Pergamon

Computers ind. Engng Vol. 30, No. 4, pp. 919–929, 1996

S0360-8352(96)00042-3
0360-8352/96 $15.00 + 0.00

GENETIC ALGORITHMS APPLIED TO THE CONTINUOUS FLOW SHOP PROBLEM

CHUEN-LUNG CHEN, RANGA V. NEPPALLI and NASSER ALJABER

Department of Industrial Engineering, Mississippi State University, P.O. Drawer U, Mississippi State, MS 39762, U.S.A.

Abstract—This research develops an approach for applying Genetic Algorithms (GA) to scheduling problems. We generate a GA based heuristic for continuous flow shop problems with total flow time as the criterion. The effects of several crucial factors of GA on the performance of the heuristic for the problem are explored in detail. The computational experience of heuristic provides several observations of the application of GA, and strongly supports that the applications of GA are problem specific. The computational experience also shows that GA can be good techniques for scheduling problems. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

The flow shop problem is a scheduling problem which considers m different machines and n jobs; each of the jobs consists of m operations, and each of the operations requires a different machine and all the jobs are processed in the same processing order. Since Johnson's work [1] on the n-job two-machine flow shop problem, flow shop problems have been extensively treated in the literature. Many algorithms have been proposed to solve the problems, and in most of these problems, it is assumed that infinite intermediate storage exists which can hold all the partially processed jobs when the jobs cannot be further processed (because the subsequent machines are busy) [2]. However, this assumption is not practical in some cases. For instance, in rolling of steel the process has to be continuous and the intermediate storage cannot exist. Reddi and Ramamoorthy [2] denoted this special flow shop problem as FSNIS (Flow Shop with No Intermediate Storage).

The NP-hardness of the problem has been studied by Sahni and Cho [3]. Wismer [4], and Reddi and Ramamoorthy [2] modeled the problem with makespan as the criterion as the well-known traveling salesman problem (TSP), and solved the problem by using TSP techniques. Owing to the fact that no wait is allowed for jobs for the FSNIS problem, Wisner showed that the minimum difference between the start time of two consecutive jobs is independent of the sequence of the other jobs. For instance, for a n-job, 3-machine problem, M_1, M_2, and M_3 are the machines. Figure 1 displays the Gantt chart for the first two jobs, job x and job y, of a sequence of the n jobs when "wait" is allowed, while, for the same sequence, Fig. 2 displays the Gantt chart for the two jobs when "no wait" is allowed. In Fig. 2, d_{xy} is the minimum difference between the start time of the two jobs. We can see that as long as job x and job y are placed consecutively in a sequence, the value of d_{xy} will never change regardless of their positions in the sequence since "no wait" is allowed for the jobs. The d value of any two consecutive jobs can be obtained by using the same procedure as Figs 1 and 2. With this property of the FSNIS problem, the makespan, M, of a sequence $\{i(1), i(2), \ldots, i(n)\}$ can be described as follows

$$M = \left(\sum_{j=1}^{n-1} d_{i(j),i(j+1)} \right) + R_{i(n)},$$

where $R_{i(n)}$ is the total processing time of the last job of the sequence.

In order to model the FSNIS problem as a (TSP), Wismer defined two dummy variables, job

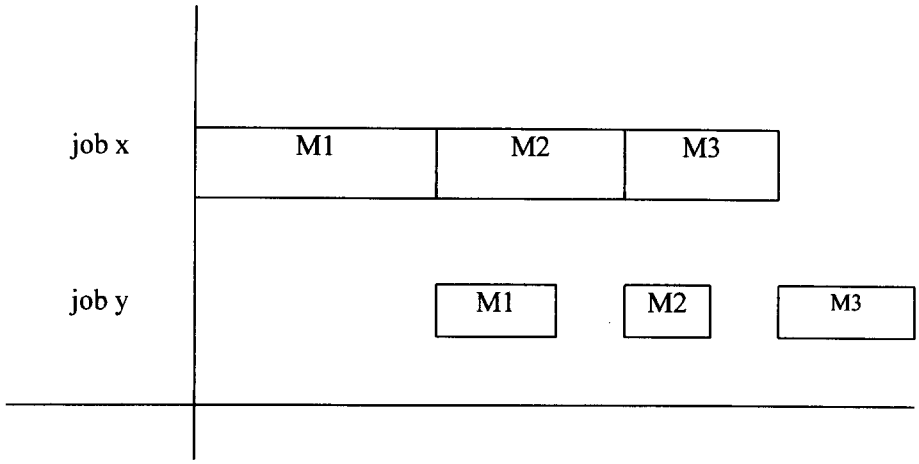

Fig. 1. Processing of job x and job y with "wait" allowed.

0 and job $n + 1$, and defined the following d values concerning the two jobs so that a sequence can always start from job 0, end at job $n + 1$, and return to job 0:

$$d_{0,j} = 0, \qquad j = 1, 2, \ldots, n;$$

$$d_{n+1,j} = d_{j,0} = \infty, \qquad j = 1, 2, \ldots, n;$$

$$d_{n+1,0} = 0,\ d_{0,n+1} = \infty, \text{ and } d_{n,n+1} = R_{i(n)}.$$

With these definitions, the objective function of the FSNIS problem with makespan as the criterion can be written as follows

$$\text{Min}\left\{M = \sum_{j=0}^{n} d_{i(j),i(j+1)}\right\}.$$

Apart from the research with makespan as the criterion, Van Deman and Baker [5], Bonny and Gundry [6], Panwalker and Woollam [7], and Rajendran and Chaudhuri [8] solved the problem with total flow time as the criterion. Based on the Wismer's idea, Van Deman and Baker [5] generated the objective function for the problem with total flow time as the criterion as follows:

$$\text{Min}\left\{Z = \sum_{j=1}^{n}\sum_{k=1}^{j} d_{i(k-1),i(k)}\right\}, \text{ where } d_{i(0),i(1)} = 0.$$

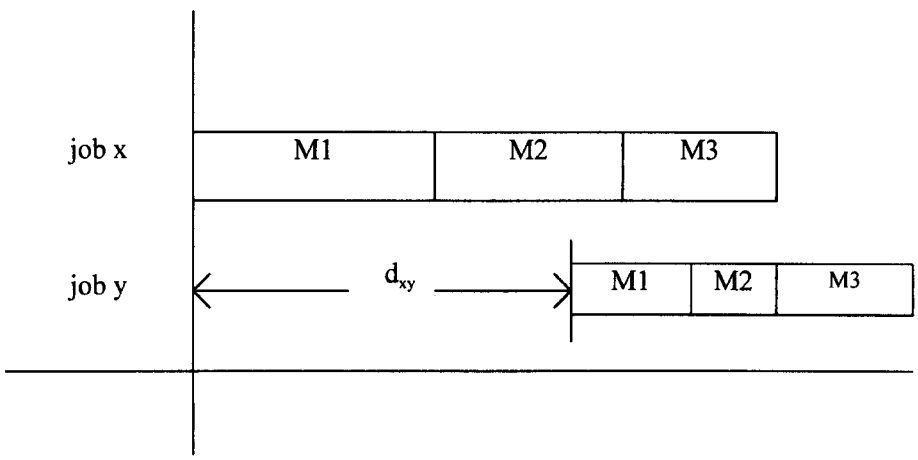

Fig. 2. Processing of job x and job y with "no wait" allowed.

They then developed a branch and bound algorithm for the problem. Their experimental results suggested that the FSNIS problem with mean flow time as the criterion can be solved as fast as the traditional flow shop problem with makespan as the criterion. Bonny and Gundry [6] proposed a slope matching algorithm in which geometrical relationships between the cumulative process times were utilized to derive start and end slopes for each job in order to use these slopes to find a job sequence. Panwalker and Woollam [7] considered a special case of the FSNIS problem which they denoted as OFSNW (Ordered Flow Shop problem with No Waiting) and they proved that SPT ordering of jobs minimizes mean flow times for all OFSNW problems. Rajendran and Chaudhuri [8] developed a job insertion heuristic for the problem. They compared the performance of their heuristic with that of the other existing heuristics, and showed that their heuristic dominates all the existing heuristics.

The objective of this research is to develop a Genetic Algorithms (GA) based heuristic for the FSNIS problem with total flow time as the criterion. We propose an approach for analysing the crucial factors for the application of GA to the candidate problem. The proposed approach can be extended to several of the related flow shop problems. The remainder of the paper is organized as follows. Section 2 presents an introduction to GA and Section 3 describes the methodology of the application of GA to the FSNIS problem. Finally, in Section 4 the analysis of the performance of the GA based heuristic is provided followed by the Conclusion in Section 5.

2. GENETIC ALGORITHMS

GA are probabilistic search techniques, which mimic the process of evolution. The fundamental principles of genetics lead to the development of GA. In order to apply GA to a problem, generally the solution space of the problem is represented by a population of structures where each structure is a possible solution to the problem. Then, a certain number of structures are chosen to form the initial generation. The structures of the next generation are generated by applying simple genetic operators to the parent structures selected from the existing generation. According to the idea that "good parents produce better offspring", a structure with higher fitness value in the current generation will have higher probability of being selected as a parent (similar to the concept of survival). When we repeat this process, we can observe a continuous improvement in the structures' performance from one generation to the next.

After being introduced by Holland in the 1970s, GA have been applied to a wide variety of problems. Some of the typical application areas of GA are as follows: the travelling salesman problem (TSP) (Grefenstette *et al.* [9]), the scheduling Problem (Davis [10], Cleveland and Smith [11], Biegal and Davern [12], Chen *et al.* [13], and Vempati *et al.* [14]), the VLSI circuit layout design problem (Fourman [15], Cohoon [16]), the computer-aided gas pipeline operation problem (Goldberg [17, 18]), the communication network control problem (Cox *et al.* [19]), the robot control problem (Davidor [20]), and the real time control problem in manufacturing systems (Grefenstette [21] and Bowden [22]). In most of the applications of GA, the results provided an insight to the robustness of GA in terms of its applicability and quality of the solutions.

The following brief outline of GA illustrates the functioning of GA, where the notation $S(t)$ is the population in the t^{th} generation; $s_i(t)$ is the i^{th} member in $S(t)$; $f(s_i(t))$ is the fitness value of $s_i(t)$, and TOTFIT (t) is the sum of $f(s_i(t))$ for all $s_i(t)$ and $S(t)$.

Step 1

Generate the initial population, $S(t)$, where $t = 0$. Determine the size of the population, POPSIZ, and the number of generations, GENER.

Step 2

Calculate the fitness value of each member, $f(s_i(t))$, for population, $S(t)$.

Step 3

Calculate the selection probability for each $s_i(t)$, where the selection probability is defined as $P(s_i(t)) = f(s_i(t))/\text{TOTFIT}$.

Step 4

Select a pair of members (parents) that will be used for reproduction via the selection probability.

Step 5

Apply genetic operators (crossover, mutation, inversion) to the parents. Replace the parents with the resulting offspring to form a new population, $S(t+1)$, for generation $t+1$. If the size of the new population is equal to the POPSIZE, then go to step 6, else go to step 4.

Step 6

If current generation, $t+1$, is equal to GENER, then stop, else go to step 2.

3. METHODOLOGY

According to the outline of the GA approach, an application of GA should consider the following elements: (1) Representation of Structure, (2) Initial Population, (3) Population Size, (4) Selection Probability, (5) Genetic Operators, and (6) Termination.

Each of the above elements affects the performance of GA and it is necessary to effectively estimate these fundamental elements of GA. The methodology adopted in this paper presents the procedure for determining these elements of GA for the application to the FSNIS problem. The procedure can be divided into two parts: (i) implementation of GA to the problem, and (ii) optimization of control parameters of GA. The first part is concerned with the determination of the representation of structure, the generation of initial population, the approach for determining the selection probability, the genetic operators, and the termination criteria. The second part is concerned with optimizing the control parameters of GA which will affect the performance of the implementation of GA. The control parameters include the population size, the rates of genetic operators, the generation gap, etc.

3.1. Implementation of GA

The representation of the structure affects the application of the genetic operators. A common representation of the structure for permutation scheduling problems (such as single machine problems and flow shop problems) is simply the sequence of jobs in the problems. For instance, for an 8-job flow shop problem, a structure can be represented as any sequence of the eight jobs such as 1 2 3 4 5 6 7 8. Several efficient genetic operators have been developed for scheduling problems with this structure representation (Cleveland and Smith [11]). Therefore, we use the same method to represent the structure of the FSNIS problem. The initial population can be generated in several ways. Most applications of GA generate the initial population randomly. According to the fundamental concept of GA, the significance of initial population should consist of a diverse set of solutions. It is believed that the more diverse population initiates more effective search. It is also believed that improving the average fitness value of the initial population will reduce the computation time. So in generating the initial population with different procedures, our attempt is to effectively improve the average fitness value of the population, and not to significantly affect the diversity of the population. In our proposed procedure, half of the members in the initial population is generated randomly, and the other half is generated using some well known heuristics. The heuristics include two heuristics to the general flow shop problem: the CDS method, and the Danninbring's method, and the best existing heuristic for the continuous flow shop problem: the Rajendran and Chaudhuri's [8] Job Insertion Based (JIB) method. To generate the members in the initial population using these heuristics, the first member is generated using Rajendran and Chaudhuri's method. The following $m-1$ members are generated by the CDS method (m is the number of machines considered in the problem), and an additional member is generated by the Danninbring method. If the number of the members generated is less than one-half of the population size, a member is randomly selected and two randomly selected positions of the member are swapped to generate a new member for the initial population. This procedure will be repeated until the number of the members generated equals one-half of the population size. In order to estimate the effect of the proposed generating procedure for the initial population, a second method which randomly generates all the members of the initial population is also considered. A comparison between the two approaches will be discussed later in the section of computational experience.

Selection probability is another important factor to consider in implementing GA. According to the general definition, the selection probability of a member should project the performance measure of the member in the population. So in the present problem, the fitness value and the selection probability of a member with lower total flow time should be high. The following procedure is one of the common procedures available for estimating the fitness value and selection probability of each member in a population.

(1) Calculate the total flow time for each member in the population.
(2) Find Fmax, which is the maximum total flow time in the population.
(3) Calculate the fitness value of each member, which is equal to the difference between the Fmax and the total flow time of each member.
(4) Calculate the selection probability of each member based on its fitness value. This is equal to the member's fitness value divided by the sum of each member's fitness value in the population.

The selection probability of each member is then used as a criterion for the selection parents for the reproduction of offspring. Members with large selection probabilities are selected for reproduction more often than members with low selection probabilities. The process for selecting parents is implementing via the common roulette wheel selection procedure outlined by Goldberg [23].

There are three well known basic genetic operators found in the GA literature: (i) crossover, (ii) mutation, and (iii) inversion. Among these three, crossover and mutation are the most commonly used operators. As mentioned earlier, the application of crossover operators varies accordingly based on the representation of the structure. When we represent the structure using the sequence of jobs in the problem, some of the popular crossover operators are Goldberg's Partially Mapped (PMX) operator, edge recombination operator, subtour-swap operator, subtour chunk operator, subtour replace operator and weighted chunking operator. Among these operators, we have used Goldberg's PMX operator which is one of the better performers among the above operators (Cleveland and Smith [11]).

The following 8-job example is generated to illustrate the procedure of the PMX operator [13]. Let structures A and B be the parent structures chosen for crossover. The elements in the structures are jobs.

$$\begin{aligned} \mathrm{A} &= 2\ 8\ 6\ 4\ 5\ 7\ 1\ 3, \\ \mathrm{B} &= 8\ 7\ 2\ 1\ 3\ 4\ 6\ 5. \end{aligned}$$

In applying the PMX operator to A and B, we first randomly choose a common interval from A and B. In this example, an interval from positions three to five is randomly chosen. Then the mappings of the elements in the two selected intervals are determined. In this example, the mappings between the selected intervals are 6 to 2, 4 to 1, and 5 to 3. Next, we swap the two intervals in A and B. The following structures show the temporary results after the swap. Obviously, both of them are not feasible because some jobs occur more than once in the sequence.

$$\begin{aligned} \mathrm{A} &= 2\ 8|2\ 1\ 3|7\ 1\ 3, \\ &\qquad\ \ |\quad\ \ | \\ \mathrm{B} &= 8\ 7|6\ 4\ 5|4\ 6\ 5. \end{aligned}$$

Therefore, we have to exchange the mapping elements which are determined in the previous step and do not locate in the selected intervals in A and B so that the new structures can be feasible. In this example, the 2, 1, and 3 in the positions 1, 7, and 8 of structure A are replaced by 6, 4, and 5, respectively. And the 4, 6, and 5 in the positions 6, 7, and 8 of structure B are replaced by 1, 2, and 3, respectively. The new structures generated are shown as follows.

$$\begin{aligned} \mathrm{A}' &= 6\ 8|2\ 1\ 3|7\ 4\ 5, \\ &\qquad\ \ |\quad\ \ | \\ \mathrm{B}' &= 8\ 7|6\ 4\ 5|1\ 2\ 3. \end{aligned}$$

The mutation operation used in this research is an operation which randomly picks two positions

in a structure and swaps the jobs in these two positions to generate a new structure. For instance, when applying the mutation operation to the following structure,

$$C = 3\ 1\ 4\ 2\ 5\ 7\ 6\ 8,$$

we randomly pick two positions of the structure (positions 2 and 6) and swap the jobs in these two positions (jobs 1 and 7) to generate a new structure,

$$C' = 3\ 7\ 4\ 2\ 5\ 1\ 6\ 8.$$

It is difficult to develop the termination criteria in the operation of GA. There are two conflicting factors to be considered for developing a termination criteria: the degree of the GA search and the CPU time required for the search. If we terminate the GA search after few generations (the CPU time is short, but the degree of search is poor), there is scope for an error of terminating the search in a premature stage. However, if we let the search terminate through natural convergence (the CPU time is long, but the degree of search is better), then the CPU time required for the search may be unreasonable. So it is necessary to develop a logical criteria of termination to obtain advantages of both short CPU time and better search. Based on the above two factors, we have used two termination criteria: (i) if the number of structures in the population with the lowest total flow time (fittest members) are more than 60% of the population (which means the search is moreover converged), terminate the search. (ii) if the number of generations exceeds 60, then terminate the search. The number of generations in the second case is selected as 60 on the basis of several experiments as it is observed that for most problem sizes, the GA based heuristic converged within 60 generations.

3.2. Optimization of control parameters of GA

Even though GA based approaches proved to be robust and efficient, another important factor to be considered in making GA search efficient is the tuning of parameter values. According to Grefenstette [24], the parameters considered in GA include population size (N), crossover rate (C), mutation rate (M), generation gap (G), scaling window (W), and selection strategy (S). In his research, Grefenstette denoted a GA with some specific parameter values as GA (N, C, M, G, W, S).

Population size is the number of structures in a population. Crossover rate is the probability that the crossover operator applies to the chosen parents, and mutation rate is the probability that the mutation operator applies to the chosen parents. Generation gap is the percentage of the population to be replaced during each generation. Scaling window is the number of generations during which the value of f' is updated, where f' is used as a base for calculating the fitness value of each structure. For a minimizing problem, the value of f' is defined as the maximum objective value of the structures evaluated so far, and the fitness of a structure is defined as the difference between f' and the objective value of the structure. The value of W is defined to be from 0 to 7. A detailed explanation of each of the values can be found in [24]. Selection strategy is the approach in selecting structures for the next generation. Grefenstette defined two selection strategies, E and P. If $S = E$, then the best performing structure will always survive intact into the next generation. If $S = P$, then a pure selection strategy is employed.

It should be noted that tuning the parameter values of GA is a complex process and can affect the efficiency of the algorithms. In most applications of GA, these parameter values are tuned based on some trial examples (hand optimization technique). The concept of optimal design of parameter values was studied by DeJong [25] when he analysed the basic behavior of GA on typical environments. DeJong's approach is considered as a hand optimization technique in which he experimented with several combinations of parameter values on a test bed of five functions. The set of the optimal parameter values generated by DeJong is GA (50, 0.6, 0.001, 1.0, 7, E).

After DeJong, only a few researchers focused on optimal design of parameter values of GA. Grefenstette's [24] research is the most important one. He developed a metalevel GA to optimize the parameter values of the actual GA, where the actual GA is the GA used to solve the problems considered with the parameter values assigned from the metalevel GA. In his approach, the structure in the metalevel GA is represented as the combination of the possible values of the six

parameters. The possible values of the six parameters are as follows: population size ranges from 10 to 160 in increments of 10, crossover rate ranges from 0.25 to 1.00 in increments of 0.05, mutation rate is allowed to be one of the eight values increasing exponentially from 0.0 to 1.0, generation gap is ranged from 0.30 to 1.0 in increments of 0.1, scaling window ranges from 0 to 7 in increments of 1, and selection strategy can only be S or P. DeJong's parameter values are adopted in the metalevel GA. Therefore, 50 structures are randomly generated for the initial population of the metalevel GA, and each structure, which contains six parameter values, is then passed down to the actual GA for solving the same test functions that DeJong considered. The fitness value of each structure is defined as a function of its performance on the test functions. The procedure of metalevel GA was implemented based on DeJong's parameter values, and terminated in the twentieth generation. The set of the optimal parameter values generated by the metalevel GA is GA (30, 0.95, 0.01, 1.0, 1, E). Grefenstette also compared the parameter values with DeJong's parameter values. The comparison was made on a test function that is not used in the test bed of the five standard functions, and the results showed that his parameter values are slightly more effective than DeJong's parameter values.

In this paper, we have applied the GA based heuristic with DeJong's and Grefenstette's optimal parameter values to the FSNIS problem, respectively. It was observed that both sets of the parameter values did not perform well for the FSNIS problem. So we have modified Grefenstette's metalevel GA, called modified metalevel GA, for tuning the parameter values of the GA based heuristic. However, unlike Grefenstette, we kept $W = 1.0$, $G = 1$, and $S = E$ in this approach, and considered the other three elements to represent the structure in the population of the metalevel GA: (1) population size, (2) crossover rate, and (3) mutation rate. The population size is defined in a range from 40 to 160 in increments of 1; the crossover rate is defined in a range from 0.5 to 1 in increments of 0.025, and the mutation rate is defined in a range from 0.005 to 0.05 in increments of 0.001. Grefenstette's parameter values are used in this modified metalevel GA. The initial population of the modified metalevel GA is generated randomly, and each structure in the population is then passed down to the actual GA for solving a set of problems, where the set of problems includes 10 different larger size FSNIS problems (see the section of computational experience). The fitness value of the structure of the modified metalevel GA is defined as a function of the solutions provided in the actual GA. The parameter values obtained using this approach are that population size is equal to 95, crossover rate is equal to 0.725, and mutation rate is equal to 0.009. The performance of this set of parameter values is discussed in the following section.

4. COMPUTATIONAL EXPERIENCE

In order to examine the effectiveness of the GA based heuristic, four comparisons were made over a wide range of jobs and machines. For each of these comparisons, 200 problems were generated for 20 different combinations of jobs ranging from 10 to 25 with the number of machines ranging from 5 to 25. The processing times were randomly sampled from a uniform distribution ranging from 1 to 99. The results of the four comparisons are presented in the following four tables.

Table 1 presents a comparison between the two approaches for generating the initial population of the GA based heuristic. One approach is to fill the initial population with heuristic solutions combined with random solutions, and the other approach is to generate the initial solution completely with random solutions (the conventional way). These two approaches are denoted by HRIP and RIP respectively. In HRIP, the heuristics that we used to generate the initial population are some well known heuristics to the general flow shop problem, such as the CDS method and Danninbring along with the Job Insertion Based (JIB) [8] heuristic developed by Rajendran and Chaudhury which has been proven to be the best existing heuristic algorithm to solve the problem under consideration. In Table 1, the first column is a couple of the number of jobs 'n', and the number of machines 'm'. The third and fourth columns illustrate the number of times that the best solution was obtained using the corresponding approach. The results in the fifth column show the relative performance of the two approaches which was computed by $\text{average}_{\text{HRIP}}/\text{average}_{\text{RIP}}$. The results in Table 1 indicate that the HRIP yields better solutions than the RIP. The number of times in which the HRIP gives better results than the RIP is 188 out of 200, and the superiority increases

Table 1. Analysis of the effect of initial population on the performance of GA

$n \times m$	No. prob.	HRIP better	RIP better	Ave (HRIP/RIP)
10 × 5	10	8	2	0.9860
10 × 10	10	8	2	0.9920
10 × 15	10	9	1	0.9920
10 × 20	10	6	4	0.9930
10 × 25	10	10	0	0.9900
15 × 5	10	9	1	0.9700
15 × 10	10	9	1	0.9450
15 × 15	10	10	0	0.9670
15 × 20	10	10	0	0.9740
15 × 25	10	9	1	0.9540
20 × 5	10	10	0	0.9330
20 × 10	10	10	0	0.9400
20 × 15	10	10	0	0.9400
20 × 20	10	10	0	0.9540
20 × 25	10	10	0	0.9220
25 × 5	10	10	0	0.9050
25 × 10	10	10	0	0.9030
25 × 15	10	10	0	0.9130
25 × 20	10	10	0	0.9070
25 × 25	10	10	0	0.9015
Total	200	188	12	—

rapidly when the problem size increases. These results also illustrate a very important point: the initial population significantly affects the performance of the GA based heuristic, and furthermore, the inclusion of some specific knowledge of the problem under consideration, which are the heuristics for the problem in this research, in the initial population may improve the performance of GA.

Table 2 presents the comparison between the GA based heuristic using the HRIP to generate the initial population and the Job Insertion Based (JIB) heuristic proposed by Rajendran and Chaudhury. This Table does not contain a column of the number of times that the best solution was obtained by the JIB heuristic because the solution from this heuristic is included in the initial population, and the solution is stored in every generation if it is the best solution in the generation. This means that the JIB heuristic would never give a solution better than the best solution obtained by the GA based heuristic. Column 5 of Table 2 shows the relative performance of the GA based heuristic to the JIB heuristic which was computed by $\text{average}_{GA}/\text{average}_{JIB}$.

According to the results in Table 2, the GA based heuristic yields better results than the JIB heuristic 168 times out of 200. This means that the GA based heuristic can usually improve the results obtained using the JIB heuristic. However, the ratios in the fifth column show that the improvement is not substantial and is reducing when the problem size is increased. This can be attributed to the fact that the JIB heuristic is a good heuristic to the FSNIS problem.

Tables 1 and 2 also reveal that if RIP is used to generate the initial population for the GA based

Table 2. Comparison of JIB with the GA heuristic

$n \times m$	No. prob.	GA better	Even (=JIB)	Ave (GA/JIB)
10 × 5	10	9	1	0.9512
10 × 10	10	10	0	0.9571
10 × 15	10	10	0	0.9828
10 × 20	10	10	0	0.9828
10 × 25	10	9	1	0.9810
15 × 5	10	10	0	0.9730
15 × 10	10	10	0	0.9860
15 × 15	10	9	1	0.9633
15 × 20	10	7	3	0.9950
15 × 25	10	8	2	0.9960
20 × 5	10	9	1	0.9890
20 × 10	10	9	1	0.9970
20 × 15	10	7	3	0.9963
20 × 20	10	7	3	0.9943
20 × 25	10	8	2	0.9923
25 × 5	10	8	2	0.9935
25 × 10	10	8	2	0.9935
25 × 15	10	7	3	0.9921
25 × 20	10	6	4	0.9963
25 × 25	10	7	3	0.9970
Total	200	168	32	—

Table 3. Comparison of optimal parameter values (Grefenstette vs modified metalevel)

$n \times m$	No. prob.	Grefenstette better	Modified metalevel better	Ave (GA/Grefenstette)
10 × 5	10	3	7	0.9755
10 × 10	10	2	8	0.9855
10 × 15	10	1	9	0.9885
10 × 20	10	2	8	0.9868
10 × 25	10	3	7	0.9920
15 × 5	10	1	9	0.9700
15 × 10	10	0	10	0.9630
15 × 15	10	0	10	0.9300
15 × 20	10	1	9	0.9650
15 × 25	10	1	9	0.9830
20 × 5	10	0	10	0.9310
20 × 10	10	0	10	0.9380
20 × 15	10	0	10	0.9420
20 × 20	10	0	10	0.9460
20 × 25	10	0	10	0.9470
25 × 5	10	0	10	0.9040
25 × 10	10	0	10	0.9080
25 × 15	10	0	10	0.9180
25 × 20	10	0	10	0.9160
25 × 25	10	1	9	0.9130
Total	100	15	185	—

heuristic, the final solution of the heuristic will be worse than that obtained using the JIB heuristic. This confirms the importance of the initial population for the GA based heuristic.

In Tables 3 and 4 the performance of GA using the parameter values obtained from the modified metalevel GA approach is compared with the performance of GA using Grefenstette's and DeJong's parameter values, respectively. These two comparisons strongly support the argument that the parameter values of GA are problem specific. Out of the 200 test problems, the GA using our parameter values produced 185 better results than the GA using Grefenstette's parameter values. Similarly, the GA using our parameter values produced 195 better results than the GA using DeJong's parameter values. The more significant point is that the relative performance of the GA using our parameter values can be up to 10 and 20% better than the GA using Grefenstette's and DeJong's parameter values, respectively. This clearly indicates the domination of the parameter values obtained by the modified metalevel GA on both Grefenstette's and DeJong's parameter values.

The GA based heuristic was implemented in FORTRAN 77, and run on a SUN 4/490 machine. Table 5 exhibits the average CPU times required to solve the problems in each of the selected

Table 4. Comparison of optimal parameter values (DeJong vs modified metalevel)

$n \times m$	No. prob.	DeJong better	Modified metalevel better	Ave (GA/DeJong)
10 × 5	10	0	10	0.9660
10 × 10	10	0	10	0.9700
10 × 15	10	1	9	0.9760
10 × 20	10	3	7	0.9870
10 × 25	10	1	9	0.9840
15 × 5	10	1	9	0.9450
15 × 10	10	0	10	0.9540
15 × 15	10	0	10	0.9000
15 × 20	10	0	10	0.9520
15 × 25	10	1	9	0.9570
20 × 5	10	0	10	0.9200
20 × 10	10	0	10	0.9020
20 × 15	10	0	10	0.9100
20 × 20	10	0	10	0.9315
20 × 25	10	0	10	0.9264
25 × 5	10	0	10	0.8120
25 × 10	10	0	10	0.8600
25 × 15	10	0	10	0.8950
25 × 20	10	0	10	0.8880
25 × 25	10	0	10	0.8910
Total	200	7	193	—

Table 5. Average CPU times (s)

	m				
n	5	10	15	20	25
10	0.64	0.74	0.92	1.03	1.22
15	1.10	1.30	1.60	1.79	2.01
20	1.77	2.19	2.52	2.83	3.24
25	2.81	3.33	3.80	4.32	4.82

problem sizes presented earlier in this section. It is clear that the CPU time required to solve the FSNIS problem using the GA based heuristic is very reasonable and practical.

5. CONCLUSIONS

In this paper, we developed a GA based heuristic for the FSNIS problem. The computational experience has shown that the GA based heuristic can usually improve the results obtained using the JIB heuristic for the candidate problem. The approach discussed in this paper can be easily extended to most of the other flow shop problems, or other types of scheduling problems.

This research provided three important points for the application of GA. First, the parameter values of GA are problem specific. Different parameter values of GA may yield significant different results for the same problem. So we have to generate parameter values of GA for different problems, and we should not just adapt parameter values from some other research. We believe that the modified metalevel GA approach is a good choice.

Second, some specific knowledge of the problem under consideration may improve the performance of GA. For instance, the initial population in this paper includes several solutions obtained from the three existing heuristic for the problem. However, the randomness of the members in the initial population should not be overlooked. Otherwise, GA will generally rapidly converge to some solution which may not be good. We tested this idea using another approach to generate the initial population. We filled the initial population by first generating several solutions using the three heuristics, then randomly choosing a solution from the existing member in the population and applying the mutate operator to the solution to generate a new member in the initial population. This procedure continues until the population size is reached. Using this initial population, GA always converges in no more than 20 generations, but the results are not as good as the approach used in our GA based heuristic. The reason for this phenomena may be that the members in the initial population are gathered in some specific area of the solution space of the problem. So GA cannot move out of the area, and can only converge to a good solution in this area.

Third, completely randomly generating the initial population for GA may not be a good approach, specifically, for the larger size problem. So it is worth while studying different approaches for generating the initial population for GA.

The application of GA to optimization problems has received a lot of attention in the past decade. However, there are still a lot of unknowns behind it, specifically, for the non-binary coding optimization problem such as the scheduling problem. In this research, we have observed the effects of several elements of GA on the application of GA to a flow shop problem. Even though the results are very encouraging, we believe that the results can still be improved.

REFERENCES

1. S. M. Johnson. Optimal two-and-three-state production schedules with setup times included. *Naval Res. Logist. Q.* **1**, 61–68 (1954).
2. S. S. Reddi and C. V. Ramamoorthy. On the flow-shop sequencing problem with no wait in process. *Opl. Res. Q.* **23**, 323–331 (1972).
3. S. Sahni and Y. Cho. Complexity of scheduling shop with no-wait process. *Math. Opns. Res.* **1**, 448–457 (1979).
4. O. A. Wismer. Solution of the flowshop-sheduling problem with no intermediate queues. *Opns. Res.* **20**, 689–697 (1972).
5. J. M. Van Deman and K. R. Baker. Minimizing mean flowtime in the flow shop with no intermediate queues. *AIIE Trans.* **6**, 28–34 (1974).
6. M. C. Bonny and S. W. Gundry. Solutions to the constrainted flowshop sequencing problem. *Opnl Res. Q.* **27**, 869–883 (1976).

7. S. S. Panwalker and C. R. Woollam. Ordered flow shop problems with no in-process waiting: further results. *J. Opl. Res. Soc.* **31,** 1039–1043 (1980).
8. C. Rajendran and D. Chaudhuri. Heuristic algorithms for continuous flow-shop problem. *Naval Res. Logist. Q.* **37,** 695–705 (1990).
9. J. J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht. Genetic algorithm for the travelling salesman problem. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, 160–168 (1985).
10. L. Davis. Job shop scheduling with genetic algorithms. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, 136–140 (1985).
11. G. A. Cleveland and S. F. Smith. Using genetic algorithms to schedule flow shop release. *Proc. 3rd. Int. Conf. on Genetic Algorithms Applications*, 160–169 (1989).
12. J. E. Biegal and J. J. Davern. Genetic algorithms and job shop scheduling. *Comput. ind. Engng* **19** (1990).
13. C. L. Chen, V. S. Vempati and Aljaber N. An application of genetic algorithms for flow shop problems. *Europ. J. Opl. Res.* **80,** 389–396 (1995).
14. V. S. Vempati, C. L. Chen and S. F. Bullington. An application of genetic algorithms to the flow shop problem. *Proc. 15th Int. Conf. Comput. Ind. Eng.* (1993).
15. M. P. Fourman. Comparison of symbolic layout using genetic algorithms. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, 141–155 (1985).
16. J. P. Cohoon. Distributed genetic algorithms for the floorplan design problem. *IEEE Trans. Computer-Aided Design* **10,** 483–491 (1991).
17. D. E. Goldberg. Computer-aided pipeline operation using genetic algorithms and rule learning. Part I: Genetic algorithms in pipeline optimization. *Engr. Comput.* **3,** 35–45 (1987).
18. D. E. Goldberg. Computer-aided pipeline operation using genetic algorithms and rule learning. Part II: Rule learning control of a pipline under normal and abnormal conditions. *Engr. Comput.* **3,** 47–58 (1987).
19. L. A. Cox, L. Davis and Y. Qiu. Dynamic anticipatory routing in circuit-switched telecommunication networks. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York (1991).
20. Y. Davidor. A genetic algorithms applied to robot trajectory generation. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York (1991).
21. J. J. Grefenstette. A system for learning control strategies with genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms*, 160–168 (1989).
22. R. O. Bowden Jr. Genetic algorithm based machine learning applied to the dynamic routing of discrete parts. Ph.D Dissertation, Department of Industrial Engineering, Mississippi State University (1992).
23. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA (1989).
24. J. J. Grefenstette. Optimized control of parameters for genetic algorithms. *IEEE Trans. Syst., Man, Cyber.* **SMC-16,** 122–128 (1986).
25. K. A. DeJong. Adaptive systems design: a genetic approach. *IEEE Trans. Syst., Man, Cyber.* **SMC-16,** 566–574 (1980).