International Journal of Production Research

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

An improved genetic algorithm for the flowshop scheduling problem

R. Rajkumar [a] & P. Shahabudeen [b]

[a] Department of Mechanical Engineering , Mepco Schlenk Engineering College , Virudhunagar DE, Tamil Nadu, India

[b] Department of Industrial Engineering , Anna University , Chennai, India

Published online: 15 Nov 2008.

To cite this article: R. Rajkumar & P. Shahabudeen (2009) An improved genetic algorithm for the flowshop scheduling problem, International Journal of Production Research, 47:1, 233-249, DOI: 10.1080/00207540701523041

To link to this article: http://dx.doi.org/10.1080/00207540701523041

International Journal of Production Research,
Vol. 47, No. 1, 1 January 2009, 233–249

An improved genetic algorithm for the flowshop scheduling problem

R. RAJKUMAR*† and P. SHAHABUDEEN‡

†Department of Mechanical Engineering, Mepco Schlenk Engineering College, Virudhunagar DE, Tamil Nadu, India
‡Department of Industrial Engineering, Anna University, Chennai, India

(*Revision received May 2007*)

This paper considers the permutation flowshop scheduling problem with the objective of minimizing makespan. Genetic algorithm (GA) is one of the search heuristics used to solve global optimization problems in complex search spaces. It is observed that, the efficiency of GA in solving a flowshop problem can be improved significantly by tailoring the various GA operators to suit the structure of the problem. In this paper, an effective Improved Genetic Algorithm (IGA) for flowshop scheduling, incorporating multi-crossover operators, multi-mutation operators and hypermutation is proposed. Computation results based on some permutation flowshop scheduling benchmark problems (OR-Library) show that the IGA gives a better solution when compared with the earlier reported results.

Keywords: Flowshop scheduling; Makespan; Genetic algorithm

Nomenclature

n	number of jobs
m	number of machines
p_{ij}	processing time of job i on machine j
$C_{\max}$, C^*	makespan, optimal makespan value or lower bound value
k	current generation number
M_k	best makespan at k_{th} generation
p_s	population size
p_c	probability of crossover
p_m	probability of mutation
p_{ro}	probability for roulette wheel selection
p_{to}	probability for tournament selection
p_t	probability for two-point crossover
P_p	probability for PMX crossover
P_{sj}	probability for SJOX crossover
p_l	probability for LOX crossover
p_{3j}	probability for arbitrary three-job change mutation
p_{2j}	probability for arbitrary two-job change mutation

*Corresponding author. Email: rrkumarau@gmail.com

International Journal of Production Research
ISSN 0020–7543 print/ISSN 1366–588X online © 2009 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540701523041

p_{sh}	probability for Shift change mutation
ψ	constant for mutation multiplier
$P(k)$	population at k_{th} generation
X_i, $X_i^{'}$, $X_i^{''}$	chromosome and temporary chromosomes
X^*, X^{**}	best and second best chromosomes
r	real random number between 0 and 1
N_g	maximum generation
G_p	generation limit for inserting new randomly generated chromosomes
G_m	generation limit constant for changing mutation probability
RE	relative error of the result obtained to C^*
BRE	relative error of the best result obtained to C^*
ARE	relative error of the average result obtained to C^*
WRE	relative error of the worst result obtained to C^*

1. Introduction and literature review

Flowshop scheduling problems focus on processing a given set of jobs, where all jobs have to be processed in an identical order on a given number of machines. The flowshop scheduling problem has the additional restriction that the processing of each job has to be continuous, i.e. once the processing of a job begins, there must not be any waiting times between the processing of any consecutive tasks of this job. Such a no-wait constraint is usually due to technological restrictions of the production process. The permutation flowshop scheduling problem (PFSP) with n jobs and m machines, as studied by many researchers, is commonly defined as follows. Each of n jobs is to be sequentially processed on machine $1, \ldots, m$. The processing time p_{ij} of job i on machine j is given. At any time, each machine can process at most one job and each job can be processed on at most one machine. The sequence in which the jobs are to be processed is the same for each machine. The widely used objective is to find a sequence for the jobs so that the makespan or the completion time is minimum (Baker 1974, Nawaz *et al.* 1983). Due to the complexity and NP-hardness of scheduling problems (Garey *et al.* 1976), it is required to develop efficient and effective advanced manufacturing and scheduling technologies and approaches. Flowshop scheduling is a class of widely studied scheduling problems with a strong engineering background, which illustrates at least some of the demands required by a wide range of real-world problems and has earned a reputation for being difficult to solve (Baker 1974, Garey *et al.* 1976). Due to its significance both in theory and applications, it still represents an important issue for demonstrating the efficiency and effectiveness of newly proposed optimization methods, although it has been widely studied so far.

Since Johnson (1954) presented his study of two-machine and special three-machine flowshop problems in 1954, issues concerning the content and scope of flowshop problems have attracted many researchers and practitioners in the last five decades. These approaches for a solution to the problem of finding an optimal or near-optimal sequence have yielded both exact and approximate solution techniques. The exact techniques usually make use of enumeration methods such as branch and bound to obtain an optimal schedule (Ignall and Schrage 1965, Lomnicki 1965). However, in most cases, these methods may require a prohibitive amount of

computation, even for small-sized problems (Wang and Zheng 2003b). On the other hand, approximate solution techniques, although they do not necessarily provide the optimal solution to the problem, are for the most part an efficient and economical way of getting a good solution to the problem. Currently available approximation algorithms can be classified as either constructive algorithms or improvement methods. Up to now the best constructive algorithms have been proposed by Palmer 1965, Campbell *et al*. 1970, Nawaz *et al*. 1983, and Koulamas 1998. Improvement methods include several general approaches, such as simulated annealing (Ogbu and Smith 1990, Ishibuchi *et al*. 1995, Chinyao Low *et al*. 2004), tabu search (Nowicki and Smutnicki 1996, Ben-Daya and Al-Fawzan 1998), and genetic algorithm (Murata and Ishibuchi 1994, Reeves 1995, Ruben Ruiz *et al*. 2003). Rajendran (1994) developed a new heuristic for a flowshop and flow-line-based manufacturing cell with the bi-criteria of minimizing makespan and total flow time of jobs. Murata *et al*. (1996) applied GA to a PFSP and examined the hybridization of the GA with other search algorithms. Wang and Zheng (2003b) uses modified evolutionary programming (MEP) and they showed that their MEP is superior to the simple evolutionary programming and NEH heuristic (see Nawaz *et al*. 1983) method. Wang and Zheng (2003a) incorporate the NEH heuristic into random initialization of a GA and propose effective hybrid heuristic for flowshop scheduling, and point out that their hybrid heuristic is comparable or even superior to that of tabu search, simulated annealing and GA.

During the last decade, GA has been widely applied to many engineering fields, especially in the production scheduling field (Wang and Zheng 2003a). GA-exhibit parallelisms contain certain redundant and historical information of the past solutions and are suited to implementation for large parallel architectures (Goldberg 1989). However, it is not easy to regulate the convergence of GA and to choose suitable parameters and operators, so GA often suffers from premature convergence (Leung *et al*. 1997). A large amount of work has been done to enhance the performance of GA.

In this paper an attempt is made to improve the existing GA procedure to apply to permutation flowshop scheduling to get better results.

2. GA for flowshop scheduling

2.1 *Outline of the Simple Genetic Algorithm (SGA)*

GAs are search algorithms based on the mechanics of natural selection and natural genetics. They work with a population of solutions and attempt to guide the search toward improvement, using a survival of the fittest principle (Goldberg 1989).

In general, GA consists of the following steps:

Step 1: Initialize a population of chromosomes.

Step 2: Evaluate the fitness of each chromosome.

Step 3: Create new chromosomes by applying genetic operators such as crossover and mutation to current chromosomes.

Step 4: Evaluate the fitness of the new population of chromosomes.

Step 5: If the termination condition is satisfied, stop and return the best chromosome; otherwise, go to Step 3.

2.2 *Framework of the Improved Genetic Algorithm (IGA)*

GA is naturally parallel and exhibits implicit parallelism, which does not evaluate and improve a single solution, but analyzes and modifies a set of solutions simultaneously (Goldberg 1989). The ability of a GA to operate on many solutions simultaneously and gather information from all current solutions to direct search reduces the possibility of being trapped in a local optimum. Unfortunately, a GA may lose solutions and substructures because of disruptive effects of genetic operators, and it is not easy to regulate a GA's convergence and hence a pure GA may easily produce premature and poor results (Rudolph 1994, Leung *et al*. 1997). To enhance the performance of genetic searches and to avoid premature convergence, an IGA is proposed with the following changes.

NEH heuristic's sequence (Nawaz *et al*. 1983) is incorporated in the generation of initial population since the NEH sequence can generate suboptimal solution rapidly. The diversity of the initial population can be maintained to a certain extent because the other solutions are still generated randomly.

For reproduction of population, two different selection methods are used each with the given probability.

In the SGA, a single type of crossover operator is applied to the whole population from start to finish, which is not good for retaining useful information and maintaining diversity if the evolution tends to be premature. In IGA, a set of crossover operators are used each with the given probabilities. Multiple crossover operators ensure that the diversity can be enhanced and the search region can be extended.

Similarly a set of mutation operators are applied in IGA. This will enrich the search template so that the exploration and exploitation abilities can be simultaneously enhanced on the basis of the advantage of combining several different search mechanisms.

Computation shows that when the best solution remains unimproved for a certain number of generations in the GA process, the solution quality will be difficult to be improved further even if the generation continues. Therefore, in this IGA the following changes are incorporated to avoid premature convergence:

(a) Elitist strategy: Remove the worst chromosome from the current population and add the best chromosome into that population.
(b) Hypermutation: When the number of generations without improving the best solution is greater than a pre-specified constant (G_m), premature convergence can be assumed then increase the probability of mutation (Hypermutation), and continue the search (Venkata Ranga Neppalli *et al*. 1996). The purpose of increasing the probability is to diversify the population of IGA.
(c) Re-Assign (Insert) new randomly generated population if makespan is not converging for the generation G_p.

Thus, we provide a framework for the IGA. IGA preserves the generality of SGA and can be easily implemented and applied to any kind of optimization problems.

3. IGA for flowshop scheduling

3.1 *Steps of IGA*

Step 0: Given the parameters required, such as population size P_s, crossover probability P_c, mutation probability P_m, etc., set $k=1$ and randomly generate an initial population of size P_s-1 plus NEH heuristic solution. $P(k)=\{X_1(k), X_2(k), \ldots, X_{Ps}(k)\}$.

Step 1: Evaluate Makespan for all the chromosomes in $P(k)$, and set the best two chromosomes X^* and X^{**}, respectively.

Step 2: Set current best makespan as M_k.

Step 3: If $M_k = M_{k-1}$, then set $countmut = countmut + 1$ and $cntrndpop = cntrndpop + 1$. Else set $countmut = 0$ and $cntrndpop = 0$.

Step 4: Set $q=0$.

Step 5: Generate a random number r. If $r \le p_{to}$, select two parents X_1 & X_2 from $P(k)$ by tournament selection, else select the parents by roulette wheel selection.

Step 6: Generate r. If $r \le p_c$, go to step 6.1 else go to step 6.2.

6.1. Perform any one of the following crossover operations.

6.1.1. Generate r. If $r \le p_t$, perform two-point crossover.
6.1.2. If $p_t < r \le (p_t + p_p)$ perform Partially Mapped crossover (PMX).
6.1.3. If $(p_t + p_p) < r \le (p_t + p_p + p_{si})$ perform Similar Job Order crossover (SJOX).
6.1.4. Else perform Linearly Ordered Crossover (LOX).

Let X_1' and X_2' be the offsprings of crossovering parents X_1 and X_2. Go to step 7.

6.2. Let $X_1' = X_1$ and $X_2' = X_2$.

Step 7: Generate r. If $r \le p_m$, go to step 7.1 else go to step 7.2.

7.1. Perform any one of the following mutation operations.

7.1.1. Generate r. If $r \le p_{3j}$ perform arbitrary three-job change mutation.
7.1.2. If $p_{3j} < r \le (p_{3j} + p_{2j})$ perform arbitrary two-job change mutation.
7.1.3. Else perform Shift change mutation.

Mutate for, X_1' to generate chromosome X_1'' and X_2'' to generate chromosome X_2''. Then, put X_1'' and X_2'' into $P(k+1)$ and let $q = q+1$. Go to step 8.

7.2. Let $X_1'' = X_1'$ and $X_2'' = X_2'$. Then, put X_1'' and X_2'' into $P(k+1)$ and let $q = q+1$.

Step 8: If $q < P_s/2$ then go to step 5; otherwise, go to step 9.

Step 9: If $countmut > G_m$, then change the mutation probability $p_m = \psi * p_m$, else no change in p_m value.

Step 10: If $cntrndpop > G_p$, then generate new population of size $0.75 \times p_s$ randomly and replace the current population.

Step 11: Update X^*, X^{**} and M_k in $P(k)$.

Step 12: Adopt Elitist Strategy. Insert the two best chromosomes X^* and X^{**} into the current population by removing two worst chromosomes (having max. makespan).

Step 13: If $k > N_g$ go to step 14. Else set $k = k + 1$ and go to step 3.

Step 14: Output makespan value and the corresponding sequence as the result. Stop.

3.2 *Implementation of the IGA for flowshop scheduling*

3.2.1 Solutions encoding and population initialization. A job-permutation-based encoding scheme has been widely used by many authors for permutation flowshop scheduling. Hence similar encoding scheme is adopted in this work. For example, in a seven-job problem a chromosome could be [4751632]. This represents the sequence where job 4 gets processed first on all the machines followed by job 7 and so on. The permutation has to be feasible, i.e. there cannot be neither missing nor repetition of jobs.

Traditionally, in GA the initial population is generated at random. However Ruben Ruiz *et al*. (2003) have pointed out that a random initialization of the population may not give good results for PFSP. NEH heuristic's sequence is incorporated in the generation of initial population since the NEH heuristic appears to be the best polynomial constructive heuristic in practice, as mentioned in many papers. Remaining chromosomes of the initial population are generated at random.

3.2.2 Fitness evaluation function. In order to mimic the natural process of the survival of the fittest, the fitness evaluation function assigns to each member of the population a value reflecting their relative superiority (or inferiority). Each chromosome has an evaluation criterion based on the objective function. Since IGA is used for maximization problems, a minimization problem can be suitably converted into a maximization problem using a fitness function. The fitness function is:

$$F = \frac{1}{C_{\max}} \tag{1}$$

Where $C_{\max}$ is the makespan, which has to be minimized.

3.2.3 Selection scheme. Selection scheme specifies the methodology employed to select the chromosome from the current population for regeneration. There are

various selection operators available that can be used to select the parents. In IGA, two classical selection schemes, namely roulette wheel selection and tournament selection (Goldberg 1989) are employed.

In roulette wheel selection, parents are selected according to their fitness value. The better the fitness, the more chances to be selected. The following procedure is used for selection.

Chromosome X_i is selected if

$$\frac{\sum_{j=1}^{i-1} f(X_j)}{\sum_{j=1}^{p_s} f(X_j)} < r \leq \frac{\sum_{j=1}^{i} f(X_j)}{\sum_{j=1}^{p_s} f(X_j)}$$

where $f(X_i)$ is fitness value of chromosome and r is the random number between 0 and 1.

In Tournament Selection, predetermined numbers of chromosomes are randomly selected from the population and the chromosome with the best fitness value is considered to be regenerated. Here selection is based on a competition within a subset of the population.

3.2.4 Crossover. Many different general and specific crossover operators have been proposed for the PFSP in the literature. Since the sequence is a permutation of elements there should be neither repeated nor missing elements to maintain feasibility. In this work, the following types of crossover operators are used, which are described as follows. A probability is attached to each type of crossover and each time any one type is selected using Monte-Carlo simulation.

1. Two point crossover

Step 1: Two points are randomly selected for dividing the parents (figure 1).

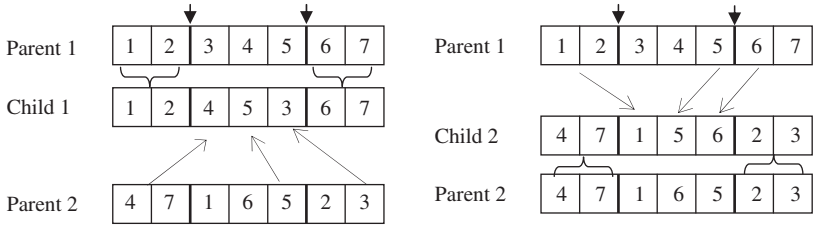

Figure 1. Two point crossover.

Step 2: The jobs outside the selected two points are directly inherited from the parent to the child.

Step 3: The remaining elements in the child are filled by scanning the other parent from left to right and entering the elements not already present.

2. PMX (partially mapped crossover)

Step 1: Two points are randomly selected for dividing the parents. The section of the parents between these two points is called the mapping section (figure 2).

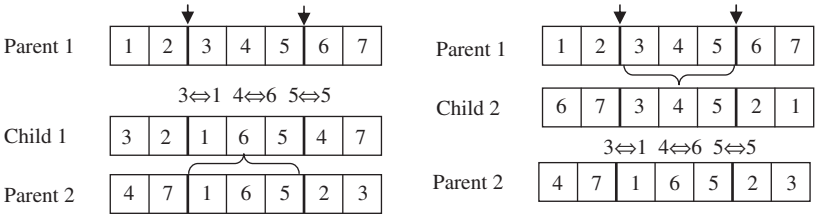

Figure 2. Partially mapped crossover.

Step 2: Exchanges the mapping section of the parent to the offspring. i.e. mapping section of the first parent is copied into the second offspring and so on.

Step 3: Define one-to-one mapping between genes of mapping section of the two parents. For the above example mapping is as follows (figure 3).

3⟷1 4⟷6 5⟷5

Figure 3. One-to-one mapping.

Step 4: Offspring is filled up by copying the genes from direct parent by scanning from left to right. In case a gene is already present in the offspring, it is replaced according to the mapping.

3. SJOX (similar job order crossover)

SJOX crossover is based on the idea of identifying and maintaining building blocks in the offspring (Ruben Ruiz *et al*. 2003). In this way similar blocks or occurrences of jobs in both parents are passed over to child unaltered. If there are no similar blocks in the parents the crossover operator will behave like the single-point order crossover. The SJOX crossover operator can be explained as follows:

Step 1: Both parents are examined on a position-by-position basis. Identical jobs at the same positions are copied over to both the offspring (figure 4(a)).

Step 2: The offspring directly inherits all jobs from the corresponding parents up to a randomly chosen cut point. That is, Child1 inherits directly from Parent1 and Child 2 from Parent 2 (figure 4(b)).

Step 3: Missing elements at each offspring are copied in the relative order of the other parent (figure 4(c)).

4. Linear order crossover (LOX)

LOX, initially suggested by Falkenauer and Bouffouix (1991), works as follows:

Step 1: Select a subsequence of operations from one parent at random.

Step 2: Produce a proto-offspring by copying the subsection sequence into the corresponding positions of it.

Step 3: Delete the operations which are already in the subsequence from the second parent. The resulted sequence of operations contains the operations that the proto-offspring needs.

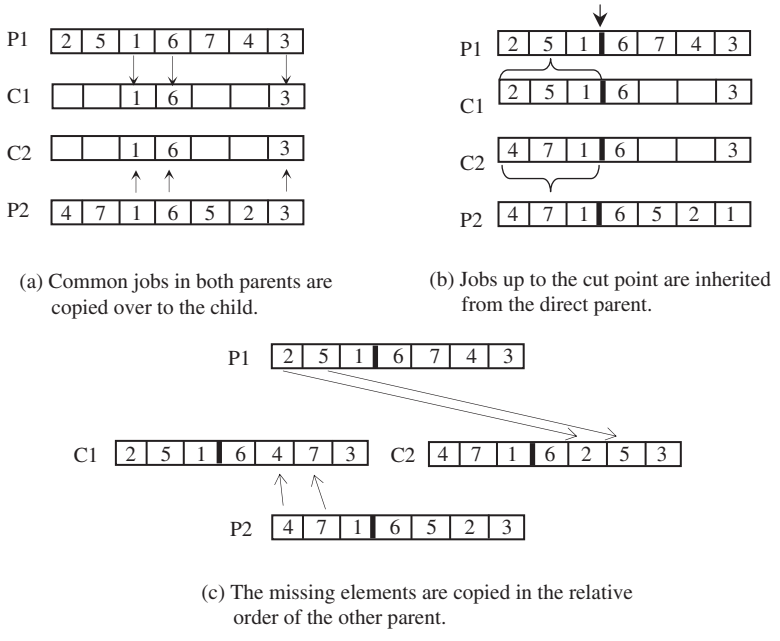

Figure 4. Similar Job Order Crossover (SJOX).

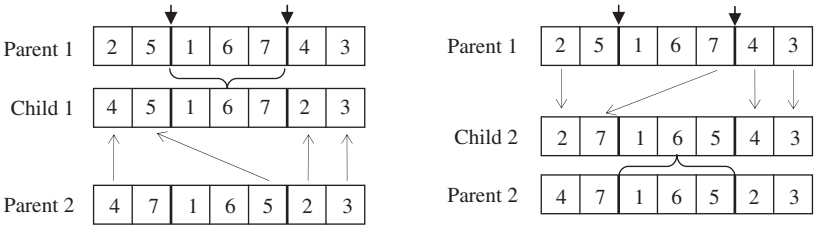

Figure 5. LOX Crossover operator.

Step 4: Place the operations into the unfixed positions of the proto-offspring from left to right according to the order of the sequence to produce an offspring.

This procedure is illustrated in figure 5. Crossover LOX tries to preserve as much as possible both the relative positions between genes and the absolute positions relative to the extremities of parents.

3.2.5 Mutation. Mutation generates an offspring solution by randomly modifying the parent's feature. It helps to preserve a reasonable level of population diversity, and provides a mechanism to escape from local optima. For each child obtained from crossover, the mutation operator is applied independently with a probability p_m. In this work, three types of mutation operators are used, which are described as

follows. A probability is attached to each type of mutation and each time any one type is selected using Monte-Carlo simulation.

1. Arbitrary three-job change

The three jobs are randomly selected, and replaced at random locations amongst the selected jobs (figure 6).

2. Arbitrary two-job change

Two jobs are selected at random and their positions are interchanged as shown below (figure 7).

3. Shift change

In this type of mutation, a job at a random position is removed and inserted at another random position as shown below (figure 8).

3.2.6 Restart scheme. Finally, to avoid premature convergence and to escape local optimum, two counters called *countmut* and *cntrndpop* are introduced as explained below.

(I) (a) Let M_k = Current generation best makespan and M_{k-1} = Previous generation best makespan.
If $M_k = M_{k-1}$ then, set $countmut = countmut + 1$ and $cntrndpop = cntrndpop + 1$. Else set $countmut = 0$ and $cntrndpop = 0$.

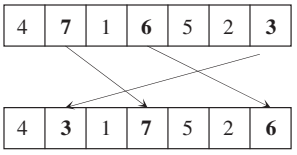

Figure 6. Arbitrary 3-job change mutation.

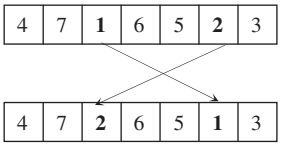

Figure 7. Arbitrary 2-job change mutation.

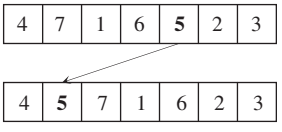

Figure 8. Shift change mutation.

(b) If there is no improvement in the best solution so far for the pre-specified generations G_m, then increase the probability of mutation and continue the search. The purpose of increasing the probability is to diversify the population of IGA i.e. dynamically change the mutation rate if makespan not converging for the generation G_m i.e. if $countmut > G_m$, then $p_m = \psi^* p_m$.

(II) Re-Assign (i.e. replace) new randomly generated chromosomes into population if makespan not converging for the generation G_p. i.e. if $cntrndpop > G_p$, then, replace 75% of current population by randomly generated new chromosomes.

3.2.7 Elitist strategy. In the process of crossover, mutation and restart scheme, there is a chance of losing the current best chromosome. Elitism refers to a method that copies the best chromosome (or few best chromosomes) to the new population. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best found solution. In IGA, two worst chromosomes from the current population are replaced by the current best chromosomes.

3.2.8 Termination criteria. Theoretically, it requires a long enough Markov chain to guarantee the convergence of GA (Rudolph 1994), which may lead to huge computations. Now that there is no practicable rule to set suitable stopping condition and it is also impossible for GA to evolve with too long time in real application, the usual way is to set a limit to a number of generations.

3.3 *Parameter settings for IGA*

Parameters used in the IGA are shown in table 1. Among these parameters, the quality of solution is mostly influenced by probability of crossover, probability of mutation and the number of generations.

4. Computational results and comparisons

4.1 *Benchmarks selected*

In this work, 29 problems that were contributed to the OR-Library by D. C. Mattfeld and R. J. M. Vaessens are used. The first eight problems called car1, car2, ..., car8, respectively, were by Carlier (1978). The other 21 problems called rec01, rec03, ..., rec41, respectively, were by Reeves (1995), who used them to compare the performances of SA, GA and neighbourhood search and found these problems to be particularly difficult. All these problems can be downloaded from http://mscmga.ms.ic.ac.uk.

4.2 *Result and analysis*

Based on the implementation discussed in section 3, the SGA and IGA are coded in C and each instance is replicated 20 times with different random number seeds.

Table 1. Parameters in IGA.

Parameter	Value
Population size p_s	70
Probability of crossover p_c	0.9
Probability of mutation p_m	0.4
Probability for roulette wheel selection p_{ro}	0.2
Probability for tournament selection p_{to}	0.8
Probability for two-point crossover p_t	0.7
Probability for PMX crossover p_p	0.125
Probability for SJOX crossover p_{sj}	0.05
Probability for LOX crossover p_l	0.125
Probability for arbitrary three-job change mutation p_{3j}	0.05
Probability for arbitrary two-job change mutation p_{2j}	0.15
Probability for shift change mutation p_{sh}	0.8
A constant Mutation multiplier ψ	1.2
Number of generations N_g	10000
Generation limit for population re-assignment G_p	2500
Generation limit for changing mutation probability G_m	1500

The performance of the IGA for the benchmark problems is compared with the result reported in the literature by Wang and Zheng (2003a, b) and SGA with the following notations.

IGA	Proposed Improved Genetic Algorithm
SGA	Simple Genetic Algorithm
WZ-MEP	Modified Evolutionary Programming of Wang and Zheng (2003b)
WZ-HH	Hybrid Heuristic of Wang and Zheng (2003a)
NEH	Heuristic of Nawaz *et al.* (1983)

4.2.1 Percentage improvement. A comparison of makespan obtained using IGA with WZ-MEP, popular NEH method and SGA are summarized in table 2, where the last three columns are the percentage improvement of IGA over the SGA, WZ-MEP and NEH, respectively. The percentage improvement is calculated as follows:

$$\text{Percentage Improvement} = \frac{(C_X - C_{IGA})}{C_X} \times 100 \tag{2}$$

where

C_X = Makespan reported by SGA, NEH and WZ-MEP
C_{IGA} = Makespan obtained using IGA

From table 2 and figure 9, the IGA clearly outperforms the earlier reported literature results in WZ-MEP (Wang and Zheng 2003b), NEH heuristic (Nawaz *et al.* 1983) and SGA for nearly all of the benchmarks. It can be observed that for all the Carlier problems quite consistent results are obtained since the problem size is very

Table 2. Percentage improvement in makespan using IGA over the earlier literature results.

Problem	n	m	C^*	Proposed IGA	SGA	WZ-MEP	NEH Heuristic	Imp.% over SGA	Imp.% over WZ-MEP	Imp.% over NEH
Car1	11	5	7038	7038.0	7038.0	7038.0	7038	0.000	0.000	0.000
Car2	13	4	7166	7166.0	7166.0	7166.0	7376	0.000	0.000	2.715
Car3	12	5	7312	7312.0	7312.0	7312.0	7443	0.000	0.000	1.760
Car4	14	4	8003	8003.0	8003.0	8003.0	8034	0.000	0.000	0.386
Car5	10	6	7720	7720.0	7720.0	7720.0	8047	0.000	0.000	4.064
Car6	8	9	8505	8505.0	8505.0	8508.4	8813	0.000	0.040	3.495
Car7	7	7	6590	6590.0	6590.0	6590.0	7008	0.000	0.000	5.965
Car8	8	8	8366	8366.0	8366.0	8366.0	8457	0.000	0.000	1.076
Rec01	20	5	1247	1248.9	1248.9	1248.7	1352	0.000	−0.016	7.626
Rec03	20	5	1109	1109.0	1109.1	1110.0	1182	0.009	0.090	6.176
Rec05	20	5	1242	1244.85	1245.00	1245.6	1302	0.012	0.060	4.389
Rec07	20	10	1566	1571.40	1572.30	1576.8	1650	0.057	0.342	4.764
Rec09	20	10	1537	1538.15	1543.75	1546.8	1641	0.363	0.559	6.268
Rec11	20	10	1431	1437.45	1437.50	1446.7	1549	0.003	0.639	7.201
Rec13	20	15	1930	1946.75	1956.80	1962.4	2077	0.514	0.797	6.271
Rec15	20	15	1950	1961.35	1969.20	1971.8	2046	0.399	0.530	4.137
Rec17	20	15	1902	1932.55	1954.80	1946.1	2044	1.138	0.696	5.453
Rec19	30	10	2093	2115.85	2128.50	2120.6	2232	0.594	0.224	5.204
Rec21	30	10	2017	2047.70	2049.85	2048.7	2109	0.105	0.049	2.907
Rec23	30	10	2011	2025.85	2041.50	2028.5	2212	0.767	0.131	8.415
Rec25	30	15	2513	2570.35	2599.20	2576.8	2688	1.110	0.250	4.377
Rec27	30	15	2373	2414.70	2428.15	2416.4	2575	0.554	0.070	6.225
Rec29	30	15	2287	2336.25	2357.60	2348.7	2411	0.906	0.530	3.100
Rec31	50	10	3045	3082.45	3129.60	3085.8	3358	1.507	0.109	8.206
Rec33	50	10	3114	3138.35	3143.65	3141.1	3262	0.169	0.088	3.791
Rec35	50	10	3277	3277.00	3279.95	3277.0	3441	0.090	0.000	4.766
Rec37	75	20	4890	5127.95	5296.61	5129.6	5337	3.212	0.032	3.917
Rec39	75	20	5043	5181.80	5351.33	5183.7	5479	3.207	0.037	5.424
Rec41	75	20	4910	5110.95	5277.27	5151.6	5435	3.313	0.789	5.962

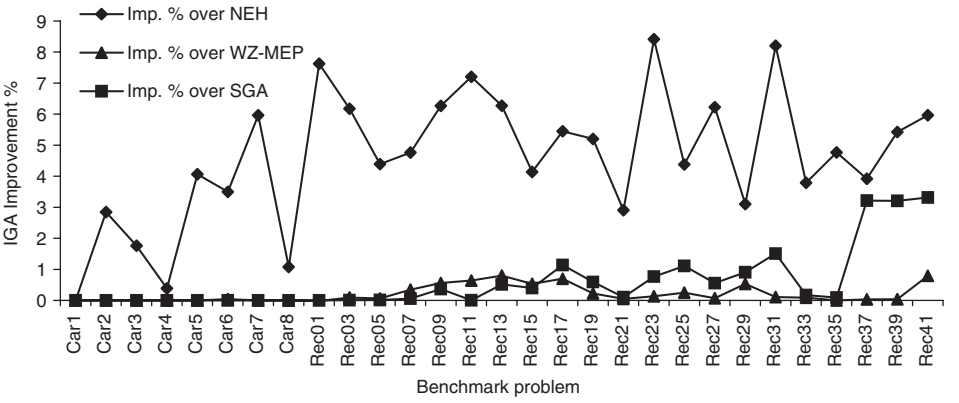

Figure 9. Improvement percentage of IGA with NEH, WZ-MEP and SGA.

small when compared with Reeves. Moreover Carlier results are already attained the lower bound makespan optimal values (C^*). Also it can be observed that for the Reeves problems, the IGA provides 5.46% improvement with respect to NEH and 0.29% with respect to WZ-MEP. Meanwhile, the overall improvement percentage for both Carlier and Reeves problems of IGA are 4.63% and 0.21% with respect to NEH and WZ-MEP. However, the overall improvement percentage for both Carlier and Reeves problems of IGA is 0.62% with respect to SGA.

The average difference of makespan value between IGA and earlier reported results of WZ-MEP and NEH heuristic is 5.08 and 154.43, respectively, for both Carlier and Reeves problems. In addition, the average difference of makespan value between IGA and SGA is found as 24.60. Reeves problems are of a considerably large size and therefore the solution space is also large. Hence, in this work considerable reduction of makespan time is achieved from IGA.

Table 2 also indicates that for small-size problems (car1 to car 8) the performances of SGA, WZ-MEP and IGA are quite close; they are also superior to the NEH heuristics. With the number of job increases percentage improvement becomes larger, i.e. the performance of IGA is quite superior to the other two methods for the large-size scheduling problems.

4.2.2 Relative error calculation. In order to evaluate the performance of the IGA, another comparison measure is also adopted i.e. the relative error against the lower bound. The various relative errors are calculated by using the following formulae.

$$\text{Relative error to } C^*,\ \text{RE} = \frac{(C_{IGA} - C^*)}{C^*} \times 100\% \tag{3}$$

$$\text{Best relative error, BRE} = \frac{(C_{IGA\_\text{best}} - C^*)}{C^*} \times 100\% \tag{4}$$

$$\text{Average relative error, ARE} = \frac{\left(C_{IGA\_avg} - C^*\right)}{C^*} \times 100\% \tag{5}$$

Table 3. Comparison of relative errors obtained using IGA with earlier literature results.

			NEH	WZ-HH			SGA			Proposed IGA		
Problem	n	m	RE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE
Car1	11	5	0	0	0	0	0	0	0	0	0	0
Car2	13	4	2.931	0	0	0	0	0	0	0	0	0
Car3	12	5	1.792	0	0	0	0	0	0	0	0	0
Car4	14	4	0.387	0	0	0	0	0	0	0	0	0
Car5	10	6	4.236	0	0	0	0	0	0	0	0	0
Car6	8	9	3.621	0	0.04	0.76	0	0	0	0	0	0
Car7	7	7	6.343	0	0	0	0	0	0	0	0	0
Car8	8	8	1.088	0	0	0	0	0	0	0	0	0
Rec01	20	5	8.42	0	0.14	0.16	0	0.15	0.16	0	0.15	0.16
Rec03	20	5	6.583	0	0.09	0.18	0	0.01	0.18	0	0	0
Rec05	20	5	4.831	0	0.29	1.13	0.24	0.24	0.24	0	0.23	0.24
Rec07	20	10	5.364	0	0.69	1.15	0	0.40	1.15	0	0.34	1.15
Rec09	20	10	6.766	0	0.64	2.41	0	0.44	2.41	0	0.07	0.59
Rec11	20	10	8.246	0	1.10	2.59	0	0.45	2.24	0	0.45	1.47
Rec13	20	15	7.617	0.36	1.68	3.06	0.52	1.39	3.47	0.41	0.87	1.40
Rec15	20	15	4.923	0.56	1.12	2.00	0.10	0.98	1.64	0	0.58	1.64
Rec17	20	15	7.466	0.95	2.32	3.73	1.16	2.78	4.73	0	1.61	2.68
Rec19	30	10	6.641	0.62	1.32	2.25	0.91	1.70	2.58	0.67	1.09	2.29
Rec21	30	10	4.561	1.44	1.57	1.64	1.44	1.63	2.03	0.74	1.52	1.64
Rec23	30	10	9.995	0.40	0.87	1.69	0.50	1.52	2.78	0.35	0.74	1.09
Rec25	30	15	6.964	1.27	2.54	3.98	1.91	3.43	5.93	1.03	2.28	3.34
Rec27	30	15	8.512	1.10	1.83	4.00	1.05	2.32	3.58	0.97	1.76	3.33
Rec29	30	15	5.422	1.40	2.70	4.20	1.09	3.09	5.33	1.01	2.15	3.28
Rec31	50	10	10.279	0.43	1.34	2.50	2.27	2.78	4.37	0.49	1.23	2.50
Rec33	50	10	4.753	0	0.78	0.83	0.83	0.95	2.02	0.22	0.78	0.83
Rec35	50	10	5.005	0	0	0	0	0.09	0.49	0	0	0
Rec37	75	20	9.141	3.75	4.90	6.18	6.67	8.35	10.00	4.09	4.87	5.34
Rec39	75	20	8.646	2.20	2.79	4.48	4.72	6.16	7.81	2.24	2.75	3.73
Rec41	75	20	10.692	3.64	4.92	5.91	6.05	7.66	8.94	3.44	4.09	5.34

$$\text{Worst relative error, WRE} = \frac{(C_{IGA\_worst} - C^*)}{C^*} \times 100\% \tag{6}$$

where

C^* = Lower bound Makespan (Wang and Zheng 2003b)
C_{IGA} = Makespan obtained using IGA
C_{IGA_best} = Lowest makespan obtained
C_{IGA_avg} = Average makespan obtained
C_{IGA_worst} = Highest makespan obtained

Relative error values are summarized in table 3. It can be observed that:

The average error obtained by IGA is not more than 5% of best-known results.

The error values obtained by IGA are less than corresponding errors reported by NEH (Nawaz *et al.* 1983) and WZ-HH (Wang and Zheng 2003a), except in a few cases.

Even the worst relative error obtained by IGA is much less than the error reported by NEH.

The error values obtained by IGA are less than corresponding errors reported by SGA for all the problems.

The moderate increase in computational time should not discourage practitioners from considering the method because it is possible to carry out the computations using high-speed computers (Wang and Zheng 2003a). In this work also, computing time is not taken into account for comparison, owing to possible variations in configurations of hardware and software used.

Based on the percentage improvement analysis and the relative error analysis, it can be concluded that the proposed IGA has produced results better than existing heuristics.

5. Conclusions

In this work, the framework of the IGA has been proposed and implemented for the flowshop sequencing problems with the objective of minimizing makespan. To have a good seed, NEH heuristic solution is incorporated into the random initialization of IGA. To avoid premature convergence of solution, multi-crossover, hypermutation and re-assignment strategy are incorporated. Numerical computation based on benchmarks demonstrated the effectiveness of the proposed method. The results are compared with earlier reported results and were found to produce better performance. The IGA framework is general enough to be applied to other optimization problems also. This IGA can also been extended to multi-objective scheduling problems.

References

Baker, K.R., *Introduction to Sequencing and Scheduling*, 1974 (Wiley: New York).

Ben-Daya, M. and Al-Fawzan, M., A tabu search approach for the flow shop scheduling problem. *European J. Operat. Resarch.*, 1998, **109**, 88–95.

Campbell, H.G., Dudek, R.A. and Smith, M.L., A heuristic algorithm for the n-job, m-machine sequencing problem. *Manag. Sci.*, 1970, **16/B**, 630–637.

Carlier, J., Ordonnancements a contraintes disjonctives. *RAIRO Recherche Operationelle/ Operat. Resear.*, 1978, **12**, 333–351.

Chinyao, L., Jinn-Yi, Y. and Kai-I, H., A robust simulated annealing heuristic for flow shop scheduling problems. *The Inter. J. Advanc. Manufact. Tech.*, 2004, **23**, 762–767.

Falkenauer, E. and Bouffouix, S., A genetic algorithm for job shop. in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991, 824–829.

Garey, E.L., Johnson, D.S. and Sethi, R., The complexity of flowshop and job-shop scheduling. *Math. Operat. Resear.*, 1976, **1**, 117–129.

Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, 1989 (Addison Wesley: Reading, MA).

Ignall, E. and Schrage, L.E., Application of branch-and-bound techniques to some flow shop problems. *Operat. Resear.*, 1965, **13**, 400–412.

Ishibuchi, H., Misaki, S. and Tanaka, H., Modified simulated annealing algorithms for the flow shop sequencing problem. *European J. Operat. Resarch.*, 1995, **81**, 388–398.

Johnson, S.M., Optimal two and three-stage production schedules with setup times included. *Nav Res Log*, 1954, **1**, 61–68.

Koulamas, C., A new constructive heuristic for the flow shop scheduling problem. *European J. Operat. Resear.*, 1998, **105**, 66–71.

Leung, Y., Gao, Y. and Xu, Z.B., Degree of population diversity – a perspective on premature convergence in genetic algorithms and its Markov-chain analysis. *IEEE Trans. Neural Networks*, 1997, **8**, 1165–1176.

Lomnicki, Z., A branch-and-bound for the exact solution of the three machine scheduling problem. *Operat. Resear.*, 1965, **16**, 89–107.

Murata, T. and Ishibuchi, H., Performance evaluation of genetic algorithms for flowshop scheduling problems. in *IEEE world congress on computational intelligence, proceedings of the first IEEE conference*, 1994, **2**, 812–817.

Murata, T., Ishibuchi, H. and Tanaka, H., Algorithms for flowshop scheduling problems. *Comput. Indust. Eng.*, 1996, **30**, 1061–1701.

Nawaz, M., Enscore, E. and Ham, I., A heuristic algorithm for the m- machine, n- machine flow shop sequencing problem. *Omega*, 1983, **11**, 91–95.

Nowicki, E. and Smutnicki, C., A fast tabu search algorithm for the permutation flow-shop problem. *European J. Operat. Resarch.*, 1996, **91**, 160–175.

Ogbu, F.A. and Smith, D.K., The applications of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem. *Comput. Operat. Resear.*, 1990, **17**, 243–253.

Palmer, D.S., Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining a near optimum. *Operat. Resear.*, 1965, **16**, 101–107.

Rajendran, C., A heuristic for scheduling on flow shop and flow line based manufacturing cell with multicriteria. *Inter. J. Prod. Resear.*, 1994, **32**, 2541–2558.

Reeves, C.R., A genetic algorithm for flowshop sequencing. *Comput. Operat. Resear.*, 1995, **22**, 5–13.

Ruben, R., Concepcion, M. and Javier, A., New genetic algorithms for the permutation flowshop scheduling problem, in *Proceedings of the Fifth Metaheuristics International Conference*, Kyoto, Japan, August 25–28, 2003, pp. 63.1–63.8.

Rudolph, G., Convergence properties of canonical genetic algorithms. *IEEE Transa. Neural Networks*, 1994, **5**, 96–101.

Venkata Ranga Neppalli, C.-L.C. and Jatinder, N.D. Gupta, Genetic algorithms for the two-stage bicriteria flowshop problem. *European J. Operat. Resarch.*, 1996, 95, 356–373.

Wang, L. and Zheng, D.Z., An effective hybrid heuristic for flow shop scheduling. *The Inter. J. Advanc. Manufact. Tech.*, 2003a, **21**, 38–44.

Wang, L. and Zheng, D.Z., A modified evolutionary programming for flow shop scheduling. *The Inter. J. Advanc. Manufact. Tech.*, 2003b, **22**, 522–527.