Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 31 (2004) 593–606
computers & operations research
www.elsevier.com/locate/dsw

Improved genetic algorithm for the permutation flowshop scheduling problem

Srikanth K. Iyer[a,*], Barkha Saxena[b]

[a] *Department of Mathematics, Indian Institute of Technology, Kanpur 208016, India*
[b] *Fair Isaac, Santa Barbara, USA*

Received 1 June 2000; received in revised form 1 September 2002; accepted 1 December 2002

Abstract

Genetic algorithms (GAs) are search heuristics used to solve global optimization problems in complex search spaces. We wish to show that the efficiency of GAs in solving a flowshop problem can be improved significantly by tailoring the various GA operators to suit the structure of the problem. The flowshop problem is one of scheduling jobs in an assembly line with the objective of minimizing the completion time or makespan. We compare the performance of GA using the standard implementation and a modified search strategy that tries to use problem specific information. We present empirical evidence via extensive simulation studies supported by statistical tests of improvement in efficiency.

Scope and purpose

Genetic algorithms (GAs) are intelligent random search strategies which have been used successfully to find near optimal solutions to many complex problems. Implementation of GA in solving problems often overlooks certain information available in a particular problem. Making use of this information may need modification of the coding of the search space and of the operators constituting GA. This is a problem specific task. This paper hopes to address this issue with regard to solving the *permutation flowshop* problem. Henceforth we will refer to this as the flowshop problem, since we consider only the permutation flowshops. The questions we pose in order to implement the improved search heuristic can be extended easily to a host of scheduling problems with single and multi-objective optimization criteria.

Flowshops are useful tools in modeling manufacturing processes. A permutation flowshop is a job processing facility which consists of several machines and several jobs to be processed on the machines. In a permutation flowshop all jobs follow the same machine or processing order. Flowshop refers to the fact that job processing is not interrupted once started. Our objective is to find a sequence for the jobs so that the makespan or the completion time is minimum. It is well known that this is a difficult problem to solve in a reasonable amount

* Corresponding author. Tel.: +91-512-59-7635; fax: +91-512-59-7500.
E-mail addresses: skiyer@iitk.ac.in (S.K. Iyer), barkhasaxena@yahoo.com (B. Saxena).

0305-0548/04/$ - see front matter
doi:10.1016/S0305-0548(03)00016-9

of time. In this paper we propose a modification to the existing GA implementation which incorporates the underlying "physics" of the flowshop problem.

Keywords: Genetic algorithms; Permutation flowshop scheduling; Longest common subsequence; Design of experiments

1. Introduction

Genetic algorithms (GAs) have proved to be highly successful in solving optimization problems where the search space is highly unstructured or the standard techniques like the branch and bound method, etc. fail to provide efficient solutions. GAs are search techniques based on mechanics of natural selection and natural genetics. They combine survival of fittest with a structured yet randomized information exchange to form a search algorithm.

The basic implementation of a GA assumes the existence of a search space whose elements are strings (chromosomes) and a function defined on this space whose optimum value needs to be evaluated. However, in any practical implementation of GA, there may be no obvious choice of the code to characterize the search space. Another important question is the way in which the crossover and mutation operators should be defined. In this work, we try to pose questions that one may ask in order to arrive at appropriate solutions to the above issues. In order to maintain focus, we undertake all discussions in the context of a permutation flowshop scheduling problem.

Flowshop scheduling is one of the most well-known problems in the area of scheduling. Various approaches to this problem have been proposed since the pioneering work of Johnson [1]. GAs have been applied (Holland [2], Goldberg [3], Davis [4]) to combinatorial optimization problems such as the traveling salesman problem (Jog et al. [5], Starkweather et al. [6], and Ulder et al. [7]), and scheduling problems (see for example, Fox and McMahon [8], Ishibuchi et al. [9]). A simulated annealing (SA) approach to the flowshop problem was proposed by Osman and Potts [10] and by Ogbu and Smith [11]. This approach was shown to produce high quality solutions. A GA for flowshop scheduling was proposed by Reeves [12], which was then tested on several categories of problems with time gradients and job correlations and some hard test problems proposed by Taillard [13]. GA was overall seen to produce results comparable to SA for the flowshop sequencing problem for most types and sizes of problems. Further, GA was shown to perform relatively better for large problems and reaches near-optimal solutions earlier. Both GA and SA out-perform other heuristics. Murata et al. [14] compare various crossover and mutation operators and show that the two point crossover and shift mutation operators are effective for this problem. They consider two hybrid variants of the GA, namely the genetic local search and genetic simulated annealing algorithms and show high performance for the hybrid models. Bagchi [15] lists several elitist versions of the GA and reports improvement in performance.

In this paper, we focus on the improvement in the standard implementation of the GA, by using the structure of the problem. We propose a new crossover operator and show using *statistical tests*, of a significant improvement in performance. Various crossover operators considered in literature (Reeves [12], Murata et al. [14], Pinedo [16]) adapt a crossover operator that is used in GAs, where the search space is coded in the form of binary strings. Since a direct application of this crossover

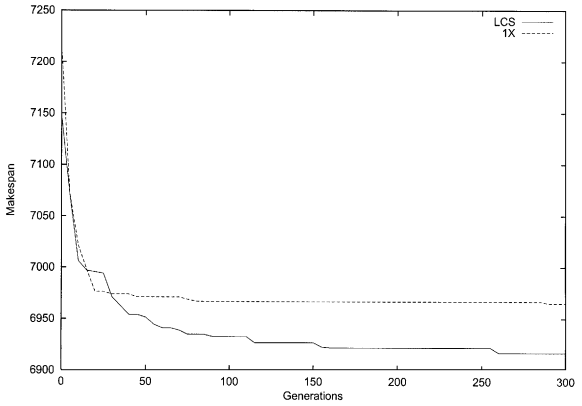

Fig. 1. Comparison between 1X and LCS crossover for a 10 m/c 10 job flowshop.

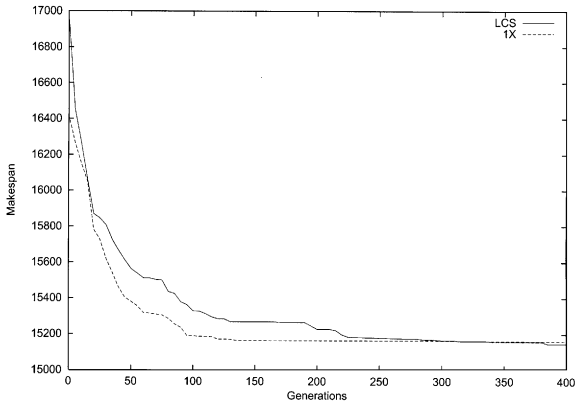

Fig. 2. Comparison between 1X and LCS crossover for a 10 m/c 20 job flowshop.

operator can produce infeasible strings, a modification is made so that crossover produces elements that belong to the search space. Neither the binary crossover nor the modified operator seem to make any use of the structure of the flowshop problem. There are a large number of variations of the basic GA implementation for the flowshop problem, and comparisons are available with the standard implementation.

Djerid and Portmann [17] and Portmann and Vignier [18] developed indicators (we mention two of the relevant ones here) for measuring the efficacy of crossover operators in keeping good schemata for permutation problems. One is the precedence constraint based indicator (PCBI) which is based on the following property: If job i precedes job j in both parents, then this precedence is maintained in the offsprings. The PCBI measures the extent to which the above property is satisfied by the crossover operator. The other is the position-based indicator which is based on a distance measure between two permutations. In using this indicator, the effort is to minimize the distance between the parents and offsprings. Several operators like the OX (order cross operator, Davis [19]), LOX (linear

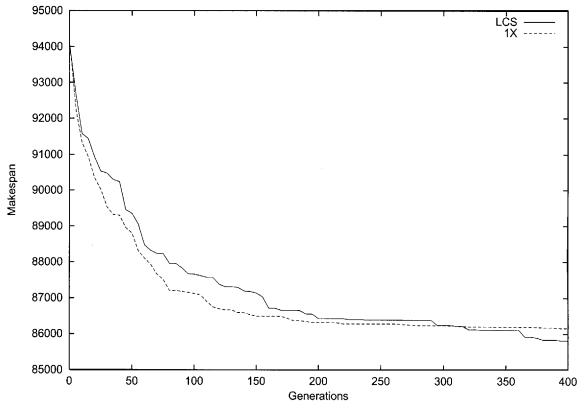

Fig. 3. Comparison between 1X and LCS crossover for a 15 m/c 49 job flowshop.

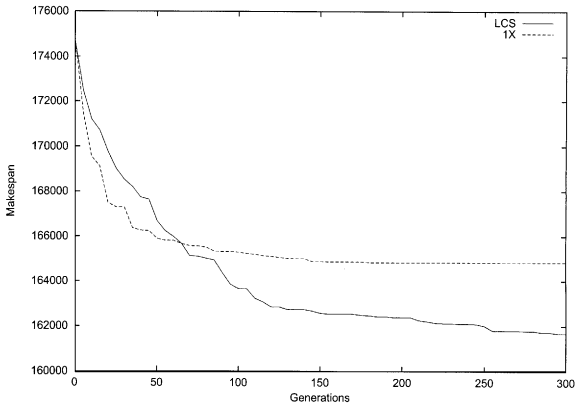

Fig. 4. Comparison between 1X and LCS crossover for a 25 m/c 60 job flowshop.

order cross operator, Falkenauer and Bouffouix [20]), 1X (Davis [19]) and kX (Caux, Pierreval and Portmann [21], Djerid et al. [22]) are examined using simulations (Portmann and Vignier [18]) and analytically (Djerid and Portmann [17]), to measure their efficacy with regard to the above indicators.

It is shown that the 1X operator is always the best among the above crossover operators for giving the greatest percentage of children who have got quality similar to quality of both parents. In fact the 1X operator always satisfies the precedence constraint property. Thus, PCBI is maximized for the 1X operator. So we compare the proposed crossover operator with the 1X operator only. This allows us to explore the use of optimal parameters derived using statistical tests (Bagchi and Deb [23]) and also run the algorithms on a large variety of problem sizes and distribution of processing times. We shall refer to the GA based on the 1X operator as the standard implementation.

The basic idea behind the crossover is that each of the following generations carries over the good genes from the previous generation. So the crossover operator must preserve that part of the genetic code of the parents which is responsible for their high fitness. Exchange of the remaining genes

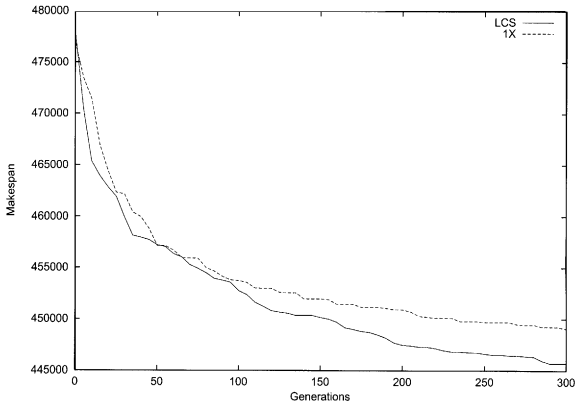

Fig. 5. Comparison between 1X and LCS crossover for a 40 m/c 100 job flowshop.

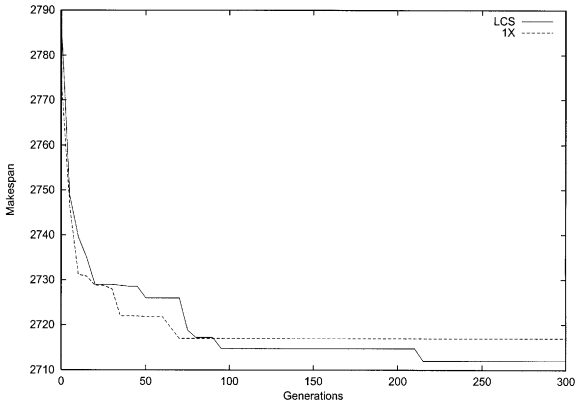

Fig. 6. Comparison between 1X and LCS crossover for a 10 m/c 10 job flowshop.

facilitates searching the space for improving the fitness. The 1X operator preserves the precedence constraint for all subsets of the parent sequences. That is, for any k, if jobs $i_1, i_2, \ldots, i_k$ occur in the order given in both parents, they will preserve this order in the offspring. This restricts the search space and slows down the convergence of the algorithm. We propose a crossover operator (referred henceforth as the LCS operator) that preserves only the longest common subsequence present in the parents. This allows us to search over a larger space, while preserving the good part of the parent chromosomes. It is seen (see Figs. 1–7), that while the 1X operator may perform well initially, the LCS operator consistently overtakes the 1X operator and continues to dominate thereafter. The LCS operator demonstrates an ability to produce better solutions even after a large number of iterations, while the 1X operator stagnates for long durations.

It must be noted that while Djerid and Portmann [17] and Portmann and Vignier [18] develop indicators to measure the efficacy of existing operators, we develop a new crossover operator by asking the same questions that they do. Our results show that while operators may score highly in

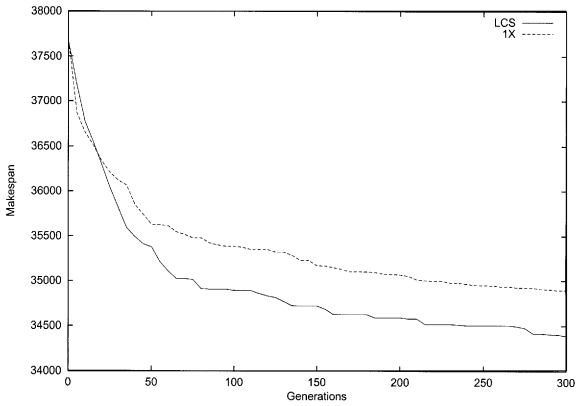

Fig. 7. Comparison between 1X and LCS crossover for a 15 m/c 49 job flowshop.

terms of preserving good schemata, they may be restricting the search space by preserving more than the desirable amount of information present in the parents.

2. Permutation flowshop scheduling

This is a typical assembly line problem where m different jobs have to be processed on n different machines. All jobs are processed on all the machines in the same order. The processing times of the jobs on machines are fixed irrespective of the order in which the processing is done. The problem is characterized by a matrix $P = (p_{ij})$, $p_{ij} > 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, of processing times. Each machine processes exactly one job at a time and each job is processed on exactly one machine at a time. The problem then is to find a sequence of jobs such that the makespan, that is the completion time of the last job in the sequence on the last machine is minimum. If C_i denotes the completion time for job i, then we are trying to minimize $\max_i C_i$. There are many other criterion that can be considered for optimization. We refer the reader to Bagchi [15] for a detailed discussion of multi-objective scheduling using GA. For details of the flowshop and other scheduling and sequencing problems we refer the reader to Baker [24].

3. Genetic algorithms

The first step in applying GA to a particular problem is to convert the feasible solutions of that problem into a string type structure called chromosome. In order to find the optimal solution of a problem, a standard GA starts from a set of assumed or randomly generated solutions (chromosomes) called initial population and evolves different but better sets of solution (chromosomes) over a sequence of generations (iterations). In each generation the objective function (fitness measuring criterion) determines the suitability of each chromosome and, based on these values, some of them are selected for reproduction. For the flowshop scheduling problem we take the fitness value of each chromosome to be the reciprocal of the makespan. The number of copies reproduced by an individual

parent is expected to be directly proportional to its fitness value, thereby embodying the natural selection procedure, to some extent. The procedure thus selects better (highly fitted) chromosomes and the worse one are eliminated. Genetic operators such as crossover and mutation are applied to these (reproduced) chromosomes and new chromosomes (offspring) are generated. These new chromosomes constitute the next generation. These iterations continue till some termination criterion is satisfied. A good reference for understanding how GAs work is Goldberg [3].

4. GA and the flowshop problem

The standard implementation for solving a permutation flowshop problem using GA can be found in Pinedo [1]. The components of a standard GA applied to solve the flowshop problem may be briefly described as follows.

4.1. Encoding scheme

Here one views the job sequences themselves as chromosomes. For example, in a five job problem a chromosome could be [13245]. This represents the sequence where job 1 gets processed first on all the machines followed by job 3 and so on. This seems to be the natural choice for this problem.

4.2. Initial population

The initial population consists of N randomly generated job sequences. N is referred to as the population size.

4.3. 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). Let r_i, $i=1,\ldots,N$ denote the reciprocal of the makespan of the strings in the population. The fitness value assigned to string i would then be proportional to $f_i = r_i/\sum_j r_j$. Recall that our objective here is to minimize the makespan.

4.4. Reproduction

Individual chromosomes from the present population are copied according to their fitness values. This is done by randomly selecting (with replacement) chromosomes with probability proportional to their fitness value. Roughly speaking, one would expect $N \times f_i$ number of copies of string i in the gene pool used to create the next generation. This forms the mating pool to which the crossover and mutation operators are applied to form the next generation.

4.5. Crossover

The main purpose of crossover is to exchange information between randomly selected parent chromosomes with the aim of producing better offspring. It recombines genetic material of two parent chromosomes to produce offspring for the next generation that retain the good properties of the parent chromosomes. The exchange is also intended to search for better genes.

To perform crossover, two parents are selected from the mating pool at random. With probability $1 - p_c$, the parents are copied as they are. With probability p_c, called the crossover probability the following operation is performed: In case of a standard GA a single point crossover (called 1X operator) between two parents is carried out by choosing a number k at random between 1 and $\ell - 1$, where ℓ is the length of the string ($\ell > 1$). Two new strings are created by swapping all characters from position $k + 1$ to ℓ. However, doing so for the above coding scheme could result in infeasible strings. For example, in a eight job problem crossing strings [12345678] and [58142376] with $k = 3$ will yield [12342376] and [58145678] which are infeasible solutions because of repetition of jobs. Hence a modification has to be carried out in the standard crossover procedure. To do this the first child is created by copying all characters of the chromosome of the first parent upto location k. The remaining places are filled by scanning parent 2 from left to right and entering the machine numbers not already present. Child 2 is created in a similar way by reversing the roles of the parents. In the above example, this will result in the children [12358476] and [58123467]. The above procedure is repeated till N children are obtained. Though the 1X operator is the best among the existing crossover operators in terms of the PCBI indicator, it is not an outcome of consciously trying to maximize this indicator.

4.6. Mutation

Mutation is done by selecting two different locations on the chromosome at random and interchanging the jobs at these locations. For each child obtained from crossover, the mutation operator is applied independently with a small probability p_m. Mutation is a way of enlarging the search space. It acts to prevent the selection and crossover from focusing on a narrow area of the search space or from the GA getting stuck in a local optimum.

4.7. Termination criterion

In our implementation of GA, we terminate the algorithm after 300 iterations.

5. Designing a better crossover operator

We wish to design a crossover operator that preserves that part of the genetic code of the parent chromosomes which is responsible for their relatively high fitness and at the same time provide freedom to search a greater part of the domain for good strings. In order to do this the fundamental question that needs to be answered is "what constitutes the good part of the parent chromosomes?" To motivate the answer let us consider the trivial case of a two job problem where the processing times for job 1 is less than that for job 2 on all machines. It is obvious that job 1 must go in

before job 2, else job 2 will "block" job 1. Extending this idea to the general case, what we are trying to do is to minimize the amount of blocking. So the important information in a chromosome or job sequence is the *relative positions* of the jobs. Changing the relative positions of the jobs may increase or decrease the amount of blocking and thus result in greater or lesser fitness. Hence the crossover operator must preserve the locations of those jobs whose relative positions in both the parents are the same. The 1X operator, while preserving all relative positions (and hence the positions of all common subsequences), turns out to be too restrictive. With each subsequent generation, the common subsequences increase, thereby allowing less space for exchange of information. We propose an operator that preserves only the *longest common subsequence* in the parents. It preserves those orders of the jobs in the string of job sequences which are expected to give good results. As the number of iterations increase, the longest common subsequence will grow and will not be disturbed. By not preserving lower order common subsequences the search space is enlarged. Elements not belonging to the longest common subsequence are then swapped in the following way. The first element of the parent2 which is not present in the longest common subsequence is copied at the first available place in the child1 and the element available at that position in the parent1 is copied at the first available position in child2. For example, consider two chromosomes from a 9 job problem:

Parent1: 4 6 9 3 7 2 8 1 5

Parent2: 1 7 4 2 9 3 8 6 5

The longest common subsequence is 49385. This gives the two children to be

Child1: 4 1 9 3 7 2 8 6 5

Child2: 6 7 4 2 9 3 8 1 5

Copy the jobs 4, 9, 3, 8 and 5 in the same positions as they are in the parents. In parent1, 6 is the first element that does not belong to the longest common subsequence. This goes into the first available slot in child2 which is location 1. Likewise, job 1 from parent2 goes into the first available location in child1 which is slot number 2. This procedure is continued to get the two children.

The problem of finding the longest common subsequence can be solved using dynamic programming in polynomial time (Corman [25]). Thus the above scheme can be implemented efficiently. It is important to note here that the time complexity for finding the longest common subsequence and the average time taken for implementing the standard coding is $O(n^2)$. The GA implementation we propose differs from the standard only in the way the crossover is carried out. The modified implementation will be abbreviated as LCS.

We make a small modification to the standard GA norm in both the algorithms. We always retain the best solution obtained thus far. That is, once we obtain generation $k+1$ from generation k using selection, crossover and mutation, we knock off the worst solution in generation $k+1$ and replace it with the best solution present among generations k and $k+1$.

6. Computational experiments

To implement the two search strategies we choose the values of the various parameters using the statistical technique called design of experiments. This is described below.

Table 1
DOE factor levels

Sl. no.	Factors	Level 1	Level 2
1	p_s	100	20
2	p_c	0.9	0.7
3	p_m	0.05	0.005

We consider a wide variety of problems ranging from one with 10 machines and 10 jobs to a 40 machine 100 job problem. All programming was done using the C programming language and programs were run on PCs with pentium processors.

6.1. Choice of GA parameters

The efficiency of GAs depend greatly on the control parameters, the population size p_s, the probability of crossover p_c, and the probability of mutation p_m. To parameterize the GA optimally Bagchi and Deb [23] suggested a design of experiment (DOE) approach employing pilot GA runs. To make a choice of these parameters endorsed by statistical experiments we plan a layout of 2^3 factorial experiment. To remove the bias possible due to the initial GA sequences (population) and also to assure robustness in the GA, Bagchi and Deb [23] suggest that performance of 5 randomly produced initial sequences (5 seeds) be tested in each experiment, providing the necessary replication. Moreover, these 5 random starting sequences are kept unaltered in each of the 8 full factorial experiments to involve the principal of variance reduction. A more robust approach would be to consider 5 randomly generated problems and start with a random initial sequence for each problem. The problem and the initial sequence will be kept unaltered while running the algorithms for the 8 parameter combinations as before. The parameter combination for a particular problem dimension is chosen as follows. For each choice of the parameter combination and for each trial run observe the value for the minimum makespan obtained after 300 generations. Choose that parameter combination that gives the smallest average minimum makespan, the average being taken over the 5 runs.

Table 1 shows the two levels of factors considered in the DOE run. Each experiment is run for 300 generations assuming that GA would get sufficiently close to the true solutions.

6.2. Simulation of test problems

We consider five different problem dimensions which include small, moderately large and large problems. The problem dimensions we consider are 10 m/c 10 job, 10 m/c 20 job, 15 m/c 49 job, 25 m/c 60 job and 40 m/c 100 job. We use two methods to simulate the matrices of processing times; one using uniform distributions and the other using normal distributions. In the first five problem sizes given in Table 2, we use uniform distributions. For each of the m jobs we generate a random number say A_i, $i = 1, 2, \ldots, m$, and simulate n numbers from a uniform $(0, A_i)$ distribution. These correspond to the processing times for job i on the n machines. In the last set of five problems listed in Table 2 for the two schemes we use normal distributions. This is done in order to create

Table 2
Best parameter combination to apply GA for 1X and LCS crossover schemes for 10 flowshop problems. First set of five problems are simulated using uniform distributions and the other five using normal

Sl. no.	Coding	Problem dimension	Parameter		
			p_s	p_c	p_m
1	Standard	10 m/c 10 job	100	0.7	0.05
		10 m/c 20 job	100	0.9	0.05
		15 m/c 49 job	100	0.7	0.05
		25 m/c 60 job	100	0.9	0.05
		40 m/c 100 job	100	0.7	0.005
		10 m/c 10 job	100	0.7	0.05
		10 m/c 20 job	100	0.9	0.005
		15 m/c 49 job	100	0.9	0.005
		25 m/c 60 job	100	0.7	0.05
		40 m/c 100 job	100	0.9	0.05
2	LCS	10 m/c 10 job	100	0.9	0.005
		10 m/c 20 job	100	0.7	0.05
		15 m/c 49 job	100	0.7	0.005
		25 m/c 60 job	100	0.9	0.05
		40 m/c 100 job	100	0.7	0.05
		10 m/c 10 job	100	0.9	0.005
		10 m/c 20 job	20	0.9	0.05
		15 m/c 49 job	100	0.9	0.005
		25 m/c 60 job	100	0.9	0.005
		40 m/c 100 job	100	0.9	0.05

problems where the processing times will exhibit widely differing behavior. Corresponding to each job, simulate two positive numbers from uniform distributions having large range. Using these as the mean and variance we then simulate n normal variates and take their absolute values. This gives the processing times for a particular job on the n machines. The problem now becomes much more difficult. A small perturbation in a string can result in large variations in the makespan.

6.3. Comparison of the algorithms

The result of the DOE trial runs are summarized in Table 2. It describes the best parameter combination to use for different problem dimensions for the two crossover schemes. Before we describe the statistical tests we give a rough summary of the outcome of the DOE experiment. Out of the 300 runs of the GA for each coding scheme (we have 5 problem dimensions, two types of problems (generated using uniform or normal distributions), 5 cases (replicates) and 8 parameter combinations), 89% of the runs resulted in the new coding scheme outperforming the standard method.

Figs. 1–7 give plots of the best performing GAs using both the schemes for seven different problems. The first five figures are cases where normal distribution is used to simulate the problem while the last two figures correspond to uniform distribution. The figures reveal the way in which the

two GAs evolve over the generations. The dotted line represents the makespan for the best solution obtained upto any generation using the 1X operator and the solid line that for the LCS operator. The LCS dominates the 1X operator in most of the simulation runs. The LCS demonstrates an ability to improve even after a large number of iterations, while the 1X operator stagnates or improves very slowly.

Remark. Bagchi [15] showed that in case of a flowshop problem with multiobjective optimization criterion, an elitist selection criterion works better than the one described above. The elitist selection process works by pooling together the parent and the child population and then choosing the best 50% of the sequences as the next generation. Simulation runs similar to the one described above show that the LCS crossover scheme outperforms the standard coding scheme in case of the elitist GA also.

6.4. Statistical tests

In this section we use statistical tests to show that there is a statistically significant improvement in performance of the LCS code over the standard algorithm.

We wish to test the hypothesis $\mathbf{H_0}$: $\mu_1 = \mu_2$, against the alternative $\mathbf{H_a}$: $\mu_1 > \mu_2$, where μ_1 and μ_2 are the average optimal makespan obtained by the standard and LCS coding schemes, respectively.

For each of the problem dimensions given in Table 2 (recall that, for the first five problems we use the uniform distributions as described in Section 7, and for the subsequent five cases we use normal distributions), we consider a sample of 15 problems, and observe the optimal makespan using the two GA algorithms after 300 generations. For each algorithm we use the parameter values as given in Table 2. A paired difference test is then performed, since we run both the algorithms on the same set of problems with the same initial population. Note that the optimal parameter values are part of the design of the algorithm. If we denote by d_i, $i = 1, \ldots, 15$, the optimal makespan of standard GA minus that for the LCS code, then the test statistic is given by

$$t = \frac{\bar{d}}{s_{\mathrm{d}}/\sqrt{n}},$$

where $\bar{d}$ is the average of the observed d_i values and s_{d} their standard deviations.

6.5. Result of statistical tests

The t-statistics for the seven different problem dimensions given in Table 2 is summarized in Table 3. Let α denote the level of significance (type I error) of the test and let t_α denote the value for which the area to the right of a student's t-distribution with $n-1=14$ degrees of freedom equals α. If the observed t value is larger than t_α, then we reject $\mathbf{H_0}$. The value $\alpha = 0.01$ is indicative of a highly significant result and $\alpha = 0.05$ implies a significant result. $t_{0.01} = 2.624$, and $t_{0.05} = 1.761$. It can be seen from Table 3 that in all but three cases we reject the null hypothesis with $\alpha = 0.05$, and in all but two cases we reject it at $\alpha = 0.01$. Thus the tests suggest that the LCS code yields a statistically significant improvement over the standard implementation.

Table 3
Results of statistical tests. The first five correspond to problems simulated using uniform distribution and the subsequent five using normal

Sl. no.	Problem dimension	$\bar{d}$	s_d	Test statistic (t)
1	10 m/c 10 job	2.9330	3.7122	3.0602
2	10 m/c 20 job	68.7332	96.972	2.7451
3	15 m/c 49 job	475.4667	602.0693	3.0586
4	25 m/c 60 job	565.2656	904.9646	2.4191
5	40 m/c 100 job	2063.0667	1968.6517	4.0587
6	10 m/c 10 job	7.0	23.5586	1.1508
7	10 m/c 20 job	163.7332	208.8705	3.0360
8	15 m/c 49 job	1795.2667	1092.1808	6.3662
9	25 m/c 60 job	2252.0667	2579.1809	3.3818
10	40 m/c 100 job	870.3333	6128.76732	0.5500

7. Concluding remarks

We considered solving a flowshop problem with the objective of minimizing the makespan. We use GA to solve this problem since it is difficult to obtain exact optimal solutions in a reasonable amount of time. We redesign the standard GA implementation by using structural information from the problem. We do this by examining the way in which the jobs in a sequence interact with each other to produce better or worse solutions. Computational results show that the redesigned implementation performs better than the standard GA.

The modification we propose can be easily adapted to solve similar sequencing and scheduling problems. Using information about the problem structure seems essential for better use of search algorithms like GAs.

Acknowledgements

We would like to thank a referee for bringing the important work of Djerid and Portmann [17] to our attention. We are grateful to Marie-Claude Portmann for supplying us with several papers in addition to the above one. We would like to thank S.K. Gupta for pointing out the existence of a polynomial time algorithm for solving the longest common subsequence problem. Our thanks also go to Tapan Bagchi for many stimulating discussions.

References

[1] Johnson SM. Optimal two and three stage production schedules with setup times included. Naval Research and Logistics Quarterly 1954;1:61–8.
[2] Holland JH. Adaption in natural and artificial systems. Ann Arbor: The University of Michigan Press, 1975.
[3] Goldberg DE. GAs in search, optimization and machine learning. Reading, MA: Addison-Wesley, 1989.
[4] Davis L, editor. Handbook of genetic algorithm. New York: Van Nostrand Reinhold, 1991.

[5] Jog P et al. The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the travelling salesman problem. Proceedings of the Third IGCA, San Mateo 1989, p. 110–5.
[6] Starkweather T et al. A comparison of genetic sequencing operators. Proceedings of Fourth IGCA, 1991, p. 69–76.
[7] Ulder NLJ. Genetic local search algorithms for the travelling salesman problem. In: Schwefel HP, Manner R, editors. Parallel problem solving for nature. Berlin: Springer, 1991. p. 109–16.
[8] Fox BR, McMahon MB. In: Rawlins GJE, editor. Genetic operations for sequencing problems, foundations of genetic algorithms. San Mateo: Morgan Kaufmann Publishers; 1991, p. 284–300.
[9] Ischibuchi H, et al. Genetic algorithms and neighbourhood search algorithms for fuzzy flowshop scheduling problems. Fuzzy Sets and Systems 1994;67:81–100.
[10] Osman IH, Potts CN. Simulated annealing for permutation flow-shop scheduling. OMEGA 1989;17:551–7.
[11] Ogbu FA, Smith DK. The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. Computers and Operations Research 1989;17:243–53.
[12] Reeves CR. Genetic algorithm for flowshop sequencing. Computers and Operations Research 1995;15:5–23.
[13] Taillard E. Some efficient heuristics methods for flow-shop sequencing problem. European Journal of Operations Research 1991;47:65–74.
[14] Murata T, et al. Genetic algorithm for flowshop scheduling problems. Computers and Industrial Engineering 1996;30(4):1061–71.
[15] Bagchi TP. Multi-objective scheduling by genetic algorithms. Boston: Kluwer Academic Publishers, 1999.
[16] Pinedo M. Scheduling theory, algorithms and systems. Englewood Cliffs: Prentice-Hall, 1995.
[17] Djerid L, Portmann M-C. How to keep good schemata using crossover operators for the permutation problems. International Transactions in Operations Research 2000;7(6):637–51.
[18] Portmann MC, Vignier A. Performance study on crossover operators keeping goos schemata for some scheduling problems. 2002, preprint.
[19] Davis L. Job-shop scheduling with genetic algorithms. Proceedings of the First International Conference on Genetic Algorithms and their Applications. Hillsdale: Lawrence Erlbaum. p. 136–40.
[20] Falkenauer E, Bouffouix S. A genetic algorithm for job shop. Proceedings of the IEEE International Conference on Robotics and Automation, vol. 1. Sacramento. p. 824–9.
[21] Caux C, Pierreval H, Portmann M-C. Les Algorithms Gntiques et leur application aux problems d'ordonnancement. Rairo-APII 1995;29(4–5):409–43.
[22] Djerid L, Portmann M-C, Villon P. Performance analysis of permutation crossover genetic operators. Journal of Decision Systems 1996;4(1/2):157–77.
[23] Bagchi TP, Deb K. Calibration of GA parameters: the design of experiment approach. Computer Science and Informatics 1996;26(3):46–56.
[24] Baker KR. Introduction to sequencing and scheduling. New York: Wiley, 1974.
[25] Corman TH, Leiserson CE, Rivest RL. Introduction to algorithms. New York: MIT Press, McGraw-Hill, 1990.

Srikanth K. Iyer is faculty of the Department of Mathematics at the Indian Institute of Technology, Kanpur, India. He received his Ph.D. in Probability and Applied Statistics from the University of California at Santa Barbara. His research interests include the use of statistical tools for improving performance of search methods, modeling of network traffic, branching particle systems, estimation for censored data, mathematical finance, queueing systems and estimation and testing for heavy tailed distributions.

Barkha Saxena completed her M.Sc. from the Indian Institute of Technology, Kanpur, India. She did her Master's dissertation under the guidance of Srikanth Iyer. She is currently working as Analytics Engineer for Fair Isaac, Santa Barbara, USA.