Pergamon

0305-0548(93)E0015-L

Computers Ops Res. Vol. 22, No. 1, pp. 15–24, 1995

0305-0548/95 $7.00 + 0.00

A GENETIC ALGORITHM FOR THE JOB SHOP PROBLEM†

FEDERICO DELLA CROCE,‡ ROBERTO TADEI§ and GIUSEPPE VOLTA¶
D.A.I., Politecnico di Torino, Corso duca degli Abruzzi 24, 10129 Torino, Italy

Scope and Purpose—Job shop scheduling is a strongly NP-hard problem of combinatorial optimization and one of the most well-known machine scheduling problems. Scope of this paper is to present some improvements obtained in dealing with this problem using a heuristic technique based on Genetic Algorithms.

Abstract—In this paper we introduce a genetic algorithm whose peculiarities are the introduction of an encoding based on preference rules and an updating step which speeds up the evolutionary process. This method improves on the results gained previously with Genetic Algorithms and has shown itself to be competitive with other heuristics. The same algorithm has been applied to flow shop problems, revealing itself to be considerably more effective than Branch and Bound techniques.

1. INTRODUCTION

Deterministic machine scheduling problems can be defined [1] in terms of the following characteristics: the machine environment, the kind of job involved and the optimality criteria. Using this classification we consider the problem $J//C_{max}$ which can be defined as follows. A set of m machines and a set of n jobs are given; job i consists of a set of operations that have to be processed in a specified sequence; each operation $o_{i,j}$ has to be processed on a definite machine j and has a processing time $t_{i,j}$ which is deterministically known. Neither release times nor due dates are specified, only one machine is available for each kind, no preemption is admitted. The cost function to minimize is the makespan, i.e. the maximum of job completion times (C_{max}).

A schedule defines the time intervals in which the operations are processed and it is feasible only if it complies with the following constraints: each machine processes only one operation at a time and the operation sequence is respected for every job.

The problem involves finding a feasible schedule which minimizes the makespan. It has been demonstrated [2] that this problem is NP-hard. Even a relatively simple case, like the Muth and Thompson instance with only 10 jobs and 10 machines [3], proved very difficult to solve with exact methods.

Nakano and Yamada [4] have proposed an approach based on Genetic Algorithms while Falkenauer and Bouffouix [5] dealt with a job shop problem with release times and due dates by choosing a specially defined cost function. Among the other attempts, significant results have been obtained with Simulated Annealing [6] and Tabu-Search [7].

In Section 2 the backbones of the Genetic Algorithms are presented and special attention is paid to applications involving ordering problems; in Section 3 the basic scheme of our genetic algorithm is introduced; in Section 4 a technique to extend the search space is presented; in Section 5 an

†This work has been partially supported by Camera di Commercio di Torino and by Ministero dell'Università e della Ricerca Scientifica, Progetto di Recerca Scientifica MURST 40%—M.O.D.

‡Federico Della Croce obtained his Ph.D. degree in Computer Science and Systems Engineering at Politecnico di Torino. His interests are in production scheduling. Presently he is at the Department of Automatica and Informatica, Politecnico di Torino. He has published papers in international journals and conference proceedings.

§Roberto Tadei is associate professor of Operations Research at Politecnico di Torino. His main research interests are sequencing and scheduling, logistics and graphs. He is member of international scientific associations and belongs to the editorial board of international journals. He has published two books and many papers in journals and conference proceedings.

¶Giuseppe Volta received the Laurea degree in electronic engineering from Politecnico di Torino in 1992. He is presently at the Department of Automatica and Informatica, Politecnico di Torino. His research interests are in the area of machine scheduling.

updating technique is described and in Sections 6 and 7 the computational results are presented and discussed.

2. GENETIC ALGORITHMS

Genetic Algorithms are an optimization technique for functions defined over finite domains. They were first proposed by John Holland in his text "Adaption in natural and artificial systems" [8] and have been successfully applied to a great variety of problems [9]. Their name refers to their imitation of natural evolutionary processes.

2.1. Scheme of a genetic algorithm

Genetic algorithms are applied to optimization problems in the following way:

(1) The search space of all possible solutions is mapped onto a set of finite strings; the alphabet of symbols is of finite dimensions, in a typical case it includes two symbols (0 and 1). Each string (*chromosome*) has a corresponding point in the search space. A symbol in the string is called a *gene*. An iteration threshold is chosen, typically a number of generations.

(2) A set of solutions is selected (usually at random) and this constitutes the initial *population*, i.e. the first generation.

(3) A *fitness* is computed for each of the individuals; *fitness* is a parameter which reflects the quality of the solution represented by the individual.

(4) A set of individuals is selected to reproduce itself, the selection takes place in a random way but with a probability proportional to fitness.

(5) From the selected set some progeny is generated by applying different genetic operators.

(6) Some of the old individuals are eliminated from the population to allow the entrance of the new ones.

(7) The new individuals' fitness is computed and they are included in the population. This step marks the end of a generation.

(8) IF the iteration threshold is reached, THEN the process is finished and the best chromosome is returned, ELSE GO TO step 4.

2.2. Operators

Among the classic genetic operators the most commonly used are:

Crossover. Crossover produces two chromosomes ("children"), starting from a couple of chromosomes ("parents"). The simplest one is the one-point crossover: a cutting site is chosen at random and the "children" are obtained by each taking the first part from one parent and the second from the other.

```
parent 1   1 0 1 1|1 1
parent 2   1 1 1 0|0 0
                   ^ cutting site
child 1    1 0 1 1|0 0
child 2    1 1 1 0|1 1
```

In the two-point crossover two cutting sites are chosen and children are obtained from parents by exchanging the part situated between the two cutting sites.

Mutation. Mutation operates on a single chromosome; one element is chosen at random from the chain of symbols and is exchanged with another symbol from the alphabet.

```
1 0 0 1 1 1
      ^ mutation
1 0 0 1 0 1
```

Parameters. There are several parameters which characterize a genetic algorithm, but their

optimum values cannot be ascertained by applying fixed rules. A number of studies [10] have been made on the standard genetic algorithm. Among the most important parameters are the population size, the number of new individuals created in each generation and the probability of applying the different operators. These same operators can be redefined in order to adjust themselves to the chosen encoding or to the problem characteristics.

2.3. Why genetic algorithms work?

To understand how Genetic Algorithms work we have to analyze the effects of selection and reproduction over the population, finding out the way in which good characters are transmitted from parents to children. Let us consider a chromosome, i.e. a string of symbols taken from an alphabet, we call *scheme* [8] a string containing symbols from the alphabet plus the # "*don't care*" symbol. We will say that a chromosome contains a schema if the two strings are coincident a part of the # symbol.

Example: chromosome 0 1 1 0 1 contains 32 *schemata* including:

0	#	#	0	#		#	1	#	#	#		0	1	1	#	1
#	#	#	#	#		0	1	1	0	1		0	#	1	#	1

If a *schema* is highly related to high fitness function values it will be called *building block*. Holland's Schema Theorem [8] says that, if reproduction chances are proportional to chromosome fitness, *schema* occurring in chromosome with above-average fitness will tend to occur more frequently in the next generation, while those with below-average fitness will tend to occur less frequently [11]. It has been showed [12] that the number of *schemata* that are processed during each generation in a population of n strings is $O(n^3)$: this is called *implicit parallelism.*

2.4. Genetic algorithms and scheduling

In scheduling and ordering problems, encoding is usually such that the chromosomes are the permutation of a basic string. A classic example is the "Travelling Salesman Problem" (TSP). Let us consider for instance the TSP with 7 cities. A permutation of the numbers from 1 to 7 is a chromosome which codifies a possible solution; this means that every symbol has to appear only once so that the chromosome makes sense. It is obvious that by applying mutation operators and traditional crossover, illegal chromosomes are obtained.

On the contrary the inversion is applicable without any difficulty since it produces a permutation of the starting chromosome.

It is then necessary to redefine the crossover and mutation so that feasible solutions are obtained by their application. Partially Mapped Crossover (PMX) is suitable for operating on permutations. It has been demonstrated [13] that this new operator satisfies the fundamental properties enunciated by Holland, namely that it behaves in such a way that the best schemata reproduce themselves better than the others. Goldberg himself [12] describes a variant called OX (Order Crossover) which tends to transmit the relative positions of genes rather than the absolute ones. In the OX the chromosome is considered to be circular since the operator is devised for the TSP. In the job shop problem relative position are important, but, as pointed out by Falkenauer and Bouffouix [5] the chromosome cannot be considered circular. For this reason they developed a variant of the OX called Linear Order Crossover (LOX), where the chromosome is considered linear instead of circular. The LOX preserves as much as possible the relative positions between the genes and the absolute positions relative to the extremities of the chromosome. On the set of problems considered by Falkenauer and Bouffouix LOX is more effective than PMX, especially for the bigger problems.

The LOX works as follows:

two cutting sites are chosen randomly;

parent 1	1 2 3 | 4 5 6 | 7 8 9
parent 2	7 6 9 | 4 3 2 | 1 5 8

the symbols that appear in parent 1's *cross-section* (the area situated between the two cutting sites)

are removed from parent 2 leaving some "holes" (showed with H)

parent 2 7 H 9|H 3 2|1 H 8

the holes are slid from the extremities towards the centre until they reach the cross section

parent 2 7 9 3|H H H|2 1 8

the cross section is then substituted with that of parent 1: child number 1 is obtained; child number 2 is obtained with a symmetrical proceeding.

child 1 7 9 3|4 5 6|2 1 8

child 2 1 5 6|4 3 2|7 8 9

The *mutation* operator that we apply (also called *swap mutation*) is defined as follows: two genes are chosen randomly and their positions are exchanged.

7 9 3 4 5 6 2 1 8

^ ^

7 9 1 4 5 6 2 3 8

3. A GENETIC ALGORITHM FOR JOB SHOP

3.1. Encoding

The first step in building a genetic algorithm is the definition of an encoding which enables us to map the solutions as a whole on a set of symbol strings. This is a crucial phase that conditions all the subsequent steps.

A possible encoding consists of defining a string of symbols for each machine (one symbol for each operation). These symbols describe the sequence of the operations. This encoding, besides being easy to read, benefits from the operators defined for TSP, a problem on which several experiments have already been carried out. Unfortunately when problems with more than one machine are considered, the sequence of operations defined by a chromosome may be incompatible with the precedence constraints of the operations, therefore not all the chromosomes define feasible schedules.

Here is an easy example with two jobs and two machines:

(the machine on which the operation takes place is shown in brackets)

Job 1: op1(M1) op2(M2)
Job 2: op3(M2) op4(M1)

Suppose the chromosome is:

M1: op4 op1
M2: op2 op3

Machine M1 should execute operation op4 first but it cannot do this until op3 has been completed; likewise M2 should execute op2 first, but it cannot do it until op1 (which comes after op4) has been completed. A deadlock situation has arisen, therefore the chromosome does not define any feasible schedule.

There are two possible ways of solving this problem: by modifying genetic operators so that they can always produce (through suitable manipulations) chromosomes to which feasible schedules correspond, or by defining a different encoding where all chromosomes produce feasible schedules. We have chosen to adopt the second alternative and to use an encoding scheme which has recently been proposed by Falkenauer and Bouffouix [5] and is defined as follows:

—Each chromosome is formed of several subchromosomes, one for each machine.
—Each subchromosome is a string of symbols, each symbol identifying an operation that has to be processed on the relevant machine.

—Subchromosomes do not describe the operations sequence on the machine, they are *preference* lists; each machine has its own *preference* list which is used at the end of an operation when it has to choose from among the different operations waiting to be scheduled; in that case the operation which appears first in the preference list will be chosen.
—The actual scheduling is deduced from the chromosome through a simulation which analyzes the state of the waiting queues in front of the machines and, if necessary, uses the preference lists to determine the schedule. The proceedings through which the scheduling is obtained is the same as that of a system based on priority rules [14], with the difference that the rules are different for each machine.

3.2. Operators

There is a single composed operator which selects two parents, produces two children using crossover and, eventually, applies mutation to them.

Crossover involves applying LOX operator to each pair of subchromosomes of the parents. Mutation involves applying the *swap mutation* operator described in Section 2.4 to a randomly selected subchromosome.

4. LIMITS OF THE ENCODING SCHEME

It is important to note that Falkenauer and Bouffouix's encoding [5] (see Section 3.1) cannot describe an arbitrary schedule. In particular it cannot describe a schedule where a machine lies idle when there are operations waiting to be scheduled. In order to understand clearly the limits of this scheme it is useful to recall some definitions:

Definition 1

A scheduling is *active* if no operation can be started earlier without delaying another operation or violating the precedence constraints. It has been demonstrated that the optimal schedule is an active one.

Definition 2

A schedule is *non-delay* if no machine lies idle when there is at least one job waiting to be scheduled on that machine [14]. The set of non-delay schedules is a proper subset of the active ones, therefore an optimal solution, which must be an active schedule, is not necessarily non-delay: an instance of this case is reported in [14].

To conclude we can observe that the adopted encoding scheme only generates non-delay schedules, so we cannot be sure that the optimal solution is encoded.

4.1. The lookahead evaluation

We applied the algorithm to a series of problems where the optimal solution was known, finding a certain variability in the performance in the different problems. Particularly in problems where the results were worse than average, no improvement was obtained even if the number of repetitions was notably increased. This fact is easily explained because for those problems the minimum makespan value of the set of non-delay schedules is a long way from the optimal value. In these cases the encoding sets a limit to the performance.

To overcome this limit we have tried to extend the search space outside the non-delay schedules set using a new evaluation function which we will call *lookahead evaluation*; the chromosome shape is unchanged but the interpretation changes, i.e. we change the simulation method through which the actual schedule is derived.

In normal encoding, when a machine ends an operation (or at the beginning when all machines are free) the next scheduled operation is the one with the highest priority among those waiting. Using the lookahead-evaluation when a machine j ends an operation, $o_{\bar{i},j}$ is the *eligible* operation, the one with the highest priority among those waiting and $t_{\bar{i},j}$ is the completion time of $o_{\bar{i},j}$ in case it is really scheduled.

Before scheduling a "Look Ahead" control is performed: this control verifies if operations with a higher priority than $o_{\bar{i},j}$ will become schedulable within time $t_{\bar{i},j}$ and, if the answer is negative, $o_{\bar{i},j}$ is scheduled. Otherwise the machine lies idle until a new operation will become schedulable, at this

time a new $o_{\bar{i},j}$ will be computed and "Look ahead" control performed again. Substantially the simulation method using lookahead evaluation works as follows:

The lookahead simulation method

1. $\forall$ machine j, set $T_j=0$.
2. Choose machine j such that T_j is minimum (in the case of a tie choose at random).
3. Compute set W_j including all the candidate operations ready to work at time T_j (operation $o_{i,j}$ is ready to work if there are no previous operations belonging to the same job i that have not yet finished being processed).
4. Compute the eligible operation $o_{\bar{i},j} \in W_j$, with duration $d_{\bar{i},j}$; $t_{\bar{i},j}=T_j+d_{\bar{i},j}$ will be the completion time of $o_{\bar{i},j}$ if scheduled at time T_j.
5. Compute set $\overline{P_{\bar{i},j}}$ including all the operations that are positioned before $o_{\bar{i},j}$ in the original sequence and that will be ready to start before the instant $t_{\bar{i},j}$.
6. IF $\overline{P_{\bar{i},j}}=\varnothing$ THEN schedule $o_{\bar{i},j}$ and set $T_{\text{old}}=T_j$, $T_j=t_{\bar{i},j}$
   ELSE set machine j to an idle state.
7. Let $o_{x,j}$ be the operation ended at time T_{old} on machine j (if this operation exists). Let y be the machine where the operation following $o_{x,j}$ in the same job x (if exists) is going to be processed, IF y lies idle set $T_y=T_j$.
8. IF all operations have not been scheduled, GOTO 2
   ELSE STOP.

Lookahead evaluation in most cases widens the search space, which in the worst case remains the same (proof will be given in Section 5.1), but does not codify all the active schedules and does not guarantee to reach an optimal schedule.

5. UPDATING

In Section 3.1 we pointed out that the chromosome does not describe an actual schedule, but it gives preference lists. We can add a remark about the equivalence of chromosomes.

5.1. Classes of equivalence

Denote by C the chromosomes space. As different chromosomes can produce equal schedules, a partition of C into classes of equivalence exists. The following proposition holds true:

Proposition 1

Let C be the chromosome space and let S be the space of non-delay schedules and let F: $C \to S$ be the function which generates a non-delay schedule starting from a chromosome according to the rules previously described. Let $A \in C$, where $A=\{A_{i,j}\}$ is a matrix with rows which identify the machines and columns for the operations, so that in a row j we have the preference list for all the operations on machine j, if $F(A)=B$ then $F(B)=B$.

Proof

We observe first that a schedule obtained from a chromosome can be read itself as a chromosome.

Let us assume that $F(B)=D$. We will demonstrate that $B=D$.

To do this it is sufficient to show that defining by O_j the set of operations schedulable on a given machine j at a given time t_0, if O_j is the same for both A and B, then the chosen operation will also be the same.

(a) Let A be the current chromosome and suppose we are scheduling ith operation on machine j at time t_0.

Let O_j be the set of operations schedulable on machine j at time t_0.

Let $o_{x,j}$ be such that:

$o_{x,j} \in O_j$ and $o_{x,j}$ precedes every other operation $\in O_j$ in the jth row of A.

$o_{x,j}$ will be scheduled, so $B_{i,j}=o_{x,j}$.

(b) Now let B be the current chromosome and suppose we are operating in the same conditions as (a), i.e. we have to schedule the ith operation on machine j at time t_0 with the same set O_j of schedulable operations.

So $o_{x,j} \in O_j$ and $o_{x,j}$ precedes every other operation $\in O_j$ in the jth row of B as B is the consequence of the simulation of chromosome A.

Thus $o_{x,j}$ will be scheduled and $D_{i,j} = o_{x,j} = B_{i,j}$.

We have demonstrated that B is equivalent to A if the set O_j is the same. As at the very first scheduling decision, set O_j is independent from the chromosome, at the beginning of the simulation we will have the same O_j in case (a) and in case (b). For each scheduled operation, the set O_j will depend only on the previous decisions, so it continues to be the same for case (a) and (b) as the operations scheduled are the same. The proposition follows.

For the case of the lookahead evaluation (see Section 4.1), the proof is the same, simply substituting set O_j with $O_e = O_j +$ (every operation that becomes schedulable within time $t_{\bar{i},j}$), where $t_{\bar{i},j}$ is the time defined in Section 4.1.

From the above proposition it follows that in each class of equivalence there is one chromosome which is an eigenvector of F with eigenvalue equal to 1, i.e. it coincides with the schedule it produces.

Now we can prove that using lookahead evaluation the search space is widened or, at least, remains the same.

Proposition 2

If a schedule can be deduced from a chromosome using normal evaluation, it can also be deduced from a chromosome using lookahead evaluation.

Proof

As it has been showed proving proposition 1, for each schedule S' in the search space of the normal evaluation, exists a class C' of chromosomes including an eigenvector. If we consider all the chromosomes belonging to that class and we evaluate them using lookahead evaluation they will produce a set of solution $S_1 \ldots S_n$, the partition of all chromosomes into classes of equivalence changes, but if we prove that at least one of the chromosomes belonging to the original class C' still generates schedule S', proposition 2 will follow.

We are going to prove that the eigenvector of class C' is the one that still produces S'.

Let us consider the eigenvector of C' and the normal evaluation procedure, as explained in the proof of proposition 1, the choice of the operation to be scheduled on machine j at a given time, depends on the set O_j of schedulable operations and on the preference list specified by the jth row of the chromosome. Being the chromosome an eigenvector, let x be the first operation in the preference list (excluding the already scheduled operations), x will be in the set O_j and will be scheduled. If we use lookahead evaluation the set of schedulable operations is $O_e + O_j +$ (every operation that becomes schedulable within time $t_{\bar{i},j}$), but x is still the operation with the greatest priority, so the extension of set O_j has no effect on the decision.

The proposition follows.

5.2. *Eigenvectors and updating*

We have supposed that among all the elements in a class, the one with the greatest power to transmit the specific characters of the class to the descendents is the eigenvector; consequently we have modified the algorithm in such a way as to substitute each new chromosome with the eigenvector of its own class of equivalence (this is the *updating* step). The computational experience shows that the updating technique improves on the results gained previously with genetic algorithms and shows this method to be competitive with other heuristics.

6. COMPUTATIONAL RESULTS

After testing the algorithm over a set of problems we have chosen the following parameters: crossover proability 1, mutation probability 0.03, population 300, we used *steady-state reproduction* [11, 15] inserting in the population ten (10) new individuals at each generation (in the traditional GA the population is completely replaced at each generation). The number of generations (2971) was chosen in order to produce a total of 30,000 individuals for each run of the algorithm for the problems of Tables 1 and 2, while for the problems of Tables 3 and 4 the number of generation

Table 1. Comparison of our genetic algorithm with Nakano's

Problem	*n*	*m*	OPT	Nakano	Our GA
MT06	6	6	55	55	55
MT10	10	10	930	965	946
MT20	20	5	1165	1215	1178

Table 2. Comparisons between the heuristics

							Our GA		
Problem	*n*	*m*	OPT (UB)	SA best	TS best	SB best	Best	AV	Tav
MT06	6	6	55	55	55	55	55	55.0	223
MT10	10	10	930	930	935	930	946	965.2	628
MT20	20	5	1165	1165	1165	1178	1178	1199.0	675
LA01	10	5	666	666	666	666	666	666.0	282
LA06	15	5	926	926	926	926	926	926.0	473
LA11	20	5	1222	1222	1222	1222	1222	1222.0	717
LA16	10	10	945	956	945	978	979	989.0	637
LA21	15	10	(1048)	1063	1048	1084	1097	1113.6	1062
LA26	20	10	1218	1218	1218	1224	1231	1248.0	1542
LA31	30	10	1784	1784	1784	1784	1784	1784.0	2762
LA36	15	15	1268	1293	1278	1305	1305	1330.4	1880

Table 3. Comparison of our genetic algorithm with Falkenauer and Bouffouix's (remember we are dealing with a maximization problem)

					Our GA	
Problem	*n*	*m*	op	F&B	Best	AV
SMALL	3	3	24	14	14	14
BIG1	6	10	60	61	59	59
BIG2	6	10	60	120	121	120.6
BIG3	6	10	60	117	117	117
GIANT	40	28	250	1818781.8	2279999	2279706.2

was bounded to 971, that means 10,000 individuals. We coded our algorithm in C language and ran it on a PC 486/25.

In the well-known Muth & Thompson 10×10 problem, the makespan of the best schedule found over five runs with our algorithm is 946, while the optimum is 930 and the previous best result obtained with a genetic algorithm was 965 [4]. Our algorithm shows a fairly quick convergence since a makespan of 989 is found after generating 6000 individuals and a makespan of 961 after 9000. Nakano uses a population of 1000 and 150 generations, which with a traditional generational replacement technique means 150,000 individuals. The comparison with Nakano is showed in Table 1 where number of jobs and machines are shown in column *n* and column *m* and the optimum value in column OPT.

Table 2 shows the computational results obtained on a set of 11 problem instances, compared with the best obtained with Shifting Bottleneck [16] (column SB), Simulated Annealing [6] (column SA), Tabu-Search [7] (column TS) and optimum value (the optimum value, when known, is in column OPT otherwise the best upper bound is shown in brackets). For our genetic algorithm (GA) there are three columns indicating the best performance over five runs (Best), the average performance over five runs (AV), and the average computation time in seconds (Tav). A column indicating the worst computation time was not inserted as there was no significant variation from one run to another for any given problem. Problems MT06–MT20 are from [3], LA01–LA36 from [17].

Table 3 shows the performance obtained over five problems instances compared with Falkenauer's results [5] (column F&B). Note that the objective function defined by Falkenauer is:

$$\max \; f = c \sum \text{advances} - \sum \text{tardiness}^2,$$

where $c = 1$ ($\sum$ *advances* corresponds to the sum of the earlinesses), the total number of operation is showed in column op.

Table 4. Comparisons on flow shop problems of our genetic algorithm with the B&B of Applegate and Cook

				Our GA		
Problem	n	m	B&B	Best	AV	Tav
CAV10X10	10	10	*	1100	1102.6	165
CAV10X10B	10	10	(1135)	1162	1166.6	161
CAV10X10C	10	10	*	1031	1031.0	165
CAV5X20	5	20	1349	1349	1349.0	195
CAV5X20B	5	20	1282	1285	1285.0	200
CAV5X20C	5	20	1321	1357	1357.0	199
CAV20X5	20	5	*	1256	1258.0	172
CAV20X5B	20	5	*	1090	1090.6	173
CAV20X5C	20	5	(1120)	1121	1134.2	172
V15X15	15	15	*	1594	1609.4	370
V15X15B	15	15	*	1683	1685.6	369
V15X15C	15	15	*	1563	1576.2	371
V10X20	10	20	*	1696	1696.0	344
V10X20B	10	20	*	1621	1631.8	340
V10X20C	10	20	*	1700	1704.8	344
V20X10	20	10	*	1645	1651.8	330
V20X10B	20	10	*	1590	1598.4	333
V20X10C	20	10	*	1561	1575.8	331

Problems "SMALL", "BIG1", "BIG2", "BIG3", and "GIANT" are the same used in [5].

Our genetic algorithm has also been applied without variations to flow shop problems. For a flow shop problem a chromosome always describes a feasible schedule, therefore it would not be necessary to interpret the chromosome as a preference list and obtain the schedule through a simulation. However the reduction in the search space obtained by this encoding produces a considerable improvement in performance, all the more significant when the problem complexity increases.

Table 4 shows the performance of our genetic algorithm over a set of 18 flow shop problems compared with Applegate & Cook's Branch and Bound method [18] (column B&B). These problems were randomly generated with an operation length between 1 and 100 and a uniformly distributed probability. With the exception for 5 instances, the Branch and Bound, running for 10 million iterations and using the value found by our algorithm as upper bound, does not find any better schedule (in this case the B&B column is asterisked). If the Branch and Bound found the optimum value this is written in column B&B, otherwise the best upper bound is put in parentheses.

7. CONCLUSIONS

In this paper we introduce a genetic algorithm in which the encoding is based on preference rules. The peculiarity of this encoding system is that it always produces feasible chromosomes. The computational results are better than those obtained with a previous genetic approach [4]. We also present lookahead evaluation that widens the search space and an updating technique based on classes of equivalent chromosomes. Both these techniques produce an improvement in the performance of the algorithm. The results are comparable with those obtained using the shifting bottleneck [16], simulated annealing [6], and tabu-search methods [7], but at cost of longer computation times. The algorithm also performs well on flow shop problems. We remark the robustness of this approach as it can be applied just with slight modifications to all types of job shop problems with regular performance measures.

Improvements in effectiveness could be obtained by widening the search space even further so that all active schedules are included, even though this would probably slow down the algorithm. To speed up the convergence one possibility would be the hybridization [11] of the algorithm with job shop specific techniques, for example choosing an initial population partially produced with other quicker heuristics.

Acknowledgements—The authors are grateful to M. Dorigo, M. Trubian and M. Pirlot for their useful suggestions and to E. Falkenauer who provided us with data of his instances.

REFERENCES

1. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, Sequencing and scheduling: algorithms and complexity. Report BS-R8909, Department of Operations Research, Statistic, and System Theory, Centre for Mathematics and Computer Science, Amsterdam (1989).
2. M. R. Garey, D. S. Johnson and R. Sethi, The complexity of flowshop and job-shop scheduling. *Math. Opl Res.* **1**, 117–129 (1976).
3. J. F. Muth and G. L. Thompson, *Industrial Scheduling*. Prentice Hall, Englewood Cliffs, N.J. (1963).
4. R. Nakano and T. Yamada, Conventional genetic algorithm for job shop scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Application* (1991).
5. E. Falkenauer and S. Bouffoix, A genetic algorithm for job shop. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* (1991).
6. P. J. M. Van Laarhoven, E. H. L. Aarts and J. K. Lenstra, Job shop scheduling by simulated annealing. *Opns Res.* **40**, 113–125 (1992).
7. M. Dell'Amico and M. Trubian, Applying tabu search to the job shop scheduling problem. *Ann. Ops Res.* **41**, 231–252 (1993).
8. J. H. Holland *Adaption in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor (1975).
9. M. Dorigo, Genetic Algorithms: the state-of-the-art and some research proposals. Report No. 89-058, Dilartimento di Elettronica, Politecnico di Milano (1989).
10. K. A. De Jong, An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. dissertation, University of Michigan (1975).
11. L. Davis, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, Amsterdam (1991).
12. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, New York (1989).
13. D. E. Goldberg and R. Lingle, Alleles, loci and the traveling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and Their Application* (1985).
14. S. French, *Sequencing and Scheduling*. Ellis Horwood, Chichester (1982).
15. G. Syswerda, Uniform Crossover in genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications* (1989).
16. J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling. *Mgmt Sci.* **34**, 391–401 (1988).
17. S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie Mellon University (1984).
18. D. Applegate and W. Cook, A computational study of the job shop scheduling problem. *ORSA J. Comput.* **3**, 149–156 (1991).