Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 2892–2907
computers & operations research
www.elsevier.com/locate/cor

A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems

Jie Gao[a, b], Linyan Sun[a], Mitsuo Gen[b,*]

[a]*School of Management, Xi'an Jiaotong University, Xi'an 710049, China*
[b]*Graduate School of Information, Production & Systems, Waseda University, Kitakyushu 808-0135, Japan*

Available online 2 February 2007

Abstract

This paper addresses the flexible job shop scheduling problem (fJSP) with three objectives: min makespan, min maximal machine workload and min total workload. We developed a hybrid genetic algorithm (GA) for the problem. The GA uses two vectors to represent solutions. Advanced crossover and mutation operators are used to adapt to the special chromosome structure and the characteristics of the problem. In order to strengthen the search ability, individuals of GA are first improved by a variable neighborhood descent (VND), which involves two local search procedures: local search of moving one operation and local search of moving two operations. Moving an operation is to delete the operation, find an assignable time interval for it, and allocate it in the assignable interval. We developed an efficient method to find assignable time intervals for the deleted operations based on the concept of earliest and latest event time. The local optima of moving one operation are further improved by moving two operations simultaneously. An extensive computational study on 181 benchmark problems shows the performance of our approach.

Keywords: Flexible job shop scheduling; Genetic algorithms; Variable neighborhood descent; Critical path

1. Introduction

In the job shop scheduling problem (JSP), there are n jobs that must be processed on a group of m machines. Each job i consists of a sequence of m operations $(o_{i1}, o_{i2}, \ldots, o_{im})$, where o_{ik} (the kth operation of job i) must be processed without interruption on a predefined machine m_{ik} for p_{ik} time units. The operations $o_{i1}, o_{i2}, \ldots, o_{im}$ must be processed one after another in the given order and each machine can process at most one operation at a time.

In this paper we study a generalization of JSP called the flexible job shop scheduling problem (fJSP), which provides a closer approximation to a wide range of scheduling problems encountered in real manufacturing systems.

In a flexible job shop, each job i consists of a sequence of n_i operations $(o_{i1}, o_{i2}, \ldots, o_{in_i})$. The fJSP extends JSP by allowing an operation o_{ik} to be executed by one machine out of a set A_{ik} of given machines. The processing time of operation o_{ik} on machine j is $p_{ikj} > 0$. The fJSP problem is to choose for each operation o_{ik} a machine $M(o_{ik}) \in A_{ik}$ and a starting time s_{ik} at which the operation must be performed.

* Corresponding author. Tel./fax: +81 93 692 5273.
E-mail addresses: calebgao@yahoo.com (J. Gao), lysun@mail.xjtu.edu.cn (L. Sun), gen@waseda.jp (M. Gen).

0305-0548/$ - see front matter © 2007 Published by Elsevier Ltd.
doi:10.1016/j.cor.2007.01.001

fJSP is therefore made more complex than JSP by the need to determine a routing policy (i.e., which machine to assign for each operation) other than the traditional scheduling decisions (i.e., to determine the starting time of each operation). In this study, we consider minimizing the following three criteria:

(1) Minimize the makespan (C_M) of the jobs.
(2) Minimize the maximal machine workload (W_M), i.e., the maximum working time spent at any machine. This objective is to prevent a solution from assigning too much work on a single machine and to keep the balance of work distribution over the machines.
(3) Minimize the total workload (W_T), which represents the total working time assigned over all machines. This objective is of interest if machine efficiencies differ.

In contrary to those multiobjective optimization problems in which there are no priorities among the objectives, makespan is given the first importance, maximal machine workload is given the secondary importance, and total workload is given the least importance in this study. When two solutions with different makespans are compared, we always prefer the solution with a smaller makespan regardless of its maximal machine workload and total workload.

In this paper, a hybrid genetic and variable neighborhood descent (VND) algorithm is used to treat the fJSP problem. The genetic algorithm (GA) uses two vectors to express the machine assignment and operation sequence information for fJSP solution candidates. The operation sequence vector uses two representation methods: Gen et al.'s [1] representation and permutation representation. Under the framework of the GA, a VND is applied to each newly generated offspring to improve its quality before injecting it into the population. The VND improves a solution first by moving one operation. When the local optimum of moving one critical operation is found, it is further improved by moving two operations simultaneously.

In Section 2, an overview of previous research is provided. Section 3 presents the representation method, decoding procedure and genetic operators of the GA. The VND and the framework of the hybrid GA are presented in Section 4. In Section 5, we provide an extensive computational study on 181 benchmark problems where our approach is compared with previous approaches. Some final concluding remarks are given in Section 6.

2. Literature review

Bruker and Schlie [2] gave a polynomial algorithm for solving flexible job shop scheduling problems with two jobs. Based on the observation that fJSP turns into the classical job shop scheduling problem when a routing is chosen, early literature [3–6] proposed hierarchical strategies for the complex scheduling problem, where job routing and sequencing are studied separately.

Jurisch [7] considered the routing and scheduling decisions simultaneously. Hurink et al. [8] and Chambers [9] developed tabu search algorithms to solve the problem. Dauzère-Pérès and Paulli [10] extended the classical disjunctive graph model for job shop scheduling to take into account the fact that operations have to be assigned to machines in the fJSP problem. Based on the extended disjunctive graph, a new neighborhood structure is defined and a tabu search procedure is provided to solve the problem. Mastrolilli and Gambardella [11] proposed two neighborhood functions for this problem. They proposed a tabu search procedure and provided an extensive computational study on 178 fJSP problems and 43 JSP problems. Their approach found 120 new better upper bounds and 77 optimal solutions over the 178 fJSP benchmark problems and it was outperformed in only one problem instance.

Yang [12] presented a genetic algorithm (GA)-based discrete dynamic programming approach. Kacem and Borne [13] proposed a localization approach to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for the fJSP problem. Zhang and Gen [14] proposed a multistage operation-based GA to deal with the problem from a point view of dynamic programming. Xia and Wu [15] treated this problem with a hybrid of particle swarm optimization (PSO) and simulated annealing (SA) as a local search algorithm. Wu and Weng [16] considered the problem with job earliness and tardiness objectives, and proposed a multiagent scheduling method.

Vaessens [17] defined a neighbor for the fJSP problem by deleting an operation r from the machine ordering, identifying a set of feasible insertions containing the optimal one and inserting r in the best-possible way. Vaessens' algorithm spends a considerable amount of computing time defining such a set of feasible insertions. Brucker and Neyer [18] also proposed the best insertion of an operation in their neighborhood function, but the algorithm they suggested is too time consuming. In order to reduce the computational effort, they proposed a faster algorithm that guarantees only

the feasibility of an insertion. Mastrolilli and Gambardella [11] reduced the set of possible neighbors to a subset which always contains neighbors that are possible to be superior to the incumbent one. Yet, the quality of a feasible insertion has to be judged by calculating the makespan of the solution after insertion. They used an approximation algorithm to evaluate the makespan of the neighbor solutions.

3. A GA for fJSP

3.1. Two-vector representation

The fJSP problem is a combination of machine assignment and operation scheduling decisions, so a solution can be expressed by the assignment of operations on machines and the processing sequence of operations on the machines. Using GA terminology, a chromosome is therefore composed of two parts:

(1) machine assignment vector (hereafter called v_1),
(2) operation sequence vector (hereafter called v_2).

In each machine assignment vector v_1, $v_1(r)$ represents the machine selected for the operation indicated at position r. Fig. 1 illustrates a machine assignment vector. For example, in Fig. 1, position 2 indicates $o_{1,2}$, and $v_1(2)$ represents the machine assigned for $o_{1,2}$.

A permutation representation is perhaps the most natural way to express operation sequences. Unfortunately because of the existence of precedence constraints, not all the permutations of the operations define feasible sequences. For job shop scheduling problems, Gen et al. [1] proposed an alternative: they name all operations for a job with the same symbol and then interpret them according to the order of occurrences in the sequence of a given chromosome. Gen et al.'s method can also be used to represent operation sequences for fJSP problems. 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 depicted in Fig. 2 can be translated into a list of ordered operations below:

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

The main advantages of Gen et al.'s representation are that each possible chromosome always represents a feasible operation sequence, and that the coding space is smaller than that of permutation representation.

With the two-vector representation, a number of feasible chromosomes are generated at random to form the initial population. Afterwards, a decoding procedure translates the chromosomes into fJSP schedules, and genetic operations evolve the population towards better solutions.

3.2. Priority-based decoding and reordering

In principle, a chromosome of the fJSP problem can be decoded into an infinite number of schedules because superfluous idle time can be inserted between operations. We shift the operations to the left as compact as possible. A shift is called a local left-shift if some operation is started earlier without altering the operation sequence. A shift is called a global left-shift if some operation is started earlier without delaying any other operations even though the shift has changed the operation sequence. A schedule is semiactive if no local left-shift exists, and is active if no global left-shift exists.

Position: 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

Fig. 1. Illustration of the machine assignment vector.

Position: 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

Fig. 2. Illustration of Gen et al.'s operation sequence representation.

When considering a regular performance measure (e.g. makespan), the optimal schedule is within the set of active schedules [19]. In this paper, we use priority-based decoding to translate chromosomes into active schedules.

Because of precedence constraints among operations of the same job, idle time may exist between operations on a machine. Let s_{ik} be the starting time of o_{ik} and c_{ik} its completion time. An operation o_{ik} can only be started after its immediate job predecessor $o_{i(k-1)}$ is completed. Given a time interval $[t_j^S, t_j^E]$ beginning from t_j^S and ending at t_j^E on machine j to allocate o_{ik}, we have

$$S_{ik} = \begin{cases} \max\{t_j^S, c_{i(k-1)}\} & \text{if } k \geqslant 2, \\ t_j^S & \text{if } k = 1. \end{cases} \tag{1}$$

Time interval $[t_j^S, t_j^E]$ is available for o_{ik} if there is enough time span from the starting of o_{ik} until the ending of the interval to complete it, i.e.,

$$\begin{cases} \max\{t_j^S, c_{i(k-1)}\} + p_{ikj} \leqslant t_j^E & \text{if } k \geqslant 2, \\ t_j^S + p_{ikj} \leqslant t_j^E & \text{if } k = 1. \end{cases} \tag{2}$$

The proposed priority-based decoding allocates each operation on its assigned machine one by one in the order represented by the operation sequence vector. When operation o_{ik} is scheduled on machine j, the idle time intervals between operations that have already been scheduled on the machine are examined from left to right to find the earliest available one. If such an available interval exists, it is allocated there; otherwise, it is allocated at the end of machine j.

The priority-based decoding method allows an operation to search the earliest available time interval on the machine. Hence, operation r may be processed earlier than another operation v, which appears before r in the operation sequence vector. In order to facilitate offspring 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 operations' starting time in the decoded schedule before the chromosome involves crossover and mutation operations.

3.3. Crossover operators

In Gen et al.'s [1] representation, each allele 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 create offspring operation sequences that combine the characteristics of their parental solutions. In order to strengthen the heritability, the operation sequence vector of the original Gen et al.'s representation is transformed into the style of permutation representation.

Although it is natural to represent an operation by job number and operation sequence, an operation can also be represented through a fixed ID as shown in Fig. 3. In the permutation representation, operations are listed in the order of their priority. The operation sequence shown in Fig. 2 can be transformed into the style of permutation representation shown in Fig. 4, where $v_3(s)$ denotes the ID of operation with the sth priority.

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 [20]. Here we apply order crossover for the

Operation	$o_{1,1}$	$o_{1,2}$	$o_{1,3}$	$o_{1,4}$	$o_{2,1}$	$o_{2,2}$	$o_{2,3}$	$o_{2,4}$	$o_{3,1}$	$o_{3,2}$	$o_{3,3}$	$o_{3,4}$	$o_{4,1}$	$o_{4,2}$	$o_{4,3}$	$o_{4,4}$
Fixed ID (r)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Fig. 3. Illustration of the fixed ID for each operation.

Position: Priority (s)	1	2	3	4	5	1	6	7	8	9	10	11	12	13	14	15	6
Operation Sequence: $v_3(s)$	5	13	9	1	14	2	15	10	6	3	11	4	7	16	12	8	
Operation Indicated	$o_{2,1}$	$o_{4,1}$	$o_{3,1}$	$o_{1,1}$	$o_{4,2}$	$o_{1,2}$	$o_{4,3}$	$o_{3,2}$	$o_{2,2}$	$o_{1,3}$	$o_{3,3}$	$o_{1,4}$	$o_{2,3}$	$o_{4,4}$	$o_{3,4}$	$o_{2,4}$	

Fig. 4. Illustration of the permutation operation sequence vector.

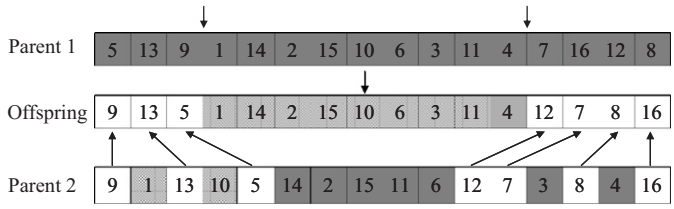

Fig. 5. Illustration of the order crossover on operation sequence.

operation sequence vectors. The 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.

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.

The procedure is illustrated in Fig. 5.

We use two crossover operators for the machine assignment vectors: extended order crossover and uniform crossover. The extended order crossover is related to crossover for operation sequence. It copies the machine assigned for an operation from the same parent where its operation sequence comes. Uniform crossover is accomplished by taking an allele from either parental machine assignment vector to form the corresponding allele of the child.

The offspring operation sequences generated by order crossover are transformed back into the format of Gen et al.'s representation by replacing each operation with its job number before they are released into the population. The order crossover does not ultimately generate any infeasible operation sequence vectors because Gen et al.'s representation repairs them into feasible ones. It is obvious that the two crossovers on machine assignment vector will not assign infeasible machines for operations.

3.4. Mutation operators

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 should be selected for mutation with a certain probability. Then, another 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 r. If allele r is to be mutated, then another allele is randomly selected to exchange with it. Immigration mutation randomly generates a number of new members of the population from the same distribution as the initial population.

3.5. Selection

For the proposed hybrid genetic algorithm (hGA), selection performs on enlarged sampling space, where both parents and offsprings have the same chance of competing for survival. In order to overcome the scaling problem of the direct fitness-based approach, ranking selection is introduced. The idea is straightforward: sort solutions in the population from the best to the worst according to their performance on the three considered criteria, and assign the selection probability of each chromosome based on the ranking. In each generation, a number of best solutions are reserved, while the rest are selected by roulette wheel selection [22] to fill the entire population. During the roulette wheel selection process, duplicate chromosomes are prohibited from entering the population.

4. Variable neighborhood descent

The evolution speed of simple GAs is relatively slow [21]. One promising approach for improving the convergence speed to improved solutions is the use of local search in GAs. Within a hybrid of GA and local search, a local optimizer

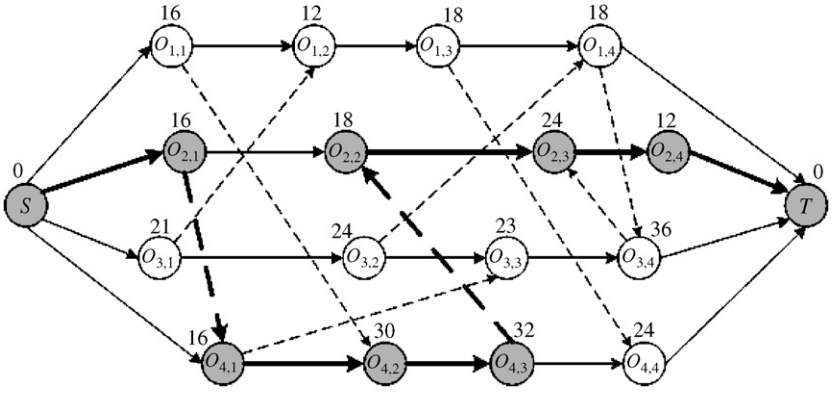

Fig. 6. Illustration of disjunctive graph.

is added to GAs and applied to every child before it is inserted into the population. GAs are used to perform global exploration among populations, while local search is used to perform exploitation around chromosomes. Because the local search can use much of the problem-specific knowledge to improve the solution quality, the hybrid approach often outperforms either method operating alone [22].

Variable neighborhood search (VNS) is based on a simple principle: systematic change of neighborhood within a possibly randomized local search [23,24]. The VND method is obtained if change of neighborhoods is performed in a deterministic way. In this study, VND works as a kind of local search method under the framework of the proposed GA.

4.1. Solution graph

The feasible schedules of fJSP problems can be represented with a directed graph $G = (N, A, E)$, with node set N, ordinary arc set A, and disjunctive arc set E. The nodes of G correspond to operations, the real arcs (A) to immediate precedence relations, and the dashed arc (E) to immediate implementation sequence of operations to be performed on the same machine. Fig. 6 gives an example.

In Fig. 6, S and T are dummy starting and terminating nodes, respectively. The number associated to each node represents the processing time of that operation. The job predecessor $PJ(r)$ of an operation r is the operation preceding r in the operation sequence of the job to which r belongs. 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 node r is a job predecessor of v, then node v is a job successor of r, $SJ(r)$. If node r is a machine predecessor of v, then node v is a machine successor of r, $SM(r)$.

4.2. Earliest event time and latest event time

Two key building blocks in directed solution graphs are the concepts of earliest starting time (s^{E}) and latest starting time (s^{L}) of an operation. The earliest starting time of operation r, represented by $s^{\mathrm{E}}(r)$, is the earliest time at which the implementation of operation r can begin. The latest starting time of operation r, represented by $s^{\mathrm{L}}(r)$, is the latest time at which the implementation of operation r can begin without delaying the completion of the jobs (i.e., the makespan). Correspondingly, we can define the earliest completion time (c^{E}) and the latest completion time (c^{L}) for an operation as follows:

$$c^{\mathrm{E}}(r) = s^{\mathrm{E}}(r) + t(r), \quad c^{\mathrm{L}}(r) = s^{\mathrm{L}}(r) + t(r), \tag{3}$$

where $t(r)$ is the processing time of node r.

To find the earliest starting time of each node in the directed solution graph, we begin by noting that since node S represents the start of the schedule, $s^{\mathrm{E}}(S) = 0$. The earliest starting time of each node is defined by the earliest

completion times of its job predecessor and its machine predecessor (if it has one):

$$s^{\mathrm{E}}(r) = \max\{c^{\mathrm{E}}[PJ(r)], c^{\mathrm{E}}[PM(r)]\}. \tag{4}$$

In case a node has no job predecessor or machine predecessor, the corresponding term is replaced with "0".

To compute the latest completion time, we begin with the terminating node and work back. The latest completion time of the terminating node is set equal to the makespan of the schedule, i.e., $c^{\mathrm{L}}(T) = C_{\mathrm{M}}$. The latest completion time of each node is determined by the latest starting times of its job successor and its machine successor if it has

$$c^{\mathrm{L}}(r) = \min\{s^{\mathrm{L}}[SJ(r)], s^{\mathrm{L}}[SM(r)]\}. \tag{5}$$

In case, a node has no job successor or machine successor, the corresponding term is replaced with "∞".

The concept of total float of an operation is the amount by which the starting time of the operation can be delayed beyond its earliest possible starting time without delaying the makespan. The total float of an operation r, represented by $F(r)$, can easily be expressed:

$$F(r) = s^{\mathrm{L}}(r) - s^{\mathrm{E}}(r). \tag{6}$$

If an operation has a total float of zero, any delay in the start of the operation will delay the completion of the schedule. Such an operation is critical to the completion of all the jobs on time. Any operation with a total float of zero is a critical one. A path from node S to T that consists entirely of critical operations connected with each other in time is called a critical path. The critical path is highlighted in Fig. 6. Here, the length of a path $(S, r_1, r_2, \ldots, r_q, T)$ is defined as the sum of the processing times of the operations r_1 up to and including r_q . The critical path is the longest path in a directed graph. The makespan of a schedule is equal to the length of the critical path [25]. There may be more than one critical path in a directed solution graph.

4.3. Moving an operation

The move of an operation r is to delete the operation from its current position and insert it at another feasible position of the solution graph. Let G be a solution graph. Operation r is deleted from G by removing the disjunctive arc from r and the disjunctive arc to r, connecting $PM(r)$ to $SM(r)$ with a dashed arc, and setting the weight of node r equal to "0". Let G^- be the graph obtained from G when an operation is removed (for the remainder, the superscript '$-$' refers to the situation after the remove of an operation, and r denotes the operation to be moved). Let $C_{\mathrm{M}}(G)$ be the makespan of solution graph G. Since G^- is obtained by deleting one operation from G, it is obvious that the makespan of G^- is no larger than $C_{\mathrm{M}}(G)$.

Let G' be the solution graph obtained from G^- after operation r is inserted again. Since operation r is deleted from G, it is necessary to find another feasible position in G^- to reallocate r so that the makespan of G' is no larger than $C_{\mathrm{M}}(G)$. In order to do this, we take $C_{\mathrm{M}}(G)$ as the "required" makespan of G^- when calculating the latest starting and completion time for each operation remained in G^-.

For a node v, it can be started as late as at $s^{\mathrm{L}-}(v)$ on machine $M(v)$ without delaying the required makespan while its immediate machine predecessor can be finished as early as at $c^{\mathrm{E}-}[PM(v)]$ in G^-. This forms a maximum idle time interval $\{c^{\mathrm{E}-}[PM(v)], s^{\mathrm{L}-}(v)\}$ before operation v on machine $M(v)$. If operation v has no machine predecessor, $c^{\mathrm{E}-}[PM(v)]$ is replaced with "0".

Given a time interval $[t_j^{\mathrm{S}}, t_j^{\mathrm{E}}]$, beginning at t_j^{S} and ending at t_j^{E} on machine j, the deleted operation r can only be started after the earliest completion of its job predecessor, and has to be finished before the latest starting time of its job successor. The interval is said to be feasible for r if,

$$\max\{t_j^{\mathrm{S}}, c^{\mathrm{E}-}[PJ(r)]\} + p_{rj} \leqslant \min\{t_j^{\mathrm{E}}, s^{\mathrm{L}-}[SJ(r)]\}, \tag{7}$$

where p_{rj} is the processing time of operation r on machine j. A time interval $[t_j^{\mathrm{S}}, t_j^{\mathrm{E}}]$ on machine j is said to be assignable for operation r if

$$\max\{t_j^{\mathrm{S}}, c^{\mathrm{E}-}[PJ(r)]\} + p_{rj} < \min\{t_j^{\mathrm{E}}, s^{\mathrm{L}-}[SJ(r)]\}. \tag{8}$$

Hence, the maximum idle time interval before v in G^- is assignable for r if

$$\max\{c^{E-}[PM(v)], c^{E-}[PJ(r)]\} + p_{rM(v)} < \min\{s^{L-}(v), s^{L-}[SJ(r)]\}. \quad (9)$$

When the maximum idle time interval before operation v is assignable for the deleted operation r, r is inserted on the machine sequence of $M(v)$ before v. This is achieved by removing the dashed arc from $PM(v)$ to v, connecting $PM(v)$ to r and r to v with dashed arcs, and setting the weight of node r equal to the processing time of operation r on machine $M(v)$.

Theorem 1. *If $C_M(G') = C_M(G)$, then r is not a critical operation in G'.*

Proof. When r is inserted before v on machine $M(v)$, v is the immediate machine successor of r, $SM(r)$. According to the definition of latest starting/completion time, the insertion of node r does not change the latest starting times of $SM(r)$ and $SJ(r)$ as long as the makespan of G' is equal to the required one of G^-. It is notable that the required makespan of G^- is set to be $C_M(G)$ when calculating the latest starting and completion times for the operations in G^-. Since $c^{L'}(r) = \min\{s^{L'}[SM(r)], s^{L'}[SJ(r)]\}$, $c^{L'}(r) = \min\{s^{L-}(v), s^{L-}[SJ(r)]\}$ if $C_M(G') = C_M(G)$.

According to the definition of earliest starting time, the insertion of node r does not change the earliest starting times of $PM(r)$ and $PJ(r)$. After the insertion of node r before v on machine $M(v)$, the machine predecessor of v in G^- becomes the machine predecessor of r in G'. Hence, $s^{E'}(r) = \max\{c^{E-}[PM(v)], c^{E-}[PJ(r)]\}$. If $C_M(G') = C_M(G)$, the total float time of r in G' is

$$F'(r) = \max\{c^{E-}[PM(v)], c^{E-}[PJ(r)]\} + p_{rM(v)} - \min\{s^{L-}(v), s^{L^-}[SJ(r)]\}. \quad (10)$$

According to inequality (9), $F'(r) > 0$. Hence r is not a critical operation in G' if $C_M(G') = C_M(G)$. □

Corollary 1. *Let $L'(P)$ be the length of path P in G'. For any P in G', if $r \in P$, then $L'(P) < C_M(G)$.*

Proof. According to Theorem 1, when the makespan of G' is equal to $C_M(G)$, r is not a critical operation. The critical path is composed all of critical operations. Therefore, if $C_M(G') = C_M(G)$, P is not a critical path because r is included in P. The length of the non-critical path will be less than the makespan. Hence, if $r \in P$, $L'(P) < C_M(G)$. □

The solution graph G' is obtained from G by changing connections associated to operation r. The new paths in G' are created by adding or removing r from the paths in G. Considering Corollary 1, the length of the added paths is no larger than the makespan of G. Hence moving an operation does not add any new path whose length is equal to $C_M(G)$ in G', and the makespan of G' is no larger than $C_M(G)$.

4.4. VND procedure

The makespan of a solution is defined by the length of its critical paths, in other words, the makespan cannot be reduced while maintaining the current critical paths. The mission of local search is to identify and break the existent critical paths one by one in order to get a new schedule with smaller makespan.

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 job predecessor. In order to reduce the computation workload, we consider only a single arbitrarily selected critical path P of G.

Local search of moving one operation consists of: (a) delete an operation of the arbitrarily selected critical path P; (b) find an assignable interval for it; and then (c) allocate it in the found interval. For the deleted operation r, it has to be started after the earliest completion time of its job predecessor ($c^{E-}[PJ(r)]$), and it has to be completed before the latest starting time of its job successor ($s^{L-}[SJ(r)]$). Hence, it is no use to insert r before an operation whose earliest completion time is less than $c^{E-}[PJ(r)]$, or to insert r after an operation whose latest starting time is larger than $s^{L-}[SJ(r)]$. When we look for the assignable time interval for the deleted operation r, we do not investigate the intervals before operations whose earliest completion times are less than $c^{E-}[PJ(r)]$ or after operations whose latest starting times are larger than $s^{L-}[SJ(r)]$.

The basic scheme for local search of moving one critical operation is presented in Fig. 7.

(i) Identify a critical path P for a given incumbent solution S;
(ii) Set r to be the first operation of P;
(iii) Repeat
  (a) Delete r from solution graph G of S to get G^-;
  (b) Search for an assignable time interval assignable for r in G^-;
  (c) If no assignable interval is found, set r to be the next operation of P;
(iv) Until an assignable interval is found or r is the dummy terminating node;
(v) If an assignable interval is found, allocate r in the interval. Otherwise, S is a local optimum of moving one operation.

Fig. 7. Local search of moving one operation.

(I) Identify a critical path P for a given incumbent solution S;
(II) Set r to be the first critical operation of P;
(III) Repeat
  (a) Set v ($v \neq r$) to be an operation in solution S;
  (b) Repeat
    i. Delete v and r from solution graph G of S to get G^-;
    ii. Search for an assignable time interval for r in G^-;
    iii. If the assignable interval for r is found, insert r in the interval to get G^-'; Otherwise, go to (vi)
    iv. Search for an assignable time interval for v in G^-';
    v. If an assignable interval for v is found, insert v in the interval to get G'
    vi. If either the assignable interval for r or v is not found, set v ($v \neq r$) to be the next operation in G
  (c) Until the assignable intervals for both r and v are found or v is the last operation in G;
  (d) If the assignable interval for either r or v is not found, set r to be the next critical operation of P;
(IV) Until both the assignable intervals for r and v are found or r is the dummy terminating node.

Fig. 8. Local search of moving two operations.

When the local optimum of moving one operation is found, there is a critical path P that cannot be broken by moving one of its operations. That is, no assignable time interval can be found for any operation $r \in P$ in the solution graph G^-. Deleting an additional operation $v (v \neq r)$ from G^- brings more idle time, and may create time intervas assignable for r.

The idea of VND is straightforward: when the local optimum of moving one critical operation is found, the solution may be further improved by moving two operations simultaneously, at least one of which is critical. In the local search of moving two operations, operations r (r is a critical operation) and v are deleted simultaneously from G to get G^-, we then search for an assignable time interval for r in G^-. When the assignable time interval is found and r is inserted to get solution G^-', we search for an assignable time interval for v in G^-'. The basic scheme for local search of moving two operations is shown in Fig. 8.

The local optima of moving two operations may be further improved by local search of moving three or more operations. Unfortunately, computational complexity increases quickly with the number of operations moved, so computation time increases rapidly. In this study, the VND only involves local search of moving one operation and local search of moving two operations.

```
procedure: Hybrid Genetic Algorithm
input: fJSP data set, GA parameters
output: a near-optimal schedule
begin
    t←0;
    initialize P(t) with two-vector 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 their local optima to yield P'(t) and C'(t) by local search
            of moving one operation;
        improve P'(t) and C'(t) to yield P''(t) and C''(t) by local search of moving two
            operations;
        fitness eval(P''(t), C''(t)) 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←t+1;
    end
    output a near-optimal schedule;
end
```

Fig. 9. Framework of the proposed hybrid genetic algorithm.

4.5. Framework of the hybrid GA

The main positive effect of hybridizing GAs with local search is the improvement in the convergence speed to local optima. On the other hand, the main negative effect is the increase in the computation time per generation. For the fJSP problem, moving two operations is much more computationally complex than moving only one. In this study, local search of moving one operation is repeatedly implemented until local optima are found while local search of moving two operations is implemented only once in each iteration of the genetic search. Hence, local search of moving two operations does not tend to find the local optima in one iteration. The framework of the proposed hGA is illustrated in Fig. 9.

5. Computational results

The hGA was implemented in Delphi on a 3.0 GHz Pentium and tested on a large number of problem instances from the literature. We compare our results with results obtained by other authors.

Several sets of problem instances were considered:

(i) The first data set (XWdata) is a set of three problems from Kacem et al. [13,26] and Xia and Wu [15].
(ii) The second data set (BRdata) is a set of 10 problems from Brandimarte [6]. The data were randomly generated using a uniform distribution between given limits.
(iii) The third data set (BCdata) is a set of 21 problems from Barnes and Chambers [27]. As a job shop can be extended to a flexible job shop by simply adding some machines that already exist in the job shop, the data were constructed from three of the most challenging classical job shop scheduling problems [28,29] (mt10, la24, la40) by replicating machines. The processing times for operations on replicated machines are assumed to be identical to the original.

Table 1
Results on XWdata

Problem ($n \times m$)	Objective	AL+CGA		PSO+SA		moGA	Proposed hGA		
							Result	popSize	AV(CPU)
8×8	C_M	15	16	15	16	15	14		
	W_M			12	13	14	12	300	22.4
	W_T	79	75	75	73	73	77		
10×10	C_M	7		7		7	7		
	W_M	5		6		5	5	300	43.1
	W_T	45		44		43	43		
15×10	C_M	24		12			11		
	W_M	11		11			11	500	112.2
	W_T	91		91			91		

(iv) The fourth test sample (DPdata) is a set of 18 problems from Dauzère-Pérès and Paulli [10]. The set of machines capable of performing an operation was constructed by letting a machine be in that set with a probability that ranges from 0.1 to 0.5.

(v) The fifth test sample (HUdata) is a set of 129 problems from Hurink et al. [8]. The problems were obtained from three problems by Fisher and Thompson [28] (mt06, mt10, mt20) and 40 problems from Lawrence [29] (la01-la40). Each set A_{ik} is equal to the machine to which operation o_{ik} is assigned in the original problem, plus any of the other machines with a given probability. Depending on this probability, Hurink et al. generated three sets of test problems: edata, rdata and vdata. The first set contains the problems with the least amount of flexibility, whereas the average size of A_{ij} is equal to 2 in rdata and $m/2$ in vdata.

The non-deterministic nature of our algorithm makes it necessary to carry out multiple runs on the same problem instance in order to obtain meaningful results. We ran our algorithm five times. The parameters used in the hGA are chosen experimentally in order to get a satisfactory solution quality in an acceptable time span. Depending on the complexity of the problems, the population size of the GA ranges from 300 to 3000. The number of maximal generations is limited to 200. Other parameters are summarized in the following:

Enhanced order crossover probability: 0.4 Immigration mutation probability: 0.4
Uniform crossover probability: 0.4 Allele-based mutation probability: 0.4

AV(CPU) stands for average computer-independent CPU times in seconds. These values were computed using the normalization coefficients of Dongarra [30], as interpreted by Vaessens et al. [31].

The proposed hGA is first tested on the three problems of XWdata. For each problem, the best solutions we got at each of the five runs are different in terms of machine assignment and operation sequence, yet they all have the same performance on the three considered objectives. Kacem et al. [13,26], Zhang and Gen [14], and Xia and Wu [15] treated exactly the same problems. They have reported the obtained schedules without the CPU time used. Table 1 gives the performance of the proposed method compared with other algorithms. "AL + CGA" is the algorithm by Kacem et al. "PSO + SA" is the algorithm by Xia and Wu. "moGA" is the algorithm by Zhang and Gen. Column popSize gives the population size of GA. The solution obtained by our algorithm for problem 15×10 is illustrated in Fig. 10.

We also conducted experiments on the other four data sets. Because the data sets have been tested with a single objective (makespan) by other authors, we only compare the performance of our algorithm with previous ones on makespan. For the HUdata problems we consider the lower bounds of makespan computed by Jurisch [7]. For the remaining data sets only lower bounds computed in a straightforward way are available, therefore they are generally not very close to the optimum. (LB, UB) denotes the optimum makespan if known, otherwise, the best lower and upper bound found to date. Flex. denotes the average number of equivalent machines per operation. AV(C_M) stands for the average makespan out of five runs. $C_M^{\#}$, $W_M^{\#}$ and $W_T^{\#}$ denote makespan, maximal machine workload and total workload of the best solution found by our algorithm over five runs. M&G is the approach proposed by Mastrolilli and

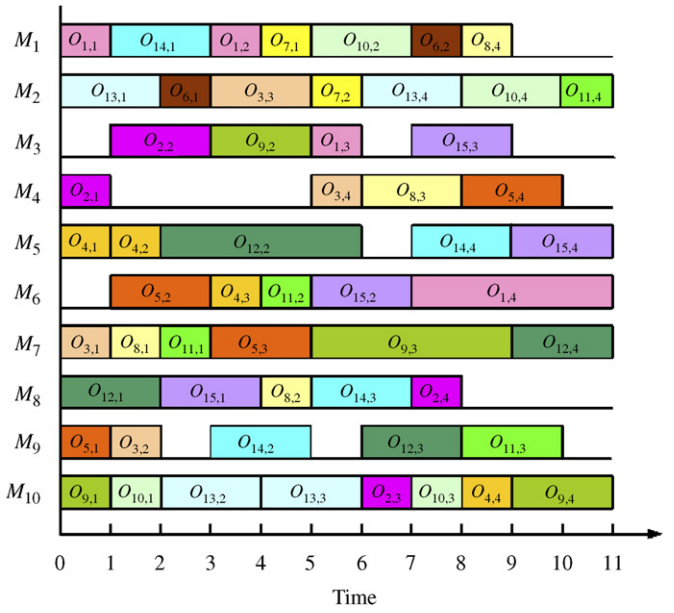

Fig. 10. Gantt chart of the obtained solution for problem 15 × 10.

Table 2
Results on BRdata

Problem	$n \times m$	Flex.	(LB, UB)	M&G		Proposed hGA					
				C_M	AV(C_M)	popSize	$C_M^{\#}$	AV(C_M)	$W_M^{\#}$	$W_T^{\#}$	AV(CPU)
Mk01	10 × 6	2.09	(36, 42)	*40	40.0	3000	*40	40	36	167	1.47
Mk02	10 × 6	4.10	(24, 32)	*26	26.0	3000	*26	26	26	151	3.44
Mk03	15 × 8	3.01	(204, 211)	*204	204.0	2000	*204	204	204	850	17.47
Mk04	15 × 8	1.91	(48, 81)	*60	60.0	3000	*60	60	60	375	3.42
Mk05	15 × 4	1.71	(168, 186)	173	173.0	1000	*172	172	172	687	6.82
Mk06	10 × 15	3.27	(33, 86)	*58	58.4	2000	*58	58	56	427	5.30
Mk07	20 × 5	2.83	(133, 157)	144	147.0	1000	*139	139	139	693	6.23
Mk08	20 × 10	1.43	523	*523	523.0	1000	*523	523	523	2524	8.44
Mk09	20 × 10	2.53	(299, 369)	*307	307.0	1000	*307	307	299	2312	18.71
Mk10	20 × 15	2.98	(165, 296)	198	199.2	2000	*197	197	197	2029	19.36

Gambardella [11]. Tables 2,3 and 4 give results on BRdata, BCdata and DPdata, respectively. The makespan associated with an asterisk is the best known upper bound by far. We only report summary results on 129 instances of HUdata.

We compute the relative error of our algorithm for each instance of HUdata, i.e., the percentage by which the best makespan (UB) obtained is above the best known lower bound (LB), that is $100 * (\text{UB} - \text{LB})/\text{LB}$. Table 5 gives the mean relative error of best makespan and average makespan over five runs on HUdata.

A solution obtained by our algorithm is illustrated in the Appendix. Detailed information about the results on HUdata and the solutions (including the assigned machine and the starting/completion time of each operation) obtained by our algorithm for each instance is available from the corresponding author (calebgao@yahoo.com).

A comparative overview of the hGA's best makespan, average makespan and computer-independent CPU time out of five runs is given in Table 6. Column Better, Equal and Worse represents the number of examples for which hGA's makespan is better, equal or worse than those found by Mastrolilli and Gambardella [11]. For some instances, their makespan found by the hGA is equal to the corresponding known lower bound, hence is the optimal value. The bracketed number denotes the number of optimal solutions. Column CPU time gives the sum of the average computer-independent CPU time out of five runs to compute all the solutions of each data set.

Table 3
Results on BCdata

Problem	$n \times m$	Flex.	(LB, UB)	M&G		Proposed hGA					
				C_M	AV(C_M)	popSize	$C_M^{\#}$	AV(C_M)	$W_M^{\#}$	$W_T^{\#}$	AV(CPU)
mt10c1	10 × 11	1.10	(655, 927)	928	928.0	3000	*927	927.2	631	5109	12.87
mt10cc	10 × 12	1.20	(655, 914)	*910	910.0	3000	*910	910	631	5109	12.24
mt10x	10 × 11	1.10	(655, 929)	*918	918.0	3000	*918	918	556	5109	12.69
mt10xx	10 × 12	1.20	(655, 929)	*918	918.0	3000	*918	918	556	5109	11.70
mt10xxx	10 × 13	1.30	(655, 936)	*918	918.0	3000	*918	918	556	5109	11.52
mt10xy	10 × 12	1.20	(655, 913)	906	906.0	3000	*905	905	548	5109	12.24
mt10xyz	10 × 13	1.30	(655, 849)	*847	850.0	3000	849	849	534	5109	10.71
setb4c9	15 × 11	1.10	(857, 924)	919	919.2	3000	*914	914	857	7727	45.99
setb4cc	15 × 12	1.20	(857, 909)	*909	911.6	3000	914	914	857	7727	38.79
setb4x	15 × 11	1.10	(846, 937)	*925	925.0	3000	*925	931	846	7727	43.20
setb4xx	15 × 12	1.20	(846, 930)	*925	926.4	3000	*925	925	846	7727	42.57
setb4xxx	15 × 13	1.30	(846, 925)	*925	925.0	3000	*925	925	846	7727	36.09
setb4xy	15 × 12	1.20	(845, 924)	*916	916.0	3000	*916	916	845	7727	41.49
setb4xyz	15 × 13	1.30	(838, 914)	*905	908.2	3000	*905	905	838	7727	39.15
seti5c12	15 × 16	1.07	(1027, 1185)	*1174	1174.2	3000	1175	1175	1027	11 472	72.27
seti5cc	15 × 17	1.13	(955, 1136)	*1136	1136.4	3000	1138	1138	888	11 472	63.00
seti5x	15 × 16	1.07	(955, 1218)	*1201	1203.6	3000	1204	1204	938	11 472	66.78
seti5xx	15 × 17	1.13	(955, 1204)	*1199	1200.6	3000	1202	1203	938	11 472	63.72
seti5xxx	15 × 18	1.20	(955, 1213)	*1197	1198.4	3000	1204	1204	938	11 472	63.36
seti5xy	15 × 17	1.13	(955, 1148)	*1136	1136.4	3000	*1136	1136.5	888	11 472	63.18
seti5xy	15 × 18	1.20	(955, 1127)	*1125	1126.6	3000	1126	1126	835	11 472	57.96

Table 4
Results on DPdata

Problem	$n \times m$	Flex.	(LB, UB)	M&G		Proposed hGA					
				C_M	AV(C_M)	popSize	$C_M^{\#}$	AV(C_M)	$W_M^{\#}$	$W_T^{\#}$	AV(CPU)
01a	10 × 5	1.13	(2505, 2530)	*2518	2528	3000	*2518	2518	2505	11 137	102.71
02a	10 × 5	1.69	(2228, 2244)	*2231	2234	3000	*2231	2531	2231	11 137	140.98
03a	10 × 5	2.56	(2228, 2235)	*2229	2229.6	1000	*2229	2229.3	2229	11 137	106.53
04a	10 × 5	1.13	(2503, 2565)	*2503	2516.2	3000	2515	2518	2503	11 085	95.93
05a	10 × 5	1.69	(2189, 2229)	*2216	2220	1000	2217	2218	2217	11 045	143.95
06a	10 × 5	2.56	(2162, 2216)	2203	2206.4	500	*2196	2198	2196	10 962	111.83
07a	15 × 8	1.24	(2187, 2408)	*2283	2297.6	1000	2307	2309.8	2287	16 485	356.32
08a	15 × 8	2.42	(2061, 2093)	*2069	2071.4	1000	2073	2076	2070	16 485	330.08
09a	15 × 8	4.03	(2061, 2074)	*2066	2067.4	500	*2066	2067	2065	16 485	327.49
10a	15 × 8	1.24	(2178, 2362)	*2291	2305.6	1000	2315	2315.2	2263	16 532	345.19
11a	15 × 8	2.42	(2017, 2078)	*2063	2065.6	500	2071	2072	2069	16 418	360.45
12a	15 × 8	4.03	(1969, 2047)	2034	2038	300	*2030	2030.6	2030	16 172	329.71
13a	20 × 10	1.34	(2161, 2302)	2260	2266.2	500	*2257	2260	2254	21 610	462.85
14a	20 × 10	2.99	(2161, 2183)	*2167	2168	500	*2167	2167.6	2164	21 610	587.13
15a	20 × 10	5.02	(2161, 2171)	2167	2167.2	200	*2165	2165.4	2165	21 610	669.92
16a	20 × 10	1.34	(2148, 2301)	*2255	2258.8	1000	2256	2258	2242	21 593	452.41
17a	20 × 10	2.99	(2088, 2169)	2141	2144	500	*2140	2142	2138	21 307	616.34
18a	20 × 10	5.02	(2057, 2139)	2137	2140.2	200	*2127	2130.7	2127	21 204	667.01

Totally, we found 38 new better solutions in terms of best solutions out of five runs, among which four solutions are optimal ones. Because GAs are by nature multipoint stochastic search methods, the proposed hGA seems to be quite stable and time consuming. The average makespan of the hGA over five runs is better than that of M&G on 87 test instances. Yet, the CPU time of the hGA is also much longer than that of M&G.

Table 5
Mean relative error on HUdata

Problem class	$n \times m$	edata		rdata		vdata	
		M&G	hGA	M&G	hGA	M&G	hGA
mt6/10/20	6 × 6 10 × 10 20 × 5	0.00 (0.10)	0.00 (0.10)	0.34 (0.36)	0.34 (0.34)	0.00 (0.00)	0.00 (0.00)
la01-05	10 × 5	0.00 (0.00)	0.00 (0.00)	0.11 (0.24)	0.07 (0.07)	0.00 (0.11)	0.00 (0.00)
la06-10	15 × 5	0.00 (0.00)	0.00 (0.00)	0.03 (0.08)	0.00 (0.00)	0.00 (0.03)	0.00 (0.00)
la11-15	20 × 5	0.29 (0.29)	0.29 (0.29)	0.02 (0.02)	0.00 (0.00)	0.00 (0.01)	0.00 (0.00)
la16-20	10 × 10	0.00 (0.00)	0.02 (0.02)	1.73 (1.77)	1.64 (1.64)	0.00 (0.00)	0.00 (0.00)
la21-25	15 × 10	5.62 (5.93)	5.60 (5.66)	3.82 (4.38)	3.57 (3.69)	0.70 (0.85)	0.60 (0.68)
la26-30	20 × 10	3.47 (3.76)	3.28 (3.32)	0.59 (0.76)	0.64 (0.72)	0.11 (0.18)	0.11 (0.13)
la31-35	30 × 10	0.30 (0.32)	0.32 (3.32)	0.09 (0.14)	0.09 (0.12)	0.01 (0.03)	0.00 (0.00)
la36-40	15 × 15	8.99 (9.13)	8.82 (8.95)	3.97 (4.47)	3.86 (3.92)	0.00 (0.00)	0.00 (0.00)

Table 6
Summary results

Data set	#	C_M			AV(C_M)			CPU time	
		Better	Equal	Worse	Better	Equal	Worse	M&G	hGA
XWdata	3	2	1	0	2	1	0	–	177
BRdata	10	3 (0)	7 (3)	0	4	6	0	74	91
BCdata	21	3 (0)	10 (0)	8	7	6	8	356	821
DPdata	18	6 (0)	4 (0)	8	13	0	5	2467	6206
Hudata	129	24 (4)	97 (73)	8	61	64	4	16 568	70 745

6. Conclusions and future study

We have developed a new approach hybridizing genetic algorithm with variable neighborhood descent to exploit the "global search ability" of genetic algorithm and "the local search ability" of variable neighborhood descent for solving multiobjective flexible job shop scheduling problem. An innovative two-vector representation scheme is proposed and an effective decoding method interprets each chromosome into an active schedule. A reorder procedure is used to unify the operation sequence in the chromosome with the sequence in the decoded schedule to facility genetic operators to pass down the good traits of the parents to their children. In order to enhance the heritability of crossover operation, operation sequence of Gen et al.'s representation is transformed into the format of permutation representation. Two crossover methods and two mutation operators were proposed for the genetic algorithm.

In order to enhance the search ability, VNS works under the framework of genetic search. Two VNSs are used: (a) moving one operation, and (b) moving two operations. Both local searches aim to break critical paths in a solution one by one so as to acquire a new solution with smaller makespan. Local search of moving one operation breaks critical

paths by deleting a critical operation from its current position and reallocating it on an assignable interval elsewhere. A novel method based on the concept of earliest and latest event time is used to find assignable intervals for the deleted operations. When the local optimum of moving one operation is found, the solution may be further improved by moving two operations simultaneously.

The proposed algorithm was tested on 181 benchmark problems. These results were compared with the results obtained by other algorithms. We got the same best solution as that obtained by the combined efforts of previous works for 119 instances, and found 38 new better solutions.

Future research directions include local search of moving three or more operations, the balance between genetic search and local search.

Acknowledgments

The authors would like to say thanks to the two anonymous reviewers and Prof. Linus Schrage for their comments. 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.

Appendix

Below is the schedule we obtained for problem la 21 of vdata of HUdata. Since the schedule is too complex to be shown in a Gantt chart, the starting and completion times of the operations assigned on each machine are as follows:

M_1: ($o_{12,1}$: 0–28) ($o_{14,2}$: 28–78) ($o_{13,3}$: 82–109) ($o_{13,4}$: 109–171) ($o_{11,3}$: 171–191) ($o_{12,4}$: 191–236) ($o_{8,4}$: 236–290) ($o_{14,5}$: 290–370) ($o_{9,5}$: 370–407) ($o_{13,7}$: 407–455) ($o_{1,7}$: 455–508) ($o_{8,7}$: 508–595) ($o_{8,8}$: 595–636) ($o_{1,9}$: 636–657) ($o_{3,8}$: 657–694) ($o_{11,10}$: 694–712) ($o_{4,10}$: 712–805).

M_2: ($o_{11,1}$: 0–31) ($o_{3,1}$: 31–50) ($o_{3,2}$: 50–133) ($o_{15,2}$: 133–188) ($o_{2,4}$: 188–230) ($o_{2,5}$: 230–309) ($o_{11,5}$: 309–334) ($o_{13,6}$: 334–401) ($o_{6,6}$: 401–408) ($o_{5,5}$: 408–425) ($o_{10,5}$: 425–499) ($o_{12,7}$: 500–522) ($o_{12,8}$: 523–594) ($o_{6,9}$: 595–689) ($o_{5,9}$: 690–738) ($o_{2,10}$: 739–804).

M_3: ($o_{5,1}$: 0–79) ($o_{4,2}$: 79–166) ($o_{4,3}$: 166–190) ($o_{4,4}$: 190–267) ($o_{15,4}$: 273–281) ($o_{11,4}$: 287–298) ($o_{9,4}$: 304–329) ($o_{7,6}$: 330–379) ($o_{2,6}$: 380–456) ($o_{13,8}$: 457–498) ($o_{10,6}$: 499–526) ($o_{7,8}$: 530–553) ($o_{7,9}$: 557–612) ($o_{4,9}$: 616–695) ($o_{3,9}$: 699–759) ($o_{13,10}$: 763–776) ($o_{5,10}$: 780–801).

M_4: ($o_{14,1}$: 0–12) ($o_{10,1}$: 12–106) ($o_{10,2}$: 106–190) ($o_{9,2}$: 193–225) ($o_{9,3}$: 226–303) ($o_{15,5}$: 306–353) ($o_{15,6}$: 356–432) ($o_{15,7}$: 435–473) ($o_{2,7}$: 476–550) ($o_{3,7}$: 553–612) ($o_{5,8}$: 615–687) ($o_{15,10}$: 730–762) ($o_{3,10}$: 762–805).

M_5: ($o_{4,1}$: 0–60) ($o_{2,2}$: 66–97) ($o_{14,3}$: 97–177) ($o_{6,3}$: 177–186) ($o_{6,4}$: 186–196) ($o_{13,5}$: 196–294) ($o_{1,5}$: 294–315) ($o_{1,6}$: 315–386) ($o_{4,6}$: 386–424) ($o_{4,7}$: 424–511) ($o_{6,8}$: 511–572) ($o_{14,9}$: 572–666) ($o_{8,9}$: 666–709) ($o_{10,10}$: 709–805).

M_6: ($o_{8,1}$: 0–9) ($o_{8,2}$: 9–29) ($o_{12,2}$: 29–126) ($o_{11,2}$: 126–150) ($o_{8,3}$: 150–189) ($o_{10,3}$: 190–268) ($o_{7,5}$: 278–315) ($o_{8,5}$: 324–360) ($o_{11,6}$: 369–441) ($o_{11,7}$: 450–517) ($o_{10,7}$: 526–595) ($o_{10,8}$: 595–664) ($o_{10,9}$: 664–709) ($o_{8,10}$: 716–723) ($o_{9,9}$: 730–765) ($o_{9,10}$: 772–798).

M_7: ($o_{1,1}$: 0–34) ($o_{13,2}$: 34–82) ($o_{6,2}$: 82–177) ($o_{5,3}$: 182–275) ($o_{5,4}$: 280–371) ($o_{12,6}$: 376–470) ($o_{5,6}$: 475–514) ($o_{4,8}$: 519–555) ($o_{1,8}$: 560–607) ($o_{15,9}$: 612–625) ($o_{9,8}$: 630–714) ($o_{12,10}$: 719–800).

M_8: ($o_{7,1}$: 0–28) ($o_{7,2}$: 29–87) ($o_{5,2}$: 88–164) ($o_{9,1}$: 165–192) ($o_{3,3}$: 195–226) ($o_{7,4}$: 229–269) ($o_{6,5}$: 272–304) ($o_{4,5}$: 307–373) ($o_{8,6}$: 376–444) ($o_{7,7}$: 447–496) ($o_{14,8}$: 499–559) ($o_{11,8}$: 562–646) ($o_{11,9}$: 649–678) ($o_{1,10}$: 681–704) ($o_{14,10}$: 707–802).

M_9: ($o_{6,1}$: 0–35) ($o_{1,2}$: 35–90) ($o_{7,3}$: 91–106) ($o_{2,3}$: 107–118) ($o_{1,3}$: 119–213) ($o_{1,4}$: 214–229) ($o_{15,3}$: 230–266) ($o_{3,4}$: 267–358) ($o_{3,5}$: 359–412) ($o_{6,7}$: 413–440) ($o_{14,7}$: 441–468) ($o_{9,7}$: 469–534) ($o_{5,7}$: 535–577) ($o_{13,9}$: 578–623) ($o_{12,9}$: 624–713) ($o_{7,10}$: 714–804).

M_{10}: ($o_{13,1}$: 0–27) ($o_{2,1}$: 27–66) ($o_{15,1}$: 66–127) ($o_{12,3}$: 127–185) ($o_{14,4}$: 185–235) ($o_{12,5}$: 236–312) ($o_{10,4}$: 312–393) ($o_{14,6}$: 393–412) ($o_{9,6}$: 417–420) ($o_{3,6}$: 425–499) ($o_{15,8}$: 504–571) ($o_{2,8}$: 576–669) ($o_{2,9}$: 674–724) ($o_{6,10}$: 729–800).

References

[1] Gen M, Tsujimura Y, Kubota E. Solving job-shop scheduling problem using genetic algorithms. In: Proceedings of the 16th international conference on computer and industrial engineering, Ashikaga, Japan; 1994. p. 576–9.
[2] Brucker P, Schlie R. Job-shop scheduling with multi-purpose machines. Computing 1990;45(4):369–75.
[3] Stecke KS. Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. Management Science 1983;29(3):273–88.
[4] Akella R, Choong Y. Performance of a hierarchical production scheduling policy. IEEE Transactions on Components, Hybrids and Manufacturing Technology 1984;7(3):225–48.
[5] Bona B, Brandimarte P. Hybrid hierarchical scheduling and control systems in manufacturing. IEEE Transactions on Robotics and Automation 1990;6(6):673–86.
[6] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 1993;41:157–83.
[7] Jurisch B. Scheduling jobs in shops with multi-purpose machines. Ph.D. dissertation, Fachbereich Mathematik/Informatik, Universitat Osnabruck; 1992.
[8] Hurink E, Jurisch B, Thole M. Tabu search for the job shop scheduling problem with multi-purpose machine. Operations Research Spektrum 1994;15:205–15.
[9] Chambers JB. Classical and flexible job shop scheduling by tabu search. Ph.D. dissertation, University of Texas at Austin, USA; 1996.
[10] Dauzère-Pérès S, Paulli J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Annals of Operations Research 1997;70(3):281–306.
[11] Mastrolilli M, Gambardella LM. Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling 2000;3(1):3–20.
[12] Yang J-B. GA-based discrete dynamic programming approach for scheduling in FMS environments. IEEE Transactions on Systems, Man, and Cybernetics—Part B 2001;31(5):824–35.
[13] Kacem I, Hammadi S, Borne P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Transaction Systems, Man, and Cybernetics—Part C 2002;32(1):1–13.
[14] Zhang H-P, Gen M. Multistage-based genetic algorithm for flexible job-shop scheduling problem. Journal of Complexity International 2005;11:223–32.
[15] Xia W, Wu Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problem. Computers & Industrial Engineering 2005;48:409–25.
[16] Wu Z, Weng M-X. Multiagent scheduling method with earliness and tardiness objectives in flexible job shops. IEEE Transactions on System, Man, and Cybernetics—Part B 2005;35(2):293–301.
[17] Vaessens RJM. Generalized job shop scheduling: complexity and local search. Ph.D. dissertation, Eindhoven University of Technology; 1995.
[18] Brucker P, Neyer J. Tabu-search for the multi-mode job-shop problem. OR Spectrum 1998;20:21–8.
[19] Baker K. Introduction to sequencing and scheduling. New York: Wiley; 1974.
[20] Gen M, Cheng R. Genetic algorithms & engineering design. New York: Wiley; 1997.
[21] Moscato P, Norman M. A memetic approach for the traveling 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, Amsterdam; 1992.
[22] Gen M, Cheng R. Genetic algorithms & engineering optimization. New York: Wiley; 2000. p. 1–30.
[23] Mladenovic N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[24] Hansen P, Mladenovic N. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130: 449–67.
[25] Balas E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Operations Research 1969;17:941–57.
[26] Kacem I, Hammadi S, Borne P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Mathematics and Computers in Simulation 2002;60:245–76.
[27] Barnes JW, Chambers JB. Flexible job shop scheduling by tabu search. Graduate program in operations research and industrial engineering, The University of Texas at Austin 1996; Technical Report Series: ORP96-09; ⟨http://ww.cs.utexas.edu/users/jbc/⟩.
[28] Fisher H, Thompson GL. Probabilistic learning combinations of local job shop scheduling rules. Englewood Cliffs, NJ: Prentice-Hall; 1963. p. 225–51.
[29] Lawrence S. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie Mellon University, Pittsburgh, PA; 1984.
[30] Dongarra JJ. Performance of various computers using standard linear equations software. Computer Science Department, University of Tennessee, Knoxville, TN; 2006.
[31] Vaessens RJM, Aarts EHL, Lenstra JK. Job shop scheduling by local search. INFORMS Journal on Computing 1996;8(3):302–17.