Int J Adv Manuf Technol (2006) 28: 532–540
DOI 10.1007/s00170-004-2403-1

ORIGINAL ARTICLE

K.S. Amirthagadeswaran · V.P. Arunachalam

Improved solutions for job shop scheduling problems through genetic algorithm with a different method of schedule deduction

Received: 1 July 2004 / Accepted: 6 September 2004 / Published online: 1 June 2005
© Springer-Verlag London Limited 2005

Abstract Job shop scheduling problems are one of the challenging combinatorial problems that have drawn the attention of researchers for the last three decades. It is observed that genetic algorithm (GA) is gaining more importance over the past several years. An attempt has been made through GA to solve job shop scheduling problems with job-based, operation-based, and proposed methods of representation and schedule deduction with the make-span objective. Computational experiments of this attempt have yielded better solutions coupled with appreciable reduction in computer processing time. A set of selected benchmark problems have been used with the proposed heuristic for validation and the results show the better performance of the proposed method of representation of jobs and schedule deduction.

Keywords Edge recombination operator · Genetic algorithm · Job shop scheduling · Schedule deduction

1 Introduction

1.1 Job shop problems

Job shop problems have a set of *n* jobs to be processed on a set of *m* machines. Each job has a set of operations to be performed on each machine in a particular order. Each machine can process at most one operation at a time. Job shop scheduling deals with the allocation of jobs to various machines with the objective of minimizing the make-span, the time to complete all jobs [1], or minimizing the tardiness (not meeting the due date) in jobs or any other required objectives.

K.S. Amirthagadeswaran (✉)
Department of Mechanical Engineering,
Government College of Technology,
Coimbatore, India
E-mail: ksagp@yahoo.co.in

V.P. Arunachalam
Department of Mechanical Engineering,
Government College of Technology,
Coimbatore, India

There are several constraints on jobs and machines [2]. They are:

(1) A job does not visit the same machines twice;
(2) There are no precedence constraints among operations of different jobs;
(3) Operation cannot be interrupted;
(4) Each machine can process only one job at a time;
(5) Neither release times nor due dates are specified.

Job shop problems occur mainly in industries where each customer has specified characters and order sizes are relatively small. In general, an infinite number of feasible schedules are possible for any job shop problem, as one can insert any arbitrary amount of idle time at any machine between adjacent pairs of operations [3]. Job shop scheduling problem (JSP) is well known as one of the most difficult NP-hard combinatorial problems.

With the advent of high speed computing systems, soft computing techniques like GA are being applied to job shop scheduling problems.

1.2 Genetic algorithms

Genetic algorithms (GA) are an attractive class of computational models that are based on the mechanics of natural selection and natural genetics. GA was introduced by Holland in 1975, and has been extensively researched and applied to many combinatorial optimization problems [4].

A GA comprises a set of individual elements (the population) and a set of biologically inspired operators defined over the population itself. According to the evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring, thus transmitting their biological heredity to new generations. In computing terms, a GA maps a problem onto a set of (typically binary or other coding methods) strings, each string representing a potential solution. The GA then manipulates the most promising strings in its search for improved solutions.

A GA operates through a simple cycle of stages:

1. Creation of "population" of strings,
2. Evaluation of each string,

3. Selection of best strings, and
4. Genetic manipulation to create the new population of strings.

At the first stage, an initial population of potential solutions is created as a starting point for each search. Each element of the population is encoded onto a string (the chromosome) to be manipulated by the genetic operators. In the next stage, the performance (or fitness) of each individual is evaluated with respect to the constraints imposed by the problem. Based on each individual's fitness, a selection mechanism chooses "mates" for the genetic manipulation process. The selection policy is ultimately responsible for assuring survival of the best-fitted individuals. The combined evaluation and selection process is called reproduction.

The manipulation process uses genetic operators to produce a new population of individuals (offspring) by manipulating the "genetic information" referred as genes, possessed by members (parents) of the current population. It comprises two operations: crossover and mutation. Crossover recombines a population's genetic material. Mutation operation introduces new genetic structures in the population by randomly modifying some of the building blocks, helping the search algorithm escape from local minima traps. Since the modification is not related to any previous genetic structure of the population, it creates different structures representing other sections of the search space.

The creation, evaluation, selection and manipulation cycle repeats until a satisfactory solution to the problem is found or some other termination criteria are met.

2 Brief survey of existing representation methods

2.1 Representation methods

This paper deals with a representation and deduction method using a heuristic technique based on GA to solve job shop scheduling problems. During the last few years, the following nine representations for the job shop scheduling problem have been proposed [5]:

- Operation-based representation
- Job-based representation
- Preference list-based representation
- Job pair relation-based representation
- Priority rule-based representation
- Disjunctive graph-based representation
- Completion time-based representation
- Machine-based representation
- Random keys representation.

These representations can be classified into the following two basic encoding approaches:

- Direct approach
- Indirect approach.

In direct approach, a schedule (the solution of JSP) is encoded into a chromosome and GAs are used to evolve those chromosomes to find out a better schedule. The representations, such as operation-based representation, job-based representation, job pair relation-based representation, completion time-based representation, and random keys representation belong to this class.

In indirect approach, such as priority rule-based representation, a sequence of dispatching rules for job assignment, but not a schedule, is encoded into a chromosome and GAs are used to evolve those chromosomes to find out a better sequence of dispatching rules. A schedule is then constructed with the sequence of dispatching rules. Preference list-based representation, priority rule-based representation, disjunctive graph-based representation, and machine-based representation belong to this class.

Schedule deduction for job-based representation and operation-based representation approaches have been compared with the proposed method of representation and schedule deduction.

2.2 Problem definition

Let us consider a 4 jobs, 3 machines problem [6].

Table 1. Time matrix of the problem

Jobs	Operations		
	1	2	3
j1	2	3	4
j2	4	4	1
j3	2	2	3
j4	3	3	1

Table 2. Routing matrix of the problem

Jobs	Operations		
	1	2	3
j1	1	2	3
j2	3	2	1
j3	2	3	1
j4	1	3	2

3 Schedule deduction using various methods

3.1 Job-based representation and deduction of schedule

This representation consists of a list of n jobs and a schedule is constructed according to the sequence of jobs. For a given sequence of jobs, all operations of the first job in the list are scheduled first, and then the operations of the second job in the list are considered. The first operation of the job under treatment is allocated in the best available processing time for the corresponding machine the operation requires; and then the second operation is allocated, etc., until all operations of the job are scheduled. The process is repeated with each of the jobs in the list considered in the appropriate sequence. Any permutation of jobs corresponds to a feasible schedule.

Considering the problem given in Tables 1 and 2 and supposing a chromosome is given as [2 4 3 1], the first job to be processed is job j2. The operation precedence constraint for j2 is [m3 m2 m1] and the corresponding processing time for each machine is [4 4 1]. First, job j2 is scheduled as shown in Fig. 1a. Then, job j4 is processed. Its operation precedence among the machines is [m1 m3 m2] and the corresponding processing time for each machine is [3 3 1]. Each of its operations is scheduled in the best available processing time for the corresponding machine, as shown in Fig. 1b. Then, job j3 is processed. Its operation precedence among the machines is [m2 m3 m1] and the corresponding processing time for each machine is [2 2 3]. Each of its operations is scheduled in the best available processing time for the corresponding machine, as shown in Fig. 1c. Finally, job j1 is scheduled, as shown in Fig. 1d.

The Gantt chart of the schedule for the sequence 2 4 3 1 under job-based representation shows the make-span as 16.

3.2 Operation-based representation and deduction of schedule

This representation encodes a schedule as a sequence of operations and each gene stands for one operation. As it is observed from literature that the minimum make-span is found in operation-based representation, this representation has also been followed in solving the problem. In this method, strings (chromosomes) are coded as a sequence of numbers (genes) with each gene representing one of the operations of the jobs involved. The specific operation represented by the genes are interpreted according to the order of the genes in the chromosome [5]. Each of the *n* (jobs) differently named (coded) genes will appear *m* (machines) times, spread over the entire chromosome.

For the above problem, a coded string comprising $4 * 3$ numbers is generated randomly. The string, say, 1 1 2 4 3 1 4 2 3 2 4 3 is a typical chromosome, where 1 stands for job 1, 2 for job 2, 3 for job 3, and 4 for job 4. Because each job has four operations, each job occurs exactly four times in the chromosome. There are three 3s in the chromosome, representing the three operations of job 3. The first 3 corresponds to the first operation of job 3, which will be processed on machine 2, the second 3 corresponds to the second operation of job 3, which will be processed on machine 3, and the third 3 corresponds to the third operation of job 3, which will be processed on machine 1.

The first operation of job 1 to be performed on machine 1 is scheduled, followed by the second operation of job 1 to be processed in machine 2. Then, the first operation of job 2 to be processed in machine 3 is scheduled, and so on. The allocation of operations of jobs to various machines is shown in Fig. 2.

The Gantt chart for the chromosome (1 1 2 4 3 1 4 2 3 2 4 3) under operation-based representation and deduction of schedule shows the make-span as 17.

3.3 Proposed representation and deduction of schedule

This representation consists of a list of *n* jobs and a schedule is constructed according to the sequence of jobs in the chromosome. For a given sequence of jobs, the first operation of all the jobs, as in the sequence, is scheduled first. Then, the second operation of all the jobs, as in the sequence, is considered. The first operation of the first job in the sequence is allocated in the best available processing time for the corresponding machine the operation requires, and then the first operation of the second job in the sequence is allocated, etc., until the first operation of all the jobs are scheduled. This process is repeated with second, third, and other operations of the jobs until all the jobs in the sequence are scheduled. Considering the same problem and supposing a chromosome is given as [**2 4 3 1**], the first

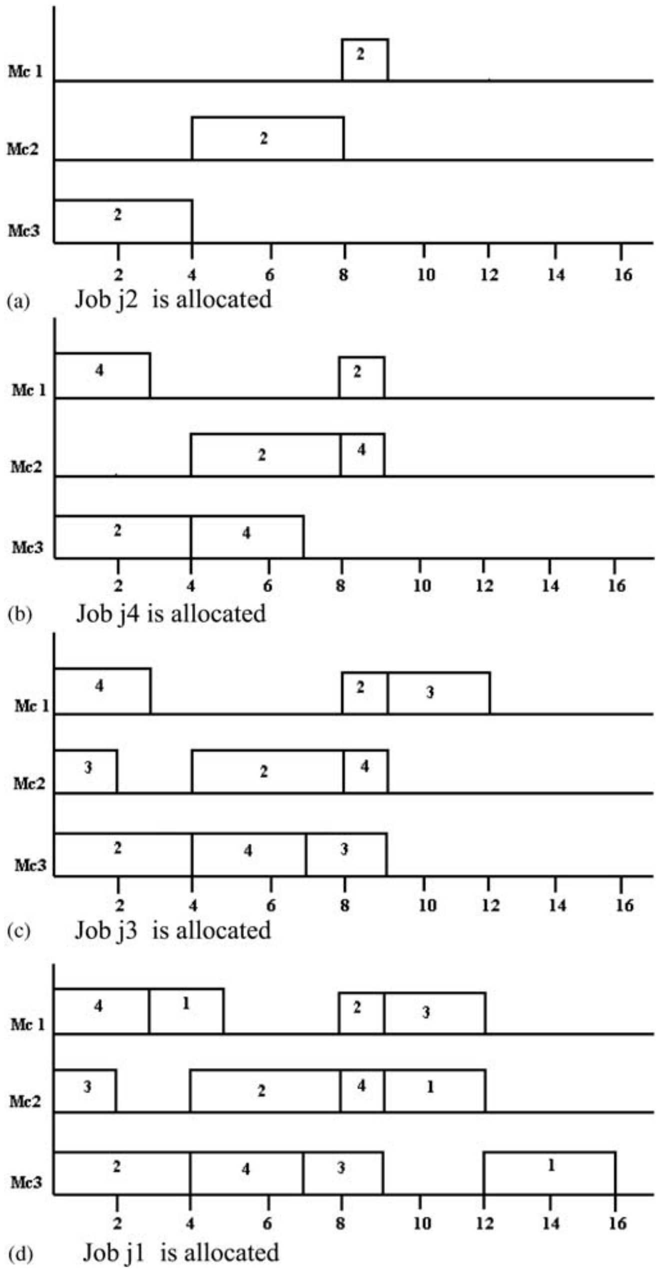
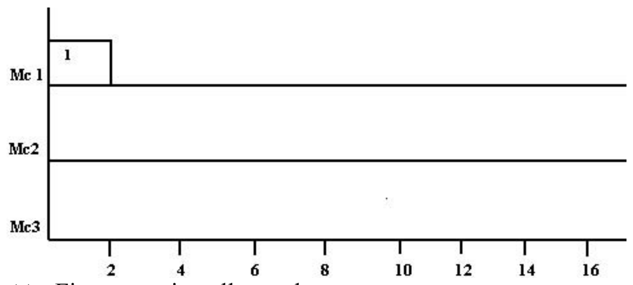
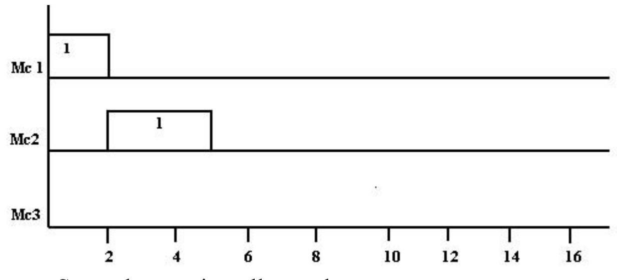


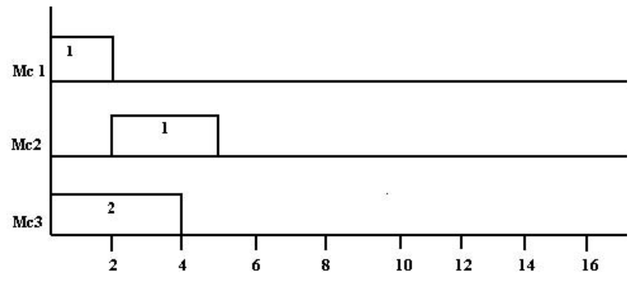
Fig. 1a–d. Scheduling of jobs as per job based representation **a** first job j2; **b** second job j4; **c** third job j3; and **d** fourth job j1



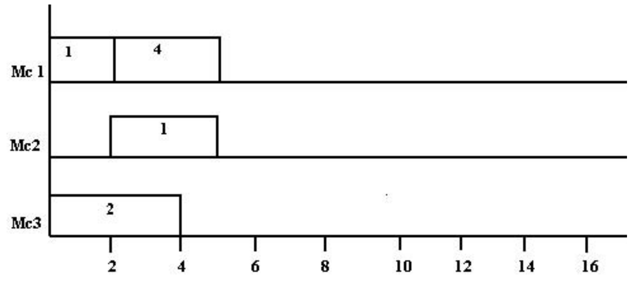
(a) First operation allocated



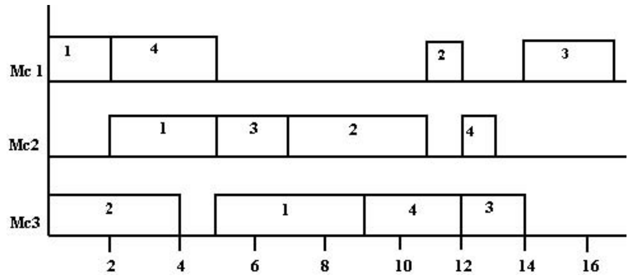
(b) Second operation allocated



(c) Third operation allocated



(d) Fourth operation allocated



(e) All operations allocated

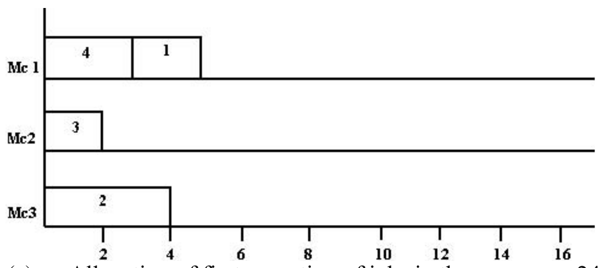
Fig. 2a–e. Scheduling of jobs as per operation sequence in the chromosome **a** first operation of job j1; **b** second operation of job j1; **c** first operation of job j2; **d** first operation of job j4; and **e** all the operations of all jobs

operation of the jobs j2, j4, j3, and j1 are in m3, m1, m2, and m1, respectively, and the corresponding times are [4 3 2 2]. The first operation m3 of job j2 is scheduled. Then, the first operation m1 of job j4 is scheduled, followed by the scheduling of the first operation m2 of job j3. Finally, the first operation m1 of job j1 is scheduled. The schedule obtained is shown in Fig. 3a.

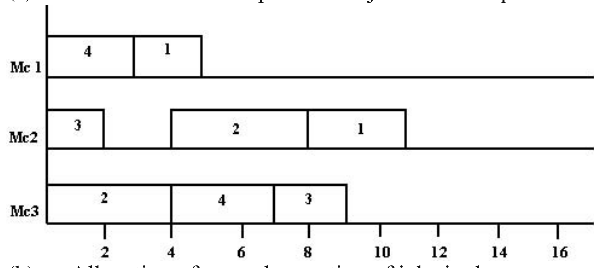
The second operation of the jobs j2, j4, j3, and j1 are in m2, m3, m3, and m2, respectively, and the corresponding times are [4 3 2 3]. The second operation m2 of job j2 is scheduled. Then, the second operation m3 of job j4 is scheduled, followed by the scheduling of the second operation m3 of job j3. Finally, the second operation m2 of job j1 is scheduled, as shown in Fig. 3b.

The third operation of the jobs j2, j4, j3, and j1 are in m1, m2, m1, and m3, respectively, and the corresponding times are [1 1 3 4]. The third operation m1 of job j2 is scheduled. Then, the third operation m2 of job j4 is scheduled, followed by the scheduling of the third operation m1 of job j3. Finally, the third operation m3 of job j1 is scheduled, as shown in Fig. 3c.

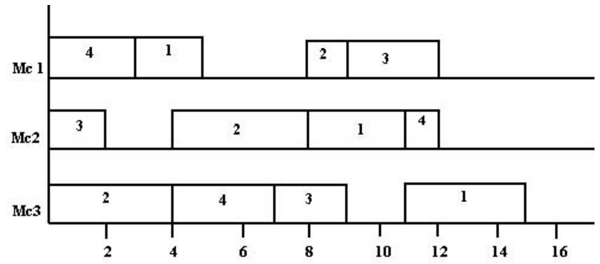
The Gantt chart of the schedule for the sequence 2 4 3 1 under proposed representation and schedule deduction scheme shows the make-span as 15.



(a) Allocation of first operation of jobs in the sequence- 2431



(b) Allocation of second operation of jobs in the sequence- 2431



(c) Allocation of third operation of jobs in the sequence- 2431

Fig. 3a–c. Scheduling of jobs as per proposed method **a** first operation of jobs j2, j4, j3, and j1 **b** second operation of jobs j2, j4, j3, and j1 **c** third operation of jobs j2, j4, j3, and j1

4 Application of GA

4.1 Objective function

The objective of the problem is to find the job sequence for which the make-span $C_{\max}$ is minimum:

Objective function $= f(x)$.

4.2 Fitness function

Since GAs are most suitable for maximization problems, the above minimization problem is converted into an equivalent maximization problem by the following transformation:

Fitness function value $F(x) = C_{\max\,p} - f(x)$,

where $C_{\max\,p}$ is taken as the largest make-span value observed in the current population. So, $C_{\max\,p}$ varies dynamically with varying population.

4.3 Input

The input data are: job numbers, operation sequences, and operation time.

4.4 Initialization

This module deals with the creation of all possible chromosomes, within the population size. The operation of GA begins with a population of random strings representing decision variables. The population size is taken as 10 and the number of generations as 1000.

For the above problem, a coded string comprising n numbers is generated randomly. The string says 2 4 3 1 is a typical chromosome. Ten random strings are created to form the initial population.

4.5 Evaluation

The objective criterion for the problems has been chosen as the make-span. The population of chromosomes is evaluated for this objective. Population size (p_size) is kept constant throughout the trial and the fitness value for each string generated is calculated. This value is used for the reproduction operation. For the pattern of chromosome 2 4 3 1, the make-span is found by placing the jobs in various machines as per the sequence dictated by the chromosome. The placement of jobs for the chromosome discussed is shown in Fig. 4.

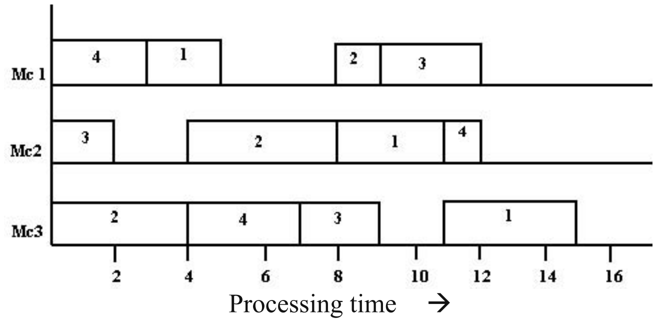


Fig. 4. Gantt chart for the chromosome 2 4 3 1 (make-span = 15)

4.6 New population creation

Creation of new population deals with selection of chromosomes and application of crossover and mutation operators.

4.6.1 Reproduction

At this step, we select good strings in the current population to form the mating pool. Roulette-wheel selection operator is used for reproduction. The action of reproduction operator is to clear the inferior points from further consideration by probabilistic elimination:

$$p(x) = F(x) / \sum_{x=1}^{x=\text{p\_size}} F(x) ,$$

where $p(x)$ is the probability of selection of chromosome x.

Through the generation of random numbers, the chromosomes are probabilistically picked in proportion to their weight and put into the mating pool. Steady state replacement policy has been adopted in this attempt to always retain the current best strings in the population [7].

4.6.2 Crossover

Crossover is the operation mainly responsible for the search of new chromosomes/strings. The probability of crossover, p_cross, is taken as 0.80 to cover 80% of the chromosome for crossover to produce children. A random number between 0 and 1 is generated for each chromosome and crossover is effected if the number is less than or equal to p_cross (0.80). The process is repeated for 9(p_size − 1) chromosomes, keeping the best of the chromosomes in the mating pool undisturbed. Edge recombination (EX) crossover operator has been used. This operator works on individual chromosomes and not on pairs. This operator, in addition to being simple to implement, gives better solutions.

If a chromosome qualifies for crossover, a random number from 1 to n is generated to select the site for crossover operation. When the crossover site is, say 2, crossover happens as indicated below:

Before crossover: 2 4 $\underline{\mathbf{3\ 1}}$
After crossover: 2 4 $\underline{\mathbf{1\ 3}}$.

4.6.3 Mutation

The next step is to perform mutation on chromosomes in the intermediate population. Mutation is to maintain diversity in the population. The probability of mutation is taken as 0.05 to cover 5% of the chromosomes for mutation. A random number between 0 and 1 is generated and mutation is effected if the number is less than or equal to p_mute(0.05). Order-based mutation (OBM) has been applied, wherein two genes at random positions in the chromosome are swapped. Two random numbers between 1 to n are generated for the selection of genes and the selected genes are interchanged for position.

When the genes selected for mutation are, for example, 2 and 4, mutation takes place as follows:

Before mutation: 2 **4** 1 **3**
After mutation: 2 *3* 1 *4* .

The above steps are repeated with the new population for the required number of generations

4.7 Output

The solution, which gets improved during each iteration/generation, is available as the result.

5 Validation

To validate the proposed heuristic, one instance from the suite JSP test instances of Fisher and Thompson (mt10), five instances set by Adams (abz5, abz6, abz7, abz8, and abz9), four instances set by Yamada and Nakano (yn1 ,yn2, yn3, and yn4), one instance set by Applegate and Cook (orb07), one instance set by Storer (swv06), and 12 instances set by Lawrance (la27, la28, la29, al30, la31, la32, la33, la34, la37, la38, la39, and la40) have been selected (problems are available in operations research (OR) library [8]). Roulette-wheel selection, edge recombination (EX) crossover, and order-based mutation (OBM) are selected as GA operators. The crossover and mutation probabilities are assigned to be 0.80 and 0.05, respectively. It has been decided to have the population size as 10 and number of generations as 1000.

A typical benchmark problem (abz6) has been taken for illustration. The first line contains the number of jobs and the number of machines, followed by details of each job through listing the machine number and processing time for each step of the job. The machines are numbered starting with zero:

```
10 10
7 62 8 24 5 25 3 84 4 47 6 38 2 82 0 93 9 24 1 66
5 47 2 97 8 92 9 22 1 93 4 29 7 56 3 80 0 78 6 67
1 45 7 46 6 22 2 26 9 38 0 69 4 40 3 33 8 75 5 96
4 85 8 76 5 68 9 88 3 36 6 75 2 56 1 35 0 77 7 85
8 60 9 20 7 25 3 63 4 81 0 52 1 30 5 98 6 54 2 86
3 87 9 73 5 51 2 95 4 65 1 86 6 22 8 58 0 80 7 65
5 81 2 53 7 57 6 71 9 81 0 43 4 26 8 54 3 58 1 69
4 20 6 86 5 21 8 79 9 62 2 34 0 27 1 81 7 30 3 46
9 68 6 66 5 98 8 86 7 66 0 56 3 82 1 95 4 47 2 78
0 30 3 50 7 34 2 58 1 77 5 34 8 84 4 40 9 46 6 44 .
```

In this section, the application of GA for the proposed method of representation and deduction of schedule is illustrated with the above example.

Objective function $= C_{\max} = f(x)$, fitness function $F(x) = C_{\max \mathrm{p}} - f(x)$, where $C_{\max \mathrm{p}} =$ Largest make-span value in the current population:

$$p(x) = F(x) / \sum_{x=1}^{x=\mathrm{p\_size}} F(x) \,,$$

where p_size is population size and $p(x) =$ Probability of selection of chromosome x.

The working of the GA is explained in Sect. 4. The GA parameters applied are roulette-wheel selection operator, edge recombination crossover operator, and order-based mutation operator. Initial population, corresponding fitness values (population evaluation), selection of chromosomes for the next generation, and chromosomes in the mating pool for the considered example are given in Table 3. The parents selected for crossover from the mating pool and corresponding offspring after crossover, chromosomes selected for mutation, new population after mutation and revised fitness values are given in Table 4.

The best make-span, corresponding schedule and CPU time are given below for the above problem for the three methods of representation and deduction of schedule considered in this paper.

Table 3. Evaluation and reproduction of chromosomes

Ch no.	Chromosomes in initial population	Obj. fun. $f(x)$	Fitness value $F(x)$	Prob. of sel. $p(x)$	Cum. seln. prob.	Rand no.	Chr. no.	Chromosomes for mating pool
1	5 1 7 8 6 3 2 4 9 0	1233	69	0.0651	0.0651	0.26	5	5 3 6 4 9 1 7 2 8 0
2	8 6 2 5 7 4 9 0 1 3	1195	107	0.1009	0.1660	0.87	9	5 1 6 4 3 0 7 9 8 2
3	5 8 0 3 7 9 6 1 2 4	1241	61	0.0575	0.2235	0.82	9	5 1 6 4 3 0 7 9 8 2
4	4 2 8 7 3 5 6 1 0 9	1302	0	0.0000	0.2235	0.26	5	5 3 6 4 9 1 7 2 8 0
5	5 3 6 4 9 1 7 2 8 0	1151	151	0.1425	0.3660	0.36	5	5 3 6 4 9 1 7 2 8 0
6	6 7 9 4 8 0 1 2 3 5	1217	85	0.0802	0.4462	0.42	6	6 7 9 4 8 0 1 2 3 5
7	9 5 1 8 0 3 7 2 6 4	1218	84	0.0792	0.5254	0.93	9	5 1 6 4 3 0 7 9 8 2
8	4 0 3 1 9 6 7 2 8 5	1076	226	0.2132	0.7386			4 0 3 1 9 6 7 2 8 5*
9	5 1 6 4 3 0 7 9 8 2	1140	162	0.1528	0.8914	0.73	8	4 0 3 1 9 6 7 2 8 5
10	1 4 7 0 2 5 8 9 6 3	1187	115	0.1085	0.9999	0.03	1	5 1 7 8 6 3 2 4 9 0

Best string is copied into mating pool and not genetically operated.

Table 4. Application of crossover and mutation operators

Population in the mating pool	Rand no.	Cr. over site, if selected	Population after crossover	Rand no.	Genes for mutation, if selected	Population after mutation	Fitness value
5 3 6 4 9 1 7 2 8 0	0.86	N.S.	5 3 6 4 9 1 7 2 8 0	0.44	N.S.	5 3 6 4 9 1 7 2 8 0	1248 – 1151 = 97
5 1 6 4 3 0 7 9 8 2	0.70	7	5 1 6 4 3 0 7 2 8 9	0.81	N.S.	5 1 6 4 3 0 7 2 8 9	1248 – 1133 = 115
5 1 6 4 3 0 7 9 8 2	0.59	5	5 1 6 4 3 2 8 9 7 0	0.94	N.S.	5 1 6 4 3 2 8 9 7 0	1248 – 1189 = 59
5 3 6 4 9 1 7 2 8 0	0.76	2	5 3 0 8 2 7 1 9 4 6	0.12	N.S.	5 3 0 8 2 7 1 9 4 6	1248 – 1240 = 8
5 3 6 4 9 1 7 2 8 0	0.88	N.S.	5 3 6 4 9 1 7 2 8 0	0.82	N.S.	5 3 6 4 9 1 7 2 8 0	1248 – 1151 = 97
6 7 9 4 8 0 1 2 3 5	0.78	5	6 7 9 4 8 5 3 2 1 0	0.04	4, 6	6 7 9 5 8 4 3 2 1 0	1248 – 1248 = 0
5 1 6 4 3 0 7 9 8 2	0.95	N.S.	5 1 6 4 3 0 7 9 8 2	0.25	N.S.	5 1 6 4 3 0 7 9 8 2	1248 – 1140 = 108
5 3 6 4 9 1 7 2 8 0	0.47	6	5 3 6 4 9 1 0 8 2 7	0.41	N.S.	5 3 6 4 9 1 0 8 2 7	1248 – 1196 = 52
4 0 3 1 9 6 7 2 8 5			Not genetically operated				1248 – 1076 = 172
5 1 7 8 6 3 2 4 9 0	0.37	1	5 0 9 4 2 3 6 8 7 1	0.66	N.S.	5 0 9 4 2 3 6 8 7 1	1248 – 1192 = 56

Probability of crossover = 0.80, Probability of mutation = 0.05, N.S. = not selected

5.1 Job-based representation and deduction of schedule

Best make-span:	1212
Best schedule:	5 7 6 4 8 2 3 9 0 1
CPU time:	1.37 s

5.2 Operation-based representation and deduction of schedule

Best make-span:	1251
Best schedule:	8 7 3 7 4 8 0 9 2 1
	7 4 4 7 7 2 2 5 5 2
	4 6 6 4 6 6 6 3 3 9
	0 0 0 3 4 4 1 6 1 4
	6 8 9 4 1 1 0 7 8 5
	9 9 6 3 9 3 7 8 2 0
	1 7 5 9 2 1 5 5 8 1
	9 8 5 2 2 5 8 0 3 7
	0 1 3 2 4 8 5 9 2 6
	3 5 8 7 0 9 6 0 3 1
CPU time:	0.44 s

5.3 Proposed representation and deduction

Best make-span:	1076
Best schedule:	4 0 3 1 9 6 7 2 8 5
CPU time:	0.38 s

All the schemes have been coded in the C++ language. The computing system used is Intel Pentium IV CPU at 1.8 GHz, 128 MB RAM processor.

6 Results and discussions

The details about the name of the problem, size of the problem, make-span found, and the CPU time for the three methods of representation and deduction of schedule are presented in Table 5. The results are also presented in the form of charts in Figs. 5 and 6. The proposed method is found to perform reasonably better when the problem size is bigger. This has been shown in Table 5.

The advantage of the proposed method of deduction is that it finds better solutions in a shorter time. The speed of finding the solutions is 3 to 18 times higher than the time taken by job-based method for the problems discussed. The priority rule-based representation and the precedence constraint-based representation are found to take longer CPU time compared to operation-based and job-based representations [9]. The best solution can be realized with operation-based representation with large number of cycles. Job-based method and proposed method of deduction may not yield the minimum make-span as there are strict rules in the allocation of operations of the jobs, in addition to operation precedence constraints.

Fig. 5. Bar chart between make-span and problem instances

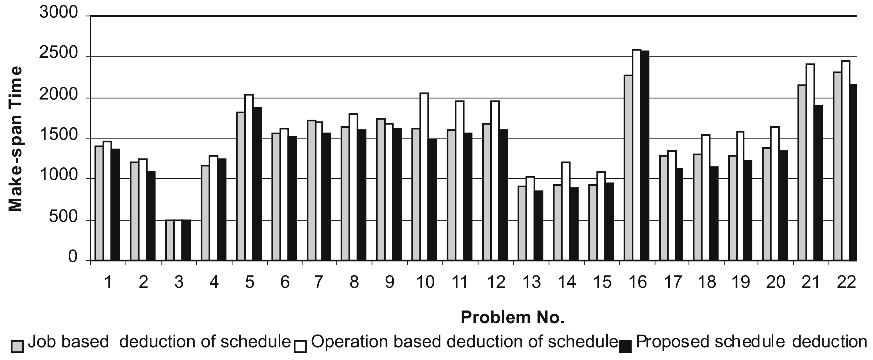


Table 5. Results of various methods of deduction

Problem no.	Problem instance	Size	Job based representation and schedule deduction Makespan	Job based representation and schedule deduction CPU time (secs)	Operation based representation and schedule deduction Makespan	Operation based representation and schedule deduction CPU time (secs)	Proposed representation and schedule deduction Makespan	Proposed representation and schedule deduction CPU time (secs)
1	abz5	10 × 10	1411	1.37	1461	0.49	1361	0.44
2	abz6	10 × 10	1212	1.37	1251	0.44	1076	0.38
3	orb07	10 × 10	487	1.37	493	0.50	499	0.44
4	mt10	10 × 10	1167	1.32	1283	0.49	1241	0.44
5	la37	15 × 15	1807	5.00	2034	0.99	1883	0.82
6	la38	15 × 15	1561	5.05	1622	0.99	1521	0.83
7	la39	15 × 15	1720	5.05	1701	1.04	1560	0.83
8	la40	15 × 15	1642	5.05	1797	0.93	1596	0.87
9	la27	20 × 10	1730	6.81	1671	0.88	1626	0.77
10	la28	20 × 10	1612	6.92	2050	0.87	1478	0.77
11	la29	20 × 10	1594	6.97	1958	0.87	1551	0.77
12	la30	20 × 10	1682	6.92	1963	0.82	1605	0.77
13	abz7	20 × 15	909	10.33	1031	1.27	846	1.05
14	abz8	20 × 15	925	10.38	1197	1.27	886	1.10
15	abz9	20 × 15	934	10.32	1080	1.27	950	1.10
16	swv06	20 × 15	2272	10.00	2582	1.26	2564	1.10
17	yn1	20 × 20	1286	13.73	1349	1.70	1123	1.43
18	yn2	20 × 20	1299	13.79	1548	1.70	1144	1.42
19	yn3	20 × 20	1278	13.85	1585	1.71	1220	1.43
20	yn4	20 × 20	1388	13.74	1637	1.65	1334	1.37
21	la31	30 × 10	2160	20.27	2410	1.26	1902	1.10
22	la32	30 × 10	2309	20.31	2439	1.21	2142	1.15
23	la33	30 × 10	2145	20.27	2251	1.26	1951	1.10
24	la34	30 × 10	2209	20.32	2341	1.26	1961	1.10

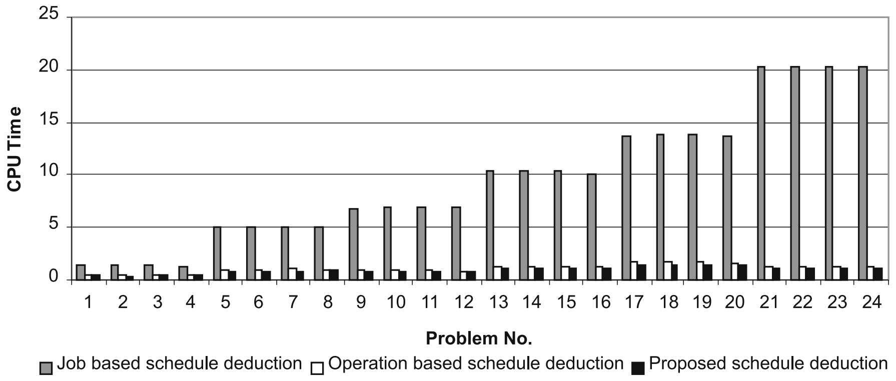


Fig. 6. Bar chart between CPU time and problem instances

In job-based method, all the operations of the first job are to be scheduled first, followed by all the operations of second job in the sequence, and so on. In the proposed method, first operation of all the jobs in the sequence are scheduled, followed by the second operation of all the jobs, and so on. Operation-based representation takes less time as the assignment of operations is easy while a schedule is being generated.

A number of trial runs were effected for the sample problem (shown in Tables 1 and 2) and the minimum make-span of 13 has been found only in the operation-based scheme and proposed scheme and not in the job-based scheme.

Almost in all the 24 instances, the proposed method of schedule deduction gives better solutions than the operation-based scheme. In 19 of the 24 instances, proposed method gives the best solutions, taking the least computational time. In four instances, proposed method takes the second position in the solution, with the best result given by job-based representation and schedule deduction scheme. It is noted that for the four instances (mt10, la37, abz9, and swv06) the CPU time taken by job-based representation and deduction scheme is 3 to 9 times more compared to proposed schedule deduction scheme. In one instance (orb07), the proposed method stands third in the solution.

7 Conclusions

An attempt has been made to assess the performance of three GA representation schemes to solve job shop problems. The proposed representation scheme has been compared with job-based and operation-based representation schemes for the make-span objective for solving the job shop problem.

The algorithms have been coded using C++ language. The make-span and CPU time are presented in the form of tables and charts. The ranking of the three schemes with reference to minimum make-span in most instances is:

(1) Proposed scheme (19 instances)
(2) Job-based representation scheme (five instances)
(3) Operation-based representation scheme.

The ranking of the three schemes with reference to CPU time for all instances is:

(1) Proposed scheme
(2) Operation-based representation scheme
(3) Job-based representation scheme.

It is suggested that the proposed representation and schedule deduction scheme may be adopted as an acceptable alternate scheme for solving job shop scheduling problems. The proposed method may be called operation precedence-based scheme.

References

1. Gen M, Cheng R (1997) Genetic algorithms and engineering design. Wiley, New York
2. Baker KR (1974) Introduction to sequencing and scheduling. Wiley, New York
3. Pinedo M, Chao X (1999) Operation scheduling with applications in manufacturing and services. McGraw-Hill, Boston
4. Goldberg DE (1989) Genetic algorithm in search, optimization, and machine learning. Addison-Wesley, Boston
5. Cheng R (1996) A tutorial survey of job-shop scheduling problems using genetic algorithm I: representation. J Comput Ind Eng 30(4):983–987
6. Panneerselvamn E (2001) Production and operations management. Prentice-Hall, New Delhi
7. Filho JLR, Treleaven PC (1994) Genetic algorithm programming environments. IEEE Comput 27(6):28–43
8. Beasley JE (1990) OR-Library. http://mscmga.ms.ic.ac.uk/info.html. Cited 1 July 2004
9. Ponnambalam SG, Aravindan P, Sreenivasa Rao P (2001) Comparative evaluation of genetic algorithms for job shop scheduling. Prod Plan Control 12(6):560–574