A Hybrid of Genetic Algorithm and Bottleneck Shifting for Flexible Job Shop Scheduling Problem

Jie Gao
School of Management,
Xi'an Jiaotong University
Xi'an, 710049, China
+86(29)8266-8748
calebgao@yahoo.com

Mitsuo Gen
Graduate School of Information, Production & Systems,
Waseda University
Kitakyushu, 808-0135, Japan
+81(93)692-5273
gen@waseda.jp

Linyan Sun
School of Management,
Xi'an Jiaotong University
Xi'an, 710049, China
+86(29)8266-8748
lysun@mail.xjtu.edu.cn

ABSTRACT

Flexible job shop scheduling problem (fJSP) is an extension of the classical job shop scheduling problem, which provides a closer approximation to real scheduling problems. We develop a new genetic algorithm hybridized with an innovative local search procedure (bottleneck shifting) for the fJSP problem. The genetic algorithm uses two representation methods to represent solutions of the fJSP problem. Advanced crossover and mutation operators are proposed to adapt to the special chromosome structures and the characteristics of the problem. The bottleneck shifting works over two kinds of effective neighborhood, which use interchange of operation sequences and assignment of new machines for operations on the critical path. In order to strengthen the search ability, the neighborhood structure can be adjusted dynamically in the local search procedure. The performance of the proposed method is validated by numerical experiments on several representative problems.

Categories and Subject Descriptors

F.2.2 Sequencing and scheduling

General Terms

Algorithms, Performance

Keywords

Flexible job shop scheduling problem; genetic algorithms; bottleneck shifting; neighborhood structure

1. INTRODUCTION

Flexible job shop is a generalization of the job shop and the parallel machine environment, which provides a closer approximation to a wide range of real manufacturing systems. Bruker and Schlie [1]were among the first to address the fJSP problem. They developed a polynomial algorithm for solving the flexible job shop scheduling problem with two jobs. Chambers [2] developed a tabu search algorithm to solve the problem. Mastrolilli and Gambardella [3] proposed two neighborhood functions for the fJSP problem. Yang [4] presented a new genetic algorithm (GA)-based discrete dynamic programming approach. Kacem and Borne [5] proposed the approach by localization to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for the fJSP problem. Wu and Weng [6] considered the problem with job earliness and tardiness objectives, and proposed a multiagent scheduling method. Xia and Wu [7] treated this problem with a hybrid of particle swarm optimization and simulated annealing as a local search algorithm. Zhang and Gen [8] proposed a multistage operation-based genetic algorithm to deal with the fJSP problem from a point view of dynamic programming.

In this paper, a hybrid genetic algorithm (hGA) is employed to solve the fJSP problem. The hGA uses two representations to adapt to the nature of this problem. One representation is used in initialization and mutation, and the other is used for crossover operation. In order to strengthen the search ability, bottleneck shifting serves as a kind of local search method under the framework of GA, which only investigates the neighbor solutions that have possibilities to improve the initial solution.

We formulate the fJSP problem in Section 2. Section 3 presents the representation method, decoding procedure and the overall procedure of the proposed hGA. The details of the genetic algorithm and the bottleneck shifting are presented in Section 4 and 5, respectively. In Section 6, we present computational study on several well-known fJSP benchmark problems and compare our results with the results obtained by previous approaches. Some final concluding remarks are given in Section 7.

2. FLEXIBLE JOB SHOP SCHEDULING PROBLEM

The flexible job shop scheduling problem is as follows: n jobs are to be scheduled on m machines. Each job i represents n_i ordered operations. The execution of each operation k of job i (noted as o_{ik}) requires one machine j selected from a set of available machines for o_{ik} called A_{ik}, and will occupy that machine t_{ikj} time units until the operation is completed. The fJSP problem is to assign operations on machines and to schedule operations assigned on each machine, subject to the constraints that:

1. The operation sequence for each job is prescribed;
2. Each machine can process only one operation at a time.

In this study, we manage to minimize the following three criteria:

1. Makespan (c_{M}) of the jobs;

2. Maximal machine workload (w_M), *i.e.*, the maximum working time spent at any machine;
3. Total workload (w_T), which represents the total working time over all machines.

The notation used in this paper is summarized in the following:

- Indices

  i, h: index of jobs, $i, h = 1, 2, \ldots, n$;
  j: index of machines, $j = 1, 2, \ldots, m$;
  k, g: index of operation sequence, $k, g = 1, 2, \ldots, n_i$

- Parameters

  n: total number of jobs;
  m: total number of machines;
  n_i: total number of operations of job i;
  o_{ik}: the k-th operation of job i;
  A_{ik}: the set of available machines for the operation o_{ik};
  t_{ikj}: processing time of the operation o_{ik} on machine j

- Decision variables

$$x_{ikj} = \begin{cases} 1, \text{ if machine } j \text{ is selected for the operation } o_{ik} \\ 0, \text{ otherwise} \end{cases}$$

  c_{ik} : completion time of the operation o_{ik}

The fJSP model is then given as follows:

$$\min \quad c_M(\boldsymbol{x}, \boldsymbol{c}) = \max_{1 \le i \le n} \left\{ c_{in_i} \right\} \tag{1}$$

$$\min \quad w_M(\boldsymbol{x}, \boldsymbol{c}) = \max_{1 \le j \le m} \left\{ \sum_{i=1}^{n} \sum_{k=1}^{n_i} t_{ikj} x_{ikj} \right\} \tag{2}$$

$$\min \quad w_T(\boldsymbol{x}, \boldsymbol{c}) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} \sum_{j=1}^{m} t_{ikj} x_{ikj} \tag{3}$$

$$\text{s.t.} \quad c_{ik} - c_{i(k-1)} \ge t_{ikj} x_{ikj}, \quad k = 2, \cdots, n_i; \ \forall\, i, j \tag{4}$$

$$[(c_{hg} - c_{ik} - t_{hgj}) x_{hgj} x_{ikj} \ge 0] \vee [(c_{ik} - c_{hg} - t_{ikj}) x_{hgj} x_{ikj} \ge 0], \ \forall\, (i,k), (h,g), j \tag{5}$$

$$\sum\nolimits_{j \in A_{ik}} x_{ikj} = 1, \ \forall\, k, i \tag{6}$$

$$x_{ikj} \in \{0, 1\}, \quad \forall\, j, k, i \tag{7}$$

$$c_{ik} \ge 0, \ \forall\, k, i \tag{8}$$

Inequity (4) describes the operation precedence constraints. In inequity (5), since one or the other constraint must hold, it is called disjunctive constraint. It represents the operation un-overlapping constraints. Equation (6) states that one machine must be selected from a set of available machines for each operation.

3. HYBRID GENETIC ALGORITHM FOR fJSP

Ever since the genetic algorithms were introduced by Holland in 1975, they have been proven to be powerful techniques for constrained optimization and combinatorial optimization problems. One promising approach for improving the convergence speed to the global optimum is the use of local search in genetic algorithms.

3.1 Two-Vector Gen *et al.*'s Representation

The GA's structure and parameter setting affect its performance. However, the primary determinants of a GA's success or failure are the coding by which its genotypes represent candidate solutions and the interaction of the coding with the GA's recombination and mutation operators.

As mentioned above, the fJSP problem is a combination of machine assignment and operation scheduling decisions. A solution can be described by the assignment of operations on machines and the processing sequence of operations on the machines. In this paper, the chromosome is therefore composed of two parts:

1) Machine assignment vector (hereafter called v_1);
2) Operation sequence vector (hereafter called v_2).

Consider a flexible job shop scheduling problem with four machines and four jobs, where each job requires four operations. It is rather easy to represent the machine assignment in a row. In each machine assignment vector v_1, $v_1(r)$ represents the machine selected for the operation indicated at locus r (hereafter, we call it operation r for shortness). An example of the machine assignment vector is shown in figure 1.

Locus: (r)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operation Indicated	$o_{1,1}$	$o_{1,2}$	$o_{1,3}$	$o_{1,4}$	$o_{2,1}$	$o_{2,2}$	$o_{2,3}$	$o_{2,4}$	$o_{3,1}$	$o_{3,2}$	$o_{3,3}$	$o_{3,4}$	$o_{4,1}$	$o_{4,2}$	$o_{4,3}$	$o_{4,4}$
Machine Assignment: $v_1(r)$	4	3	3	1	2	4	1	4	3	1	2	1	2	4	4	3

Figure 1. Illustration of the machine assignment vector

Permutation representation is perhaps the most natural representation of operation sequences. Unfortunately because of the existence of precedence constraints, not all the permutations of the operations define feasible sequences. For job shop scheduling problem, Gen *et al.* proposed an alternative: they name all operations for a job with the same symbol and then interpret them according to the order of occurrence in the sequence of a given chromosome [9][10]. Gen *et al.*'s method can also be used to represent the operation sequences for the fJSP problem. Each job i appears in the operation sequence vector (v_2) exactly n_i times to represent its n_i ordered operations. For example, the operation sequence represented in figure 2 can be translated into a list of ordered operations below:

$$o_{2,1} \succ o_{4,1} \succ o_{3,1} \succ o_{1,1} \succ o_{4,2} \succ o_{1,2} \succ o_{4,3} \succ o_{3,2} \succ o_{2,2} \succ o_{1,3} \succ o_{3,3} \succ o_{1,4} \succ o_{2,3} \succ o_{4,4} \succ o_{3,4} \succ o_{2,4}.$$

Locus: Priority (s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operation Sequence: $v_2(s)$	2	4	3	1	4	1	4	3	2	1	3	1	2	4	3	2

Figure 2. Illustration of the operation sequence vector

The main advantages of the two-vector Gen *et al.*'s representation are that each possible chromosome always represent a feasible solution candidate, and that the coding space is smaller than that of permutation representation.

3.2 Priority-Based Decoding and Operation Sequence Reordering

In this paper we use priority-based decoding, where each operation searches the earliest available time interval for implementing on its assigned machine in the order represented by operation sequence vector. The l-th time interval on machine j $[t_{jl}^{E}, t_{jl}^{L}]$ is available for o_{ik}, if

$$\begin{cases} \max\{t_{jl}^{\mathrm{E}}, c_{i(k-1)}\} + t_{ikj} \le t_{jl}^{\mathrm{L}}, & \text{if } k \ge 2; \\ t_{jl}^{\mathrm{E}} + t_{ikj} \le t_{jl}^{\mathrm{L}}, & \text{if } k = 1. \end{cases} \quad (9)$$

then, we take $\max\{t_{jl}^{\mathrm{E}}, c_{i(k-1)}\}$ (if $k \ge 2$) or t_{jl}^{E} (if k=1) as the starting time of o_{ik}. Let N be the total number of operations, $R(i, k)$ be the locus of o_{ik} in machine assignment vector. The decoding procedure is shown in figure 3.

```
procedure: Priority-Based Decoding Procedure
input: fJSP data set, chromosome $v_1(r)$, $v_2(s)$
output: an active schedule
begin
  for $i$=1 to $n$
    $k_i \leftarrow 0$;   // $k_i$ represents the no. of operations already
                             assigned for job i
  for $s$=1 to $N$
    $i \leftarrow v_2(s)$, $k_i \leftarrow k_i+1$, $j \leftarrow v_1(R(i, k_i))$;
    search the an available idle time interval from left to right
      on machine $j$ for operation $k_i$ of job $i$;
    if such an available time interval is found, then assign the
      operation there; else schedule the operation at the end of
      machine $j$;
  end
  output the active schedule;
end
```

Figure 3. Priority-based decoding procedure

The priority-based decoding method allows an operation to search the earliest available time interval between the operations already scheduled on the machine. Hence, the operation sequence represented by the operation sequence vector may be different from the sequence in terms of the operation starting time of the decoded schedule.

Chromosomes are evaluated in phenotype space, while traits of the parental solutions are passed down to their children by means of manipulating chromosomes. Hence in order to facilitate offsprings to inherit the operation sequence information of their parents, it is necessary to unify the operation sequence in the chromosome with the sequence in the corresponding decoded schedule. The operation sequence in a chromosome is reordered according to the operation starting time of the decoded schedule before the chromosome involves crossover and mutation operations.

3.3 Framework of the Hybrid Genetic Algorithm

Two individuals with high fitness values are likely to have dissimilar machine assignment and operation sequences, and the recombination may result in offsprings of poor performance. This means that the genetic operations by themselves have limited ability in finding the global optimal solution. In this study, bottleneck shifting serves as one kind of local search method and is hybridized with the genetic algorithm. The framework of the proposed hybrid genetic algorithm is illustrated in figure 4. The details of the proposed genetic algorithm and the bottleneck shifting are discussed in the next two sections.

```
procedure: Hybrid Genetic Algorithm
input: fJSP data set, GA parameters
output: a best schedule
begin
  $t \leftarrow 0$;
  initialize $P(t)$ with two-vector Gen et al.'s representation;
  fitness eval($P$) by priority-based decoding;
  reorder operation sequence according to operation starting
    time;
  while (not termination condition) do
    crossover $P(t)$ to yield $C(t)$ by exchange crossover and
      enhanced order crossover;
    mutation $P(t)$ to yield $C(t)$ by allele-based mutation and
      immigration mutation;
    improve $P(t)$ and $C(t)$ to yield $P'(t)$ and $C'(t)$ by bottleneck
      shifting;
    fitness eval($P'$, $C'$) by priority-based decoding;
    select $P(t+1)$ from $P'(t)$ and $C'(t)$ by mixed sampling;
    reorder operation sequence according to operation starting
      time;
    $t \leftarrow t+1$;
  end
  output a best schedule;
end
```

Figure 4. Framework of the proposed hybrid genetic algorithm

4. GENETIC OPERATORS

In this study, the initial population is generated randomly without using any information about the fJSP domain, in order to maintain the diversity of individuals. Starting from the initial population, genetic operations then evolve the population to converge to the optimal solution.

4.1 Crossover Operators

In genetic algorithms, crossover recombines two chromosomes to generate a number of children. Offspring of crossover should represent solutions that combine substructures of their parental solutions. Compared to other metaheuristics, such as: tabu search (TS), simulated annealing (SA), ant colony optimization (ACO), crossover may be the most distinct operation of GAs, making heritability especially critical.

In the two-vector Gen *et al*'s representation, each gene of the operation sequence vector does not indicate a concrete operation of a job but refers to an operation that is context-dependent. Hence, it is hard for crossover to generate offsprings that combine the characteristics of their parental solutions. In order to strengthen the heritability, a chromosome of the original two-vector Gen *et al*'s representation is transformed into the style of the so called two-vector permutation representation.

In a chromosome of the two-vector permutation representation as shown in figure 5, $v_3(s)$ denotes the operation with the s-th priority in the operation sequence vector, and $v_4(s)$ represents the machine assigned for operation $v_3(s)$. The machine assignment shown in figure 1 and the operation sequence shown in figure 2 can be transformed into the format of the permutation representation shown in figure 5. For example, operation 5 holds the first priority and machine 2 is selected for it in figure 5. Based on the new representation scheme, we propose two kinds of crossover operators.

Locus: Priority (s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operation Sequence: $v_3(s)$	5	13	9	1	14	2	15	10	6	3	11	4	7	16	12	8
Machine Assignment: $v_4(s)$	2	2	3	4	4	3	4	1	4	3	2	1	1	3	4	4

Figure 5. Illustration of the operation sequence vector

1) *Exchange crossover*. A good schedule could be expected by exchanging machine assignment and operation sequence schemes between a pair of parents. This kind of crossover is accomplished by selecting two parents and exchanging the machine assignment vectors of the two parents to generate offsprings.

2) *Enhanced order crossover*. During the past decades, several crossover operators have been proposed for permutation representation, such as partial-mapped crossover, order crossover, cycle crossover, and so on. Yet, these methods are proposed for permutation chromosomes with only one vector. An enhanced order crossover expanded from the original order crossover [11][12] is devised for the two-vector permutation representation. The enhanced order crossover works as follows:

Step 1. Select a subsection of operation sequence from one parent at random.

Step 2. Produce a proto-child by copying the substring of operation sequence into the corresponding positions, and then copy the machines assigned for these operations from the same parent.

Step 3. Delete the operations that are already in the substring from the second parent. The resulted sequence of operations contains operations that the proto-child needs.

Step 4. Place the operations into the unfixed positions of the proto-child from left to right according to the order of the sequence in the second parent. Then, copy the machines selected for these operations from the second parent to produce an offspring.

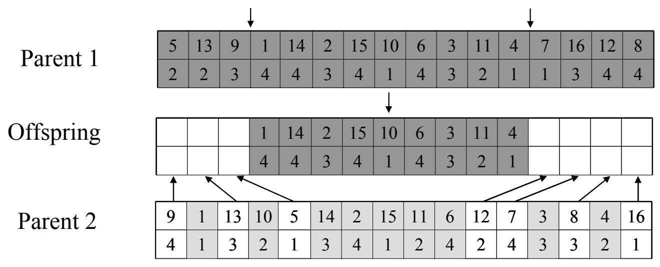

Figure 6. Illustration of the order crossover

The procedure is illustrated in figure 6. The offsprings generated by the enhanced order crossover are transformed back into the format of the two-vector Gen *et al*'s representation before they are released into the population. Machine assignment vector can be rearranged into the format of v_1 easily. Permutation operation sequence (v_3) is translated to Gen *et al.*'s format (v_2) by replacing each operation with its job no..

4.2 Mutation Operators

Mutation operators create new solutions that are slightly different from their parents. In this study, two kinds of mutation operations are implemented: allele-based mutation and immigration mutation. For machine assignment vectors, allele-based mutation randomly decides whether an allele r ($1 \le r \le N$) should be selected for mutation in a certain probability. Then, a new available machine will be assigned for the operation indicated by the selected allele. For operation sequence vectors, allele-based mutation randomly decides whether to mutate an allele s ($2 \le s \le N$) in a certain probability. If an allele s is to be mutated, then swap $v_2(s\text{-}1)$ and $v_2(s)$.

In contrary to the canonical gene-by-gene mutation with very small probability at each generation, immigration mutation randomly generates one or more new members of the population from the same distribution as the initial population. This process prevents premature convergence of the population, and leads to a simple statement of convergence.

4.3 Fitness Function

Different from most other multiobjective optimization problems, the three considered objectives do not conflict with one another in that a small makespan (c_M) requires a small maximal workload (w_M) and a small maximal workload implies a small total workload (w_T). During evaluation, the fitness of a solution is calculated by synthesizing the three objectives into a weighted sum. We have to normalize the objective values on the three criteria before they are summed since they are of different scales. Let $c_\mathrm{M}(l)$ be the makespan of the l-th chromosome. The scaled makespan ($c_\mathrm{M}'(l)$) of a solution l is as follows:

$$c_\mathrm{M}'(l) = \begin{cases} \dfrac{c_\mathrm{M}(l) - c_\mathrm{M}^{\min}}{c_\mathrm{M}^{\max} - c_\mathrm{M}^{\min}}, \text{ if } c_\mathrm{M}^{\max} \neq c_\mathrm{M}^{\min}, \\ 0.5, \qquad \text{otherwise,} \end{cases} \quad \text{for all } l \tag{10}$$

where:

$$c_\mathrm{M}^{\min} = \min_{1 \le l \le P} \{c_\mathrm{M}(l)\};$$

$$c_\mathrm{M}^{\max} = \max_{1 \le l \le P} \{c_\mathrm{M}(l)\}$$

where P is the total number of solution candidates to be evaluated in a generation. With the same method, we can scale maximal workload $w_\mathrm{M}(l)$ and total workload $w_\mathrm{T}(l)$ for each solution l. After scaling, the three objectives all take values from the range of [0, 1].

In order to guide the genetic and local search to the most promising area, makespan is given a very large weight since the other two objectives heavily depend on it. Additionally, it is typically the most important criterion in practical production environments. For the fJSP problem, a number of solutions with different maximal workloads or total workloads may have the same makespan. From this point of view, we firstly find the solutions with the minimum makespan, then minimize the maximal workload and the total workload in the presence of the minimum makespan. The fitness of a solution l then is:

$$f(l) = \alpha_1 c_\mathrm{M}'(l) + \alpha_2 w_\mathrm{M}'(l) + \alpha_3 w_\mathrm{T}'(l) \tag{11}$$

where $\alpha_1 > \alpha_2 > \alpha_3 > 0$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

5. BOTTLENECK SHIFTING

5.1 Defining Neighborhood

A central problem of any local search procedure for combinatorial optimization problems is how to define the effective neighborhood around an initial solution. In this study, the effective neighborhood is based on the concept of critical path. To define neighborhood using critical path is not new for job shop scheduling problem and has been employed by many researchers [13] [14] [15].

The feasible schedules of an fJSP problem can be represented with disjunctive graph $G = (N, A, E)$, with node set N, ordinary (conjunctive) arc set A, and disjunctive arc set E. The nodes of G correspond to operations, the real arcs (A) to precedence relations, and the dashed arc (E) to pairs of operations to be performed on the same machine. For example, the following schedule of the 4×4 problem can be illustrated in the disjunctive graph shown in figure 7:

The Schedule={($o_{1,1}$, M_4: 0-16), ($o_{1,2}$, M_3: 21-33), ($o_{1,3}$, M_3:33-51), ($o_{1,4}$, M_1: 51-69), ($o_{2,1}$, M_2:0-16), ($o_{2,2}$, M_4:94-112), ($o_{2,3}$, M_1: 112-136), ($o_{2,4}$, M_4: 136-148), ($o_{3,1}$, M_3: 0-21), ($o_{3,2}$, M_1: 21-45), ($o_{3,3}$, M_2: 45-68), ($o_{3,4}$, M_1: 69-105), ($o_{4,1}$, M_2: 16-32), ($o_{4,2}$, M_4: 32-62), ($o_{4,3}$, M_4: 62-94), ($o_{4,4}$, M_3: 94-118)}.

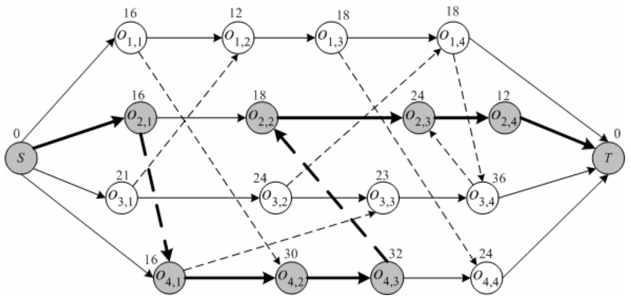

Figure 7. Illustration of disjunctive graph

In figure 7, S and T are dummy starting and terminating nodes respectively. The number above each node represents the processing time of that operation. The critical path is the longest path in a graph. For an fJSP schedule, its makespan is equal to the length of the critical path in the corresponding disjunctive graph. The critical path is highlighted with broad-brush arcs in figure 7. Any operation on the critical path is called a critical operation. A critical operation cannot be delayed without increasing the makespan of the schedule.

The job predecessor $PJ(r)$ of an operation r is the operation preceding r in the operation sequence of the job that r belongs to. The machine predecessor $PM(r)$ of an operation r is the operation preceding r in the operation sequence on the machine that r is processed on. If an operation r is critical, then at least one of $PJ(r)$ and $PM(r)$ must be critical, if they exist. In this study, if a job predecessor and a machine predecessor of a critical operation are both critical, then choose the predecessor (from these two alternatives) that appears first in the operation sequence.

A new schedule that is slightly different from the initial solution can be generated by changing the processing sequence of two adjacent operations performed on the same machine, *i.e.*, reversing the direction of the disjunctive arc that links the two operations. The neighborhood created in this way is named as type I here. Neighbor solutions can also be generated by assigning a different machine for one operation. This kind of neighborhood is named as type II.

The makespan of a schedule is defined by the length of its critical path, in other words, the makespan is no shorter than any possible path in the disjunctive graph. Hence, for a neighbor solution of type I, only when these two adjacent operations are on the critical path, the new solution is possible to be superior to the old one. Likewise, for a neighbor solution of type II, it cannot outperform the initial solution if the operation is not a critical one.

For the fJSP problem, we can only swap the operation sequence between a pair of operations that belong to different jobs. It is possible to decompose the critical path into a number of blocks, each of which is a maximal sequence of adjacent critical operations that require the same machine. As a result, the possible swaps are further confined as follows:

- In each block, we only swap the last two and first two operations;
- For the first (last) block, we only swap the last (first) two operations in the block. In case where the first (last) block contains only two operations, these operations are swapped.
- If a block contains only one operation, then no swap is made.

Due to the strict restrictions above, possible swaps occur only on a few pairs of adjacent operations that belong to different jobs on the critical path. Neighbor solutions of type I are actually generated by implementing these possible swaps. Figure 8 shows the critical path, critical blocks and the possible swaps in a schedule. The total number of the neighbors of type I (N^{I}) is less than the total number of critical operations (N^{C}) since some critical operations can not involve the possible swaps.

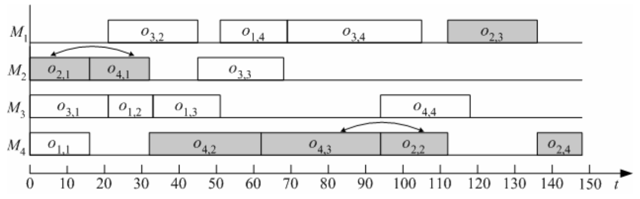

Figure 8. Neighborhood of type I

A neighbor solution of type II can be created by assigning a different machine $j \in A_{ik}$ for a critical operation o_{ik}. Let n_l^{II} be the number of machines on which the l-th critical operation can be assigned. $n_l^{\mathrm{II}} - 1$ neighbors can be generated by assigning the operation on any of the other $n_l^{\mathrm{II}} - 1$ available machines. Hence, the total number of neighbors of type II (N^{II}):

$$N^{\mathrm{II}} = \sum_{l=1}^{N^{\mathrm{C}}} n_l^{\mathrm{II}} - 1 \qquad (12)$$

Since N^{I} is less than N^{C}, N^{II} generally represents a much larger number than N^{I}.

5.2 Local Search Transition Mechanism

During the local search, the original schedule will transit to a better neighbor solution, if it exists. This gives rise to a new problem: what is an improved solution. For the fJSP problem, there may be more than one critical path in a schedule, in which the makespan is determined by the length of the critical path. A solution with a smaller number of critical paths may provide more potential to find solutions with less makespan nearby because the makespan cannot be decreased without breaking all the current critical paths. An important problem of any local search method is how to guide to the most promising areas from an initial solution. In this study, a solution is taken to be an improved solution if it satisfies either of the two alternative requirements:

1) An improved solution has a larger fitness value than the initial solution; or
2) The improved solution has the same fitness value as the initial solution, yet it has less critical paths.

5.3 Adjusting Neighborhood Structure

Let $N(i)$ denote the set of neighborhood of solution i. The enlarged two-pace neighborhood can be defined as the union of the neighborhood of each neighbor of the initial solution. Let $N^2(i)$ be the two-pace neighborhood of solution i, then,

$$N^2(i) = \cup_{j \in N(i)} N(j) \tag{13}$$

A larger neighborhood space size generally indicates a higher quality of the local optima because in each step of the local search, the best solution among a larger number of neighbor solutions is selected as the initial solution for the next local search iteration. On the other hand, a larger neighborhood space size would bring a greater computational load because more neighbor solutions have to be evaluated and compared. That is, each step of the local search will take longer time. Hence, the number of the local search iterations is decreased when the time spent on local search is limited. As a result, the deep search ability is not fully utilized.

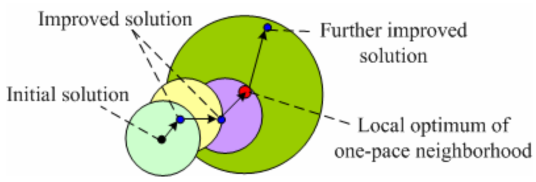
Improved solution
Further improved solution
Initial solution
Local optimum of one-pace neighborhood

Figure 9. Broad search over the enlarged two-pace neighborhood

In order to enhance the search ability of the local search without incurring too much computational load, during the search process over type II neighborhood, the local search procedure will implement over the enlarged two-pace neighborhood only when it reaches the local optimum of the one-pace neighborhood. The broad search process is illustrated in figure 9.

6. COMPUTATIONAL RESULTS

In order to test the effectiveness and performance of the proposed hybrid genetic algorithm, three representative instances (represented by problem $n \times m$) were selected for simulation. The works by Kacem *et al.*[5][16], Xia and Wu [7], and Zhang and Gen [8] are among the most recent progresses made in the area of fJSP. Unfortunately, the simulation results of [8] are not included in their work. Hence, the results obtained by our method are compared with the results from [5][16] and [7]. All the simulation experiments were performed with Delphi on Pentium 4 processor (2.6-GHz clock). The adopted parameters of the hGA are listed in table 1.

Table 1. Parameters of the hGA

Parameters	Value	Parameters	Value
population size	1500	immigration mutation prob.	0.15
maximal generation	300	α_1	0.85
exchange crossover prob.	0.2	α_2	0.10
order crossover prob.	0.3	α_3	0.05
allele-based mutation prob.	0.10	L	5

6.1 Problem 8×8

This is an instance of partial flexibility. In the flexible job shop, there are 8 jobs with 27 operations to be performed on 8 machines. For more details about this problem, refer to [7]. Experimental simulations were run for 20 times. The 20 runs all converge to optimal solutions with the same objective values on the three considered criteria. One of the optimal solutions is shown in figure 10. This test instance seems to be oversimplified. It takes averagely 16.4 generations for the hGA to converge to the optimal solutions. The computation time averages at 5 minutes. By adopting different weights of the three considered objectives, we got another optimal solution shown in figure 11.

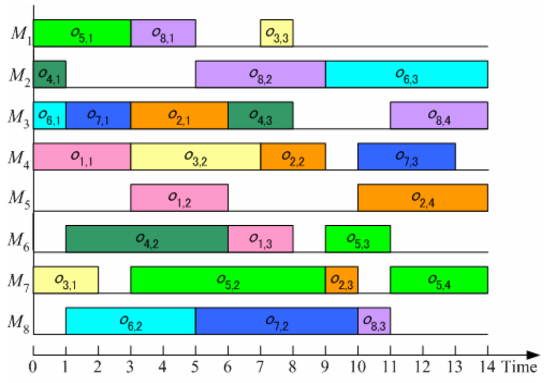

Figure 10. Optimal solution 1 of problem 8×8 (c_M=14, w_M=12, w_T=77)

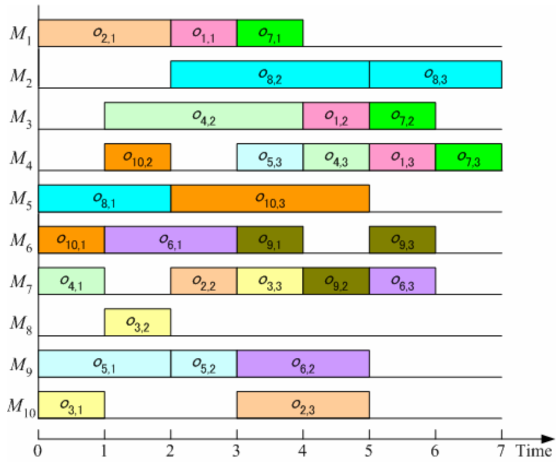

Figure 11. Optimal solution 2 of problem 8×8 (c_M=15, w_M=12, w_T=75)

6.2 Problem 10×10

For this test instance, there are 10 jobs with 30 operations to be performed on 10 machines. For more details about this problem, refer to [7]. Experimental simulations were run for 20 times for this problem. The 20 runs all converge to optimal solutions with the same objective values. One of the optimal solutions is shown in figure 12. Averagely, the hGA takes 26.50 evolution generations and about 17 minutes to find the optimal solutions.

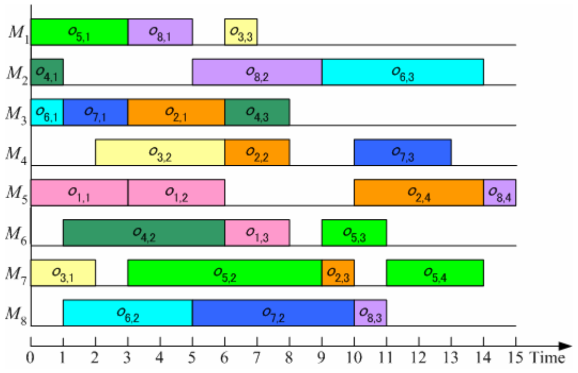

Figure 12. Optimal solution of problem 10×10 (c_M=7, w_M=5, w_T=43)

6.3 Problem 15×10

A larger-sized problem is chosen to test the performance of our hybrid genetic algorithm. This problem contains 15 jobs with 56 operations that have to be processed on 10 machines with total flexibility (for more details about this problem, refer to [7]). This problem has been viewed as a challenge for other algorithms.

Experimental simulations were run for 20 times for this problem. The 20 runs all converge to optimal solutions with the same fitness values. Not only different solutions with the same optimal fitness value but also distinct solutions with the same optimal objective values in the three considered criteria are found in one run of the experiment. To provide more than one good solution for decision-makers is a main advantage of genetic algorithms. Figure 13 shows one of the optimal solutions.

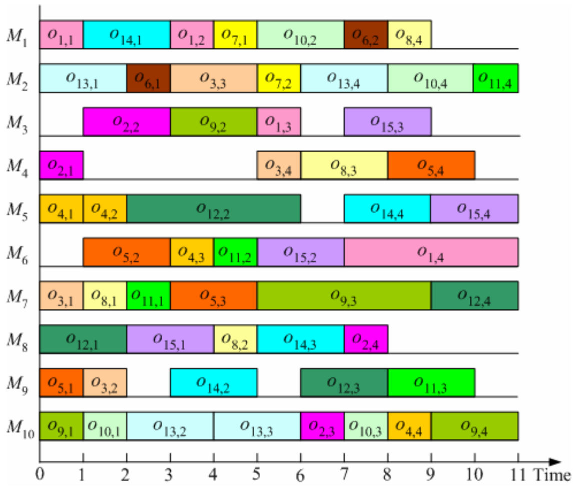

Figure 13. Optimal solution of problem 15×10 (c_M=11, w_M=11, w_T=91)

It takes about 97.75 generations to converge to the optimal solutions. Table 5 gives statistical information about convergence generations. It seems quite easy for our algorithm to find the optimal solutions even for such a large-sized fJSP problem. Yet, the time spent on finding the optimal solutions is rather long and averages at 135.47 minutes because the local search consumes large amount of computation time during the evolutionary process. In comparison with the scheduling horizon, the computation time is acceptable in the real world.

Table 2. Convergence generations on problem 15×10

Mean	95% confidence interval for mean		Std. deviation
	Lower bound	Upper bound	
97.75	89.40	106.10	17.85

Table 3 gives the performance of the proposed method compared with other algorithms. "Approach by Localization" and "AL+CGA" are two algorithms by Kacem *et al.* [5] [16]. "PSO+SA" is the algorithm by Xia and Wu [7].

Table 3. Performance of the hGA for the three fJSP problems

Problem		Classical GA	AL+CGA		PSO+SA		Proposed hGA
8×8	c_M	16	15	16	15	16	15
	w_M				12	13	12
	w_T	77	79	75	75	73	75
10×10	c_M	7	7		7		7
	w_M	7	5		6		5
	w_T	53	45		44		43
15×10	c_M	23	24		12		11
	w_M	11	11		11		11
	w_T	95	91		91		91

7. CONCLUSIONS

We have developed a new approach hybridizing genetic algorithm with bottleneck shifting to fully exploit the "global search ability" of genetic algorithm and "the local search ability" of bottleneck shifting for solving multiobjective flexible job shop scheduling problem. An innovative two-vector Gen *et al.*'s presentation scheme is proposed and an effective decoding method is used to interpret each chromosome into an active schedule. The initialization and mutation operations operate chromosomes of the two-vector Gen *et al.*'s presentation. However, in order to enhance the heritability of crossover operation, chromosomes of the two-vector Gen *et al.*'s presentation are transformed into the format of the two-vector permutation presentation, and then an enhanced order crossover is proposed to implement recombination operation on the chromosomes of the two-vector permutation presentation.

Two kinds of neighborhood are defined based on the concept of critical path for the fJSP problem. The two kinds of neighborhood are quite effective in that they only contain solutions that are likely to improve the initial solution. In the local search, the number of critical paths serves as one kind of intermediate objective besides the three original criteria in order to guide the local search to the most promising areas. The neighborhood structure can be dynamically adjusted during the local search process so that the quality of the local optima can be improved without incurring too much computational load.

Several well-known benchmark problems of different scales are solved by the proposed algorithm. The simulation results obtained in this study are compared with the results obtained by other authors' algorithms. The results demonstrate the performance of the proposed algorithm.

ACKNOWLEDGMENTS

This work is partly supported by Waseda University Grant for Special Research Projects 2004 and the Ministry of Education, Science and Culture, the Japanese Government: Grant-in-Aid for Scientific Research (No.17510138). This paper is also supported in part by National Natural Science Foundation of China (NSFC) under Grant 70433003.Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

REFERENCES

[1] Bruker, P. and Schlie, R., Job-shop scheduling with multi-purpose machines. *Computing*, 45, 369-375, 1990.

[2] Chambers, J. B., *Classical and Flexible Job Shop Scheduling by Tabu Search*. PhD thesis, University of Texas at Austin, Austin, U.S.A., 1996.

[3] Mastrolilli, M. and Gambardella, L. M., Effective neighborhood functions for the flexible job shop problem. *J. Sched.*, 3, 3-20, 2000.

[4] Yang, J.-B., GA-based discrete dynamic programming approach for scheduling in FMS environments. *IEEE Trans. Systems, Man, and Cybernetics—Part B*, 31(5), 824-835, 2001.

[5] Kacem, I., hammadi, S. and Borne, P., Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Trans. Systems, Man, and Cybernetics—Part C*, 32(1), 1-13, 2002.

[6] Wu, Z. and Weng, M. X., Multiagent scheduling method with earliness and tardiness objectives in flexible job shops. *IEEE Trans. System, Man, and Cybernetics—Part B*, 35(2), 293-301, 2005.

[7] Xia, W. and Wu, Z., An effective hybrid optimization approach for muti-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 48, 409-425, 2005.

[8] Zhang, H. and Gen, M., Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal of Complexity International*, 11, 223-232, 2005.

[9] Cheng, R., Gen, M. and Tsujimura, Y., A tutorial survey of job-shop scheduling problems using genetic algorithms-I. Representation. *Computers & Industrial Engineering*, 30(4), 983-997, 1996.

[10] Cheng, R., Gen, M. and Tsujimura, Y., A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2), 343-364, 1999.

[11] Gen, M. and Cheng, R., *Genetic Algorithms & Engineering Design*, New York: Wiley, 1997.

[12] Gen, M. and Cheng, R., *Genetic Algorithms & Engineering Optimization* , New York: Wiley, 2000.

[13] Adams, J., Balas, E. and Zawack, D., The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3), 391-401, 1988.

[14] Balas, E. and Vazacopoulos, A., Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2), 262-275, 1998.

[15] Goncalves, J. F., Mendes, J. J. M., Resende, M. G.. C., A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167, 77-95, 2005

[16] Kacem, I., Hammadi, S. and Borne, P., Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60, 245-276, 2002.