0.02	0.74%	10	2.44%	10	3.54%	10
18x16x6	5.43	12.91	0.02	0.02	0.73%	8	2.83%	10	3.42%	10
19x12x10	7.37	17.52	0.02	0.02	0.53%	9	2.18%	10	3.02%	10
20x15x3	2.64	12.25	0.02	0.02	0.94%	10	3.39%	10	4.73%	10
Average	5.22	16.09	0.02	0.02	0.6%	9.4	2.29%	10	3.11%	10

4.2.2 Medium Problems

The parameters were the same as those in small problems, except that the number of generations was 200. There were 10 types of problems with 10 instances in each type. The column $C_{\max}$(HGA) < $C_{\max}$(NEH) in Table 2 is the number of times that the minimizing $C_{\max}$ of HGA was better than that of NEH in each instances.

Table 2 shows that the computational times of NEH and CDS were lower than that of HGA. The HGA solution quality, however, was better than that of NEH and CDS. Also, the minimizing $C_{\max}$ in 100 instances was smaller than that of NEH and CDS. The comparison results of HGA and pure GA in RPFS medium problems are shown in Table 2. Although the experimental results show that the solutions quality of the non-hybrid version of GA are close to those of HGA (about 0.6%), minimizing $C_{\max}$ in each instance of HGA is better than that of pure GA in almost every test (about 94 times).

4.2.3 Large Problems

The parameters were the same as those in medium problems, except that the number of generations was 400. There were 5 types of problems with 10 instances in each type. Table 3 shows that the computational times of NEH, CDS, and pure GA were lower than that of HGA. The solution quality of HGA, however, was better than that of NEH, CDS, and pure

GA. All of the best values in 100 instances of HGA were the best among all four of these methods. In Table 3, we find that the larger the number of job, the greater the improvement quality of HGA over pure GA.

Table 3. Comparison Results in Large Problems

Types	CPU time(s)				HGA versus GA		HGA versus NEH		HGA versus CDS	
	GA	HGA	NEH	CDS	The improvement rate of HGA over GA	C_{max}(HGA) < C_{max}(GA)	The improvement rate of HGA over NEH	C_{max}(HGA) < C_{max}(NEH)	The improvement rate of HGA over CDS	C_{max}(HGA) < C_{max}(CDS)
20×20×10	5.92	26.45	0.02	0.2	0.93%	10	2.29%	10	3.09%	10
30×30×5	10.03	51.57	0.21	0.25	1.75%	10	2.70%	10	4.18%	10
40×40×5	17.22	100.29	0.31	0.37	1.71%	10	2.16%	10	3.62%	10
50×50×6	37.49	285.86	0.76	0.33	1.83%	10	2.26%	10	3.40%	10
80×80×3	44.19	635.35	0.95	0.55	2.01%	10	2.32%	10	4.58%	10
Average	22.97	219.90	0.45	0.31	1.65%	10	2.35%	10	3.77%	10

5. CONCLUSIONS

In this study, we developed a HGA for RPFS problems with makespan as the criterion. The experimental results show that HGA outperforms the other algorithms (NEH and CDS). Moreover, computational results show that HGA is superior to pure GA for large and medium size problems. GA is inspired by nature phenomena. If it exactly mimics nature, however, it takes an unexpected long computational time. Therefore, the effect of parameters must be studied thoroughly in order to obtain a good solution in a reasonable amount of time. As the probability of obtaining a near-optimal solution increases, so do the costs of longer computational time resulting from the increase in the number of generations or size of population. When dealing with large size problems or large reentrant times, setting larger population sizes or generations will increase the probability of obtaining near-optimal solutions.

In conclusion, GA provides a variety of options and parameter settings that still must be fully investigated. This study has demonstrated the potential for solving RPFS problems using a GA, and it clearly suggests that such procedures are well worth exploring in the context of solving large and difficult combinatorial problems.

6. REFERENCES

1. Baker, K. R., (1974). Introduction to sequencing and scheduling. John Wiley & Sons, New York.
2. Beasly, J. E., (1990). OR-library: Distribution Test Problems by Electronic Mail. Journal of the Operational Research Society, 41(11): 1069-1072.
3. Bispo, C. F. and Tayur, S., (2001). Managing simple re-entrant flow lines: theoretical foundation and experimental results. IIE Transactions, 33: 609-623.
4. Bowden, Jr. R. O., (1992). Genetic algorithm based machine learning applied to the dynamic routing of discrete parts. Dissertation, Department of Industrial Engineering, Mississippi State University.
5. Campbell, H. G., Dudek, R. A., and Smith, M. L., (1970). A heuristic algorithm for the *n* job, *m* machine sequencing problem. Management Science, 16(10): B630–B637.
6. Cheng, R., Gen, M., and Tsujimura, Y., (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms, part I. Computers and Operations Research, 36: 343-364.
7. Cheng, R., Gen, M., and Tsujimura, Y., (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic: genetic search strategies. Computers and Operations Research, 36: 343-364.
8. Dannenbring, D. G., (1977). An evaluation of flow shop sequencing heuristics. Management Science, 23(11): 1174-1182.
9. Gillies, A., (1985). Machine learning procedures for generating image domain feature detectors. Ph. D. thesis, University of Michigan, Ann Arbor.
10. Goldberg, D. E., (1989). Genetic algorithms in search. Optimization and Machine Learning. Addison-Wesley, Reading, Ma.
11. Grefenstette, J. and Baker, J., (1989). How genetic algorithms work: A critical look at implicit parallelism. ICGA: 20-27.
12. Gupta, J. N. D., (1971). A functional heuristic algorithm for the flowshop scheduling problem. Operational Research Quarterly, 22(1): 39-47.
13. Holland, J., (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.
14. Johnson, S. M., (1954). Optimal two- and three-stage production schedules with set up times included. Naval Research Logistics Quarterly, 1: 61-68.

15. Kubiak, W., Lou, S. X. C., and Wang, Y., (1996). Mean flow time minimization in reentrant job-shops with a hub. Operations Research, 44: 764-776.
16. Manne, A. S., (1960). On the job-shop scheduling problem. Operations Research, 8: 219-223.
17. Michalewicz, Z., (1994). Genetic Algorithm + Data Structure = Evolution Programs. Second Ed., Springer-Verlag, New York.
18. Murata, T., Ishibuchi, H., and Tanaka, H., (1996). Genetic algorithms for flow shop scheduling problems. Computers and Industrial Engineering, 30: 1061-1071.
19. Nawaz, M., Enscore E. E., and Ham, I., (1983). A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. OMEGA The International Journal of Management Science, 11(1): 91-95.
20. Palmer, D. S., (1965). Sequencing jobs through a multi-stage process in the minimum total time— A quick method of obtaining a near optimum. Operational Research Quarterly, 16(1): 101-107.
21. Pinedo, M., (2002). Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, New Jersey.
22. Pan, J. C. H. and Chen, J. S., (2003). Minimizing makespan in re-entrant permutation flow-shops. Journal of the Operational Research Society, 54: 642–653.
23. Pan, J. C. H. and Chen, J. S., (2004). A comparative study of schedule-generation procedures for the reentrant shops. International Journal of Industrial Engineering– Theory, Applications and Practice, 11: 313-321.
24. Rinnooy Kan, A. H. G., (1976). Machine Scheduling Problems: Classification, Complexity and Computations. Martinus Nijhoff, The Hague, Holland.
25. Vargas-Villamil, F. D. and Rivera, D. E. (2001). A model predictive control approach for real-time optimization of reentrant manufacturing lines. Computers in Industry, 45: 45-57.
26. Wagner, H. M., (1959). An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly, 6: 131-140.
27. Wang, M. Y., Sethi, S. P., and Van De Velde, S. L., (1997). Minimizing makespan in a class of reentrant shops. Operations Research, 45: 702-712.
28. Wilson, J. M., (1989). Alternative formulations of a flow-shop scheduling problem. Journal of the Operational Research Society, 40: 395-399.

BIOGRAPHICAL SKETCH

Jen-Shiang Chen is a Professor at the Department of Business Administration, Far East University. He received his Ph.D. from National Taiwan University of Science and Technology, Taiwan. His current research interests include production scheduling and supply chain management. He has published articles in European Journal of Operational Research, OMEGA, Journal of the Operational Research Society, Computers & Operations Research, Engineering Optimization, International Journal of Systems Science, International Journal of Advance Manufacturing Technology, International Journal of Industrial Engineering・ Theory, Applications and Practice, and others.

Jason Chao-Hsien Pan is a Professor at the Department of Business Administration, Takming University of Science and Technology. He completed his Ph.D. degree in Industrial Engineering from the University of Houston. His current research interests include production scheduling and inventory management. His recent publications have appeared in European Journal of Operational Research, International Journal of Production Research, Journal of the Operational Research Society, Computers & Operations Research, International Journal of Systems Science, Production Planning & Control, and others.

Chien-Min Lin completed his mater degree in Industrial Management from the National Taiwan University of Science and Technology. His research interest is production scheduling.