UDC 005.8:518.876.3, DOI:10.2298/CSIS090608017L

A Hybrid Variable Neighborhood Search Algorithm for Solving Multi-Objective Flexible Job Shop Problems

Jun-qing Li[1], Quan-ke Pan[1], and Sheng-xian Xie[1]

[1]College of Computer Science, Liaocheng University
Liaocheng, 252059, People's Republic of China
{Lijunqing, qkpan, xsx}@lcu.edu.cn

Abstract. In this paper, we propose a novel hybrid variable neighborhood search algorithm combining with the genetic algorithm (VNS+GA) for solving the multi-objective flexible job shop scheduling problems (FJSPs) to minimize the makespan, the total workload of all machines, and the workload of the busiest machine. Firstly, a mix of two machine assignment rules and two operation sequencing rules are developed to create high quality initial solutions. Secondly, two adaptive mutation rules are used in the hybrid algorithm to produce effective perturbations in machine assignment component. Thirdly, a speed-up local search method based on public critical blocks theory is proposed to produce perturbation in operation sequencing component. Simulation results based on the well-known benchmarks and statistical performance comparisons are provided. It is concluded that the proposed VNS+GA algorithm is superior to the three existing algorithms, i.e., AL+CGA algorithm, PSO+SA algorithm and PSO+TS algorithm, in terms of searching quality and efficiency.

Keywords: Flexible Job Shop Scheduling Problem; Multi-objective; Genetic Algorithm; Variable Neighborhood Search.

1. Introduction

The job-shop scheduling problem (JSP) is one of the most popular scheduling models existing in practice, which has been proven to be among the hardest combinatorial optimization problems [1, 2] and has got more and more research focus in recent years. The Flexible job-shop problem (FJSP), an extension of the classical JSP, is harder than the latter because the addition of assignment of a suitable machine from a set of candidate machines for each operation.

The FJSP recently has captured the interests of many researchers. The first paper about solving the FJSP was proposed by Brucker and Schlie (Brucker & Schlie, 1990) [3], which gives a simple FJSP model with only two jobs and

each operation performed on each machine with the same processing time. The first author with the hierarchical idea to solve the FJSPs was Brandimarte (Brandimarte, 1993) [4], who solved the first stage with some existing dispatching rules and the second stage with tabu search heuristic algorithms. Kacem (Kacem, Hammadi & Borne, 2002) [5] solved the two stage problems with the genetic algorithm (GA). Gao (Gao & Gen et al., 2006) [6] used bottleneck shifting method in genetic algorithm for solving the FJSP. Saidi-mehrabad (Saidi-mehrabad, 2007) [7] gave a detailed solution with tabu search method. Li et al. (2009) [8] presented a hybrid particle swarm optimization (PSO) combining with a fast neighborhood structure algorithm for the problem.

The research on the multi-objective FJSP is much less than the mono-objective FJSP. Kacem et al. (2002a, 2002b) [5, 9] developed an effective evolutionary algorithm controlled by an assigned model based on the approach of localization (AL). Xia and Wu (2005) [10] presented a practical hierarchical solution approach by making use of PSO to assign operations on machines and simulated annealing (SA) algorithm to schedule operations on each machine. Zhang et al. (2009) [11] developed a hybrid algorithm combining PSO with tabu search (TS) algorithm. Most of the above algorithms solved the multi-objective FJSP problem by transforming it to a mono-objective one through giving each objective a different weight.

In this paper, we propose a novel hybrid variable neighborhood search algorithm combining with the genetic algorithm (VNS+GA) for solving the multi-objective FJSP problems to minimize the makespan, total workload of all machines, and workload of the busiest machine. In the hybrid algorithm, the three objectives are also combined into a single objective by assigning each objective a different weight. GA is used to produce a swarm of candidate solutions, whereas VNS is introduced to obtained more optimal solutions around the given candidate solutions. A mix of two machine assignment rules and two operation sequencing rules are developed to create high quality initial solutions. To produce effective perturbations in the machine assignment module, two adaptive mutation rules are used in the hybrid algorithm. A speed-up local search method based on public critical blocks theory is proposed to produce perturbation in operation sequencing component.

The rest of this paper is organized as follows: In section 2, we briefly describe the problem formulation. Then, the framework of our hybrid algorithm is presented in Section 3. The GA for perturbation in machine assignment component is introduced in Section 4. Section 5 illustrates the VNS approach for local searching in operation sequencing component. Section 6 shows the experimental results compared with other algorithms. Finally, Section 7 gives the conclusion of our works.

2. Problem formulation

FJSP is an extension of the classical JSP; therefore, we can formulate the FJSP based on the JSP. Consider a set of *n* jobs, denoted as $J = \{J_1, J_2, ...J_n\}$, each job J_i $(1 \le i \le n)$ in *J* has a pre-defined number of operations, and should be operated on a selected machine from a machine set named $M = \{M_1, M_2, ...M_m\}$. The main different between FJSP and JSP lies in two aspects: first, in the classical JSP problem, with *n* jobs and *m* machines, there are $n \times m$ operations, whereas in FJSP, given *n* jobs and *m* machines, the number of operations may large or small than $n \times m$; second, in the classical JSP, an operation should be operated on a pre-defined machine, whereas in FJSP, an operation can be operated by a set of machines. Therefore, FJSP is harder than JSP. There are two kinds of FJSP problems, i.e., T-FJSP (Total Flexible Job-shop Scheduling Problem) and P-FJSP (Partial Flexible Job-shop Scheduling Problem) [8, 9]. For the T-FJSP, each job can be operated on every machine from the set *M*, whereas for the P-FJSP, there is an additional problem constraint, that is, one operation of a job can be processed by a sub set of machines $M' \subset M$.

In this paper, the following objectives are to be minimized:

(1) c_{M}. Maximal completion time of all machines, i.e., the makespan;

(2) w_{T}. Total workload of all machines;

(3) w_{M}. Workload of the critical machine or the busiest machine.

The weighted sum of the above three objective values is taken as the objective function in this study:

$F(c) = w_1 \times c_{\mathrm{M}} + w_2 \times w_{\mathrm{T}} + w_3 \times w_{\mathrm{M}}$

Where, w_1, w_2, w_3 represent the weight assigned to the objective c_{M}, w_{T} and w_{M}, respectively.

The following assumptions are given in this study [8-11]:

(1) Each machine can perform at most one operation at any time and can not be interrupted during its work.
(2) Each operation can not be interrupted during its performance.
(3) Setting up time of machines and move time between operations are negligible.
(4) Jobs are independent from each other.
(5) Machines are independent from each other.

Some useful notations are given as follows:

- Let J = $\{J_i\}_{1 \le i \le n}$, indexed *i*, be as set of *n* jobs to be scheduled.
- Let $M = \{M_k\}_{1 \le k \le m}$, indexed *k*, be a set of *m* machines.
- Each job J_i can be operated on a given set of machines M_i.
- The $O_{i,j}$ represents the *j*th operation of J_i.

- The set of candidate machines waiting for processing $O_{i,j}$ is denoted as $M_k \subseteq M$.
- $p_{i,j,k}$ represents the processing time of $O_{i,j}$ operated on the *k*th machine.
- Two sub-problems of the FJSP: T-FJSP and P-FJSP.

$$FJSP = \begin{cases} T-FJSP, & \text{if } M(O_{i,h}) = M; \forall i,h \\ P-FJSP, & \text{if } M(O_{i,h}) \subset M; \forall i,h \end{cases}$$

- Decision variables

$$x_{i,h,k} = \begin{cases} 1, & \text{if machine } k \text{ is selected for the operation } O_{i,h} \\ 0, & \text{otherwise} \end{cases}$$

$c_{i,h}$: completion time of the operation $O_{i,h}$.

The formulation of the multi-objective FJSP in this study is then given in Fig.1.

$$\min c_M = \max_{1 \le i \le n} \{c_{i,n_i}\} \quad (1)$$

$$\min w_M = \max_{1 \le k \le m} \{\sum_{i=1}^{n} \sum_{h=1}^{n_i} p_{i,h,k}\} \quad (2)$$

$$\min w_T = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{h=1}^{n_i} p_{i,h,k} x_{i,h,k} \quad (3)$$

$$\text{s.t. } c_{i,h} - c_{i,h-1} \ge p_{i,h,k} x_{i,h,k}, \quad h = 2,\ldots,n_i; \forall i,k \quad (4)$$

$$\sum_{k \in M(O_{i,h})} x_{i,h,k} = 1, \forall i,h \quad (5)$$

$$M(O_{i,h}) \subseteq M, \forall i,h \quad (6)$$

$$x_{i,h,k} \in \{0,1\}, \forall i,h,k \quad (7)$$

$$c_{i,h} \ge 0, \forall i,h \quad (8)$$

Fig. 1. Problem formulation of the multi-objective FJSP

Equation (4) ensures the operation precedence constraints. Equation (5) guarantees that for each operation one and only one machine must be selected from the set of available machines. Inequity (6) indicates that the set of available machines for each operation come from the given machine set M.

3. Framework of the Hybrid Algorithm

In this study, we propose a hybrid variable neighborhood search algorithm combining with the genetic algorithm (VNS+GA) for solving the multi-objective FJSPs. The detail steps of the VNS+GA algorithm are listed as follows.

Step1. Initialization

Step1.1: Set up parameters.

Step1.2: Produce machine assignment component for each chromosome in the population C_{pop}.

Step1.3: Produce operation sequencing component for each chromosome in the population C_{pop}.

Step1.4: Evaluate each chromosome, and then obtain the best solution C_{best}.

Step1.5: if stop criteria is satisfied, then go to Step 4; otherwise, go to Step 2.

Step2: Perturbation in machine assignment component

Step2.1: Produce P_{size} child chromosomes by applying crossover operation.

For i=0 to P_{size}

(1) Generate a random number r, if $r < p_{select}$, then randomly select one parent chromosome in the population C_{pop} denoted as P_1, and select the current best solution C_{best} as P_2. Otherwise, select two parent chromosomes in the population C_{pop} at random denoted as P_1 and P_2, respectively.

(2) Produce two child chromosomes denoted as C_1 and C_2 by applying crossover function with probability p_c on the two parent chromosomes P_1 and P_2.

(3) Evaluate the two child chromosomes; if one chromosome from the two child chromosomes denoted as C_{be} which is better than C_{best}, then replace C_{best} by C_{be}.

(4) Select the best solution from the four chromosomes (i.e., P_1, P_2, C_1 and C_2), and then insert it into a temp population T_{pop}.

End for

Step2.2: Produce P_{size} child chromosomes by applying mutation operation.

For i=0 to P_{size}

(1) Select the ith chromosome in population T_{pop} as the parent chromosome PP_1.

(2) Produce a child chromosome denoted as CC_1 by applying mutation operation with probability p_m^c on the selected parent

chromosomes PP_1.

(3) Evaluate the child chromosome CC_1; if CC_1 is better than C_{best}, and then replace C_{best} by CC_1.

(4) Insert the best solution among CC_1 and PP_1 into T_{pop}.

End for

Step2.3: Select P_{size} better chromosomes in T_{pop}

(1) Sequence all $2 \times P_{size}$ chromosomes from population T_{pop} in descending order on chromosome fitness, that is, the chromosome with optimal fitness value will appear at the relative top position.

(2) Select the top P_{size} chromosomes as the current population C_{pop} for next generation.

Step3: Perturbation in operation sequencing component

For the current best solution C_{best}, operate the following steps:

Step3.1: Get all critical operations.

Step3.2: Get all public critical operations.

Step3.3: Get all public critical blocks.

Step3.4: Use the function *effectiveNeighbor*() to search the best neighbor solution of the current best solution. If the former is more optimal than the latter, then replace the current best solution C_{best} by the new neighbor solution. Then go to step 1.5.

Step4: Output the current best solution C_{best} and stop.

4. Machine assignment algorithm: the genetic algorithm

4.1. Genetic Algorithm

Genetic Algorithm (GA), proposed by J. Holland in 1975 [12, 13], has been used to solve optimization problems in recent years [13]. The GA is based on the genetic process of biological organisms. Several key factors including populations, crossover functions, mutation functions, evolution approaches and stop criterion are important for the efficiency of the GA. The main steps for the process of GA can be described as follows [13].

Step1: Let k=0. Randomly produce N chromosomes as the initial population $p(k)$.

Step2: Evaluate each chromosome in the population and get the fitness value of every solution.

Step3: If stop criterion is satisfied, then output the best solution; otherwise operate steps 4-8.

Step4: Let *m*=0.

Step5: Then, select two chromosomes in the population $p(k)$ using certain selection rules, which are named p_1 and p_2, respectively.

Step6: Randomly produce a real number $\zeta \in [0,1]$, if $\zeta < p_c$, where p_c is crossover probability, then apply given crossover function on the two selected parent chromosomes. The resulted two chromosomes are selected as two temp chromosomes, namely t_1 and t_2, respectively. Otherwise, the two parent chromosomes will be selected as the two temp chromosomes t_1 and t_2, respectively.

Step7: Randomly produce a real number $\delta \in [0,1]$, if $\delta < p_m$, where p_m is mutation probability, then apply given mutation function on t_1 and t_2 respectively. The two resulted chromosomes will be inserted into the new population $p(k+1)$.

Step8: Let *m*=*m*+2. If *m*<*N*, then go back to step 5. Otherwise, let *k*=*k*+1, and then go back to step 2.

4.2. Encoding

The FJSP problems involve two decision stages, i.e., machine assignment stage and operation sequencing stage. Therefore, a solution consists of two parts of vectors, $A_1 = \{A_1(1), A_1(2),....,A_1(\kappa)\}$ (machine assignment vector) and operation sequencing vector $A_2 = \{A_2(1), A_2(2),....,A_2(\kappa)\}$, where κ equals to the operation number. $A_1(i), 1 \le i \le \kappa$ represents the corresponding selected machine for each operation. Fig. 2 shows an example of a machine assignment vector. For example, it can be seen from Fig. 2 that the operation O_{11} is performed on machine M_4, O_{12} is performed on machine M_3, and so on. An operation sequencing vector is shown in Fig. 3, which tells us the operation sequence as follows.

$$O_{21} \succ O_{11} \succ O_{31} \succ O_{22} \succ O_{12} \succ O_{23} \succ O_{13} \succ O_{32}$$

position	1	2	3	4	5	6	7	8
operation	O_1	O_1	O_1	O_2	O_2	O_2	O_3	O_3
	1	2	3	1	2	3	1	2
machine	4	3	2	1	2	1	2	3

Fig. 2. Machine assignment vector example

position	1	2	3	4	5	6	7	8
operation	2	1	3	2	1	2	1	3

Fig. 3. Operation sequencing vector example

4.3. Initialization of Machine assignment component

Following are two approaches for the initialization of machine assignment component:

- Random rule, denoted as $\boldsymbol{MS_a}$. For each operation J_i, a random selected machine from a set of candidate machines, denoted as M_i, will be placed in position *i* in the machine assignment component.
- Local minimum processing time rule, denoted as $\boldsymbol{MS_b}$. Table 1 gives an example about the steps of this rule, the example data come from [14]. For operations of the same job, finding the minimum processing time, fixing the assignment, and then adding this processing time to every other entry in the same column.

4.4. Crossover operation

Given two parent chromosomes p_1 and p_2 in Fig. 4. The steps of the crossover operator are as follows.

Step1: Generate two random numbers r_1 and r_2, $2 \le r_1, r_2 \le (\kappa - 1)$, where κ equals the number of operations.

Step2: Select the subsection between r_1 and r_2 of one parent chromosome such as p_2.

Step3: Produce a temporary vector named c_1 by copying the selected subsection into the corresponding position.

Step4: Copy the corresponding operation from p_1 into the unfixed position.

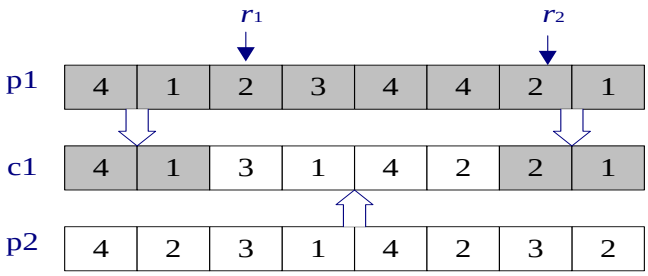

Fig. 4. Crossover operator

4.5. Mutation operation

Mutation operation is very important in GA with aim to produce population diversity. In order to obtain population diversity as well as population convergence, two mutation-operation rules are proposed as follows.

- Random rule denoted as ϖ_1. (1) randomly select an operation with more than two candidate machines, denoted O_s; (2) randomly select a machine from *Machines*(O_s) different with the current machine; (3) replace the current machine by the selected machine at the position O_s.
- Last Processing rule denoted as ϖ_2. (1) record the last release time for each machine; (2) create a vector M_{lp} including all machines with last release time equals the current makespan; (3) get all public critical operations; (4) for each machine M_{old} in M_{lp}, firstly, find a public critical operation O_s which is processed on M_{old}, secondly, select a candidate machine for processing O_s which is not in M_{lp}, denoted M_s; (5) replace the current machine M_{old} by the selected machine M_s at position O_s.

5. Operation sequencing algorithm: variable neighborhood search algorithm

5.1. Initialization of the operation sequencing component

Once the machine assignment component is fixed, we should consider how to sequence the operations on each machine, i.e., to determine the start time of every operation. In this section, we should consider two issues: the operation precedence constraint of the same job and the objective of the problem. In our hybrid algorithm, the operation sequence is obtained through a mix of following two different approaches:

- Random rule, denoted as OS_a. OS_a is the naive and direct approach for sequencing operations. The advantage of this approach is its simplicity. The disadvantage is also obvious too, that is, it can easily produce idle time interval and make the finding solution process more time consuming.
- Most Work Remaining (MWR) denoted as OS_b. This approach selects the operation with the most remaining work for each machine. The operation precedence constraint of the same job must be considered at the same time.

5.2. Public critical block theory

The critical problem of local searching is how to design an effective neighborhood around a given solution. The promising neighborhood is based on the concept of critical path, which was firstly proposed by Adams [15] in solving JSP problems.

The feasible schedules of FJSP problems can be represented with a disjunctive graph G= (N, A, E), where N is the node set, A is the conjunctive arc set, and E is the disjunctive arc set. The number aside the node indicates the processing time of this operation on the assigned machine. Each arc in A represents the operation precedence constraint. The dashed arcs (E) correspond to pairs of operations to be performed on the same machine. For example, given a chromosome {1,2,2,3,2,3,3,1 | 1,1,4,2,3,3,2,4}, Fig. 5 shows the disjunctive graph for a feasible solution of the example chromosome.

If G has more than one critical path, noted $CP_i, 1 < i \le nc$, where nc represents the number of critical path. Those critical operations belonging to all nc critical paths are called public critical operations. A public critical block is a maximal sequence of adjacent public critical operations processed on the same machine. Fig. 6 shows the Gantt chart for the feasible solution of the above chromosome example. In the example solution, there are six public operations, i.e. $\{O_{11}, O_{12}, O_{21}, O_{31}, O_{32}, O_{22}\}$, whereas there are three public critical blocks, i.e. $\{O_{11}\}, \{O_{12}, O_{21}, O_{31}\}, \{O_{32}, O_{22}\}$.

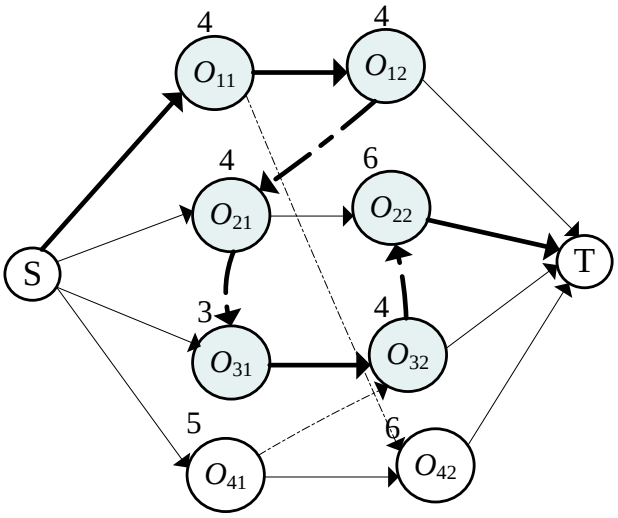

Fig. 5. The disjunctive graph for the example chromosome

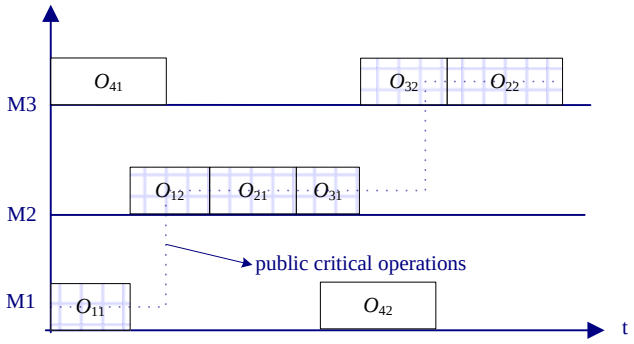

Fig. 6. The Gantt chart for the example chromosome

To develop neighborhood structure based on the public critical block theory, we give some notations as follows.

- JP_i, JS_i, MP_i, MS_i indicates the immediate job predecessor, job successor, machine predecessor and machine successor of the operation J_i, repectively.
- λu: the critical path with operation u in it.
- $O_{\lambda u}$: those operations in the same critical path λ which has operation u in it.
- $PR_{O_{\lambda u}}$: those operations belonging to the operation set $O_{\lambda u}$ and be operated before operation u.
- $L(u,v)$: the length of the longest path from the operation u to v.

Next, we give three theorems about the neighborhood structure based on public critical path theory.

Theorem 1. *If G has more than one critical path, and two operations u and v are critical operations but not public critical operations, then moving u or v cannot yield a better solution*.

Proof. First, if there exists a public critical operation in $PR_{O_{\lambda u}}$, denoted as O_k. The path subsection from O_k to u is called sub(u). Because the operation u is not a public critical operation, there exists an operation u' with the processing time interval crossover with the processing time interval of u. Therefore, u' is in another critical path but not a public critical operation either. The path subsection from O_k to u' is called sub(u'). The movement in sub(u) does not affect the length of sub(u'). So, the start time of O_k will not change and the makespan of the solution will not be improved. Second, if there does not exist any operation in $PR_{O_{\lambda u}}$, which means that the public critical operations are all operated after u. The first public critical operation after u denoted as O_k. There exists another critical operation u' which is before O_k and with the processing time crossed over with u. We name the critical path subsection

from u to O_k sub(u), and the critical path subsection from u' to O_k sub(u'). The movement in sub(u) also cannot affect the length of the sub(u'), and useless for improvement of the makespan.

Theorem 2. *If two public critical operations u and v to be performed on the same machine, v is the block rear and* $L(v,T) \geq L(JS_u,T)$ *, then inserting u right after v yields an acyclic complete selection.*

This theorem derives the idea that if two following conditions are satisfied: (1) there is no directed path from JS_u to v in G; (2) the complete time of v is not after the complete time of the immediate job successor of u. Then, inserting u right after v can produce a feasible solution as shown in Fig 7. The proof is analogous to the proof of the theorem 1 in [16].

Theorem 3. *If two public critical operations u and v to be performed on the same machine, u is the block head and* $L(0,u)+p_u \geq L(0,JP_v)+p_{JP_v}$ *, then inserting v right before u yields an acyclic complete selection.*

This theorem derives the idea that if two following conditions are satisfied: (1) there is no directed path from JP_v to u in G; (2) the complete time of u is not before the complete time of the immediate job predecessor of v. Then, inserting v right before u can produce a feasible solution as shown in Fig 8. The proof is analogous to the proof of the theorem 2 in [16].

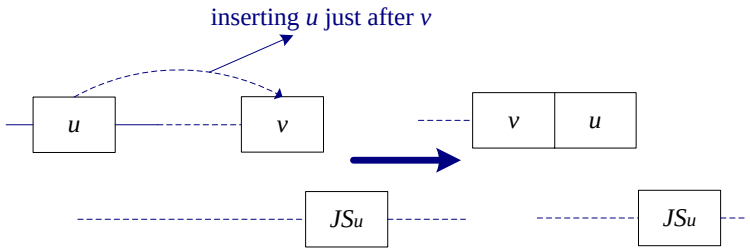

Fig. 7. A chart for inserting the inner operations just after the block rear

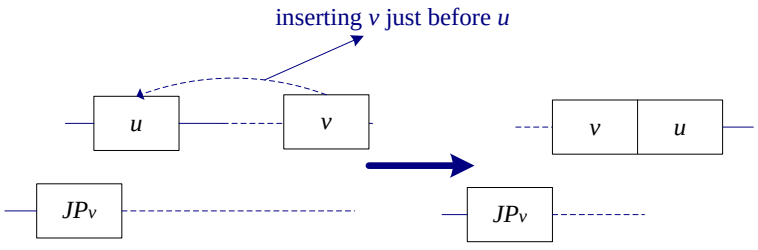

Fig. 8. A chart for inserting the inner operations just before the block head

5.3. Effective neighborhood structure

The makespan of a solution equals the length of its critical path, in other words, the makespan cannot be reduced while maintaining the current critical paths. The right direction of the local search is to identify and break the entire existent critical paths one by one in order to get a better solution.

The first successful critical path neighborhood structure for the classical JSP was introduced by Van Laarhoven et al [17], and is often denoted by N1. The N1 neighborhood is generated by swapping any adjacent pair of critical operations on the same machine. Dell Amico and Trubian [18], Nowicki & Smutnicki [19] and Balas & Vazacopoulos [20] proposed N4, N5 and N6 respectively. The N4 neighborhood is developed by moving an internal operation to the very beginning of its block or the very end of its block. The N5 neighborhood is created by swapping the first two or the last two operations in a block. The N6 neighborhood is produced by moving an operation to the beginning of the block or to the end of the block. In this study, we extend the critical path neighborhood structure for solving the FJSP, and propose some novel neighborhood structures based on the public critical blocks theory.

For discuss conveniently, we give two neighborhood categories as follows:

Definition 1: *Insert neighborhood*

First, randomly select two different positions i and i' in a feasible schedule chromosome, and then delete the ith operation and insert it before or after position i'.

Definition 2: *Swap neighborhood*

First, randomly select two different positions i and i' in a feasible schedule chromosome, and then swap the two operations at the selected position.

Based on the public critical block theory and the block structure listed above, we give an effective local search operator as shown in Fig. 9.

Procedure *effectiveNeighbor*()
Input: a set named *pb* including all public critical blocks
Output: an optimal neighbor solution

for *i*=0 to *pb.size*()
$pb_t = pb[i]$
if pb_t contains more than two public critical operations, then
k← the number of public critical operations in pb_t
u← $pb_t[0]$ //block head
v← $pb_t[k-1]$ //block rear
Step1: *Insert structures*
for *j*=0 to *k*-1
q← $pb_t[j]$
Step1.1: if ($L(0,q) + p_q \geq L(0, JP_v) + p_{JPv}$) then
Insert *v* right before *q*

$pb_t = pb[i]$

$q \leftarrow pb_t[j]$

Step1.2: if ($L(v,T) \geq L(JS_q,T)$) then

Insert *q* right after *v*

$pb_t = pb[i]$

$q \leftarrow pb_t[j]$

Step1.3: if ($L(0,u) + p_u \geq L(0,JP_q) + p_{JPq}$) then

Insert *q* right before *u*

$pb_t = pb[i]$

$q \leftarrow pb_t[j]$

Step1.4: if ($L(q,T) \geq L(JS_u,T)$) then

Insert *u* right after *q*

$pb_t = pb[i]$

end for

Step2: *Swap structures*

Step2.1: Swap $pb_t[0]$ with $pb_t[1]$

$pb_t = pb[i]$

Step2.2: Swap $pb_t[k-2]$ with $pb_t[k-1]$

end for

Evaluate each neighbor solution produced by the above two steps.

If the new solution is better than the current solution, then replace the current solution with the new one.

Output the current optimal solution.

Fig. 9. Pseudo-code of *effectiveNeighbor*()

6. Experimental results

This section describes the computational experiments to evaluate the performance of the proposed algorithm. For this purpose, we made a detail comparison with three existing algorithms, i.e., AL+CGA algorithm [9], PSO+SA algorithm [10] and PSO+TS algorithm [11]. The test samples come from [5]. The dimensions of the problems range from 4 jobs $\times$ 5 machines to 15 jobs $\times$ 10 machines. The current instantiation was implemented in C++ on a Pentium IV 1.8GHz with 512M memory.

6.1. Setting parameters

Each instance can be characterized by the following parameters: number of jobs (n), number of machines (m), and the number of operations (op_num).

Followings are the detail parameters value:

- Population size P_{size}: 1000;
- Maximum number of generations $gen_{\max}$: $n \times m$;
- Maximum number of iteration with no improvement of the best solution during the local search $iter_{\max}$: $op\_num/2$;
- Crossover probability for the machine assignment component: 45%;
- Minima mutation probability $p_m^{\min}$: 40%;
- Maxima mutation probability $p_m^{\max}$: 95%;
- Current mutation probability at *t* generation p_m^c :

$$p_m^c = p_m^{\min} + (\frac{t}{gen_{\max}}) \times (p_m^{\max} - p_m^{\min})$$

- Probability for selection between the best chromosome and a random one as a parent chromosome for crossover p_{select} :

$$p_{select} = 0.8 - (\frac{t}{gen_{\max}}) \times (0.8 - 0.2)$$

- Rate of initial assignments with MS_a: 20%;
- Rate of initial assignments with MS_b: 80%;
- Rate of initial assignments with OS_a: 20%;
- Rate of initial assignments with OS_b: 80%;
- Rate of initial assignments with ϖ_1 : 50%;
- Rate of initial assignments with ϖ_2 : 50%;

6.2. Results of the Kacem instances

The test instances come from the five Kacem instances [5] (i.e. problem 4×5, 8×8, 10×7, 10×10 and 15×10). The five tables from Table 2 to Table 5 show the comparison results for the five problems. Some notations are given as follows: The column labeled 'AL + CGA' refers to Kacem's method [9] and the column labeled 'PSO + SA' refers to the algorithm proposed by Xia and Wu [10]. The column labeled 'PSO+TS' shows the results from the hybrid algorithm developed by Zhang et al. [11]. Figs 10 to 14 show the optimal solutions obtained by our approach in the form of Gantt chart. The pair of number (in the form of [job, operation]) inside the blocks is the operation to be processed on the corresponding machine. The two numbers just below the block represent the start time and end time of the operation, respectively.

Problem 4×5

This is an instance of total flexibility, in which 4 jobs with 12 operations are to be performed on 5 machines. The obtained solutions by our hybrid algorithm are characterized by the following values:

Solution 1: c_{M} =12, w_{T} =32, w_{M} =8
Solution 2: c_{M} =11, w_{T} =32, w_{M} =10
Solution 3: c_{M} =11, w_{T} =34, w_{M} =9

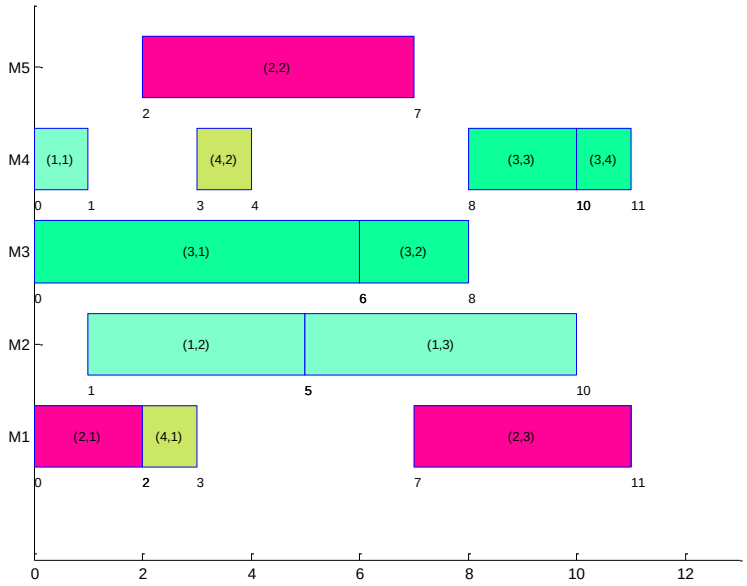

Fig. 10. The obtained optimal solution of instance 1 (4 jobs 12 operations 5 machines: F_1(c) =11, F_2(c) =34 F_3(c) =9)

Table 2. Comparison of results on problem 4 × 5 with 12 operations

	AL+CGA	PSO+TS	VNS+GA		
c_{M}	16	11	11	11	12
w_{T}	34	32	32	34	32
w_{M}	10	10	10	9	8

It can be seen from Table 2 that the VNS+GA algorithm dominate the AL+CGA algorithm in solving the problem 4×5. Our approach can obtain richer optimal solutions than the PSO+TS algorithm. Fig. 10 shows the obtained Solution 3 in the form of Gantt chart.

Problem 8×8

This is an instance of partial flexibility, in which 8 jobs with 27 operations are to be performed on 8 machines. The obtained solutions by our hybrid algorithm are characterized by the following values:

Solution 1: c_{M} =14, w_{T} =77, w_{M} =12

Solution 2: c_{M} =15, w_{T} =75, w_{M} =12

Solution 3: c_{M} =16, w_{T} =73, w_{M} =13

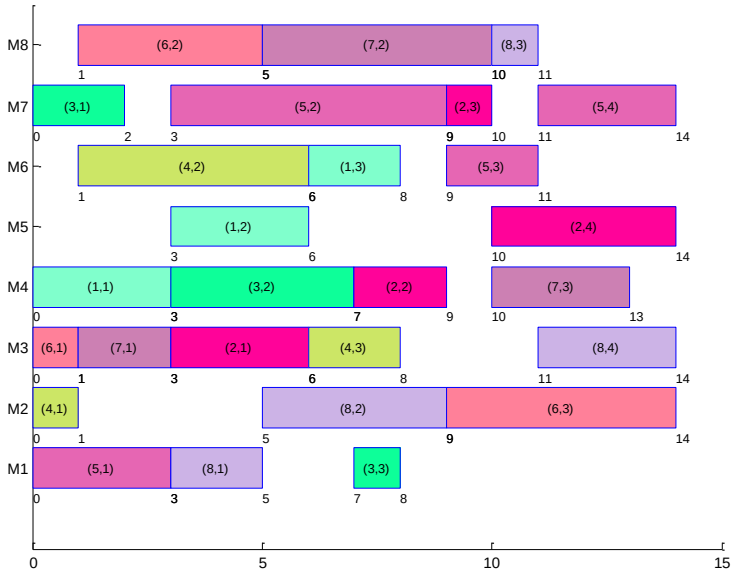

Fig. 11. The obtained optimal solution of instance 2 (8 jobs/27 operations /8 machines: F_1(c) =14, F_2(c) =77 F_3(c) =12)

Table 3. Comparison of results on problem 8×8 with 27 operations

	AL+CGA		PSO+SA		PSO+TS		HTSA		
c_{M}	15	16	15	16	14	15	14	16	15
w_{T}	79	75	75	73	77	75	77	73	75
w_{M}	13	13	12	13	12	12	12	13	12

It can be seen from Table 3 that the VNS+GA algorithm dominate the AL+CGA algorithm. The hybrid algorithm can obtain richer optimal solutions than both the PSO+TS algorithm and the PSO+SA algorithm. Fig. 11 shows the obtained Solution 1 in the form of Gantt chart.

Problem 10×7

This is an instance of total flexibility, in which 10 jobs with 29 operations are to be performed on 7 machines. The obtained solutions by our hybrid algorithm are characterized by the following values:

Solution 1: c_{M} =11, w_{T} =62, w_{M} =10

Solution 2: c_{M} =11, w_{T} =61, w_{M} =11

Fig. 12 shows the obtained Solution 2 in the form of Gantt chart.

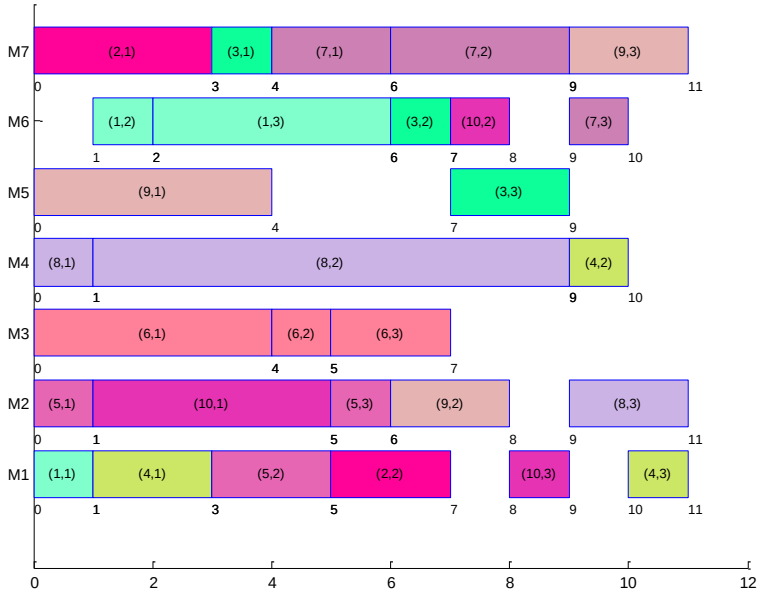

Fig. 12. The obtained optimal solution of instance 3 (10 jobs/29 operations /7 machines: $F_1(c)$ =11, $F_2(c)$ =61 $F_3(c)$ =11)

Problem 10×10

This is an instance of total flexibility, in which 10 jobs with 30 operations are to be performed on 10 machines. The obtained solutions by our hybrid algorithm are characterized by the following values:

Solution 1: c_{M} =7, w_{T} =43, w_{M} =5

Solution 2: c_{M} =7, w_{T} =42, w_{M} =6

Solution 3: c_M =8, w_T =42, w_M =5

It can be seen from Table 4 that the VNS+GA algorithm dominate the above three algorithms and can also obtain richer optimal solutions. Fig. 13 shows the obtained Solution 2 in the form of Gantt chart.

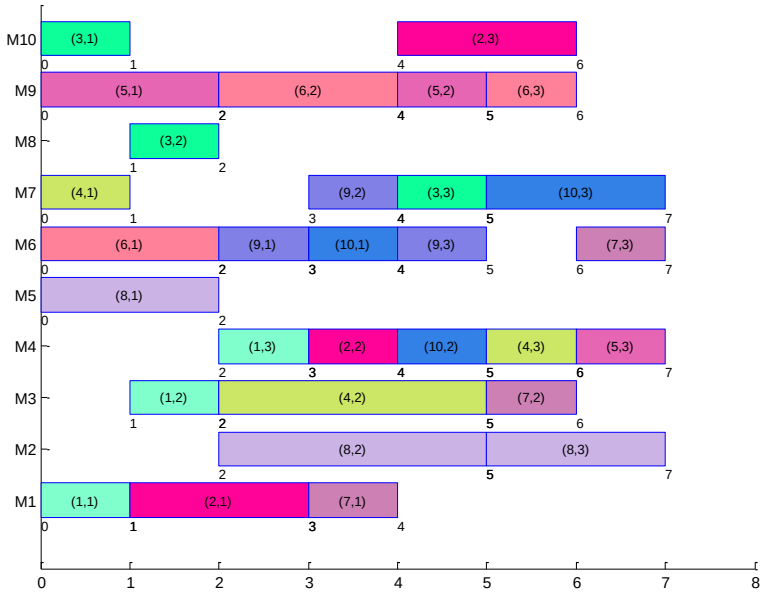

Fig. 13. The obtained optimal solution of instance 4 (10 jobs/30 operations/10 machines: F_1(c) =7, F_2(c) =42 F_3(c) =6)

Table 4. Comparison of results on problem 10 × 10 with 30 operations

	AL+CGA	PSO+SA	PSO+TS	HTSA		
c_M	7	7	7	7	8	7
w_T	45	44	43	42	42	43
w_M	5	6	6	6	5	5

Problem 15×10

This is an instance of total flexibility, in which 15 jobs with 56 operations are to be performed on 10 machines. The obtained solutions by our hybrid algorithm

are characterized by the following values:

Solution 1: c_M =11, w_T =92, w_M =11

Solution 2: c_M =12, w_T =91, w_M =11

It can be seen from Table 5 that the VNS+GA algorithm dominate both the AL+CGA and the PSO+TS algorithms and can also obtain richer optimal solutions than the PSO+SA algorithm. Fig. 14 shows the obtained Solution 2 in the form of Gantt chart.

Table 5. Comparison of results on problem 15 × 10 with 56 operations

	AL+CGA		PSO+SA	PSO+TS	HTSA	
c_M	23	24	12	11	11	12
w_T	95	91	91	93	92	91
w_M	11	11	11	11	11	11

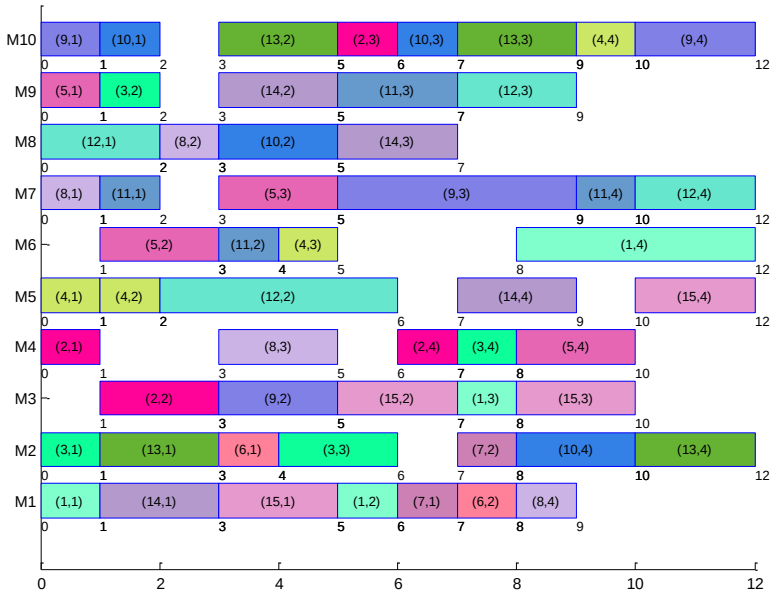

Fig. 14. The obtained optimal solution of instance 5 (15 jobs/56 operations/10 machines: F_1(c) =12, F_2(c) =91 F_3(c) =11)

From the above comparison with other three existing algorithm for solving the five Kacem instances, we can conclude that our algorithm either obtain superior solutions or can obtain richer non-dominate solutions than the other

approaches, especially for solving large scale instances.

7. Conclusions

Flexible job shop scheduling problem is very important in both fields of combinatorial optimization and engineering management. Most literature focus on proposing hybrid algorithms for solving mono-objective FJSPs. The research on the multi-objective FJSP is much less than the mono-objective FJSP. Kacem proposed a hybrid algorithm named AL+CGA combining GA and approach of localization. Xia and Wu presented a hybrid algorithm named PSO+SA which making use of PSO for solving the assignment sub problem and SA for solving the routing sub problem. Zhang et al. developed a hybrid algorithm named PSO+TS for the multi-objective FJSPs with the same three objectives. In this paper, we introduce a novel algorithm named VNS+GA combining VNS and GA for solving the multi-objective FJSPs to minimize the makespan, the total workload, and the workload of the busiest machine. There are mainly three contributions in the hybrid algorithm. Firstly, a mix of two machine assignment rules and two operation sequencing rules are developed in the initialization stage to produce enough high quality initial solutions. Secondly, an adaptive mutation rules are introduced for considering both population diversity and convergence speed in perturbation in the machine assignment component. Thirdly, a speed-up variable neighbor search operator based on public critical block theory was investigated. The new local searching approach makes the search space dwindled deeply and produces high quality neighbor solutions in very short time. Experimental results compared with the three existing algorithms (i.e., AL+CGA algorithm, PSO+SA algorithm and PSO+TS algorithm) show that our hybrid algorithm can either obtain superior solutions or obtain richer non-dominated solutions than the other algorithms, especially for larger scale instances.

The future work is to extend the initial solution rules and the public critical block method for solving other combinatorial problems. In addition, we will develop other heuristic algorithms with the public critical block neighborhood structure for solving the multi-objective FJSPs.

Acknowledgments. This research is partially supported by National Science Foundation of China under Grants 60874075, 70871065, Open Research Foundation from State Key Laboratory of Digital Manufacturing Equipment and Technology (Huazhong University of Science and Technology), Science Research and Development of Provincial Department of Public Education of Shandong under Grant J08LJ20, J09LG29, J08LJ59, and Soft Science Foundation of Shandong under Grant 2009RKB125.

References

1. Jain A.S., Meeran S.: Deterministic job-shop scheduling: Past, present and future, European Journal of Operation Research, Vol. 113, No. 2, 390-434. (1998)
2. Garey M.R., Johnson D.S., Sethi R.: The Complexity of Flowshop and Job shop Scheduling, Mathematics of Operations Research, Vol. 1, No. 2, 117-129. (1976)
3. Bruker P., Schlie R.: Job-shop scheduling with multi-purpose machines, Computing, Vol. 45, No. 4, 369-375. (1990)
4. Brandimarte P.: Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research, Vol. 22, 158-183. (1993)
5. 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, Vol. 60, No.3, 245-276. (2002a)
6. Gao L., Peng C.Y., Zhou C., Li P.G.: Solving flexible job shop scheduling problem using general particle swarm optimization, In Proceedings of the 36th CIE Conference on Computers & Industrial Engineering, Kaohsiung, Taiwan, 3018-3027. (2006)
7. Saidi-mehrabad M., Fattahi P.: Flexible job shop scheduling with tabu search algorithms, International Journal of Advanced Manufacturing Technology, Vol. 32, No. 5-6, 563-570. (2007)
8. Li J.Q., Pan Q.K., Xie S.X., Li H., Jia B.X., Zhao C.S.: An effective hybrid particle swarm optimization algorithm for flexible job-shop scheduling problem. MASAUM Journal of Computing, Vol. 1, No. 1, 69-74. (2009)
9. Kacem I., Hammadi S., Borne P.: Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems, IEEE Transactions on Systems, Man and Cybernetics, Part C, Vol. 32, No. 1, 408-419. (2002b)
10. Xia W.J., Wu Z.M.: An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Computers and Industrial Engineering, Vol. 48, No. 2, 409-425. (2005)
11. Zhang G.H., Shao X.Y., Li P.G., Gao L.: An effective hybrid swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, Computers and Industrial Engineering, Vol. 56, No. 4, 1309-1318. (2009)
12. Goldberg D.E., Holland J.H.: Genetic Algorithms and Machine Learning, Machine Learning, Vol. 3, No. 2-3, 95-99. (1988)
13. Wang L.: Intelligent Optimization Algorithms with Applications. Tsinghua University Press, Beijing, 2001.
14. Pezzella F., Morganti G., Ciaschetti G.: A genetic algorithm for the Flexible Job-shop Scheduling Problem. Computers & Operations Research, Vol. 35, No. 10, 3202-3212. (2008)
15. Adams J., Balas E., Zawack D.: The shifting bottleneck procedure for Job Shop Scheduling. Management Science, Vol. 34, No. 3, 391–401. (1988)
16. Zhang C.Y., Li P.G., Guan Z.L., Rao Y.Q.: A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers & Operations Research, Vol. 34, No. 11, 3229-3242. (2007)
17. Van Laarhoven P.J.M., Aarts E.H.L., Lenstra J.K.: Job shop scheduling by simulated annealing. Operations Research, Vol. 40, No. 1, 113-125. (1992)
18. Dell'Amico A.M., Trubian A.M.: Applying Tabu Search to the Job-shop Scheduling Problem. Annals of Operation Research, Vol. 41, No. 3, 231-252. (1993)
19. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the job-shop problem. Management Science, Vol. 42, No. 6, 797-813. (1996)

20. Balas E., Vazacopoulos A.: Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. Management Science, Vol. 44, No. 2, 262-275. (1998)

Jun-qing Li was born in Liaocheng, China, in 1976. He received the Master degree of computer science and technology in 2004 from Shandong Economic University, Shandong, China. He is currently an associate professor in Department of Computer Science and Technology, Liaocheng University, Liaocheng, China. He is now a member of IEEE and CCF. His research is related to the evolutionary optimization methods for discrete events systems, job shop systems, operational research, computer network, peer-to-peer systems and network security. He has (co-)authored around 50 research papers published. He has been serving on editorial boards or reviewers of two international journals. He has been the member of program committees of many international conferences.

Quan-ke Pan was born in Liaocheng, China, in 1971. He received the PhD of manufacturing and automation in 2003 from Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include computational intelligent, operational research and swarm optimization algorithms. He is currently a professor in Department of Computer Science and Technology, Liaocheng University, Liaocheng, China. He has (co-)authored around 120 research papers published. He has been serving on editorial boards of several international journals and has edited special issues in international journals. He has been member of program committees of many international conferences.

Sheng-xian Xie was born in Liaocheng, China, in 1957. He is currently a professor in Department of Computer Science and Technology, Liaocheng University, Liaocheng, China, and he is the chair of the department. His research interests include network security, access control and peer-to-peer systems.

Received: August 08, 2009; Accepted: February 04, 2010.

Table 1 Local minimum processing time rule (**MS_b**)

	M_1	M_2	M_3	M_4		M_1	M_2	M_3	M_4		M_1	M_2	M_3	M_4		M_1	M_2	M_3	M_4
O_{11}	7	6	4	5		7	6	**4**	5		**7**	6	4	5		7	6	**4**	5
O_{12}	4	8	5	6		4	8	**9**	6		**4**	8	5	6		**4**	8	5	6
O_{13}	9	5	4	7		9	5	**8**	7		**13**	5	4	7		9	**5**	4	7
O_{21}	2	5	1	3		2	5	**5**	3		**6**	5	1	3		2	5	**1**	3
O_{22}	4	6	8	4		4	6	**12**	4		**8**	6	8	4		4	6	8	**4**
O_{23}	9	7	2	2		9	7	**6**	2		**13**	7	2	2		9	7	2	**2**
O_{31}	8	6	3	5		8	6	**7**	5		**12**	6	3	5		8	**6**	3	5
O_{32}	3	5	8	3		3	5	**12**	3		**7**	5	8	3		**3**	5	8	3