Pergamon

Computers ind. Engng Vol. 30, No. 4, pp. 983–997, 1996

S0360-8352(96)00047-2
0360-8352/96 $15.00 + 0.00

A TUTORIAL SURVEY OF JOB-SHOP SCHEDULING PROBLEMS USING GENETIC ALGORITHMS—I. REPRESENTATION

RUNWEI CHENG, MITSUO GEN and YASUHIRO TSUJIMURA

Ashikaga Institute of Technology, Ashikaga 326, Japan

Abstract—Job-shop scheduling problem (abbreviated to JSP) is one of the well-known hardest combinatorial optimization problems. During the last three decades, the problem has captured the interest of a significant number of researchers and a lot of literature has been published, but no efficient solution algorithm has been found yet for solving it to optimality in polynomial time. This has led to recent interest in using genetic algorithms (GAs) to address it. The purpose of this paper and its companion (Part II: Hybrid Genetic Search Strategies) is to give a tutorial survey of recent works on solving classical JSP using genetic algorithms. In Part I, we devote our attention to the representation schemes proposed for JSP. In Part II, we will discuss various hybrid approaches of genetic algorithms and conventional heuristics. The research works on GA/JSP provide very rich experiences for the constrained combinatorial optimization problems. All of the techniques developed for JSP may be useful for other scheduling problems in modern flexible manufacturing systems and other combinatorial optimization problems. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

Machine scheduling problems arise in diverse areas such as flexible manufacturing system, production planning, computer design, logistics, communication, etc. A common feature of many of these problems is that no efficient solution algorithm is known yet for solving it to optimality in polynomial time. The classical job-shop scheduling problem (JSP) is one of the most well-known machine scheduling problems. Informally, the problem can be described as follows: there are a set of jobs and a set of machines. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can process at most one operation at a time. A schedule is an allocation of the operations to time intervals on the machines. The problem is to find a schedule of minimum length.

During the last three decades, the problem has captured the interest of a significant number of researchers and a huge amount of literature has been published. Among these are the books by Muth and Thompson [1], Conway *et al.* [2], Baker [3], French [4], Rinnooy Kan [5], Coffman [6], Blazewicz *et al.* [7], and most recent book by Morton and Pentico [8].

JSP is among the hardest combinatorial optimization problems [9]. Because of its inherent intractability, heuristic procedures are an attractive alternative. Most conventional heuristic procedures use a *priority rule*, i.e., a rule for choosing an operation from a specified subset of as yet unscheduled operations. In recent years, an interest in using probabilistic search methods to solve JSP has been growing rapidly, such as simulated annealing (SA) [10], tabu search (TS) [11], and genetic algorithms (GA) [12–24, 50]. These approaches comprise the emergence of promise for conquering the combinatorial explosion in a variety of decision-making arenas. Recent papers on GA/JSP are listed in Table 1.

The purpose of this paper and its companion (Part II: Hybrid Genetic Search Strategies) is to give a tutorial survey of recent works on solving classical JSP using genetic algorithms. In Part I, we put our attention on the representation schemes proposed for JSP. In Part II, we will discuss various hybrid approaches of genetic algorithms and conventional heuristics.

2. JOB-SHOP SCHEDULING PROBLEMS

The classical JSP can be stated as follows: there are m different machines and n different jobs to be scheduled. Each job is composed of a set of operations and the operation order on machines is prespecified. Each operation is characterized by the required machine and the fixed processing time. There are several constraints on jobs and machines, such as,

- a job does not visit the same machines twice;
- there are no precedence constraints among operations of different jobs;
- operation can not be interrupted;
- each machine can process only one job at a time;
- neither release times nor due dates are specified.

The problem is to determine the operation sequences on the machines in order to minimize the makespan, i.e., the time required to complete all jobs.

The JSP with makespan objective can be formulated as follows:

$$\min \max_{1 \leqslant k \leqslant m} \{\max_{1 \leqslant i \leqslant n} \{c_{ik}\}\} \tag{1}$$

$$\text{s. t.} \quad c_{ik} - t_{ik} + M(1 - a_{ihk}) \geqslant c_{ih},\ i = 1, 2, \ldots, n \text{ and } h,\ k = 1, 2, \ldots, m \tag{2}$$

$$c_{jk} - c_{ik} + M(1 - x_{ijk}) \geqslant t_{jk},\ i, j = 1, 2, \ldots, n \text{ and } k = 1, 2, \ldots, m \tag{3}$$

$$c_{ik} \geqslant 0,\ i = 1, 2, \ldots, n \text{ and } k = 1, 2, \ldots, m \tag{4}$$

$$x_{ijk} = 0 \text{ or } 1,\ i, j = 1, 2, \ldots, n \text{ and } k = 1, 2, \ldots, m \tag{5}$$

where c_{jk} is the completion time of job j on machine k, t_{jk} is the processing time of job j on machine k, M is a big positive number, a_{ihk} is an indicator coefficient defined as follows:

$$a_{ihk} = \begin{cases} 1, & \text{if processing on machine } h \text{ precedes that on machine } k \text{ for job } i \\ 0, & \text{others} \end{cases}$$

and x_{ijk} is an indicator variable defined as follows:

$$a_{ihk} = \begin{cases} 1, & \text{if job } i \text{ precedes job } j \text{ on machine } k \\ 0, & \text{others} \end{cases}$$

The objective is to minimize makespan. Constraint (2) ensures that the processing sequence of operations for each job corresponds to the prescribed order. Constraint (3) ensures that each machine can process only one job at a time.

Table 1. Recent papers on GA/JSP

Authors	Description
Davis [12]	Preference list-based representation
Nakano *et al.* [13]	Job pair relation-based representation
Falkenauer *et al.* [14]	Preference list-based representation
Bagchi *et al.* [15]	Problem specific representation
Yamada *et al.* [16]	Completion time-based representation
Tamaki *et al.* [17]	Disjunctive graph-based representation
Paredis [18]	Job pair relation-based representation
Holsapple *et al.* [19]	Job-based representation
Fan *et al.* [20]	Operation-based representation
Gen *et al.* [23]	Operation-based representation
Bean [25]	Random key representation
Dorndorf *et al.* [21]	Priority rule-based and machine-based
Croce *et al.* [24]	Preference list-based representation
Kobayashi *et al.* [26]	Preference list-based representation
Norman *et al.* [27]	Random key representation

The integer programming (IP) formulation for JSP was discussed in Baker [3] and more completely in Greenberg [28]. This new IP formulation mentioned above tries to give a better representation of the precedence constraint using a system of linear inequalities.

3. ENCODING PROBLEM

One basic feature of genetic algorithms is that it works on *coding space* and *solution space* alternatively: evolution works on coding space (chromosomes) while evaluation works on solution space as shown in Fig. 1. Natural selection is the link between chromosomes and the performance of their decoded solutions. In Holland's work, encoding is carried out using binary strings. During the last 10 years, various non-string encoding techniques have been created for particular problems, to which classical GA was difficult to apply directly [50].

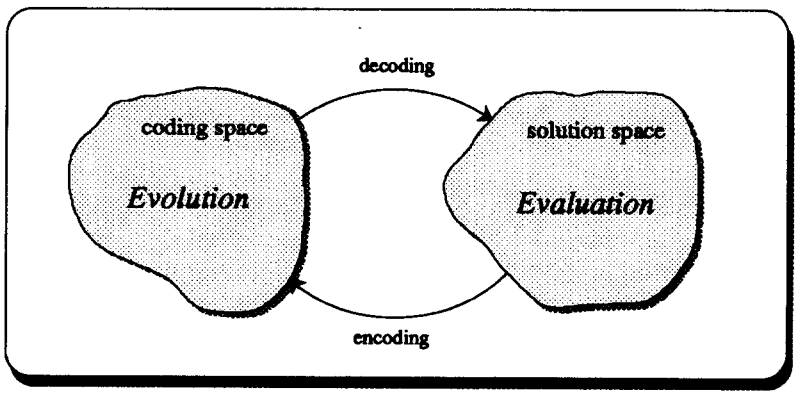

Fig. 1. Coding space and solution space.

How to encode solutions to chromosomes is a key issue for genetic algorithms. For the non-string coding approach, three critical issues emerged concerned with the encoding and decoding between chromosomes and solutions (or the mapping between phenotype and genotype):

- the feasibility of a chromosome
- the legality of a chromosome
- the uniqueness of mapping.

The *feasibility* refers to the phenomenon that whether or not a solution decoded from a chromosome lies in the feasible region of a given problem. The *legality* refers to the phenomenon whether or not a chromosome represents a solution to a given problem as shown in Fig. 2.

The *infeasibility* of chromosomes originates from the nature of the constrained optimization problem. Whatever methods, conventional ones or genetic algorithms, must handle the constraints. For many optimization problems, the feasible region can be represented as a system of equalities or inequalities (linear or non-linear). For such cases, many efficient penalty methods have been

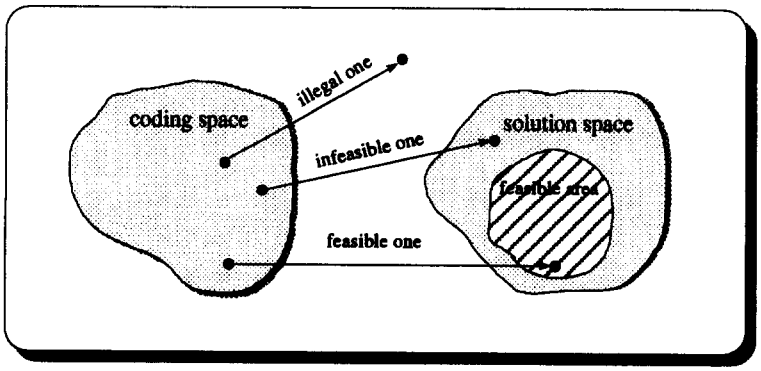

Fig. 2. Feasibility and legality.

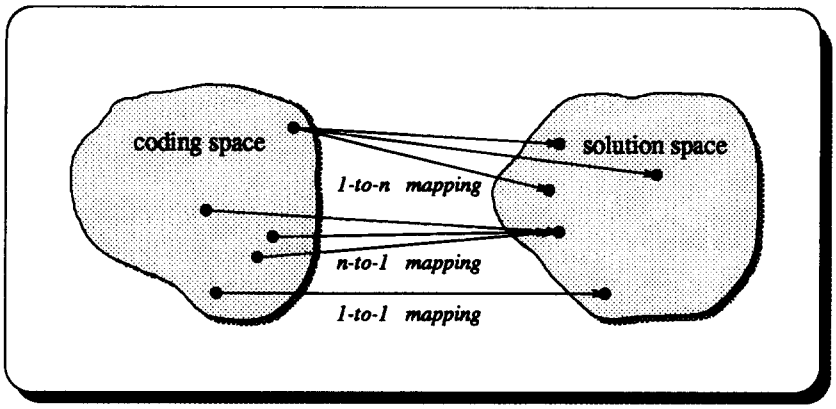

Fig. 3. The mapping from chromosomes to solutions.

proposed to handle infeasible chromosomes [29–31]. In constrained optimization problems, the optimum typically occurs at the boundary between feasible and infeasible area. The penalty approach will force genetic search to approach to optimum from both feasible and infeasible regions.

The *illegality* of chromosomes originates from the nature of encoding techniques. For many combinatorial optimization problems, problem-specific encodings are used and such encodings usually yield illegal offspring by a simple one-cut-point crossover operation. Because an illegal chromosome cannot be decoded to a solution, it means that such a chromosome cannot be evaluated, the penalty approach is inapplicable to this situation. Repairing techniques are usually adopted to convert an illegal chromosome to a legal one. For example, the well-known PMX operator is essentially a kind of two-cut-point crossover for permutation representation together with a repairing procedure to resolve the illegitimacy caused by the simple two-cut-point crossover.

The *mapping* from chromosomes to solutions (decoding) may belong to one of following three cases:

- 1-to-1 mapping
- n-to-1 mapping
- 1-to-n mapping

as shown in Fig. 3. The 1-to-1 mapping is the best one among three types and 1-to-n mapping is the most undesired one.

Because of the existence of the precedence constraints of operations, JSP is not as easy as traveling salesmen problem (TSP) to find out a nature representation. There is not a good representation for the precedence constraints with a system of inequalities. Therefore, a penalty approach is not easily applied to handle such kinds of constraints. Orvosh and Davis [32] have shown that for many combinatorial optimization problems, it is relatively easy to repair an infeasible or illegal chromosome and the repair strategy did indeed surpass other strategies such as rejecting strategy or penalizing strategy. Most GA/JSP researchers prefer to take repairing strategy to handle the infeasibility and illegality. Consequently, one very important issue in building a genetic algorithm for the job-shop problem is to devise an appropriate representation of solutions together with problem-specific genetic operations in order that all chromosomes generated in either initial phase or evolutionary process will produce feasible schedules. This is a crucial phase that conditions all the subsequent steps of genetic algorithms.

4. REPRESENTATION FOR JOB-SHOP SCHEDULING PROBLEM

During the last few years, the following nine representations for the job-shop scheduling problem have been proposed:

- 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 genetic algorithms 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 genetic algorithms 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.

We shall discuss them in turn using a simple example given in Table 2.

4.1. Operation-based representation

This representation encodes a schedule as a sequence of operations and each gene stands for one operation. One natural way to name each operation is using a natural number, like the permutation representation for TSP. Unfortunately because of the existence of the precedence constraints, not all possible permutations of these numbers define feasible schedules. Gen *et al.* proposed an alternative: they named all operations for a job with the same symbol and then interpreted it according to the order of occurrence in the given chromosome [23, 33]. For a n-job and m-machine problem, a chromosome contains $n \times m$ genes. Each job appears in the chromosome exactly m times and each repeating (each gene) does not indicate a concrete operation of a job but refers to a unique operation which is context-dependent. It is easy to see that any permutation of the chromosome always yields a feasible schedule.

A schedule is decoded from a chromosome with the following decoding procedure: (1) firstly translate the chromosome to a list of ordered operations, (2) then generate the schedule by a one-pass heuristic based on the list. The first operation in the list is scheduled first, then the second operation, and so on. Each operation under treatment is allocated in the best available processing time for the corresponding machine the operation requires. The process is repeated until all operations are scheduled. A schedule generated by the procedure can be guaranteed to be an active schedule [3].

Table 2. Example of 3-job and 3-machine JSP

	Operations		
Job	1	2	3
	Processing time		
j_1	3	3	2
j_2	1	5	3
j_3	3	2	3
	Machine sequence		
j_1	m_1	m_2	m_3
j_2	m_1	m_3	m_2
j_3	m_2	m_1	m_3

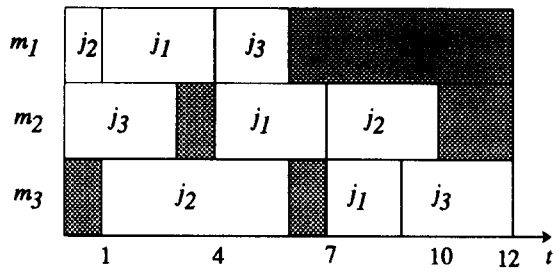

Fig. 4. Decoded active schedule.

Consider the 3-job 3-machine problem given in Table 2. Suppose a chromosome is given as [2 1 1 1 2 2 3 3 3]. Because each job has three operations, it occurs exactly three times in the chromosome. Each gene uniquely indicates an operation and can be determined according to its order of occurrence in the sequence. Let o_{jim} denote the ith operation of job j on machine m. The chromosome can be translated into a unique list of ordered operations of [o_{211} o_{111} o_{122} o_{133} o_{223} o_{232} o_{312} o_{321} o_{333}]. Operation o_{211} has the highest priority and is scheduled first, then o_{111}, and so on. The resulting active schedule is shown in Fig. 4.

Fang *et al.* also proposed a kind of operation-based representation for the job-shop schedule problem [20]. For a $j \times m$ problem, the chromosome is a string containing a $j \times m$ chunk large enough to hold the largest job number j. A chunk is atomic as far as a genetic algorithm is concerned. It is indubitable that one chunk corresponds to one operation, but the description of "chunk" given in their paper is not clear enough for us to be able to code an operation into a chunk. When constructing an actual schedule, they maintain a circular list of incomplete jobs and lists of unscheduled operations for each such job. A chunk provides the information how to find out the operation to be scheduled next among these lists. A chunk is used as the index of a job in the circular list of incomplete jobs, and the first untackled operation of the job will be scheduled into the earliest place where it can fit in.

4.2. Job-based representation

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, and so on 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. Holsapple *et al.* have used this representation to deal with the static scheduling problem in a flexible manufacturing context [19].

Suppose a chromosome is given as [2 3 1]. The first job to be processed is job j_2. The operation precedence constraint for j_2 is [m_1 m_3 m_2] and the corresponding processing time for each machine is [1 5 3]. Firstly, the operations of job j_2 are scheduled as shown in Fig. 5(a). Then the job j_3 is processed, its operation precedence among machines is [m_2 m_1 m_3] and the corresponding processing time for each machine is [3 2 3]. Each of its operations is scheduled in the best available processing time as shown in Fig. 5(b). Lastly, the operations of job j_1 are scheduled as shown in Fig. 5(c).

4.3. Preference list-based representation

This representation was originally proposed by Davis for a kind of scheduling problem [12]. Falkenauer and Bouffouix used it for dealing with a job-shop scheduling problem with release times and due dates [14]. Croce *et al.* applied it to a classical job-shop scheduling problem [24].

For an n-job m-machine job-shop scheduling problem, a chromosome is formed of m subchromosomes, each for one machine. Each subchromosome is a string of symbols with length n, and each symbol identifying an operation that has to be processed on the relevant machine. Subchromosomes do not describe the operation sequence on the machine, they are *preference lists*;

each machine has its own preference list. The actual schedule is deduced from the chromosome through a simulation, which analyses the state of the waiting queues in front of the machine and if necessary uses the preference lists to determine the schedule, that is, the operation which appears first in the preference list will be chosen.

Now we show how to deduce an actual schedule from a given chromosome. Consider the same example given in Table 2. Suppose a chromosome to be: [(2 3 1) (1 3 2) (2 1 3)]. The first gene (2 3 1) is the preference list for machine m_1, (1 3 2) for machine m_2 and (2 1 3) for machine m_3. From these preference lists we can deduce that the first preferential operations are job j_2 on machine m_1, j_1 on m_2, and j_2 on m_3. According to the given operation precedence constraints, only j_2 on m_1 is schedulable, so it is scheduled on m_1 first as shown in Fig. 6(a). The next schedulable operation is j_2 on m_3 as shown in Fig. 6(b). Now the current preferential operations are j_3 on m_1, j_1 on m_2 and m_3. Because all of them are not schedulable at the current time, we look for the second preferential operations in each list. They are j_1 on m_1, j_3 on m_2 and m_3. The schedulable operations are j_1 on m_1 and j_3 on m_2. We schedule then as shown in Fig. 6(c). The next schedulable operations are j_3 on m_1 and j_1 on m_2 as shown in Fig. 6(d). After that the schedulable operations are j_2 on m_2 and j_1 on m_3 as shown in Fig. 6(e). The last operation to be scheduled is j_3 on m_3 as shown in Fig. 6(f). Now we complete schedule deduction from a chromosome and get a feasible schedule with makespan 12. With the decoding procedure, all possible chromosomes always procedure feasible schedules.

Croce *et al.* have argued that the deduction procedure only generates non-delay schedules, so we cannot be sure that the optimal solution is encoded. They gave a rather complex lookahead evaluation procedure to help the deduction procedure get an active schedule. But the argument is not true. When generating a non-delay schedule, we must first identify a critical machine which can start earliest and then select an operation which can be processed earliest on the critical machine [3]. Because in the process of deduction, the next operation is selected according to the preference lists, we are not certain to get a non-delay schedule, but an active schedule. For example, the schedule given in Fig. 6(f) is an active schedule, but a non-delay schedule because machine m_3 remains idle at time 6 when it could start on the third operation of job j_3.

Kobayashi *et al.* also adopted this kind of representation in their work [26]. The difference with the above method is that they use Giffler and Thompson's heuristic to decode a chromosome into a schedule.

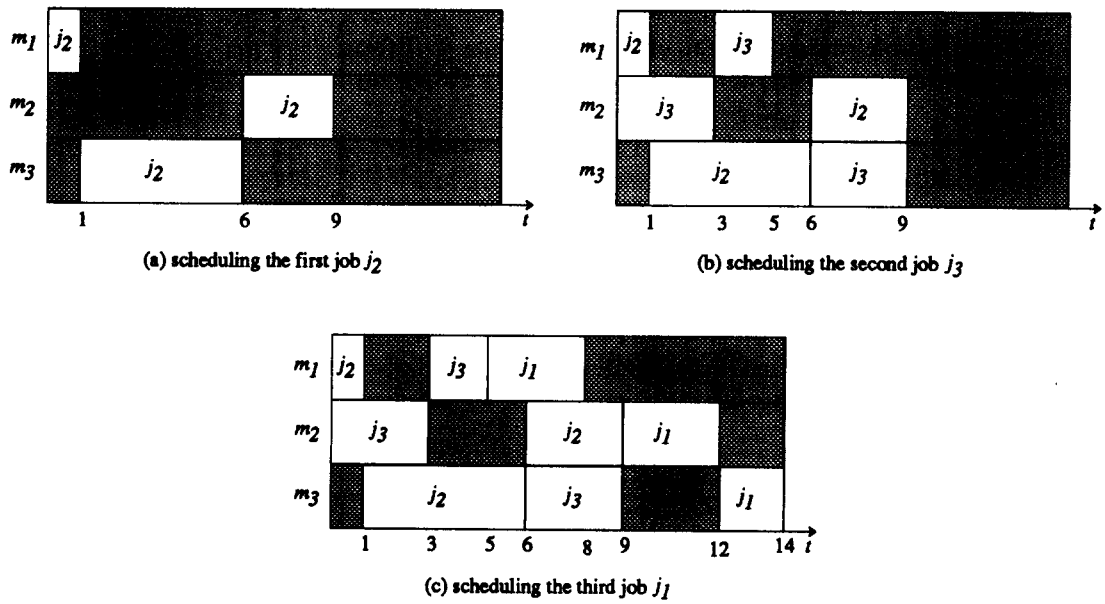

Fig. 5. Scheduling jobs: (a) for the first job j_2; (b) for the second job j_3; and (c) for the third job j_1.

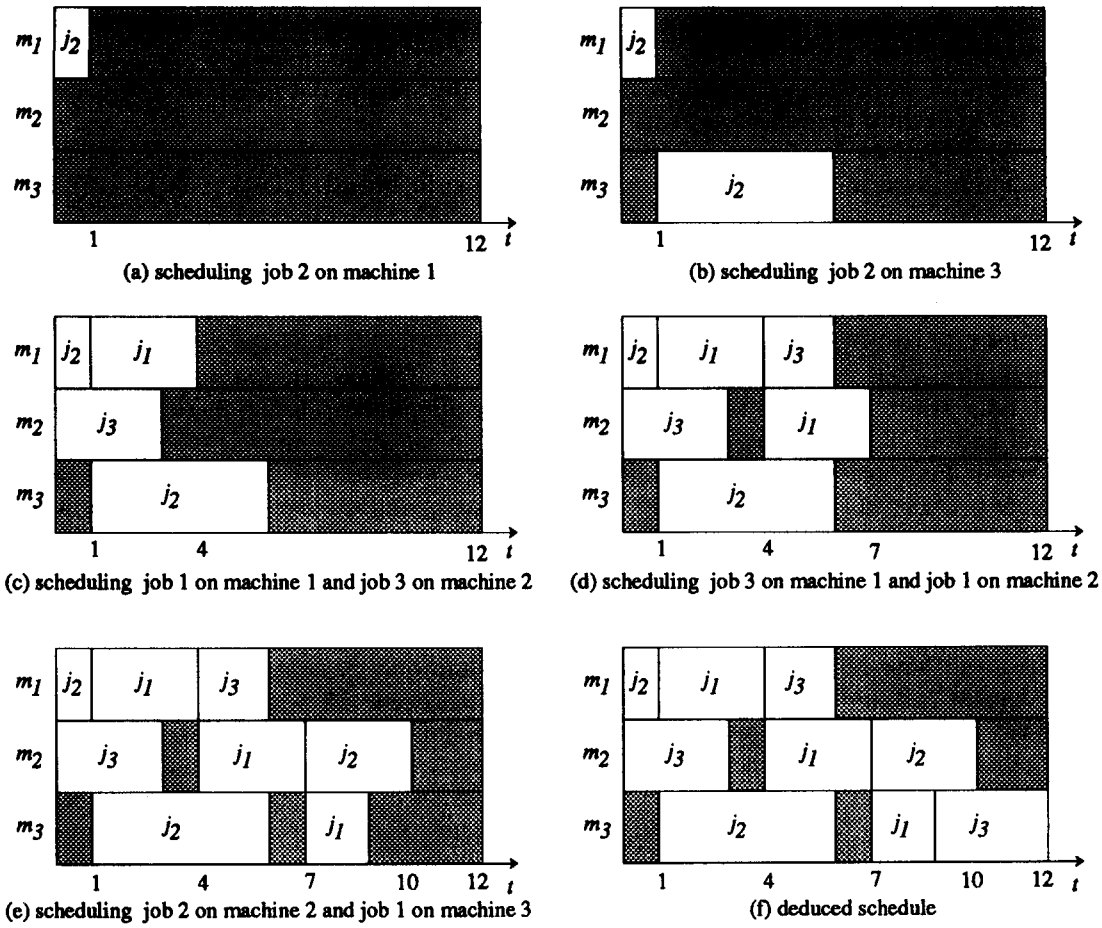

Fig. 6. Deduce a schedule from the chromosome.

4.4. Job pair relation-based representation

Nakano and Yamada used a binary matrix to encode a schedule and the matrix is determined according to the precedence relation of a pair of jobs on corresponding machines [13].

Consider the following 3-job 3-machine problem. The operation precedence constraints of the jobs and one feasible schedule of the problem are given in Table 3.

A binary variable is defined to indicate precedence relation for a pair of jobs as follows:

$$x_{ijm} = \begin{cases} 1, & \text{if job } i \text{ is processed prior to job } j \text{ on machine } m \\ 0, & \text{others} \end{cases}$$

Let us examine the precedence relation of a job pair (j_1, j_2) on machines (m_1, m_2, m_2). According to the given schedule, we have $(x_{121}\ x_{122}\ x_{123}) = (0\ 1\ 0)$. For the job pair (j_1, j_3), we have $(x_{131}\ x_{132}\ x_{133}) = (1\ 0\ 1)$, and for the job pair (j_2, j_3), the precedence relation on machines (m_1, m_3, m_2) are $(x_{231}\ x_{233}\ x_{232}) = (1\ 1\ 0)$. Note that the sequence of variable x_{ijm} for a pair of jobs should keep consistent with the sequence of operations of the first job i. For example, for job pair (j_2, j_3), the sequence of operations of job j_2 is (1 3 2), so the relative variables are sequenced as

Table 3. Example of the 3-job and 3-machine problem

Operation precedence				Feasible schedule			
Job	Machine sequence			Machine	Job sequence		
j_1	m_1	m_2	m_3	m_1	j_2	j_1	j_3
j_2	m_1	m_3	m_2	m_2	j_3	j_1	j_2
j_3	m_2	m_1	m_3	m_3	j_2	j_1	j_3

Table 4. Selected priority rules

No.	Rule	Description
1	SPT	Select an operation with the shortest processing time
2	LPT	Select an operation with the longest processing time
3	MWR	Select an operation for the job with the most total processing time remaining
4	LWR	Select an operation for the job with the least total processing time remaining

(x_{231} x_{233} x_{232}) rather than (x_{231} x_{232} x_{233}). By summarizing these results, we get a binary matrix representation for the given feasible schedule as follows:

$$\begin{array}{l}(j_1, j_2) \text{ on } (m_1, m_2, m_3): \\ (j_1, j_3) \text{ on } (m_1, m_2, m_3): \\ (j_2, j_3) \text{ on } (m_1, m_3, m_2):\end{array} \quad \begin{pmatrix} x_{121} & x_{122} & x_{123} \\ x_{131} & x_{132} & x_{133} \\ x_{231} & x_{233} & x_{232} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

The representation is perhaps the most complex one among all representations for JSP and is highly redundant [20]. Besides its complexity, with this representation, the chromosomes produced either by the initial procedure or by genetic operations are illegal in general.

4.5. Priority rule-based representation

Dorndorf and Pesch proposed a priority rule-based genetic algorithm [21], where a chromosome is encoded as a sequence of dispatching rules for job assignment and a schedule is constructed with a priority dispatching heuristic based on the sequence of dispatching rules. Genetic algorithms here are used to evolve those chromosomes to find out a better sequence of dispatching rules.

Priority dispatching rules are probably the most frequently applied heuristics for solving scheduling problems in practice because of their ease of implementation and their less time complexity. The algorithms of Giffler and Thompson can be considered as the common basis of all priority rule-based heuristics [34, 35]. The problem is to identify an effective priority rule. For an extensive summary and discussion on priority dispatching rules see Panwalker and Iskander [36], Haupt [37], and Blackstone *et al.* [38].

For an n-job m-machine problem, a chromosome is a string of $n \times m$ entry ($p_1, p_2, \ldots, p_{nm}$). An entry p_i represents one rule of the set of prespecified priority dispatching rules. The entry in the ith position says that a conflict in the ith iteration of the Giffler and Thompson algorithm should be resolved using priority rule p_i. More precisely, an operation from the conflict set has to be selected by rule p_i; ties are broken by a random choice. Let

PS_t = a partial schedule containing t scheduled operations,
S_t = the set of schedulable operations at iteration t, corresponding to a given PS_t,
σ_i = the earliest time at which operation $i \in S_t$ could be started,
ϕ_i = the earliest time at which operation $i \in S_t$ could be completed,
C_t = the set of conflicting operations in iteration t.

The procedure to deduce an offspring from a given chromosome ($p_1, p_2, \ldots, p_{nm}$) works as follows:

PROCEDURE. Deduce an offspring for priority rule-based encoding.

Step 1: Let $t = 1$ and begin with PS_t as the null partial schedule and S_t include all operations with no predecessors.

Step 2: Determine $\phi_t^* = \min_{i \in S_t}\{\phi_i\}$ and the machine m^* on which ϕ_t^* could be realized. If more than one such machine exists, tie is broken by a random choice.

Step 3: Form conflicting set C_t which includes all operations $i \in S_t$ with $\sigma_i < \phi_t^*$ that requires machine m^*. Select one operation from C_t by priority rule p_t and add this operation to PS_t as early as possible, thus creating new partial schedule PS_{t+1}. If more than one operation exists according to the priority rule p_t, tie is broken by a random choice.

Step 4: Update PS_{t+1} by removing the selected operation from S_t and adding the direct successor of the operation to S_t. Increment t by one.

Step 5: Return to step 2 until a complete schedule is generated.

Let us see an example. Four priority rules to be used are given in Table 4. Consider the following chromosome [1 2 2 1 4 4 2 1 3], where 1 stands for rule SPT, 2 for rule LPT, 3 for rule MWR, and 4 for rule LWR. At the initial step, we have

$$S_1 = \{o_{111}, o_{211}, o_{312}\}$$
$$\phi_1^* = \min\{3, 1, 3\} = 1$$
$$m^* = 1$$
$$C_1 = \{o_{111}, o_{211}\}$$

Now operations o_{111} and o_{211} compete for machine m_1. Because the first gene in the given chromosome is 1 (which means SPT priority rule), operation 211 is scheduled on machine m_1 as shown in Fig. 7(a). After updating the data we have

$$S_2 = \{o_{111}, o_{223}, o_{312}\}$$
$$\phi_2^* = \min\{4, 6, 3\} = 3$$
$$m^* = 2$$
$$C_2 = \{o_{312}\}$$

Operation o_{312} is scheduled on machine m_2 as shown in Fig. 7(b). After updating the data we have

$$S_3 = \{o_{111}, o_{223}, o_{321}\}$$
$$\phi_3^* = \min\{4, 6, 3\} = 3$$
$$m^* = 1$$
$$C_3 = \{o_{111}, o_{321}\}$$

Now operations o_{111} and o_{321} compete for machine m_1. Because the third gene in the given chromosome is 2 (which means LPT priority rule), operation o_{111} is scheduled on machine m_1 as shown in Fig. 7(c). Repeat these steps until a complete schedule is deduced from the given chromosome as shown in Fig. 7(d).

4.6. *Disjunctive graph-based representation*

Tamaki and Nishikawa proposed a disjunctive graph-based representation [17], which also can be viewed as one kind of job pair relation-based representation. The job-shop scheduling problem can be represented with a *disjunctive graph* [39, 40]. The disjunctive graph $G = (N, A, E)$ is defined

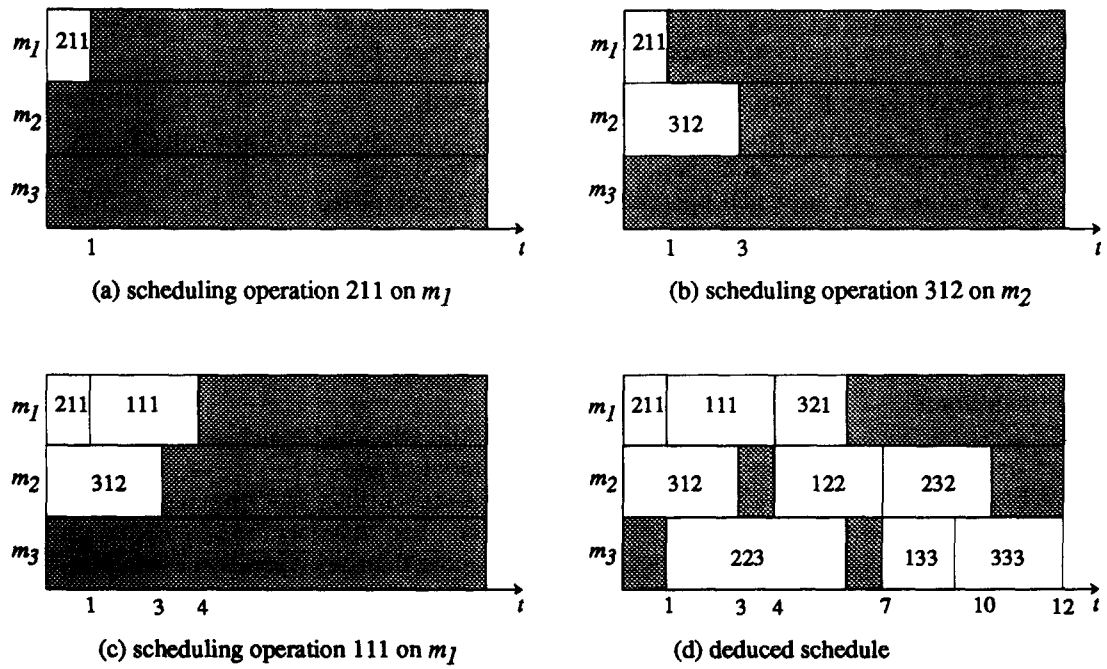

Fig. 7. Deduce a schedule from a priority rule-based encoding.

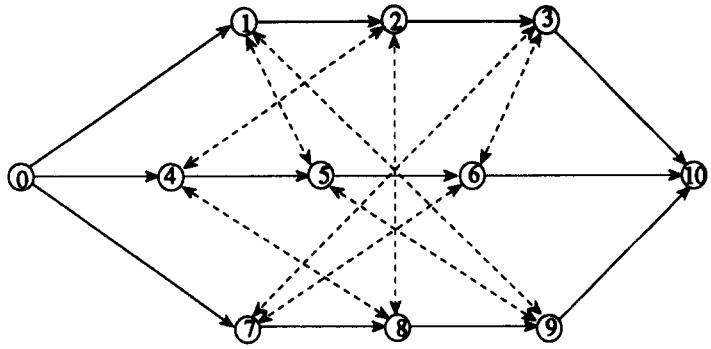

Fig. 8. Disjunctive graph of 3-job 3-machine problem.

as follows: N contains nodes representing all operations, A contains arcs connecting consecutive operations of the same job, and E contains disjunctive arcs connecting operations to be processed by the same machine. The disjunctive constraints are represented by an edge in E. A disjunctive arc can be settled by either of its two possible orientations. The construction of a schedule will settle the orientations of all disjunctive arcs so as to determine the sequences of operations on same machines. Once a sequence is determined for a machine, the disjunctive arcs will be replaced by the usual *conjunctive* arcs. Figure 8 illustrates the disjunctive graph for a 3-job 3-machine example. A chromosome consists of binary string corresponding to an order list of disjunctive arcs in E as shown in Fig. 9, where e_{ij} stands for the disjunctive arc between nodes i and j and is defined as follows:

$$e_{ij} = \begin{cases} 1, & \text{settle the orientation of the disjunctive arc from node } j \text{ to node } i \\ 0, & \text{settle the orientation of the disjunctive arc from node } i \text{ to node } j \end{cases}$$

order list of disjunctive arcs: $e_{15}\ e_{19}\ e_{59}\ e_{24}\ e_{28}\ e_{48}\ e_{36}\ e_{37}\ e_{67}$

chromosome: [0 0 1 1 0 0 0 1 1]

Fig. 9. Disjunctive graph-based representation.

The job-shop scheduling problem is to find an order of the operations on each machine, i.e., to settle the orientation of the disjunctive arcs such that the resulting graph is acyclic to guarantee that there are no precedence conflicts between operations. It is easy to see that an arbitrary chromosome may yield a cyclic graph, that means the schedule is infeasible. So this chromosome is not used to represent a schedule but only used as a decision preference. They used a critical path-based procedure to deduce a schedule. During the process of deduction, when conflict of two nodes (operations) occurs on a machine, the corresponding bit of the chromosome is used to settle the processing order of the two operations, in other words, to settle the orientation of the disjunctive arc between the two nodes.

4.7. Completion time-based representation

Yamada and Nakano proposed a completion time-based representation [16]. A chromosome is an ordered list of completion times of operations. For the same example given in Table 2, the chromosome can be represented as follows:

$$[c_{111}\ c_{122}\ c_{133}\ c_{211}\ c_{223}\ c_{232}\ c_{312}\ c_{321}\ c_{333}]$$

where c_{jir} means the completion time for operation i of job j on machine r. It is easy to know that such a representation is not suitable for most genetic operators because it will yield an illegal schedule. Yamada and Nakano designed a special crossover operator for it.

4.8. Machine-based representation

Dorndorf and Pesch proposed a machine-based genetic algorithm [21], where a chromosome is encoded as a sequence of machines and a schedule is constructed with a shifting bottleneck heuristic based on the sequence.

The *shifting bottleneck heuristic*, proposed by Adams *et al.* [41] is probably the most powerful procedure known up to now among all heuristics for the job-shop scheduling problem. It sequences the machines one by one, successively, taking each time the machine is identified as a bottleneck among the machines not yet sequenced. Every time after a new machine is sequenced, all previously established sequences are locally reoptimized. Both the bottleneck identification and the local reoptimization procedures are based on repeatedly solving a certain one-machine scheduling problem that is a relaxation of the original problem. The main contribution of their approach is the way to use this relaxation to decide upon the order in which the machines should be sequenced.

The shifting bottleneck heuristic is based on the classic idea of giving priority to bottleneck machines. Different measures of bottleneck quality of machines will yield a different sequence of bottleneck machines. The quality of the schedules obtained by the shifting bottleneck heuristic heavily depends on the sequence of bottleneck machines. Adams *et al.* also proposed an enumerative version of the shifting bottleneck heuristic to consider different sequences of machines.

Instead of enumerative tree search, Dorndorf and Pesch proposed a genetic strategy to determine the best machine sequence for the shifting bottleneck heuristic. A chromosome is a list of ordered machines. Genetic algorithms here are used to evolve those chromosomes to find out a better sequence of machines for the shifting bottleneck heuristic. The difference between the shifting bottleneck heuristic and genetic algorithm is that the bottleneck is no longer a decision criterion for the choice of the next machine, which is controlled by a given chromosome.

Let M_0 be the set of machines already sequenced and a given chromosome be $[m_1, m_2, \ldots, m_m]$. The procedure of deducing a schedule from the chromosome works as follows:

PROCEDURE. Deduce a schedule for machine-based encoding.
Step 1: Let $M_0 \leftarrow \emptyset$, $i \leftarrow 1$ and the chromosome $[m_1, m_2, \ldots, m_m]$
Step 2: Sequence machine m_i optimally. Update set $M_0 \leftarrow M_0 \cup \{m_i\}$.
Step 3: Reoptimize the sequence of each critical machine $m_i \in M_0$ in turn, while keeping the other sequences fixed.
Step 4: Let $i \leftarrow i + 1$. Then if $i > m$, stop; otherwise go to step 2.

The details of step 3 can be found in Adams *et al.* [41] or in Applegate and Cook [42].

4.9. Random key representation

Random key representation is firstly given by Bean [25]. With this technique, genetic operations can produce feasible offspring without creating additional overhead for a wide variety of sequencing and optimization problems. Norman and Bean further successfully generalized the approach to the job-shop scheduling problem [27, 43].

Random key representation encodes a solution with *random number*. These values are used as sort *keys* to decode the solution. For n-job m-machine scheduling problem, each gene (a random key) consists of two parts: an integer in set $\{1, 2, \ldots, m\}$ and a fraction generated randomly from (0, 1). The integer part of any random key is interpreted as the machine assignment for that job. Sorting the fractional parts provides the job sequence on each machine. Consider the same example given in Table 2. Suppose a chromosome to be:

$$[1.34 \ \ 1.09 \ \ 1.88 \ \ 2.66 \ \ 2.91 \ \ 2.01 \ \ 3.23 \ \ 3.21 \ \ 3.44]$$

Sorting the keys for machine 1 in ascending order results in the job sequence $2 \rightarrow 1 \rightarrow 3$, for machine 2 the job sequence $3 \rightarrow 1 \rightarrow 2$, and for machine 3 the job sequence $2 \rightarrow 1 \rightarrow 3$. Let o_{jm} denote job j on machine m. The chromosome can be translated into a unique list of ordered operations as follows:

$$[o_{21} \ \ o_{11} \ \ o_{31} \ \ o_{32} \ \ o_{12} \ \ o_{22} \ \ o_{23} \ \ o_{13} \ \ o_{33}]$$

It is easy to see that the job sequences given above may violate the precedence constraints. Accompanying this coding, a pseudocode is presented by Norman and Bean for handling the precedence constraints [27].

5. DISCUSSION

Various encoding techniques are summarized in the previous section. In this section, we give a comparative discussion from the following aspects:

- Lamarkian property of chromosome
- complexity of decoder
- the property of encoding space and mapping
- memory requirements.

5.1. Lamarkian property

The canonical genetic algorithm developed by John Holland [44], inspired by Darwin's theory of natural selection, performs robust search, but is slower than other methods. Kennedy [45, 46] introduced *Lamarkian evolution* into the canonical genetic algorithm to improve its efficiency, in which offspring organisms first pass through Darwin's biological evolution and then pass through Lamarkian's intelligent evolution to inject some "smart" into the offspring organism before returning it to be evaluated.

The Lamarkian property for an encoding technique concerns the issue whether or not chromosomes can pass on their "merits" to a future population through common genetic operation. Let us see an example. A tour of 9-city TSP can be encoded into the following chromosome:

random key: [0.34 0.09 0.88 0.66 0.91 0.01 0.23 0.21 0.44]
permutation: [6 2 8 7 1 9 4 3 5]

Suppose a subtour is formed with the second and sixth cut points. For permutation representation, the subtour is [8 7 1 9]. An offspring receives the same subtour from its parent. For random key representation, the subtour is [0.88 0.66 0.91 0.01]. Usually, it may refer to a different subtour in offspring, but not the same subtour of $8 \rightarrow 7 \rightarrow 1 \rightarrow 9$ as its parent. What subtour it refers to will depend on the values of the other genes in the offspring. In fact, the offspring receives nothing (about good subtour) from the parent, but a mere random chance.

Generally, we hope an encoding technique has the property. In fact, most encodings have Lamarkian property, that is, offspring can inherit goodness from the parent. The opposite extreme is *no Lamarkian*, that is, offspring inherits nothing from parents. The third type is called *half-Lamarkian*. In this case, part of the segments inherited from parents refers to the same things as the parents and the remaining part refers to different things.

Let us consider the encodings given in the above section. Random key representation and completion time-based representation have no Lamarkian, job-based representation and priority rule-based presentation have Lamarkian, and the remaining representations have half-Lamarkian. In general, for a n-job m-machine problem, a chromosome contains $n \times m$ genes and each gene corresponds to an operation. The 1-to-1 corresponding relation is realized mainly by the following three ways: (1) job sequence, (2) preference list, and (3) priority rule. For the first two cases, the corresponding relation is context-dependent, which means that a gene with the same value may be interpreted into the same things, or may not.

5.2. Complexity of decoder

In principle, there are an infinite number of feasible schedules for a job-shop scheduling problem. Generally, three kinds of schedules can be distinguished as follows: *semi-active schedule*, *active schedule*, and *non-delay schedule* [3]. Optimal schedule is within the set of active schedules. The non-delay schedules are smaller than active schedules, but no guarantee that it will contain an optimum. So we hope that a schedule decoded from a chromosome would be an active one. All of the encoding techniques proposed for the job-shop scheduling problem can generate an active schedule by use of the decoder. The degree of complexity of the decoder can be classified into the following four levels:

Level 0: no decoder. Completion time-based representation belongs to the class. In this case, all burdens are put on genetic operators.

Level 1: simple mapping relation. Operation-based representation, job pair relation-based representation and job-based representation belong to the class.

Level 2: simple heuristic. Preference list-based representation and priority rule-based representation belong to the class.

Level 3: complex heuristic. Preference list-based encoding used by Kobayashi *et al.*, disjunctive graph-based representation, machine-based representation belong to the class.

5.3. *The property of encoding space and mapping*

As we can see for any encoding technique, through a decoding procedure, a chromosome also corresponds to a legal, feasible and active schedule and the mapping relation is 1-to-1. But the encoding space can be classified into two classes: one contains only feasible solution space and another includes illegal solution space. This can be tested by applying the simple one-point crossover operator to an encoding and checking if the yielded offspring is illegal. Priority rule-based representation, job-based representation, and disjunctive graph-based representation belong to the first class and the remaining representations belong to the second class.

In addition, the space of job-based and machine-based representations just correspond to the partial space of whole solution space.

5.4. *Memory requirements*

For an n-job m-machine problem, if we define the standard length for a chromosome as $n \times m$ genes, the encodings for the job-shop can be classified into three classes:

(1) The encoding length is less than standard length. Job-based one, disjunctive graph-based one and machine-based one belong to the class. Note that the encoding space for job-based one just corresponds to part of the solution space, but not the whole solution space. So there is no guarantee for finding the optima with the encoding techniques.

(2) The encoding length is larger than standard length. Only job pair relation-based one belongs to the class. The representation is highly redundant.

(3) The encoding length is equal to standard length. The remaining representations belong to the class.

Lastly, we would like to point out that for giving a fair judgement on these encoding techniques, it is necessary to make further comparative studies of these techniques under standard experiment conditions using benchmark problems to reveal the advantage and disadvantage for each of them. We leave the topic about how to design a standard experiment to the companion paper: Part II. Hybrid Genetic Search Strategies.

Acknowledgement—This work was supported in part by a research grant from the Ministry of Education, Science and Culture, Japanese Government: Grant-in-Aid for Scientific Research, the International Scientific Research Program (No. 07045032).

REFERENCES

1. J. Muth and G. Thompson (Eds). *Industrial Scheduling*. Prentice Hall, Englewood Cliffs, NJ (1963).
2. R. Conway, W. Maxwell and L. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, MA (1967).
3. K. Baker. *Introduction to Sequencing and Scheduling*. Wiley, New York (1974).
4. S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*. Wiley, New York (1982).
5. A. Rinnooy Kan. *Machine Scheduling Problem: Classification, Complexity and Computation*. Martinus Nijhoff, Hague (1976).
6. E. Coffman. *Computer and Job-shop Scheduling Theory*. Wiley, New York (1976).
7. J. Blazewicz, K. Ecker, G. Schmidt and J. Weglarz. *Scheduling in Computer and Manufacturing Systems*, 2nd Edition. Springer, New York (1994).
8. T. Morton and D. Pentico. *Heuristic Scheduling Systems—With Applications to a Production Systems and Project Management*. Wiley, New York (1993).
9. M. Garey, D. Johnson and R. Sethi. The complexity of flowshop and jobshop scheduling. *Maths Ops Res.* **1**, 117–129 (1976).
10. P. Van Laarhoven, E. Aarts and J. Lenstra. Job shop scheduling by simulated annealing. *Ops Res.* **40**, 113–125 (1992).
11. M. Dell'Amico and M. Trubian. Applying tabu search to the job shop scheduling problem. *Ann. Ops Res.* **40**, 231–252 (1993).
12. L. Davis. Job shop scheduling with genetic algorithm. In *Proc. of the First Int. Conf. on Genetic Algorithms* (Edited by J. Grefenstette), pp. 136–140. Lawrence Erlbaum Associates, Hillsdale, NJ (1985).

13. R. Nakano and T. Yamada. Conventional genetic algorithms for job-shop problems. In *Proc. of the Fourth Int. Conf. on Genetic Algorithms* (Edited by Belew and Booker), pp. 477–479. Morgan Kaufman, San Mateo, Calif. (1991).
14. E. Falkenauer and S. Bouffoix. A genetic algorithm for job shop. In *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pp. 824–829 (1991).
15. S. Bagchi, S. Uckun, Y. Miyabe and K. Kawamura. Exploring problem-specific recombination operators for job shop scheduling. In *Proc. of the Fourth Int. Conf. on Genetic Algorithms* (Edited by Belew and Booker), pp. 10–17. Morgan Kaufman, San Mateo, Calif. (1991).
16. T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature*, pp. 281–290. Elsevier Science Publishers, North-Holland (1992).
17. H. Tamaki and Y. Nishikawa. A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. In *Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature*, pp. 573–582. Elsevier Science Publishers, North-Holland (1992).
18. J. Paredis. Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling. In *Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature*, pp. 281–290. Elsevier Science Publishers, North-Holland (1992).
19. C. Holsapple, V. Jacob, R. Pakath and J. Zaveri. A genetic-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Trans. Systems, Man, and Cybernetics* **23,** 953–971 (1993).
20. H. Fang, P. Ross and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pp. 375–382. Morgan Kaufmann Publishers, San Mateo, Calif. (1993).
21. U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Complex Systems* **22,** 25–40 (1995).
22. M. Gen and T. Kobayashi (Eds). *Proc. of the 16th Int. Conf. on Computer and Industrial Engineering*, Ashikaga, Japan (1994).
23. M. Gen, Y. Tsujimura and E. Kubota. Solving job-shop scheduling problem using genetic algorithms. In *Proc. of the 16th Int. Conf. on Computer and Industrial Engineering*, pp. 576–579. Ashikaga, Japan (1994).
24. F. Croce, R. Tadei and G. Volta. A genetic algorithm for the job shop problem. *Complex Systems* **22,** 15–24 (1995).
25. J. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Computing* **6,** 154–160 (1994).
26. S. Kobayashi, I. Ono and M. Yamamura. An efficient genetic algorithm for job shop scheduling problems. In *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pp. 506–511. Morgan Kaufmann Publishers, San Francisco, Calif. (1995).
27. B. Norman and J. Bean. Random keys genetic algorithm for job-shop scheduling: unabridged version. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor (1995).
28. H. Greenberg. A branch-and-bound solution to the general scheduling problem. *Ops Res.* **16,** 353–361 (1968).
29. A. Smith and D. Tate. Genetic optimization using a penalty function. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pp. 499–505. Morgan Kaufmann Publishers, San Mateo, Calif. (1993).
30. Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In *Proc. of the 4th Annual Conf. on Evolutionary Programming* (Edited by J. McDonnell *et al.*), pp. 135–155. MIT Press, Cambridge, MA (1995).
31. M. Gen and R. Cheng. A survey of penalty techniques in genetic algorithms. *Proc. of 1996 IEEE Int. Conf. on Evolutionary Computation*, pp. 804–809 (1996).
32. D. Orvosh and L. Davis. Using a genetic algorithm to optimize problems with feasibility constraints. In *Proc. of the First IEEE Conf. on Evolutionary Computation*, pp. 548–552. IEEE Press, Florida (1994).
33. A. Kubota. Study on optimal scheduling for manufacturing system by genetic algorithms. Master's thesis, Ashikaga Institute of Technology, Ashikaga, Japan (March 1995).
34. B. Giffler and G. Thompson. Algorithms for solving production scheduling problems. *Ops Res.* **8,** 487–503 (1960).
35. R. Storer, S. Wu and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *ms* **38,** 1495–1510 (1992).
36. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Ops Res.* **25,** 45–61 (1977).
37. R. Haupt. A survey of priority-rule based scheduling problem. *OR Spektrum* **11,** 3–16 (1989).
38. J. Blackstone, D. Phillips and G. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *Int. J. Prod. Res.* **20,** 27–45 (1982).
39. B. Roy and B. Sussmann. Les problemes d'ordonnancement avec contraintes disjonctives. Technical Report 9, SEMA, Note D.S., Paris (1964).
40. E. Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Ops Res.* **17,** 941–957 (1969).
41. J. Adams, E. Balas and D. Zawack. The shifting bottleneck procedure for job shop scheduling. In *Int. J. Flexible Manuf. Systems* **34,** 391–401 (1988).
42. D. Applegate and W. Cook. A computational study of the job shop scheduling problem. *ORSA J. Computing* **3,** 149–156 (1991).
43. B. Norman and J. Bean. Random keys genetic algorithm for scheduling. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor (1995).
44. D. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA (1989).
45. S. Kennedy. Five ways to a smarter genetic algorithm. *AI Expert* (1993).
46. D. Whitley, V. Gordan and K. Mathias. Lamarkian evolution, the baldwin effect and function optimization. In *Parallel Problem Solving from Nature—PPSN III* (Edited by Y. Davidor *et al.*), pp. 6–15. Springer, Berlin (1994).
47. R. Belew and L. Booker (Eds). *Proc. of the Fourth Int. Conf. on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, Calif. (1991).
48. R. Männer and B. Manderick (Eds). *Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature*. Elsevier Science Publishers, North-Holland (1992).
49. S. Forrest (Ed.). *Proc. of the Fifth Int. Conf. on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, Calif. (1993).
50. M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. Wiley, New York (1966).