A Novel Variable Neighborhood Particle Swarm Optimization for Multi-objective Flexible Job-shop Scheduling Problems

Hongbo Liu[†,§], Ajith Abraham[‡,†], Crina Grosan [‡], Ningning Li[§]
[†]School of Computer Science, Dalian Maritime University, Dalian 116026, China
[‡]Centre for Quantifiable Quality of Service in Communication Systems,
Faculty of Information Technology, Mathematics and Electrical Engineering,
Norwegian University of Science and Technology, N-7491 Trondheim, Norway.
[§]Department of Computer Science, Dalian University of Technology, Dalian 116023, China
lhb@dlut.edu.cn, ajith.abraham@ieee.org, liningning1982@gmail.com

Abstract

This paper introduces a hybrid metaheuristic, the Variable Neighborhood Particle Swarm Optimization (VNPSO), consisting of a combination of the Variable Neighborhood Search (VNS) and Particle Swarm Optimization (PSO). The proposed VNPSO method is used for solving the multi-objective Flexible Job-shop Scheduling Problems (FJSP). The details of implementation for the multi-objective FJSP and the corresponding computational experiments are reported. The results indicate that the proposed algorithm is an efficient approach for the multi-objective FJSP, especially for large scale problems.

1 Introduction

Flexible Job-shop Scheduling Problems (FJSP) is an extension of the classical JSP which allows an operation to be processed by any machine from a given set. It incorporates all the difficulties and complexities of its predecessor JSP and is more complex than JSP because of the additional need to determine the assignment of operations to the machines. The job shop is flexible, i.e. there are multiple job routes. The scheduling problem of a FJSP consists of a routing sub-problem, that is, assigning each operation to a machine out of a set of capable machines and the scheduling sub-problem, which consists of sequencing the assigned operations on all machines in order to obtain a feasible schedule minimizing a predefined objective function. It is quite difficult to achieve an optimal solution with traditional optimization approaches owing to the high computational complexity. In the literature, different approaches have been proposed to solve this problem. Mastrolilli and Gambardella [1] proposed some neighborhood functions for FJSP. Ong *et al*. [2] applied the clonal selection principle of the human immune system to solve FJSP with recirculation. Brandimarte [3], Saidi-Mehrabad and Fattahi [4] presented a tabu search algorithm that solves the flexible job shop scheduling problem to minimize the makespan. Liouane *et al*. [5] proposed a hybrid algorithm based on ant systems and local search optimization for FJSP. More researchers attempt to solve the FJSP using genetic algorithms [6, 7]. Ho *et al*. [8] proposed an architecture for learning and evolving of Flexible Job-Shop schedules to improve the computational time and quality of schedules. In many real-world FJSP, it is often necessary to optimize several criteria [9]. Minimization of makespan, lateness, tardiness, flow time, machine idle time, and such others are unusual the important criteria in the problems. Kacem *et al*. [10, 11] studied on modeling genetic algorithms for FJSP. Because of the intractable nature of the problem and its importance in both fields of production management and combinatorial optimization, it is desirable to explore other avenues for developing good heuristic algorithms for the problem.

Particle Swarm Optimization (PSO) incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the intelligence is emerged [12, 13]. It has become the new focus of research recently [14, 15, 16, 17]. However, its performance deteriorates as the dimensionality of the search space increases, especially for the multi-objective FJSP involving large scale. PSO often demonstrates faster convergence speed in the first phase of the search, and then slows down or even stops as the number of generations is increased. Once the algorithm slows down, it is difficult to achieve better scheduling solutions. By hybridizing particle swarm optimization and simulated annealing, Xia and Wu [18] developed an

hybrid approach for the multi-objective flexible job-shop scheduling problem (MFJSP). We introduce a novel hybrid meta-heuristic, the Variable Neighborhood Particle Swarm Optimization (VNPSO) for the multi-objective FJSP. The basic idea is to drive those particles by a shaking strategy and get them to explore variable neighborhood spaces for the better scheduling solutions.

2 Problem formulation

We focus on flexible job-shop scheduling problems composed of the following elements:

- Jobs. $J = \{J_1, \cdots, J_n\}$ is a set of n jobs to be scheduled. Each job J_i consists of a predetermined sequence of operations. $O_{i,j}$ is the operation j of J_i. All jobs are released at time 0.
- Machines. $M = \{M_1, \cdots, M_m\}$ is a set of m machines. Each machine can process only one operation at a time. And each operation can be processed without interruption during its performance on one of the set of machines. All machines are available at time 0.
- Flexible. MFJSP usually is classified into two types as follows:
  - Total FJSP (T-FJSP): each operation can be processed on any machine of M.
  - Partial FJSP (P-FJSP): each operation can be processed on one machine of subset of M.
- Constraints. The constraints are rules that limit the possible assignments of the operations. They can be divided mainly into the following situations:
  - Each operation can be processed by only one machine at a time (disjunctive constraint).
  - Each operation, which has started, runs to completion (non-preemption condition).
  - Each machine performs operations one after another (capacity constraint).
  - Although there are no precedence constraints among operations of different jobs, the predetermined sequence of operation for each job forces each operation to be scheduled after all predecessor operations (precedence/conjunctive constraint).
  - the machine constraints emphasize the operations can be processed only by the machine from the given set (resource constraint).
- Objective(s). Most of the research reported in the literature is focused on the single objective case of the problem, in which the objective is to find a schedule that has minimum time required to complete all operations (minimum makespan). Some other objectives, such as flow time or tardiness are also important like the makespan. Currently it has been paid more attentions to investigate the problem from a multiobjective perspective. It is desirable to generate many near-optimal schedules considering multiple objectives.

To formulate the objective, define $C_{i,j,k}$ ($i = 1, 2, \cdots, n$; $j = 1, 2, \cdots, p$; $k = 1, 2, \cdots, m$) as the completion time that the machine M_k finishes the j-th operation $O_{i,j}$ of job i; $\sum C_k$ represents the time that the machine M_k completes the processing of all the jobs. Define $C_{sum} = \sum_{k=1}^{m}(\sum C_k)$ as the flowtime, and $C_{max} = max\{\sum C_k\}$ as the makespan. The problem is thus to both determine an assignment and a sequence of the operations on all machines that minimize the criteria:

- The sum of the completion times (flowtime): C_{sum}.
- the maximum completion time (makespan): C_{max}.

Minimizing C_{sum} asks the average job finishes quickly, at the expense of the largest job taking a long time, whereas minimizing C_{max}, asks that no job takes too long, at the expense of most jobs taking a long time. Minimization of C_{max} would result in maximization of C_{sum}. The weighted aggregation is the most common approach to the problems. According to this approach, the objectives, $f_1 = \min\{C_{sum}\}$ and $f_2 = \min\{C_{max}\}$, are summed to a weighted combination:

$$F = \min(w_1 f_1 + w_2 \lambda f_2) \tag{1}$$

where λ is the scaling factor, which is the average number of machines per operation; w_1 and w_2 are non-negative weights, and $w_1 + w_2 = 1$. These weights can be either fixed or adapt dynamically during the optimization. The dynamic weighted aggregation [19] was used in the paper. Alternatively, the weights can be changed gradually according to the Eqs. (2) and (3). The variation for different values of w_1 and w_2 ($R = 200$) are illustrated in Fig. 1.

$$w_1(t) = |sin(2\pi t/R)| \tag{2}$$

$$w_2(t) = 1 - w_1(t) \tag{3}$$

Definition 1 *A flexible job-shop scheduling problem can be defined as* $\prod = (J, O, M, F)$. *The key components are jobs, operations and machines. For the sake of simplify, the scheduling problem also be represented in triple* $P = (J, O, M)$.

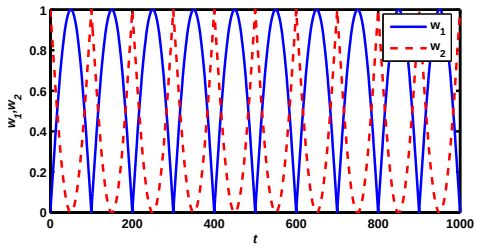

Figure 1. Dynamic weight variation.

Let $A(t)$ be the set of operations being processed at time t, and let $r_{i,j,k} = 1$ if operation j of job i is assigned on machine k to be processed and $r_{i,j,k} = 0$ otherwise. Let $d_{i,j}$ denote the duration (processing time) of operation j of job i. The conceptual model of the MFJSP can be formulated the following way:

$$\textbf{Minimize} \quad F(C_{max}, C_{sum}) \tag{4}$$

subject to :

$$C_{i,j,k} \leq C_{i,j+1,k} - d_{i,j+1}. \quad j = 1, \cdots, p-1. \tag{5}$$

$$\sum_{j \in A(t)} r_{i,j,k} \leq 1, \quad k \in M; t \geq 0. \tag{6}$$

$$C_{i,j,k} \geq 0, \quad j = 1, \cdots, n. \tag{7}$$

The complexity of FJSP increases with the number of constraints imposed and the size of search space employed [20]. Except for some highly restricted special cases, very simple special cases of FJSP are already strongly NP-hard. For the standard FJSP, the size of search space is $(n!)^m$, and for this reason, it is computationally infeasible to try every possible solution. This is because the required computation time increases exponentially with the problem size. In practice, many real-world FJSPs have a larger number of jobs and machines as well as additional constraints and flexibilities, which further increase its complexity. For the same number of machines and jobs, the P-FJSP is more difficult to solve than the T-FJSP. Therefore, the P-FJSP is transformed to the T-FJSP by adding 'infinite processing times' to the unused machines and to solve the latter instead in [10]. However, although the P-FJSP is a generalization of the T-FJSP, Ho *et al*. illustrated the distinguish between the problem types of T-FJSP and P-FJSP [8].

3 PSOs for FJSP

3.1 SPSO and its Convergence

The standard PSO model consists of a swarm of particles, which are initialized with a population of random candidate solutions. They move iteratively through the d-dimensional problem space to search the new solutions, where the fitness, f, can be calculated as the certain qualities measure. Each particle has a position represented by a position-vector $\vec{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{x}_i^{\#}$, and its j-th dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\vec{x}^*$, and its j-th dimensional value is x_j^*. During the iteration time t, the update of the velocity from the previous velocity to the new velocity is determined by Eq.(8). The new position is then determined by the sum of the previous position and the new velocity by Eq.(9).

$$\begin{aligned} v_{ij}(t) =& w v_{ij}(t-1) + c_1 r_1 (x_{ij}^{\#}(t-1) - x_{ij}(t-1)) \\ &+ c_2 r_2 (x_j^*(t-1) - x_{ij}(t-1)) \end{aligned} \tag{8}$$

$$x_{ij}(t) = x_{ij}(t-1) + v_{ij}(t) \tag{9}$$

$$v_{i,j} = sign(v_{i,j}) min(|v_{i,j}|, v_{max}) \tag{10}$$

$$x_{i,j} = sign(x_{i,j}) min(|x_{i,j}|, x_{max}) \tag{11}$$

where r_1 and r_2 are the random numbers in the interval [0,1]. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as coefficient of the social component. The variable w is called as the inertia factor, which value is typically setup to vary linearly from 1 to near 0 during the iterated processing. From Eq.(8), a particle decides where to move next, considering its current state, its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. The pseudo-code for SPSO is illustrated in Algorithm 1. The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region which is defined by each individual's previous best success and the success of some other particle. Bergh and Engelbrecht [21] overviewed the theoretical studies, and extend these studies to investigate particle trajectories for general swarms to include the influence of the inertia term. They also provided a formal proof that each particle converges to a stable point. Eberhart and Kennedy called the two basic methods as "gbest model" and "lbest model" [12]. In the lbest model, particles have information only of their own and their nearest array neighbors' best, rather than that of the entire group. Unfortunately there is a large computational cost to explore the neighborhood relation in each iteration. In the gbest model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Some previous studies has been shown that the trajectories of the particles

oscillate in different sinusoidal waves and converge quickly in the "gbest model" algorithm [22, 23]. During the iteration, the particle is attracted towards the location of the best fitness achieved so far by the particle itself and by the location of the best fitness achieved so far across the whole swarm. The "gbest model" has faster convergence. But very often for multi-modal problems involving high dimensions it tends to suffer from premature convergence [24, 25].

Algorithm 1 Particle Swarm Algorithm

```
01. Begin
02.   Parameter settings and initialize swarm
03.   Evaluation
04.   Locate leader
05.   g = 1
06.   While (the end criterion is not met) do
07.       For each particle
08.           Update velocity
09.           Update position
10.           Evaluation
11.           Update pbest
12.       EndFor
13.       Update leader
14.   g++
15.   End While
16. End
```

3.2 VNPSO

The aim of Variable Neighbourhood Search approaches is to avoid poor local optima by systematically changing neighbourhood which increases its size iteratively until a local minimum is better than the current one is found. These steps are repeated until a given termination condition is met. Hansen and Mladenović suggest various VNS strategies for a wide range of combinatorial optimization problems [26, 27]. The major difference between our approach and the VNS approach proposed by Hansen and Mladenović lies in the change of neighbourhoods by perturbing the particles effectively. The metaheuristic method we proposed in [28], the VNPSO, was originally inspired by VNS. In PSO, if a particle's velocity decreases to a threshold v_c, a new velocity is assigned using Eq.(12):

$$\begin{aligned} v_{ij}(t) =& w\hat{v} + c_1 r_1 (x_{ij}^{\#}(t-1) - x_{ij}(t-1)) \\ &+ c_2 r_2 (x_j^{*}(t-1) - x_{ij}(t-1)) \end{aligned} \tag{12}$$

$$\hat{v} = \begin{cases} v_{ij} & \text{if } |v_{ij}| \geq v_c \\ rand(-1,1) v_{max}/\eta & \text{if } |v_{ij}| < v_c \end{cases} \tag{13}$$

where $rand(-1,1)$ is the random number, normal distributed with the interval [-1,1], and η is the variable neighborhood scaling factor to control the domain of the particle's oscillation according to v_{max}. v_c is the minimum velocity threshold, a tunable threshold parameter to limit the minimum of the particles' velocity. Our algorithm scheme is summarized as Algorithm 2. The performance of the algorithm is directly correlated to two parameter values, v_c and ρ. A large v_c shortens the oscillation period, and it provides a great probability for the particles to leap over local minima using the same number of iterations. But a large v_c compels the particles in the quick "flying" state, which leads them not to search the solution and forcing them not to refine the search. The value of ρ changes directly the variable search neighborhoods for the particles. We also implemented the Multi-Start PSO (MSPSO), illustrated in Algorithm 3, to compare their performances.

Algorithm 2 Variable Neighborhood Particle Swarm Optimization

```
01. Initialize the size of the particle swarm n,
01.   and other parameters.
02. Initialize the positions and the velocities for all
02.   the particles randomly.
03. Set the flag of iterations without
03.   improvement Nohope = 0.
04. While (the end criterion is not met) do
05.   g = 1;
06.   Calculate the fitness value of each particle;
07.   x*(vec) = argmin_{i=1}^{n}(f(x*(vec)(t-1)), f(x_1(vec)(t)),
07.     f(x_2(vec)(t)), ..., f(x_i(vec)(t)), ..., f(x_n(vec)(t)));
08.   If x*(vec) is improved then Nohope = 0,
08.     else Nohope = Nohope + 1.
09.   For i= 1 to n
10.     x_i^{#}(vec)(t) = argmin_{i=1}^{n}(f(x_i^{#}(vec)(t-1)), f(x_i(vec)(t));
11.     For j = 1 to d
12.       If Nohope < 10 then
13.         Update the j-th dimension value of x_i(vec) and v_i(vec)
13.             according to Eqs.(8),(10),(9),(11);
14.       else
15.         Update the j-th dimension value of x_i(vec) and v_i(vec)
15.             according to Eqs.(13),(12),(9),(11).
16.     Next j
17.   Next i
18.   g++
19. End While.
```

3.3 Encoding Representations

Encoding representation can be extremely important when trying to find solutions to a problem in a heuristic or metaheuristic algorithm. The data structures, such as the chromosome, particle position, plus the algorithm combine

to make efficient programs. Better efficiency of search can be achieved by modifying the encoding representation and its related operators so as to generate feasible solutions and avoiding the use of a repair mechanism. A bad encoding representation can increase the size of the search space or slow down the algorithm if too many repair operators are needed to ensure the representation is valid.

Algorithm 3 Multi-start Particle Swarm Optimization

01. Initialize the size of the particle swarm n,
01. and other parameters.
02. Initialize the positions and the velocities for all
02. the particles randomly.
03. Set the flag of iterations without
03. improvement $Nohope = 0$.
04. While (the end criterion is not met) do
05. $g = 1$;
06. Calculate the fitness value of each particle;
07. $\vec{x}^* = argmin_{i=1}^{n}(f(\vec{x}^*(t-1)), f(\vec{x}_1(t)),$
07. $f(\vec{x}_2(t)), \cdots, f(\vec{x}_i(t)), \cdots, f(\vec{x}_n(t)))$;
08. If $\vec{x}^*$ is improved then $Nohope = 0$,
08. else $Nohope = Nohope + 1$.
09. For i= 1 to n
10. $\vec{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\vec{x}_i^{\#}(t-1)), f(\vec{x}_i(t))$;
11. For j = 1 to d
12. If $Nohope < 10$ then
13. Update the j-th dimension value of $\vec{x}_i$ and $\vec{v}_i$
13. according to Eqs.(8),(10),(9),(11);
14. else
15. Re-initialize the positions and the velocities
15. for all the particles randomly.
16. Next j
17. Next i
18. g++
19. End While.

Cheng *et al*. [6], Kleeman and Lamont [9] introduced a taxonomy of how Evolutionary Algorithms (EA) represent job-shop problems. These representations can be classified as either directly encoded approaches or indirectly coded approaches. With a direct approach, a schedule is encoded into the chromosome. The EA then operates on these schedules in an effort to find the best schedule. For direct approaches, there are five different ways the EA can be encoded:

- Operation-based
- Job-based
- Job pair relation-based
- Completion time-based
- Random keys

Indirect approaches are chromosome representations that do not directly encode the schedule into the chromosome. There are four indirect approaches:

- Preference list-based
- Priority rule-based
- Disjunctive graph-based
- Machine-based

For encoding representations, we have to consider time and space computational complexity, the need to maintain solution feasibility. To solve FJSP, there are the three important factors:

- Flexibility
- Length
- Availability

There are fixed sequences for operations in each job (precedence constraints). But there are no precedence constraints among operations of different jobs. If the job and the operations are ordered beforehand, there is not enough flexible between jobs. Gen *et al*. [29] proposed a perfect method: they name all operations for a job with the same symbol (for example, the corresponding job index) and then interpret them according to the order of occurrences in the sequence of a given chromosome. All permutations of the chromosome yield a valid schedule. The chromosome length is $\sum_{i=1}^{n} p_i$, where n is the number of jobs and p_i is the number of operations in job i. It would be confronted with another difficulty: how to check effectively which machine can process the assigning operations when it deals with P-FJSP. For the same number of machines and jobs, the Kacem *et al*. [10] transformed the P-FJSP to the T-FJSP by adding 'infinite processing times' to the unused machines. Some individuals would be evaluated to 'infinite'. This increases the overall time complexity due to the presence of redundant assignments. Ho *et al*. [8] proposed a new chromosomal representation, which has two components: operation order and machine selection. Operation order component is similar to Gen's method. Each individual is obtained from this schedule by replacing each operation by the corresponding job index. By reading the data from left to right and increasing operation index of each job, a feasible schedule is always obtained. The machine selection component consists of a chromosome of size $\sum_{i=1}^{n} p_i$. Each allele of the chromosome is a sub-chromosome, which lists the preference which machine would process the operation. For the problem in Table 1, one possible encoding is shown in Fig. 2. This method inherits the advantages of both the operation-based chromosome representation and the preference list-based representation. The chromosome

Table 1. An example of the P-FJSP.

		M_1	M_2	M_3
J_1	$O_{1,1}$	4	5	XXX
	$O_{1,2}$	9	2	2
	$O_{1,3}$	XXX	6	3
J_2	$O_{2,1}$	6	5	XXX
	$O_{2,2}$	3	3	5

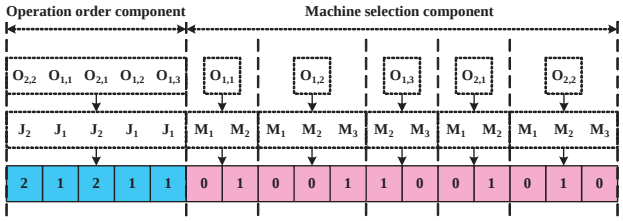

Figure 2. Operation-order-machine-selection representation.

length is $\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} p_i * q_j$, where n is the number of jobs, p_i is the number of operations in job i, q_j is the number of machines which operation $O_{i,j}$ can be assigned on. This chromosome representation has to consider the availability of machines that process operations so that the decoding processing reduces the search space size. But it is possible that one operation is assigned on more than one machine after crossover and mutation operators. Therefore, a repair mechanism to maintain feasibility is required. In addition, this representation is complex and redundant for the T-FJSP, since the machine selection component seems too long.

In our PSO algorithms, the position representation of the particles also has two components: operation order and machine selection. But it is with a variable length strategy. The first part, operation order component provides the order of operations, and all operations for a job is signed with the corresponding job index. The second part, machine selection component is variable length according to the problems. If the average number of machines per operation is larger than the half of number of machines, we encode each dimension with a random number in the interval $[1, m+1)$. Each dimension of the particle's position maps one operation, and the value of the position indicates the machine number to which this task/operation is assigned during the course of particle swarm algorithm. So the value of a particle's position should be integer. But after updating the velocity and position of the particles, the particle's position may appear real values such as 1.4, etc. It is meaningless for the assignment. Therefore, in the algorithm we usually round off the real optimum value to its nearest integer number. In this part, the sequence of the operations will be not changed during the iteration. The feasible different sequence schedule of the operations between different jobs comes from the first part. If the average number of machines per operation is less than or equal to the half of number of machines, we extend it as Fig. 2. The variable length representation allows the algorithm to maintain a balance between the flexibility of FJSP and search space, and to converge on the better results effectively.

4 Experiment Settings, Results and Discussions

To illustrate the effectiveness and performance of the proposed algorithm, three representative instances based on practical data have been selected. Three problem instances $((J8, O27, M8)$, $(J10, O30, M10)$ and $(J15, O56, M10)$ are taken from Kacem *et al*. [10, 11, 30]. In our experiments, the algorithms used for comparison were MSPSO (Multi-start PSO) and VNPSO (Variable Neighborhood PSO). In VNPSO, ρ and v_c were set to 2 and 1e-7 before 15,000 iterations, while they were set to 5 and 1e-10 after 15,000 iterations. Other parameter settings are shown in table 2. The average fitness values of the best solutions throughout the optimization run were recorded. The averages (F) and the standard deviations (std) were calculated from the 10 different trials. The standard deviation indicates the differences in the results during the 10 different trials. Usually another emphasis will be to generate the schedules at a minimal amount of time. So the completion time for 10 trials were used as one of the criteria to improve their performance. Figs. 3, 4 and 5 illustrate the performance for the three algorithms during the search processes for the three FJSPs. Empirical results are illustrated in Table 3. In general, VNPSO performs better than another approaches, although its computational time is worse than VTPSO for the low dimension problem, $(J8, O27, M8)$. VNPSO could be an ideal approach for solving the large scale problems when other algorithms failed to give a better solution.

5 Conclusions

In this paper, we introduced a hybrid metaheuristic, the Variable Neighborhood Particle Swarm Optimization (VNPSO), consisting of a combination of the Variable Neighborhood Search (VNS) and Particle Swarm Optimization(PSO), and considered its application for solving the multi-objective Flexible Job-shop Scheduling Problems (FJSP). The details of implementation for the multi-objective FJSP are provided and its performance

Table 2. Parameter settings for the algorithms.

Algorithm	Parameter name	Value
	Size of the population	20
MSGA	Probability of crossover	0.9
	Probability of mutation	0.09
	Swarm size	20
	Inertia weight w	0.7
PSOs	Self coefficient c_1	$(w+1)^2/2$
	Social coefficient c_2	$(w+1)^2/2$
	Clamping Coefficient ρ	0.5

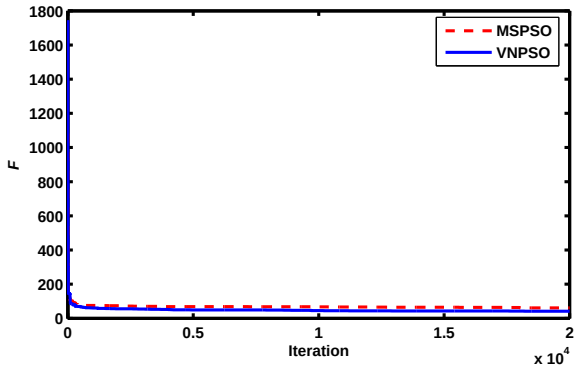

Figure 3. The performance of the algorithms for $(J8, O27, M8)$ FJSP.

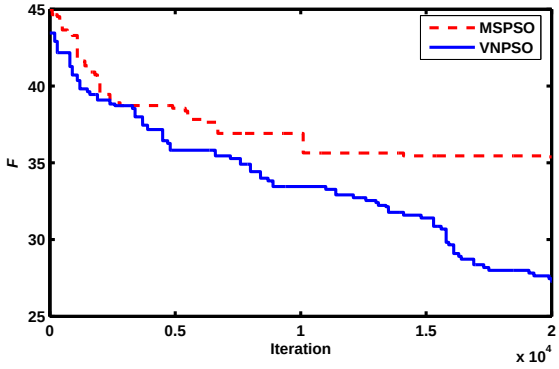

Figure 4. The performance of the algorithms for $(J10, O30, M10)$ FJSP.

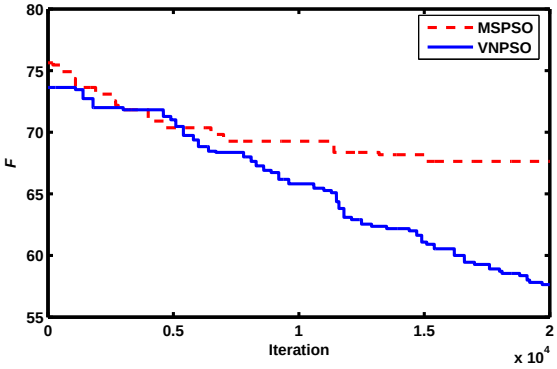

Figure 5. The performance of the algorithms for $(J15, O56, M10)$ FJSP.

Table 3. Comparing the results for FJSPs.

Instance	MSPSO	VNPSO
$(8, 27, 8)$	61.0661 $\pm$4.8259	41.1429 $\pm$4.4263
$(10, 30, 10)$	35.3840 $\pm$3.2902	27.2727 $\pm$3.3195
$(15, 56, 10)$	67.6364 $\pm$1.8778	57.6364 $\pm$5.6886

was compared using computational experiments. The empirical results have shown that the proposed algorithm is an available and effective approach for the multi-objective FJSP, especially for large scale problems.

Acknowledgments

We would like to thank Drs. Ran He and Bo Li for their scientific collaboration in this research work. This work is supported partly by NSFC (60573087) and MOE (06JJDXLX001).

References

[1] M. Mastrolilli and L. M. Gambardella. "Effective neighborhood functions for the flexible job shop problem". *Journal of Scheduling*, 2002, 3(1), pp. 3–20.

[2] Z. X. Ong , J. C. Tay and C. K. Kwoh. "Applying the Clonal Selection Principle to Find Flexible Job-Shop Schedules". *Lecture Notes in Computer Science*, 2005, 3627, pp. 442–455.

[3] P. Brandimarte. "Routing and Scheduling in a Flexible Job-Shop by Tabu Search". *Annals of Operations Research*, 1993, 2, pp. 158–183.

[4] M. Saidi-Mehrabad and P. Fattahi. "Flexible job shop scheduling with tabu search algorithms". *International Journal of Adv Manuf Technol*, 2007, 32, pp. 563–570.

[5] N. Liouane, I. Saad, S. Hammadi and P. Borne. "Ant systems & Local Search Optimization for flexible Job Shop Schedul-

ing Production". *International Journal of Computers, Communications & Control*, 2007, 2(2), pp. 174–184.

[6] R. Cheng, M. Gen and Y. Tsujimura. "A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation". *International Journal of Computers and Industrial Engineering* 1996, 30, pp. 983–997.

[7] R. Cheng, M. Gen and Y. Tsujimura. "A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies". *International Journal of Computers and Industrial Engineering*, 1996, 36, pp. 343–364.

[8] N. B. Ho, J. C. Tay and E. Lai. "An effective architecture for learning and evolving flexible job-shop schedules". *European Journal of Operational Research*, 2007, 179, pp. 316–333.

[9] M. P. Kleeman and G. B. Lamont. "Scheduling of Flow-Shop, Job-Shop, and Combined Scheduling Problems using MOEAs with Fixed and Variable Length Chromosomes". *Studies in Computational Intelligence*, 2007, 49, pp. 49–99.

[10] I. Kacem, S. Hammadi and P. Borne. "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems". *IEEE Transactions on Systems, Man and Cybernetics*, 2002, 32(1), pp. 1–13.

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

[12] J. Kennedy, and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, CA, 2001.

[13] M. Clerc. *Particle Swarm Optimization*. ISTE Publishing Company, London, 2006.

[14] A. Abraham, H. Guo and H. Liu. "Swarm intelligence: foundations, perspectives and applications". In: *Swarm Intelligent Systems, Studies in Computational Intelligence*, N. Nedjah, L. Mourelle (eds.), Springer, 2006, pp. 3–25.

[15] A. Abraham, H. Liu and Zhang W.: "Scheduling job on computational Grids using fuzzy particle swarm algorithm". *Lecture Notes in Artificial Intelligence*, 4252, 2006, pp. 500–507.

[16] C. Grosan, A. Abraham and M. Nicoara. "Search optimization using hybrid particle sub-swarms and evolutionary algorithms". *International Journal of Simulation Systems, Science & Technology*, 2005, 6(10-11), pp. 60–79.

[17] C. Grosan. "Multiobjective 0/1 Knapsack Problem using Adaptive ϵ -Dominance". *Lecture Notes in Computer Science*, 2006, pp. 547–556.

[18] W. Xia and Z. Wu. "An Effective Hybrid Optimization Approach for Multi-objective Flexible Job-shop Scheduling Problems". *Computers and Industrial Engineering*, 2005, 48, pp. 409–425.

[19] Parsopoulos K. E. and Vrahatis M. N.: "Recent Approaches to Global Optimization Problems through Particle Swarm Optimization". *Natural Computing*, 2002, 1, pp. 235–306.

[20] K. Ripon, C. Tsang and S. Kwong. "An Evolutionary Approach for Solving the Multi-Objective Job-Shop Scheduling Problem". *Studies in Computational Intelligence*, 2007, 49, 165–195.

[21] F. van den Bergh and A.P. Engelbrecht. "A study of particle swarm optimization particle trajectories". *Information Sciences*, 2006, 176, pp. 937–971.

[22] M. Clerc and J. Kennedy. "The Particle Swarm-explosion, Stability, and Convergence in A Multidimensional Complex Space". *IEEE Transactions on Evolutionary Computation*, 2002, 6, pp. 58–73.

[23] T. I. Cristian "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection". *Information Processing Letters*, 2003, 85(6), pp. 317–325.

[24] H. Liu, B. Li, Y. Ji and T. Sun. "Particle Swarm Optimisation from lbest to gbest". *Applied Soft Computing Technologies: The Challenge of Complexity*, A. Abraham, B. D. Baets, M. Köppen, B. Nickolay (Eds.), Springer, 2006, pp. 537–545.

[25] H. Liu, A. Abraham and M. Clerc. "Chaotic Dynamic Characteristics in Swarm Intelligence". *Applied Soft Computing Journal*, 2007, 7, pp.1019–1026.

[26] P. Hansen and N. Mladenović. "Variable neighbourhood search: Principles and applications". *European Journal of Operations Research*, 2001, 130, pp. 449–467.

[27] P. Hansen, N. Mladenović. "Variable Neighborhood Search". *Handbook of Metaheuristics*, F. V. Glover, G. A. Kochenberger (eds.), Kluwer Academic Publishers, 2003, pp. 145–184.

[28] H. Liu and A. Abraham. "An Hybrid Fuzzy Variable Neighborhood Particle Swarm Optimization Algorithm for Solving Quadratic Assignment Problems". *Journal of Universal Computer Science*, Vol. 13, 2007 (to appear)

[29] M. Gen , Y. Tsujimura and E. Kubota. "Solving job-shop scheduling problem using genetic algorithms". *Proceedings of the 16th international conference on computer and industrial engineering*, 1994, pp. 576–579.

[30] Taillard, E.D. "Benchmarks for Basic Scheduling Problems". *European Journal of Operational Research*, 1993, 64(2), pp. 278–285.