PERGAMON Computers & Industrial Engineering 36 (1999) 343–364

computers & industrial engineering

www.elsevier.com/locate/compindeng

A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies

Runwei Cheng[a,*], Mitsuo Gen[b], Yasuhiro Tsujimura[b]

[a]*Department of Systems Engineering, College of Information Science and Engineering, Northeastern University, Shenyang 110006, China*
[b]*Department of Industrial and Information Systems Engineering, Ashikaga Institute of Technology, Ashikaga 326-8558, Japan*

Abstract

Job-shop scheduling problem is one of the well-known hardest combinatorial optimization problems. During the last three decades, this problem has captured the interest of a significant number of researchers. 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 to address the problem.

How to adapt genetic algorithms to the job-shop scheduling problems is very challenging but frustrating. Many efforts have been made in order to give an efficient implementation of genetic algorithms to the problem. During the past decade, two important issues have been extensively studied. One is how to encode a solution of the problem into a chromosome so as to ensure that a chromosome will correspond to a feasible solution. The other issue is how to enhance the performance of genetic search by incorporating traditional heuristic methods. Because the genetic algorithms are not well suited for fine-tuning of solutions around optima, various methods of hybridization have been suggested to compensate for this shortcoming. The purpose of the paper is to give a tutorial survey of recent works on various hybrid approaches in genetic job-shop scheduling practices.

The research on how to adapt the genetic algorithms to the job-shop scheduling problem provide very rich experiences for the constrained combinatorial optimization problems. All of the techniques developed for the problem are very useful for other scheduling problems in modern flexible manufacturing systems and other difficult-to-solve combinatorial optimization problems. © 1999 Published by Elsevier Science Ltd. All rights reserved.

Keywords: Genetic algorithms; Job-shop scheduling; Combinatorial optimization problem; Hybrid heuristics

* Corresponding author.
E-mail address: runwei@yahoo.com (R. Cheng)

0360-8352/99/$ - see front matter © 1999 Published by Elsevier Science Ltd. All rights reserved.
PII: S0360-8352(99)00136-9

1. Introduction

In the job shop scheduling problem, we are given a set of jobs and a set of machines. Each machine can handle, at most, one job at a time. 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. The purpose is to find a schedule, that is, an allocation of the operations to time intervals on the machines, that has a minimum duration required to complete all jobs [1,2].

This problem 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 many solution methods have been proposed, ranging from simple and fast dispatching rules to sophisticated branch-and-bound algorithms. However, with the rapid increase in the speed of computing and the growing need for efficiency in scheduling, it becomes increasingly important to explore ways of obtaining better schedules at some extra computational costs. The genetic algorithm approach is one such kind of attempt. In the view of computational cost, genetic algorithms are not as efficient as some other heuristic methods [3]. The objective of the research is two-fold: on the one hand, we want to create an effective method based on genetic algorithms so as to obtain better schedules than traditional heuristic methods; on the other hand the problem exhibits all aspects of constrained combinatorial optimization and has served as the benchmark problem for testing new algorithmic ideas.

How to adapt the genetic algorithms to the job-shop scheduling problems is very challenging but frustrating. Many efforts have been made in order to give an efficient implementation of genetic algorithms to the problem. During the past decade, two important issues have been extensively studied. One is how to encode a solution of the problem into a chromosome. Because of the existence of complex constraints inherent in the problem, a simple binary string does not work at all, which indubitably yields to infeasible or even illegal solutions. The virgin efforts of most researchers have been devoted to the invention of a new and efficient encoding method to the problem. A comprehensive survey on this topic has been given in our former paper entitled as a tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation [4]. The other issue is how to enhance the performance of genetic search by incorporating traditional heuristic method. Because of the combinatorial nature of the problem, to find the optima to such kind of problem is as hard as to find the needle in a haystack. Genetic algorithms are not well suited for fine tuning of solutions around optima. Therefore, various methods of hybridization have been suggested to compensate for this shortcoming. The purpose of this paper is to give a tutorial survey of recent works on various hybrid approaches of the genetic algorithms proposed so far for the job-shop scheduling problem.

The organization of this paper is as follows: general comments on hybrid genetic search are given in the next section; various hybrid genetic algorithms are concisely described in subsequent sections; in the last section we give some concluding remarks.

2. Basic approaches

There are three basic approaches of applying the genetic algorithms to a given problem:

- adapt problems to the genetic algorithms
- adapt the genetic algorithms to problems
- adapt both the genetic algorithms and problems.

The genetic algorithm was first created as a kind of generic and weak method featuring binary encoding and binary genetic operators. Basically, the approach requires a modification of a given problem into an appropriate form so as that its solution can be represented with binary string. The approach is shown in Fig. 1, which includes a mapping between potential solutions and the binary representation, taking care of decoders or repair procedures, etc. For complex problems, it is hard to represent a solution with a binary string, which limits the applications of the genetic algorithms.

To overcome such problems, various non-standard implementations of the genetic algorithms have been created for particular problems, which leave the given problem unchanged and adapt the genetic algorithms by modifying a chromosome representation of solutions and applying appropriate genetic operators as shown in Fig. 2.

In general, however, it is not a good choice to use the whole original solution of a given problem as the chromosome representation, because the solution structure of many real problems are too complex to be a natural representation suitable for the genetic algorithms. Generally, the encoding methods can be either direct or indirect. In the direct encoding method, the whole of a solution for a given problem is used as a chromosome. For a complex problem, however, such a method will make almost all conventional genetic operators inapplicable due to that a vast of offspring will be infeasible or illegal. The *infeasibility* means that some constraints are violated while *illegality* means that the chromosome can not be decoded into a solution. On the contrary, in the indirect encoding method, just the necessary part of a solution is encoded in a chromosome. Solutions to a given problem then can be generated by a decoder. A decoder is a problem-specific and determining procedure. With this method, genetic algorithms will focus their search solely on the interesting part of solution space. Therefore, the third approach is to adapt both the genetic algorithms and the given problem, as shown in Fig. 3.

Essentially, the combinatorial optimizations seek a proper permutation and/or combination

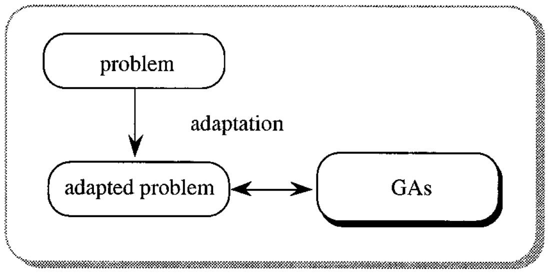

Fig. 1. The first approach: adapt a problem to the genetic algorithms.

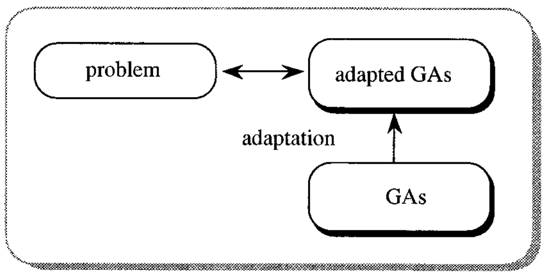

Fig. 2. The second approach: adapt the genetic algorithms to a problem.

of some items for a given problem under some constraints. A common feature of combinatorial optimization problems is that if the permutation and/or combination can be determined, a solution then can be derived with a problem-specific procedure. Therefore, the genetic algorithms can be used to evolve an appropriate permutation and/or combination of some items under consideration and a heuristic method then can be used to construct a solution subsequently according to the permutation and combination.

This approach has been successfully applied in the area of industrial engineering and is becoming the main approach for the practice of genetic algorithms [5,6]. Most of research on applying genetic algorithms to the job-shop scheduling problem have adopted this approach. The essence of the job-shop scheduling problem is to find out a permutation of operations on each machine subject to precedence constraints in order to minimize the makespan. If the permutation can be determined, a solution then can be easily derived with a problem-specific procedure. A general approach for applying the genetic algorithms to the job-shop scheduling problem is:

1. Using the genetic algorithms to evolve an appropriate permutation
2. Using a heuristic method to construct a solution subsequently according to the permutation.

Because the genetic algorithms are not well suited for fine tuning of structures close to optima, various methods of hybridization have been suggested to compensate for this shortcoming. A famous principle of hybridization given by Davis is ‘hybridize where possible’ [7]. The

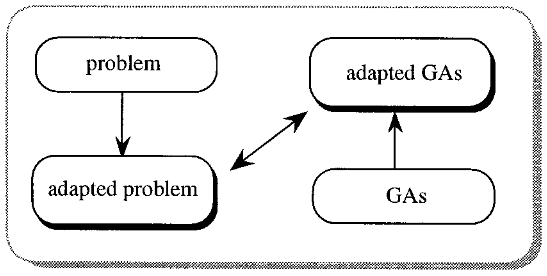

Fig. 3. The third approach: adapt both the genetic algorithms and the problem.

hybridization methods for the job-shop scheduling problem can be classified into the following three categories [7,8]:

1. Adapted genetic operators
2. Heuristic-featured genetic operators
3. Hybrid genetic algorithms.

The first approach is to revise or invent generic operators so as to meet the feature of a given encoding representation. The second approach is to create new genetic operators inspired from conventional heuristics. The third approach involves hybridizing conventional heuristics into the main loop of genetic algorithms where possible.

3. Adapted genetic operators

In general, the crossover operator is regarded as a main genetic operator and the performance of the genetic algorithms depends, to a great extent, on the performance of the crossover operator used. Conceptually, the crossover operates on two chromosomes at a time and generates offspring by combining features of both chromosomes. A simple way to achieve crossover is the well-known one-cut-point or two-cut-point crossover methods, which work well with bit string encodings. For many complex problems, it is usually very difficult to represent a solution with the bit string encodings. Therefore, many non-bit string encodings, especially various literal string encodings, have been suggested and many new genetic operators have been adapted to cope with these non-bit string encodings. Essentially the adapted crossover methods can be viewed as the revised version of one-cut-point or two-cut-point crossovers for some non-bit string encodings by incorporating with some repairing procedures in order to avoid yielding illegal or infeasible solutions.

For the ease of explanation below, we first distinguish between the following two kinds of literal strings:

1. Pure literal string
2. General literal string.

A pure literal string encoding consists of distinctive symbols, while a general string encoding allows a symbol to have its repetition of a prescribed number. In other words, duplication is prohibited in a pure literal string while the same symbol can co-exist in a general literal string. During the last ten years, the following nine representations for the job-shop scheduling problem have been proposed [4]:

- 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.

Among the nine encoding methods, the job-based encoding and the machine-based encoding are the pure literal string; the operation-based encoding, the preference list-based encoding and the priority rule-based encoding are the general literal string. Note that the preference list-based encoding is well used in many studies. It consists of several sub-strings and one substring is a pure literal string corresponding to an operation sequence to a machine. A usual way, in practice, is to apply genetic operators to each substring for this encoding. With the multiple usage of genetic operators, this encoding is essentially treated as a pure literal string. Therefore, many kinds of genetic operators for a pure literal string can be directly used for this encoding.

As we know, there are two kind of order relations in the job-shop scheduling problem:

1. The operation sequence on each machine;
2. The precedence constraints among operations for a job.

The first must be determined by a solution method, while the second must be maintained in a schedule. Accordingly, several approaches have been proposed to handle the order relations:

1. The information with respect to both operation sequence and precedence constraints is preserved in an encoding simultaneously. In such a case, all crossover methods for literal permutation encodings are not directly applicable. A chromosome may be either *infeasible* in the sense that some precedence constraints are violated or *illegal* in the sense that the repetitions of some symbols are not equal to the prescribed numbers. For a pure literal string encoding, the illegality means that some symbols are repeated more than once while other symbols get lost. For a general literal string encoding, the illegality makes its sense that some symbols may appear more than necessary while others less than necessary. A special attention must be given on how to handle infeasibility and illegality when designing a new crossover operator for such kind of encodings so as to not disorder precedence relations. Among the nine literal string encodings, the operation-based encoding is the only one which keeps both information of operation sequence and precedence constraints in a chromosome.
2. Only the information with respect to operation sequence is encoded in the encodings, while a special decoder procedure or schedule builder procedure is used to resolve the precedence constraints. In such a case, the chromosome is one kind of literal permutation and all the attention we should pay is not to produce an illegal offspring by devised genetic operators. Among the nine literal string encodings, the preference list-based encoding, the job-based encoding, and the machine-based are this kind of encoding.
3. Neither of both order relations but some guild information are encoded in the chromosome. Such kind of encoding is a pure permutation of literal strings. Although the duplications of some genes make the sense of illegality as the one above, but the order of genes has no direct corresponding to the operation sequences of the jobs. The priority rule-based encoding belongs to this kind.

3.1. Crossover operators

During the past two decades, various crossover operators have been proposed for literal permutation encodings, such as partial-mapped crossover (PMX), order crossover (OX), cycle crossover (CX), position-based crossover, order-based crossover, etc. Note that there are two different basic considerations for designing crossover operators for literal permutation encodings; (1) to make less change when crossing over so as to inherit parents' features as much as possible—all variations of two-cut-point crossover operator belong to this class; (2) to make more change when crossing over so as to explore new patterns of permutation and thereby enhance the search ability—all variations of uniform crossover belong to this class.

3.1.1. Partial-mapped crossover (PMX)

Partial-mapped crossover was proposed by Goldberg and Lingle [9]. It can be viewed as a variation of two-cut-point crossover by incorporating with a special repairing procedure to resolve possible illegitimacy. PMX has the following major steps:

1. Select two cut-points along the string at random. The substrings defined by the two cut-points are called the mapping sections.
2. Exchange two substrings between parents to produce proto-children.
3. Determine the mapping relationship between two mapping sections.
4. Legalize offspring with the mapping relationship.

3.1.2. Order crossover (OX)

Order crossover was proposed by Davis [10]. It can be viewed as a kind of variation of PMX that uses a different repairing procedure. OX has the following major steps:

1. Select a substring from one parent at random.
2. Produce a proto-child by copying the substrings into the corresponding positions as they are in the parent.
3. Delete all the symbols from the second parent, which are already in the substring. The resultant sequence contains the symbols the proto-child needs.
4. Place the symbols into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce an offspring.

3.1.3. Position-based crossover

Position-based crossover was proposed by Syswerda [11]. It is essentially a kind of uniform crossover for literal permutation encodings incorporated with a repairing procedure. Uniform crossover operator was proposed for bit-string encoding by Syswerda [11]. It firstly generates a random mask and then exchanges relative genes between parents according to the mask. A crossover mask is simply a binary string with the same size of chromosome. The parity of each bit in the mask determines, for each corresponding bit in an offspring, which parent it will receive that bit from. Because uniform crossover will produce illegal offspring for literal permutation encodings, position-based crossover uses a repairing procedure to resolve the illegitimacy. Position-based crossover has the following major steps:

1. Select a set of positions from one parent at random.
2. Produce a proto-child by copying the symbols on these positions into the corresponding positions of the proto-child.
3. Delete the symbols which are already selected from the second parent. The resultant sequence contains only the symbols the proto-child needs.
4. Place the symbols into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce one offspring.

3.1.4. Order-based crossover

Order-based crossover was also proposed by Syswerda [11]. It is a slight variation of position-based crossover in that the order of symbols in the selected position in one parent is imposed on the corresponding ones in the other parent.

3.1.5. Cycle crossover (CX)

Cycle crossover was proposed by Oliver et al. [12]. The same as the position-based crossover, it takes some symbols from one parent and the remaining symbols from the other parent. The difference is that the symbols from the first parent are not selected randomly and only those symbols are selected which defined a cycle according to the corresponding positions between parents. CX works as follows:

1. Find the cycle which is defined by the corresponding positions of symbols between parents.
2. Copy the symbols in the cycle to a child with the corresponding positions of one parent.
3. Determine the remaining symbols for the child by deleting those symbols which are already in the cycle from the other parent.
4. Fulfil the child with the remaining symbols.

3.1.6. Linear order crossover (LOX)

Falkenauer and Bouffouix proposed a modified version of order crossover, the linear order crossover [13]. Order crossover tends to transmit the relative positions of genes rather than the absolute ones. In the order crossover, the chromosome is considered to be circular since the operator is devised for the TSP. In the job-shop problem, the chromosome can not be considered to be 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 works as follows:

1. Select sublists from parents randomly.
2. Remove sublist_2 from parent p_1 leaving some 'holes' (marked with h) and then slide the holes from the extremities towards the center until they reach the cross section. Similarly, remove sublist_1 from parent p_2 and slide holes to cross section.

3. Insert sublist$_1$ into the holes of parent p_2 to form the offspring o$_1$ and insert sublist$_2$ into the holes of parent p_1 to form an offspring o$_2$.

The crossover operator can preserve both the relative positions between genes and the absolute positions relative to the extremities of parents as much as possible. The extremities correspond to the high and low priority operations.

3.1.7. Subsequence exchange crossover

Kobayashi et al. [14] proposed a subsequence exchange crossover method, inspired by the similar ideas of Brady [15] and Mühlenbein [16] for TSP. A job sequence matrix is used as encodings. For a n-job m-machine problem, the encoding is an $m \times n$ matrix where each row specifies an operation sequence for each machine. A subsequence is defined as a set of jobs which are processed consecutively on a machine for both parents but not necessarily in the same order. This method includes the following two steps:

1. Identify subsequences one for one machine for the parents.
2. Exchange these subsequences machine by machine among parents to create offspring.

Because it is difficult to maintain the precedence relation among operations in either initial population or offspring by use of the job sequence matrix encoding, a Giffler and Thompson algorithm is used to carefully adjust job orders on each machine to resolve the infeasibility and to convert offspring into active schedules.

3.1.8. Job-based order crossover

Ono et al. [17] gave a variation of their subsequence exchange crossover method, called job-based order crossover, by relaxing the requirement that all jobs in the subsequence must be processed consecutively. The job-based order crossover is also designed for the encoding of job-sequence matrix. It has the following steps:

1. Identify the sets of jobs from parents, one set for one machine.
2. Copy the selected jobs of the first parent onto the corresponding positions of the first child machine by machine. Do the same thing for the second child.
3. Fulfil the unfixed position of the first child by the not-selected jobs from left to right according to the order as they appear in the second parent. Do the same thing for the second child.

3.1.9. Partial schedule exchange crossover

Gen et al. proposed a partial schedule exchange crossover for an operation-based encoding [18]. They consider partial schedules to be the natural building blocks and intend to use such crossover to maintain building blocks in offspring in much the same manner as Holland described [19]. The method has the following steps:

1. Identify a partial schedule in one parent randomly and in the other parent accordingly.
2. Exchange the partial schedules to generate proto-offspring.

3. Determine the missed and exceeded genes for the proto-offspring.
4. Legalize offspring by deleting exceeded genes and adding missed genes.

The partial schedule is identified with the same job in the head and tail of the partial schedule.

3.1.10. Substring exchange crossover

Cheng et al. gave an another version of partial schedule exchange crossover, called substring exchange crossover [20]. It can be viewed as a kind of adaptation of two cut-points crossover for general literal string encodings. It has the following steps:

1. First, select two cut-points along the string at random. Exchange two substrings defined by the two cuts between two parents to produce proto-children.
2. Determine the missed and exceeded genes for each proto-child by making a comparison between two substrings.
3. Legalize the proto-children by replacing the exceeded genes with the missed genes in a random way.

3.2. Mutation

It is relatively easy to make some mutation operators for permutation representation. During the last decade, several mutation operators have been proposed for permutation representation, such as inversion, insertion, displacement, reciprocal exchange mutation, and shift mutation [6].

Inversion mutation selects two positions within a chromosome at random and then inverts the substring between these two positions. *Insertion mutation* selects a gene at random and inserts it in a random position. *Displacement mutation* selects a substring at random and inserts it in a random position. Insertion can be viewed as a special case of displacement in where substring just contains one gene. *Reciprocal exchange mutation* selects two positions at random and then swaps the genes on these positions. *Shift mutation* first chooses a gene randomly and then shifts it to a random position of right or left from the gene's position.

4. Heuristic-Featured Genetic Operators

Inspired by some successful heuristic methods, several heuristic-featured genetic operators have been created for the job-shop scheduling problems. The kernel procedures within these kind of genetic operators are heuristic methods. A method can be identified as a crossover just because that it will merge two parents to produce two offspring and as a mutation just because it will alter a certain amount of genes of one parent to produce an offspring.

4.1. Giffler and Thompson algorithm-based crossover

Yamada and Nakano proposed a crossover operator based on Giffler and Thompson algorithm [21]. The generation procedure of the Giffler and Thompson algorithm is a tree search approach. At each step, it essentially identifies all processing conflicts (the operations competing for the same machine) and an enumeration procedure is used to resolve these conflicts. Yamada and Nakan's procedure of crossover operator is essentially a kind of one-pass procedure but not tree search approach. When generating offspring, at each step, it identifies all processing conflicts like the way of Giffler and Thompson, and then chooses one operation from the conflict set of operations according to one of their parents schedules. Let

o_{ji} = the ith operation of job j,
S = the set of schedulable operations for a given partial schedule,
ϕ_{ji} = the earliest completion time of the ith operation of job j in S,
G_r = the set of conflicting operations in S on machine r
The procedure to generate offspring from two parents works as follows:

1. Let S include all operations with no predecessors initially.
2. Determine $\phi^* = \min\{\phi_{ji} | o_{ji} \in S\}$ and the machine r^* on which ϕ^* could be realized.
3. Let G_{r^*} include all operations $o_{ji} \in S$ that requires machine r^*.
4. Choose one of the operations from G_{r^*} as follows:
   (a) generate a random number $\epsilon \in (0, 1)$ and compare it with mutation rate p_m if $(\epsilon < p_m)$ then choose an arbitrary operation from G_{r^*} as o^*_{ji}
   (b) otherwise select one of two parents with an equal probability, say p_s, find an operation o^*_{ji} which was scheduled earliest in p_s among all the operations in G_{r^*},
   (c) schedule o^*_{ji} in offspring according to ϕ_{ji},

5. Update S as follows:
   (a) remove operation o^*_{ji} from S,
   (b) add the direct successor of operation o^*_{ji} to S,

6. Return to (2) until a complete schedule is generated.

In 4(a), conflict is resolved by choosing an operation randomly, while in 4(b), conflict is resolved by giving priority to the operation which was scheduled earliest in one of its parent p_s among all conflicting operations in G_{r^*}. The parent p_s is selected randomly with equal

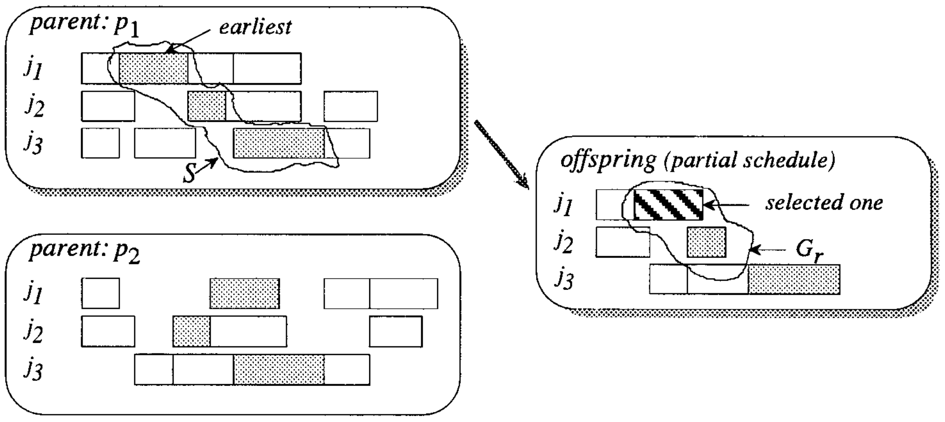

Fig. 4. Giffler and Thompson algorithm-based crossover.

probability. Therefore, Yamada and Nakano's approach is essentially based on priority dispatching heuristics not a pure Giffler and Thompson approach. At each step, one operation is selected to add into the partial schedule of offspring and conflicts among operations are resolved by specifying priority to operations according to their parents' schedules.

Fig. 4 demonstrates the selection of operation in 4(b). For the offspring, operations o_{11}, o_{21}, o_{31} and o_{32} are scheduled. The schedulable operations are $S=\{o_{12}, o_{22}, o_{33}\}$. Assume that the set of conflicting operations are $S=\{o_{12}, o_{22}\}$. Parent p_1 is selected. Because operation o_{12} was scheduled earliest in parent p_1, it is scheduled into offspring.

4.2. Neighborhood search-based mutation

In conventional genetic algorithms, mutation is a background operator, which is just used to produce small perturbations on chromosomes in order to maintain the diversity of population. Cheng et al. proposed a mutation inspired by neighbor search technique [20,22]. It is then not a background operator and is used to perform intensive search in order to find an improved offspring.

Many definitions may be considered for the neighborhood of a schedule. For operation-based representation, the neighborhood for a given chromosome can be considered as the set of chromosomes (schedules) transformable from a given chromosome by exchanging the positions of λ genes (randomly selected and nonidentical genes). A chromosome (schedule) is said to be λ-optimum, if it is better than any others in the neighborhood according to some measure. Let us see an example of a 4-job 4-machine problem. Suppose genes on positions 4, 8 and 12 are randomly selected. They are (4 3 2) and their possible permutations are (3 4 2), (3 2 4), (2 3 4), (2 4 3) and (4 2 3). The permutations of the genes together with remaining genes of the chromosome form the neighbor chromosomes shown in Fig. 5. Then we evaluate all neighbor chromosomes and the best one is used as the offspring of mutation. The overall procedure of mutation is given below:

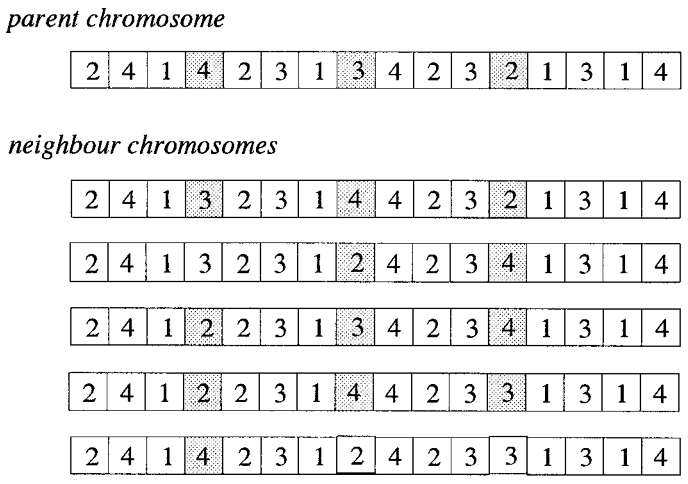

Fig. 5. Neighbor schedules.

```
begin
  $i \leftarrow 0$
  while    $(i \leq \text{pop-size} \times p_m)$ do
    choose an unmutated chromosome randomly
    pick up $\lambda$ nonidentical genes randomly from it
    make its neighbors based on all permutations of the genes
    evaluate all neighbor schedules
    select the best neighbor as offspring
    $i \leftarrow i + 1$
  end.
```

5. Hybrid genetic algorithms

The role of local search in the context of the genetic algorithms has been receiving serious consideration and many successful applications are strongly in favor of such a hybrid approach. Because of the complementary properties of genetic algorithms and conventional heuristics, a hybrid approach often outperforms either method operating alone. The hybridization can be done in a variety of ways, including:

1. Incorporate heuristics into initialization to generate well-adapted initial population. In this way, a hybrid genetic algorithm with elitism can guarantee to do no worse than the conventional heuristic does.
2. Incorporate heuristics into evaluation function to decode chromosomes to schedules.
3. Incorporate local search heuristic as an add-on extra to the basic loop of genetic algorithm, working together with mutation and crossover operators, to perform quick and localized optimization in order to improve offspring before returning it to be evaluated.

One of the most common forms of hybrid genetic algorithms is to incorporate local search techniques as an add-on extra to the main genetic algorithms loop of recombination and selection. With the hybrid approach, genetic algorithms are used to perform global exploration among the population, while heuristic methods are used to perform local exploitation around chromosomes.

5.1. Combining genetic algorithms with local search

A common form of hybrid genetic algorithms is the combination of local search with the genetic algorithm. The genetic algorithm is good at global search but slow to converge, while local search is good at fine-tuning but often falls into local optima. The hybrid approach complements the properties of the genetic algorithm and the local search heuristic methods. The genetic algorithm is used to perform global search to escape from local optima, while the local search is used to conduct fine-turning.

The local search in this context can be thought of as being analogous to a kind of learning that occurs during the lifetime of an individual string. With standard genetic algorithm, the selection of chromosomes is based on the instantaneous fitness at their birth; while with hybrid methods, the selection is based on the fitness at the end of the individual life, the life being performed by local search. The improved offspring will pass on its traits acquired during this learning (local optimization) to future offspring through common crossover [23]. This phenomenon is called *Lamarkian Evolution*. So this hybrid approach can be viewed as the combination of *Darwin's evolution* with *Lamarkian's evolution* [24,25]. The overall of the hybrid procedure (below) is shown in Fig. 6.

```
begin
  $t \leftarrow 0$
  initialize $P(t)$ (job sequences)
  local search to improve chromosomes
  evaluate $P(t)$
  while (not termination condition) do
  begin
    recombine $P(t)$
    local search to improve chromosomes
    evaluate $P(t)$
    select to next population $P(t)$
    $t \leftarrow t + 1$
end.
```

5.2. Combining genetic algorithm with Giffler and Thompson method

Dorndorf and Pesch proposed a priority rule-based encoding method for the job-shop scheduling problem and developed a hybrid version of genetic algorithms by incorporating the well-known Giffler and Thompson algorithm [26]. In the hybrid approach, the genetic algorithm is used to evolve a sequence of priority dispatching rules and the Giffler and Thompson algorithm is used to deduce a schedule from the encoding of priority dispatching rules. The overall procedure of the hybrid method is given as follows:

```
begin
  $t \leftarrow 0$
  initialize $P(t)$ (job sequences)
```

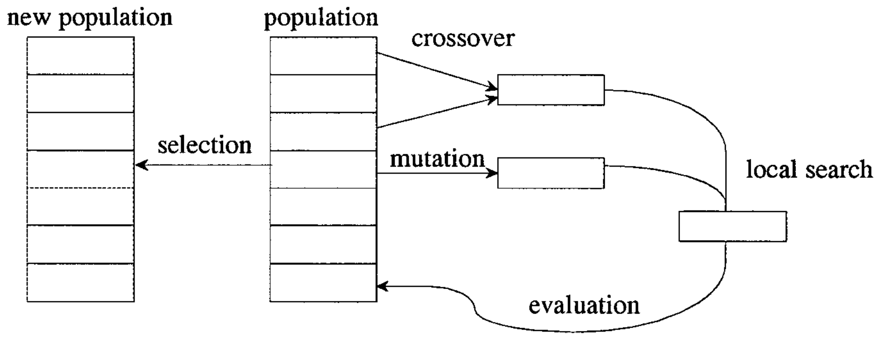

Fig. 6. Hybrid procedure: genetic algorithms + local search.

```
    local search to improve chromosomes
    evaluate $P(t)$
    while (not termination condition) do
    begin
      recombine $P(t)$
      deduce schedules with Giffler and Thompson algorithms
      evaluate $P(t)$
      select to next population $P(t)$
      $t \leftarrow t + 1$
  end
```

Priority rules are probably the most frequently applied heuristics for solving scheduling problems in practice because of their ease of implementation and their low time complexity. The algorithm of Giffler and Thompson can be considered as the common basis of all priority rule-based heuristics [27]. The generation procedure of Giffler and Thompson is a tree-structured approach. The nodes in the tree correspond to partial schedules, the arcs represent the possible choices, and the leaves of the tree are the set of enumerated schedules. For a given partial schedule, the algorithm essentially identifies all processing conflicts, i.e. operations competing for the same machine, and an enumeration procedure is used to resolve these conflicts in all possible ways at each stage [1]. Instead of enumerative tree search, Dorndorf and Pesch used priority dispatching rules to resolve these conflicts, i.e. a gene specifies a priority rule for selecting one operation among the conflicting operations at a time. Therefore, they in fact used a kind of one-pass priority dispatching heuristic, but not a pure version of the Giffler and Thompson algorithm.

Generation procedures operate on a set of schedulable operations at each stage. Schedulable operations are operations unscheduled yet with immediately scheduled predecessors. This set can be simply determined from precedence structure. The number of stages for a one-pass procedure is equal to the number of operations $m \times n$. At each stage, one operation is selected to add into *partial schedule*. Conflicts among operations are solved by priority dispatching rules specified by a corresponding gene. Following the notations of Baker [1], we have:

PS_t = a partial schedule containing t scheduled operations,
S_t = the set of schedulable operations at stage 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.

For a given active partial schedule, the potential start time σ_i is determined by the completion time of the direct predecessor of operation i and the latest completion time on the machine required by operation i. The larger of these two quantities is σ_i. The potential finishing time ϕ_i is simply $\sigma_i + t_i$, where t_i is the processing time of equation i. The procedure to generate active schedules works as follows:

1. Let $[p_1, p_2 \ldots, p_{mn}]$ be a given chromosome.
2. Let $t = 1$ and begin with PS_t as the null partial schedule and S_t include all operations with no predecessors.
3. 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.

4. 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.
5. Update S_t by removing the selected operation from it and adding the direct successor of the operation to it. Increment t by one.
6. Return to (3) until a complete schedule is generated.

The remaining problem is to identify an effective priority rule. For an extensive summary and discussion, see Panwalkar and Iskander [28], Haupt [29], and Blackstone et al. [30]. Table 1 consists of some of the priority rules commonly used in practice.

Kobayashi et al. proposed another version of hybrid genetic algorithms with the Giffler and Thompson algorithm [14,17]. The job sequence matrix encoding method is used in their studies. The Giffler and Thompson algorithm is used to deduce an active schedule from the encoding. The procedure to generate an active schedule works as follows:

1. Let $t = 1$ and S_t include all operations with no predecessors.
2. Determine $\phi_t^* = \min_{i \in S_t}\{\phi_i\}$ and the machine m^* on which ϕ_t^* could be realized.
3. If more than one such machine exists, tie is broken by a random choice.
4. Form conflict set C_t which includes all operations $i \in S_t$ with $\sigma_i < \phi_t^*$ that requires machine m^*.
5. Check the first unscheduled operation on machine m^*. If it belongs to the conflict set C_t, keep it unchanged; otherwise, swap it with an operation randomly selected from C_t.
6. Update S_t by removing the scheduled operation from it and adding the direct successor of the operation to it. Increment t by one.
7. Return to (2) until all operations are scheduled.

Table 1
A list of job shop dispatch rules

Rule	Description
SPT (Shortest Processing Time)	Select an operation with shortest processing time
LPT (Longest Processing Time)	Select an operation with longest processing time
MWR (Most Work Remaining)	Select an operation for the job with the most total processing time remaining
LWR (Least Work Remaining)	Select an operation for the job with the least total processing time remaining
MOR (Most Operations Remaining)	Select an operation for job with the greatest number of operations remaining
LOR (Least Operations Remaining)	Select an operation for the job with the smallest number of operations remaining
EDD (Earliest Due Date)	Select a job with earliest due date
FCFS (First Come First Served)	Select the first operation in the queue of jobs for the same machines
RANDOM (Random)	Select an operation at random

5.3. Combining genetic algorithms with bottleneck shifting heuristic

Dorndorf and Pesch proposed a machine-based encoding for the job-shop problem and developed a hybrid version of genetic algorithms by incorporating the well-known bottleneck shifting heuristic [26]. In this hybrid approach, genetic algorithm is used to evolve a sequence of machines and the bottleneck shifting heuristic is used to deduce a schedule from the encoding of machine sequence.

The *shifting bottleneck heuristic*, proposed by Adams et al. [31] is one of the most powerful procedures for the job-shop scheduling problem. It sequences the machines one by one, successively, taking each time the machine 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 certain one-machine scheduling problems that are 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 different sequences of bottleneck machines. The quality of the schedules obtained by shifting bottleneck heuristic heavily depends on the sequence of bottleneck machines. Adams et al. also proposed an enumerative version of shifting bottleneck heuristic to consider different sequences of machines [31].

Instead of enumerative tree search, Dorndorf and Pesch proposed a genetic strategy to determine the best machine sequence for shifting bottleneck heuristic [26]. A chromosome is a list of ordered machines. The genetic algorithms here are used to evolve those chromosomes to find out a better sequence of machines for shifting bottleneck heuristic. The difference between 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 determined by a given chromosome. Therefore, there is no need to identify one bottleneck machine at each iteration.

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:

1. Let $M_0 \leftarrow \phi$, $i \leftarrow 1$ and the chromosome $[m_1, m_2, \ldots, m_m]$
2. Sequence machine m_i optimally. Update set $m_0 \leftarrow M_0 \cup \{M_i\}$.
3. Reoptimize the sequence of each critical machine $m_i \in M_0$ in turn, while keeping the other sequences fixed.
4. Let $i \leftarrow i + 1$. Then if $i > m$, stop; otherwise go to (2).

The details of (3) can be found in Adams et al. [31] or in Applegate and Cook [32].

5.4. Combining genetic algorithm with beam search

The hybrid approach proposed by Holsapple et al. is to combine the genetic algorithm with beam search technique [33]. Roughly, beam search is used to convert a chromosome (job-based representation) to a schedule. Beam search is a heuristic refinement of breadth-first search that relies on the notion of *beam width* to restrict the number of nodes that we branch form at each stage (or level) of the search tree. At each level of the search process, all nodes are evaluated using a predefined evaluation function. Then the beam width w is used to pick the w best nodes (in terms of their evaluations) to branch out from (i.e. expand) at the current level to generate the offspring nodes at the next level in the tree. The remaining nodes at the current level are pruned out permanently. If there are only fewer than w nodes available to begin with, then all of these are chosen for expansion. From each of the chosen w nodes, all possible successor nodes are generated. The process continues until no further expansion is possible. Fig. 7(a) illustrates the beam search with a simple example.

Filtered beam search is a refinement of the beam search method. It uses one other construct called the *filter width* to further prune the state space. For each of the w nodes selected for expansion at a level, the filter width f determines the maximum number of successor nodes that could be generated. That is, we could generate as many successor nodes as possible but not exceeding f. For a given value of f, the choice of which f successors (from the set of possible successor nodes) to generate is random. Both w and f are user-specified quantities. The search process is given in Fig. 7(b).

Holsapple et al. adopt a job-based representation. Genetic algorithm is used to manipulate the job sequence and filtered beam search is used to generate the 'best' schedule for a given job sequence (or a chromosome). We can consider the beam search as a part of the evaluation function, that is, converting a job sequence (the genotype of chromosome) to a schedule (the phenotype of chromosome), but it does more than converting. They considered a kind of static scheduling problem in flexible manufacturing contexts. The difference of the problem with classic job-shop problem is that any operation of jobs can be processed on any machine. In such a case, a chromosome of the job-based representation corresponds to many feasible schedules, so they use beam search to find out the 'best' one among the feasible schedules for a

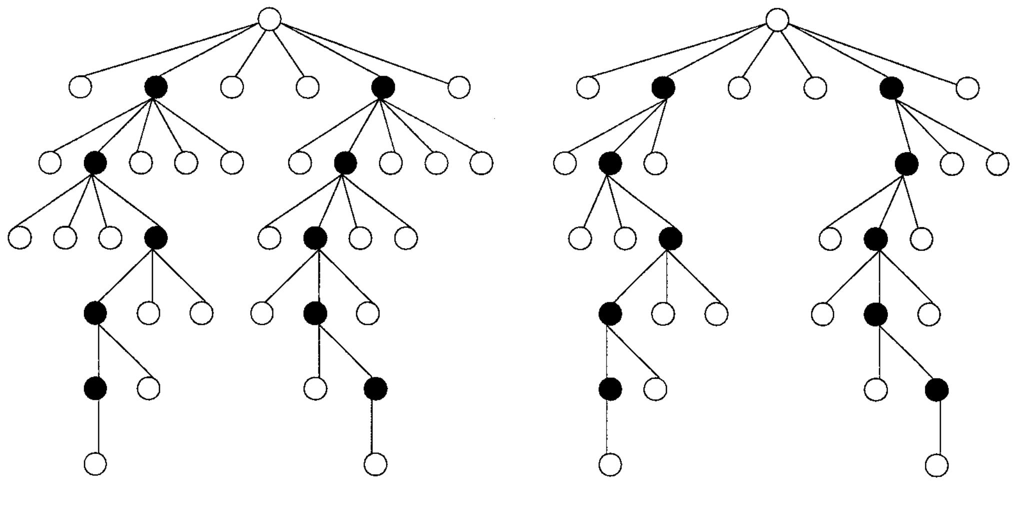
(a) beam search with $w=2$
(b) filtered beam search with $w=2$ and $f=3$

Fig. 7. Beam search and filtered beam search.

given chromosome. For a classical job-shop scheduling problem, the processing machine for each operation is predetermined, so a job-based chromosome corresponds only to one feasible schedule as illustrated in the last section. In this case, we do not need any tree search techniques such as beam search to convert a chromosome into a feasible schedule. The overall procedure of their genetic algorithms is given below:

```
begin
  $t \leftarrow 0$
  initialize $P(t)$ (job sequences)
  beam search to generate best schedules for each chromosome
  evaluate each schedule
  while (not termination condition) do
  begin
    recombine $P(t)$
    beam search to generate best schedules for each chromosome
    evaluate each schedule
    select to next population $P(t)$
    $t \leftarrow t + 1$
end
```

6. Discussion

During the past decade, the job-shop scheduling problem has become a hot topic in the genetic algorithm field. This is because this problem exhibits all aspects of constrained combinatorial problems and serves as a paradigm for testing any new algorithmic ideas [34].

Note that a solution of genetic algorithms is not necessarily a solution of a given problem. It may contain the necessary information for constructing a solution to a given problem. As we know, there are two kinds of order relations in the job-shop scheduling problem: (1) the operation sequence on each machine and (2) the precedence constraints among operations for a job. The first one must be determined by a solution method, while the second must be maintained in a schedule. If these two kinds of order relations are mingled together in the chromosomes, it will increase the burden on genetic operators of how to handle the infeasibility and the illegality of offspring. It is a well studied issue how to represent a solution of job-shop scheduling problem in genetic algorithms. Because of the existence of precedence constraints among operations, the permutation of operations usually corresponds to infeasible solutions. A common way suggested by many researchers is to cope with the trouble of precedence constraints in a special schedule builder, then the job-shop scheduling problem is treated as a permutation problem: genetic algorithms are used to evolve an appropriate permutation of operations on each machine and a schedule builder is used to produce a feasible solution according to the permutation as well as the precedence requirements.

Most encoding methods for job-shop scheduling problems belong to the kind of general literal strings in the sense that a symbol can be repeated in a chromosome. Among them, preference list-based encoding is well used in the studies, which consists of several substrings. Because each substring of the encoding is a pure literal string, a common way suggested by many studies is to apply genetic operators at each substring. In this way, the general literal string is essentially treated as several pure literal strings, and many genetic operators developed for pure literal strings can then be used directly. This multiple use of genetic operators will greatly increase the amount of computation as the problem size increases. It is hoped that the genetic search ability can be increased accordingly. Note that because multiple genetic operators are independently used in each substring, one of the consequences of such multiple use is that precedence constraints among operations may be violated. This is why we need a builder to readjust the order of operations to obtain a feasible solution. The trade-off among the complexity of encoding and builder should therefore be taken into account.

A trend in the genetic job-shop scheduling practice is to incorporate local search techniques into the main loop of the genetic algorithm to convert each offspring into an active schedule. In general, feasible permutations of operations correspond to the set of semi-active schedules. As the size of the problem increases, the size of the set of semi-active schedules will become larger and larger. Because genetic algorithms are not very good at fine-tuning, searches within the huge space of semi-active schedules will make genetic algorithms less effective. As we know, optima to the job-shop scheduling problem lies within the set of active schedules, which is much smaller than the set of semi-active ones. By using a permutation encoding, we have no way to confine the genetic search within the space of active schedule. One possible way is to leave the genetic search in the whole search space of semi-active schedules while converting each chromosome into an active schedule by an add-on extra procedure, the schedule builder. One successful attempt is to incorporate the Giffler and Thompson algorithm into the genetic algorithms to perform the conversion. The Giffler and Thompson algorithm is originally a kind of enumeration method. In most kinds of hybrid approaches, it is used in the one-pass heuristic way to readjust the operation order in order to resolve the precedence constraints and to convert each offspring into an active schedule.

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] Baker K. Introduction to sequencing and scheduling. New York: John Wiley & Sons, 1974.
[2] Broucker P. Scheduling algorithms, 2nd ed. Berlin: Springer-Verlag, 1998.
[3] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job-shop problem. Management Sciences 1996;42(6):797–813.
[4] Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation. International Journal of Computers and Industrial Engineering 1996;30:983–97.
[5] Michalewicz Z. Genetic algorithm + data structure = evolution programs, 2nd ed. New York: Springer-Verlag, 1994.
[6] Gen M, Cheng R. Genetic algorithms and engineering design. New York: John Wiley & Sons, 1997.

[7] Davis L, editor. Handbook of genetic algorithm. New York: Van Nostrand Reinhold, 1991.
[8] Renders J, Bersini H. Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: Fogel D, editor. Proceedings of the First IEEE Conference on Evolutionary Computation. FL: IEEE Press, 1994. p. 312–7.
[9] Goldberg D, Lingle R. Alleles, Loci and the traveling Salesman Problem. In: Grefenstette JJ, editor. Proceedings of the First International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum Associates, 1985. p. 154–9.
[10] Davis L. Applying adaptive algorithms to epistatic domains. In: Proceedings of the International Joint Conference on Artificial Intelligence, 1985. p. 162–4.
[11] Syswerda G. Uniform crossover in genetic algorithms. In: Schaffer J, editor. Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann Publishers, 1989. p. 2–9.
[12] Oliver I, Smith D, Holland J. A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ, editor. Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum Associates, 1987. p. 224–30.
[13] Falkenauer E, Bouffoix S. A genetic algorithm for job shop. In: Proceedings of the 1991 IEEE International Conference on Robotics and Automation, 1991. p. 824–9.
[14] Kobayashi S, Ono I, Yamamura M. An efficient genetic algorithm for job shop scheduling problems. In: Eshelman LJ, editor. Proceedings of the Sixth International Conference on Genetic Algorithms. San Francisco, CA: Morgan Kaufmann Publishers, 1995. p. 506–11 July 1995.
[15] Brady R. Optimization strategies gleaned from biological evolution. Nature 1985;317(31):804–6.
[16] Mühlenbein H, Schlouter M, Kraemer O. Evolution algorithms in combinatorial optimization, Parallel Computing 7.
[17] Ono I, Yamamura M, Kobayashi S. A genetic algorithms for job-based order crossover. In: Fogel D, editor. Proceedings of the Third IEEE Conference on Evolutionary Computation. Japan: IEEE Press, 1996. p. 547–52.
[18] Gen M, Tsujimura Y, Kubota E. Solving job-shop scheduling problem using genetic algorithms. In: Gen M, Kobayashi T, editors. Proceedings of the 16th International Conference on Computer and Industrial Engineering. Japan: Ashikaga, 1994. p. 576–9.
[19] Holland J. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press, 1975.
[20] Cheng R. A study on genetic algorithms-based optimal scheduling techniques, PhD thesis, Tokyo Institute of Technology, 1997.
[21] Yamada T, Nakano R. A genetic algorithm applicable to large-scale job-shop problems. In: Männer R, Manderick B, editors. Proceedings of the Second International Conference on Parallel Problem Solving from Nature. North-Holland: Elsevier Science Publishers, 1992. p. 281–90.
[22] Tsujimura Y, Gren M. Parts loading scheduling in a flexible forging machine using an advanced genetic algorithm. Journal of Intelligent Manufacturing 1999;10(2):149–56.
[23] Kennedy S. Five ways to a smarter genetic algorithm, AI Expert 1993:35–38.
[24] Moscato P, Norman M. A memetic approach for the travelling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Proceedings of the International Conference on Parallel Computing and Transputer Applications, 1992 Amsterdam.
[25] Fogarty T, editor. Evolutionary computing. Berlin: Springer-Verlag, 1994.
[26] Dorndorf U, Pesch E. Evolution based learning in a job shop scheduling environment. Computers and Operations Research 1995;22(1):25–40.
[27] Storer R, Wu S, Vaccari R. New search spaces for sequencing problems with application to job shop scheduling. Management Science 1992;38(10):1495–1510.
[28] Panwalker S, Iskander W. A survey of scheduling rules. Operations Research 1977;25:45–61.
[29] Haupt R. A survey of priority-rule based scheduling problem. OR Spektrum 1989;11:3–16.
[30] Blackstone J, Phillips D, Hogg G. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Researches 1982;20:27–45.
[31] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. International Journal of Flexible Manufacturing Systems 1988;34(3):391–401.
[32] Applegate D, Cook W. A computational study of the job shop scheduling problem. ORSA Journal of Computing 1991;3(2):149–56.

[33] Holsapple C, Jacob V, Pakath R, Zaveri J. A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. IEEE Transactions on Systems, Man, and Cybernetics 1993;23:953–71.
[34] Gen M, Cheng R. Genetic algorithms and engineering optimization. New York: John Wiley & Sons, 1999.