International Journal of Management Perspective
Vol.1, No.1
pp. 69-95

A novel particle swarm optimization algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times

M. Amiri[1,*]
N. Javadian[2]
H. Zare[2]

Abstract

Much of the research on operations scheduling problems has either ignored setup times or assumed that setup times on each machine are independent of the job sequence. This paper deals with the hybrid flow shop scheduling problems in which there are sequence dependent setup times, commonly known as the SDST hybrid flow shops. This type of production system is found in industries such as chemical, textile, metallurgical, printed circuit board, and automobile manufacture. With the increase in manufacturing complexity, conventional scheduling techniques for generating a reasonable manufacturing schedule have become ineffective. A particle swarm optimization algorithm can be used to tackle complex problems and produce a reasonable manufacturing schedule within an acceptable time. This paper describes a novel particle swarm optimization algorithm approach to the scheduling of a SDST hybrid flow shop. An overview of the hybrid flow shops and the basic notions of a PSO are first presented. Subsequently, the details of a NPSO approach are described and implemented. The results obtained are compared with those computed by Random Key Genetic Algorithm presented previously.

Key words: Short-term scheduling; Hybrid flow shops; Sequence dependent setup times; Makespan; Particle Swarm Optimization

1. Department of Industrial Management, Management and Accounting Faculty, Allameh Tabatabaeii University, Tehran, Iran.
2. Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran.
* Corresponding author.
E- mail address: mg_amiri@yahoo.com

1. Introduction

Several flow patterns can be encountered, depending on the number of stages (*v*) required to process a job and on the number of available machines per stage (*M(v)*). The diagram in Fig. 1 illustrates schematically the relationships between the different machine environments (Zandieh and Fatemi, 2003).

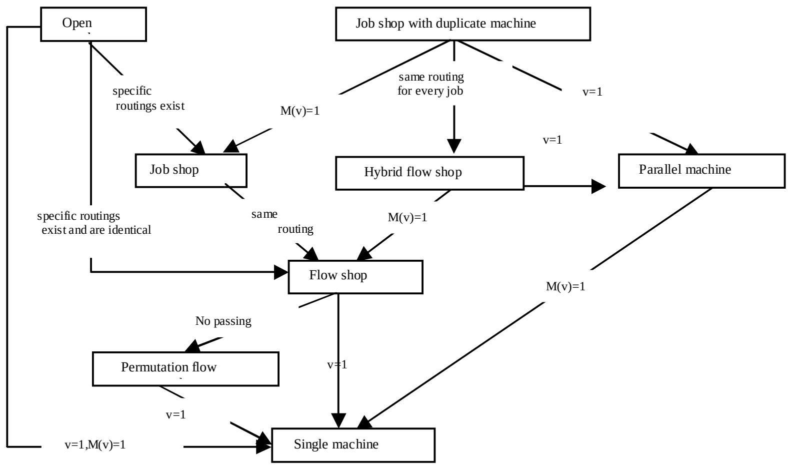


Fig. 1. A classification for scheduling problems based on resource environments.

A hybrid flow shop model, commonly known as flexible flow line, allows us to represent most of the production systems. The process industry such as chemical, pharmaceutical, oil, food, tobacco, textile, paper, and metallurgical industry can be modeled as a hybrid flow shop. In the literature, the notion of hybrid flow shop has emerged in the 70s (Arthanary and Ramaswamy, 1971). A hybrid flow shop consists of a series of production stages, each of which has several facilities in parallel (Elmaghraby and Karnoub, 1995). Some stages may have only one facility, but for the plant to be qualified as a hybrid flow shop, at least one stage must have several facilities. The flow of products in the plant is unidirectional. Each product is processed at only one facility in each stage and at one or more stages before it exits the plant. Each stage may have multiple parallel identical machines. These machines can be identical, uniform, or unrelated. Each job is processed by at most one machine at each stage.

Pinedo (1995) cited machine setup time is a significant factor for production scheduling in all flow patterns, and it may easily consume more than 20% of available machine capacity if not well handled. Also the completion time of production and machine setups are influenced by production mix and production sequence. On the one hand, processing in large batches may increase machine utilization and reduce the total setup time. On the other hand, large batch processing increases the flow time. Scheduling problems with sequence-dependent setup times are among the most difficult classes of scheduling problems. A single-machine sequence-dependent setup scheduling problem is equivalent to a traveling-salesman problem and is NP-hard (Pinedo 1995). Even for a small system, the complexity of this problem is beyond the reach of existing theories (Luh et al,1998).

Sequence-dependent setup scheduling of a hybrid flow shop system is even more challenging. Although there has been some progress reported, but the understanding of sequence-dependent setups, however, is still believed to be far from being complete (Luh et al,1998).

In recent years, with the emergence of computational intelligence, intelligence-oriented algorithms such as GA, SA, TS, and etc. have been employed to scheduling problems.

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Eberhart and Kennedy in (1995), inspired by social behavior of bird flocking or fish schooling. Similar to genetic algorithm (GA), PSO is a population based optimization tool. Original PSO is distinctly different from other evolutionary-type methods in a way that it does not use the filtering operation (such as crossover and mutation) and the members of the entire population are maintained through the search procedure so that information is socially shared among individuals to direct the search towards the best position in the search space. Clerc and Kennedy (2002) researched on the explosion stability and convergence in a multi-dimensional complex space of the particle swarm and Trelea (2003) studied convergence analysis and parameter selection of the particle swarm optimization algorithm. Eberhart and Shi (1998) compared genetic algorithms with particle swarm optimization. In recent years there have been a lot of reported works focused on the modification PSO such as Fan (2002), Kennedy and Mendes (2002), and Shi et al. (2005) to solve continuous optimization problems, but its being used to solve HFSSP does not have rich literatures.

In this paper, a particle swarm optimization evolutionary algorithm (PSOEA) is proposed to SDST hybrid flow shop problems. The paper has the following structure. Section 2 gives literature review of SDST hybrid flow shop scheduling. Section 3 is problem description. Section 4 introduces the proposed novel particle swarm optimization (NPSO) algorithm. Section 5 presents experimental design which compared the results achieved by proposed NPSO algorithm with those achieved by past genetic algorithms. Finally, section 6 consist conclusions and future work.

2. Literature review

Gupta and Tunc (1994) presented four heuristic algorithms to minimize makespan for a two stage hybrid flow shop problem with separable setup and removal times. In which, sequencing of jobs can be done using one of Sule's (1982) rule or Szwarc and Gupta's algorithm(1987) while assigning jobs to multiple machines at the second stage is done by attempting to minimize the job-waiting time at the second stage.

Robust local search improvement techniques for flexible flow-line scheduling were considered by Leon and Ramamoorthy (1997). They considered neighborhoods of problem data, using ideas from Storer et al (1992). Lee et al (1997) have applied genetic algorithms to the joint problem of determining lot sizes and sequence to minimize make span in flexible flow lines. Though this research included sequence-dependent setup times, buffers between stages were limited and a permutation schedule was required. Combining genetic algorithms with simulated annealing was also considered.

Kochhar and Morris (1987) model flexible flow lines in a more complete manner in that they allow for setups between jobs, finite buffers which may cause blocking and starvation, machine down-time, and current and subsequent state of the system. They extend a Wittrock (1985, 1988) algorithm and evaluate several policies with a deterministic simulation. Sawik (1992) has developed numerous results for the flexible flow-line scheduling problem. His basic model includes factors such as transportation time between stages and nonzero release times. However, sequence-dependent setup times are not included. Later, Sawik (1994 and 1995) extended his heuristic to the case of no buffers between stages.

Srikar and Ghosh (1986) considered a permutation flow shop with sequence-dependent setup times in their MILP model, which used many fewer variables than the previous models. Srikar and Ghosh (1986) used decision variables that focused on whether a job is scheduled any time before another job. However, Stafford and Tseng (1990) discovered several problems with Srikar and Ghosh (1986), corrected these and extended this modeling concept to non-sequence dependent setup time flow shops, no-intermediate-queue flow shops and sequence-dependent setup time, no intermediate-queue flow shops. Rios-Mercado and Bard (1998) also considered the sequence-dependent setup time flow shop and developed several valid inequalities for models based on the traveling salesman problem and the Srikar-Ghosh model.

Hung and Ching (2003) addressed a scheduling problem taken from a label sticker manufacturing company which is a two-stage hybrid flow shop with the characteristics of sequence-dependent setup time at stage 1, dedicated machines at stage 2, and two due dates. The objective was to schedule one day's mix of label stickers through the shop such that the weighted maximal tardiness is minimized. They proposed a heuristic to find the near-optimal schedule for the problem. The performance of the heuristic was evaluated comparing its solution with both the optimal solution for small-sized problems and the solution obtained by the scheduling method used in the shop.

While many papers have been written in the area of scheduling hybrid and flexible flow lines, many of them are restricted to special cases of two stages, specific configurations of machines at stages, and to simplify the problem, setups are seldom considered in the scheduling. For those ones addressing setups, the setup times are fixed and included in processing times. However, in most real world cases, the length of the setup time depends on both jobs, which is separable from processing. There seems to be published only three works addressing heuristics for flexible flow lines with sequence dependent setup times. Kurz and Askin (2003) examined scheduling rules for SDST flexible flow lines. They explored three classes of heuristics. The first class of heuristics (cyclic heuristics) is based on simplistic assignment of jobs to machines with little or no regard for the setup times. The second class of heuristics is based on the insertion heuristic for the traveling salesman problem (TSP). The third class of heuristics is based on Johnson's Rule. Note that the second class caters to setup aspects of the problem while the third derives from standard flow shops. They proposed eight heuristics (CH, RCH, SPTCH, FTMIH, CTMIH, MMIH, 1,g Johnson's Rule, g/2,g/2 Johnson's Rule) and compared the performance of those on a set of test problems. Moreover, Kurz and Askin (2004) formulated the SDST flexible flow lines as an integer programming model. Because of the difficulty in solving the IP model directly, they developed a Random Keys Genetic Algorithm (RKGA). Problem data was generated to evaluate the RKGA with other scheduling heuristics rules, which they proposed aforetime. They created a lower bound to evaluate the heuristics. Zandieh et al. (2006) proposed an immune algorithm, and showed that this algorithm outperforms the random keys genetic algorithm of Kurz and Askin (2004).

3. Problem description

Let *g* be the number of workshops in series. Let *n* be the number of jobs to be processed and m^t be the number of machines in parallel at each stage *t*. We assume that machines are initially setup for a nominal job 0 at every stage. Job *n*+1 exists at every stage only to indicate the end of the process, if needed. We have the following definitions.

p_i^t = processing times for job *i* at stage *t*

s_{ij}^t = sequence dependent setup time from job *i* to job *j* at stage *t*

$\tilde{p}_i^t$ = modified processing times for job *i* at stage *t* ($\tilde{p}_i^t = p_i^t + \min_j s_{ij}^t$)

S^t = set of jobs that visit workshop stage *t*

The processing time of job 0 is set at 0. The setup time from job 0 indicates the time to move from the nominal set point state. We assume that all jobs currently in the system must be completed at each stage before the jobs under consideration may begin setup. The completion times of job 0 at each stage are set to the earliest setup time may begin at that stage. The setup time for job *n*+1 is set at 0; this job only exists to indicate the end of the schedule. We also include the restriction that every stage must be visited by at least as many jobs as there are machines in that stage.

4. The novel particle swarm optimization (NPSO) algorithm

PSO is an evolutionary algorithm which is initialized with a population (named swarm in PSO) of random solutions and searches for optima by updating generations. Each individual or potential solution, named particle, flies in the dimensional problem space with a velocity which is dynamically adjusted according to the flying experiences of its own and its colleagues. The PSO algorithm mimics the behavior of flying birds and their means of information exchange to solve optimization problems. It has been introduced as an optimization technique in real-number spaces, but many optimization problems are set in a discrete space. Typical examples include problems that require ordering, such as HFSSP. In this section we introduce a novel PSO algorithm for HFSSP to minimize *makespan*.

4.1. Original particle swarm optimization algorithm

Suppose that the searching space is D dimensional and m particles form the colony. The ith particle represents a dimensional vector X_i (i = 1, 2, . . . , m). It means that the ith particle locates at $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ (i = 1, 2, . . . , m) in the searching space. The position of each particle is a potential result. We could calculate the particle's fitness by putting its position into a designated objective function. The ith particle's "flying" velocity is also a D dimensional vector, denoted as $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. Denote the best position of the ith particle as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$, and the best position of the colony as $P_g = (p_{g1}, p_{g2}, \ldots, p_{gD})$ respectively. The original PSO algorithm could be performed by the following equations:

$$v_{id}(k+1) = v_{id}(k) + c_1 r_1 (p_{id}(k) - x_{id}(k)) + c_2 r_2 (p_{gd}(k) - x_{id}(k)) \qquad (4.1)$$
$$x_{id}(k+1) = x_{id}(k) + v_{id}(k+1) \qquad (4.2)$$

where k represents the iterative number, c_1, c_2 are learning factors, usually $c_1 = c_2 = 2$. r_1, r_2 are random numbers between (0, 1). The termination criterion for the iterations is determined according to whether the max generation or a designated value of the fitness of P_g is reached (Kennedy and Eberhart 1995).

4.2. The NPSO algorithm for HFSSP to minimize makespan

HFSSP is set in a discrete space, so the most important issue in applying PSO successfully to HFSSP is to develop an effective 'problem mapping' and 'solution generation' mechanism. If these two mechanisms are devised successfully, its possible to find good solutions for a given optimization problem in acceptable time. According to the character of HFSSP, we will design the particle, particles' velocities and iterative formula of NPSO algorithm. The crossover and mutation operators will be used in the GAs and NPSO algorithm for HFSSP to minimize *makespan.*

4.2.1. Iterative model of the NPSO algorithm for HFSSP to minimize makespan

If the HFSSP is n-jobs and m-machines, suppose that the searching space is n-dimensional and s particles form the colony. The ith particle represents an n-dimensional vector $X_i (i = 1, 2, \ldots, s)$. It means that the ith particle locates at X_i which is one sequence in the searching space. The position of each particle is a potential result. We could calculate the particle's fitness by putting its position into a designated objective function. When the fitness is lower, the corresponding X_i is "better". The ith particle's "flying" velocity is also an n-dimensional vector, denoted as V_i. Denote the best position of the ith particle as P_i, and the best position of the colony as P_g respectively. The NPSO algorithm could be performed by the following equations:

$$V_i(k+1) = P_i(k) \oplus P_g(k) \tag{4.3}$$

$$(v_{r1}, v_{r2}, .., v_{rN})(k+1) = M(v_{r1}, v_{r2}, .., v_{rN}) \tag{4.4}$$

$$X_i(k+1) = X_i(k) \oplus V_i(k+1) \tag{4.5}$$

$$(x_{r1}, x_{r2}, \ldots, x_{rN})(k+1) = M(x_{r1}, x_{r2}, \ldots, x_{rN}) \tag{4.6}$$

where k represents the iterative generation number, and r $(1 \le r \le psize)$ is random integer which denotes mutating particle, and $\oplus$ is crossover denotation which denotes two particles making crossover operator and its detailed operator was shown in Fig. 3,4. $M(v_r)$, $M(x_r)$ mean mutating particle v_r and x_r whose detailed operator was expressed in Fig. 5,6, $N \in [psize/4, psize/2]$ denotes mutating particle numbers in every generation. The termination criterion for the iterations is determined according to whether the max generation or a designated value of the fitness of P_g is reached.

4.2.2. Step of the NPSO algorithm for HFSSP to minimize makespan

Step 1. Let initialization iterative generation be $k = 0$, initialization population size *psize*, The termination iterative generation *Maxgen*. Give birth to *psize* initializing particles as following:

Randomly generate an initial population of (*psize* - 3) particles, generate a particle with SPTCH, generate a particle with FTMIH, generate a particle with g/2, g/2 -Johnson's rule. Calculate each particle's fitness value of initialization population, and let first generation P_i be initialization particles, and choose the particle with the best fitness value of all the particles as the P_g (gBest).

Step 2. Every P_i (k) and P_g (k) crossover can get two child particles, compare them and let smaller fitness value particle be final child of predecessors. Using (4.3) obtains "flying" velocity V_i particles, then utilizing (4.4) randomly mutating N particles of them. And using (4.5) and (4.6) with the same method gives birth to the next generation particles X_i. If the fitness value is better than the best fitness value P_i (pBest) in history, let current value as the new P_i (pBest). Choose the particle with the best fitness value of all the particles as the P_g (gBest). If k= *Maxgen*, go to Step 3, or else let $k = k + 1$; go to Step 2.

Step 3. Put out the P_g.

We can learn that there are two key steps when applying NPSO algorithm to HFSSP: the representation of the solution and the fitness function. The searching is a repeat process, and the stop criterion is that the maximum iteration number is reached. In NPSO algorithm, each particle of the swarm shares mutual information globally and benefits from the discoveries and previous experiences of all other colleagues during the search process. NPSO algorithm requires only primitive and simple mathematical operators, and is computationally inexpensive in terms of both memory requirements and time.

4.2.3. Recognition of particle

Particle recognition here refers to the creation of a representation scheme to denote solutions as a number sequence for NPSO operators to operate on. In this case, a candidate solution is represented by a random key representation. The advantage of this representation is its ease of implementation. This representation was proposed by Norman and Bean (1999) to avoid infeasible solution. They used the following solution representation for an identical multiple machine problem. Each job is assigned a real number whose integer part is the machine number to which the job is assigned and whose fractional part is used to sort the jobs assigned to each machine. For an example consider a problem with five jobs (n = 5), two processes (g = 2), two machines at stage one (m^1 =

2), and three machines at stage two ($m^2 = 3$). For this problem we must generate five random numbers from uniform distribution $[1,1 + m^t]$ in each stage of process (Fig. 2).

As shown in Fig. 2, each of the blocks denotes a process. For example, the first set of numbers of block 1 implies that in process 1, job 1, job 3, and job 4 are assigned to machine 1; also job 2 and job 5 are assigned to machine 2. The order of jobs to be scheduled on machine 1 is job 1 followed by job 3 and then job 4, and the order of jobs to be scheduled on machine 2 is job 5 followed by job 2.

Process 1					Process 2				
1.41	2.13	1.23	1.65	2.01	3.27	1.19	1.87	2.61	3.05

Fig. 2. Representation of candidate solution in NPSO

4.2.4. Generation of initial particle

Usually, initial particles are randomly generated in the feasible space, but initial particles can influence the convergence time. Because of this, we incorporated some known heuristics into initialization to generate three well-adapted initial particles. Remained initial particles are randomly generated.

Kurz and Askin (2003, 2004) proposed three heuristics based on greedy methods, flow line methods and the Insertion Heuristic for the TSP. These heuristics were named SPTCH, FTMIH, and g/2, g/2 Johnson's rule.

In the SPT Cyclic Heuristic (SPTCH), the jobs are ordered at stage 1 in increasing order of the modified processing times $\tilde{p}_i^1$. At subsequent stages, jobs are assigned in earliest ready time order. Jobs are assigned to the machine in every stage that allows it to complete at the earliest time. The SPTCH has the following steps for each stage *t*:

1. Create the modified processing times $\tilde{p}_i^1$.
2. Order the jobs in non-decreasing order (SPT) of $\tilde{p}_i^1$.
3. At each stage $t = 1, 2, \dots, g$, assign job 0 to each machine in that stage.
4. For stage 1:
 (a) Let *bestmc* = 1.
 (b) For $[i] = 1$ to n, $i \in S^1$:
For *mc* = 1 to m^1:

Place job [*i*] last on machine *mc*.
Find the completion time of job [*i*]. If this time is less on *mc* than on *bestmc*,
let *bestmc* = *mc*.
Assign job [*i*] to the last position on machine *bestmc*.
5. for each stage $t = 2, \ldots, g$:
(a) Update the ready times in stage t to be the completion times in stage $t - 1$.
(b) Arrange jobs in increasing order of ready times.
(c) Let *bestmc* = 1.
(d) For $[i] = 1$ to n, $i \in S^t$:
For $mc = 1$ to m^t:
Place job [*i*] last on machine *mc*.
Find the completion time of job [*i*]. If this time is less on *mc* than on *bestmc*,
let *bestmc* = *mc*.
Assign job [*i*] to the last position on machine *bestmc*.

The FTMIH is a multiple insertion heuristic to minimize the sum of flow times (completion-ready times) at each stage. It is a multiple machine, multiple stage adaptation of the Insertion Heuristic for the TSP. Setup times are accounted for by integrating their values into the processing times using $\tilde{p}_i^t$. The FTMIH can then be performed using these modified processing times at each stage. Once jobs have been assigned to machines, the true processing and setup times can be used. The FTMIH has the following steps for each stage t:

1. Create the modified processing times.
2. Order the jobs in non-increasing order (LPT) of $\tilde{p}_i^t$.
3. For $[i] = 1$ to n, $i \in S^t$:

(a) Insert job [*i*] into every position on each machine.
(b) Calculate the true sum of flow times using the actual setup times.
(c) Place job *i* in the position on the machine with the lowest resultant sum of flow times.
4. Update the ready times in stage $t + 1$ to be the completion times in stage t.

Johnson's rule (1954) finds the optimal makespan solution for $F/2//C\max$. The $g/2$, $g/2$ Johnson's rule is an extension of Johnson's rule to take into account the setup times for the flow shop with more than two stages. The aggregated first half of the stages and the aggregated last half of the stages are considered to create the order for assignment in stage 1. The value $\tilde{p}_i^1$ is the sum of modified processing times for stages 1 to $\left\lceil \frac{g}{2} \right\rceil$ and $\tilde{p}_i^g$ is the sum over stages $\left\lceil \frac{g}{2} \right\rceil + 1$ to *g*.

1. Create the modified processing times $\tilde{p}_i^1$ and $\tilde{p}_i^g$.
2. Let $U = \left\{ j \middle| \tilde{p}_j^1 < \tilde{p}_j^g \right\}$ and $V = \left\{ j \middle| \tilde{p}_j^1 \geq \tilde{p}_j^g \right\}$.
3. Arrange jobs in *U* in non-decreasing order of $\tilde{p}_i^1$ and arrange jobs in *V* in non-increasing order of $\tilde{p}_i^g$.

   Append the ordered list *V* to the end of *U*.

4. At each stage *t* = 1, 2 ,. . . , *g*, assign job 0 to each machine in that stage.
5. For [*i*] = 1 to *n*, $i \in S^1$:
(a) For *mc* = 1 to m^1:
Place job [*i*] last on machine *mc*.
If this placement results in the lowest completion time for job [*i*], let *m* = *mc*.
(b) Place job [*i*] last on machine *m*.
6. for each stage *t* = 2,. . . ,*g*:
(a) Update the ready times in stage *t* to be the completion times in stage *t* - 1.
(b) Arrange jobs in increasing order of ready times.
(c) For [*i*] = 1 to *n*, $i \in S^t$:
(1) For *mc* = 1 to m^t :
Place job [*i*] last on machine *mc*.
If this placement results in the lowest completion time for job [*i*], let *m* = *mc*.
(2) Place job [*i*] last on machine *m*.

4.2.5. Crossover and mutation operators [75]

Crossover operators:

One-segment crossover (C_1): A pair of crossing points is randomly selected along the length of the first predecessor chromosome. The jobs inside crossing points are copied into the offspring. The remaining places of the offspring are filled up by taking in order each legitimate gene from the second predecessor. This operator is illustrated in Fig. 3 with crossing points at 4 and 10 of the string.

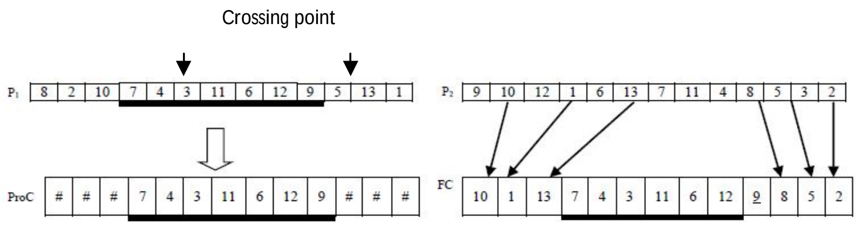
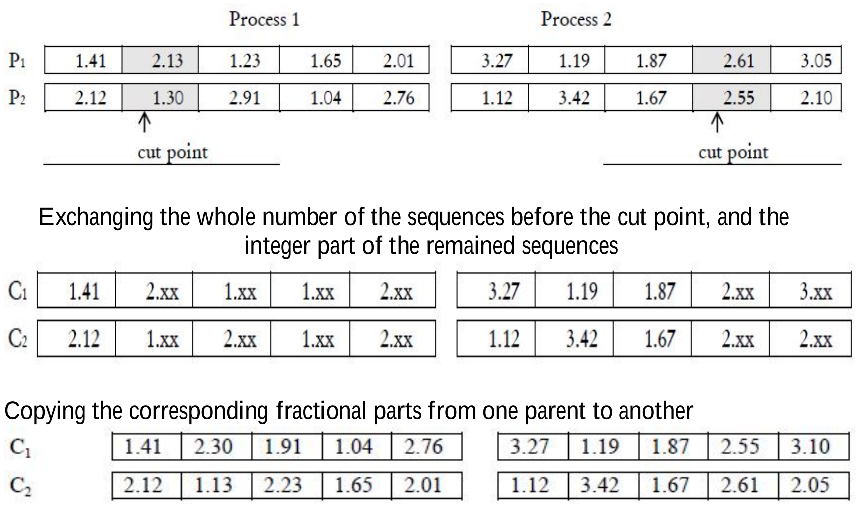


Fig. 3. Illustration of the one-segment crossover operator (C_1).

Order Crossover (C_2): The well-known Traveling Salesman Problems (TSPs) are frequently used to model and solve manufacturing scheduling (Khoo, Lee, and Yin 2000). Basically, a TSP is a class of problem where a salesman visits each of his destinations once and only once. Michalewicz (1994) showed that crossover operators such as the Partially Mapped Crossover (PMX), the Order Crossover (OX) and the Cycle Crossover (CX), can be used to handle TSPs. Among them, the OX method appears to be the most promising and has been adopted in this work. Essentially, the OX method generates a clone (or offspring) by retaining a part of a sequence from the other parent. In this work, the OX method is needed to be modified to accommodate the aforementioned representation scheme that is used.

The modified OX method is best explained using two processes, five jobs, two machines in process 1, and three machines in process 2 example. Consider two parents, P_1 and P_2, with a randomly generated cut point for each process. By exchanging the whole number of the sequences before the cut point, and the integer part of the remained sequences, two clones (or offspring) can be generated as Fig. 4, where *.xx* represents the fractional part that is currently unknown. Starting from the cut point of one parent, the corresponding fractional parts from the other parent are copied one at a time.

The crossover operator has now generated two clones (or offsprings) from the two parents. It can be seen that the clones (or offsprings) share a lot of properties with the parents. Using the modified OX method as the crossover operator, only the job order is allowed to vary. Machine assignment, on the other hand, can not be changed during a crossover operation



	Process 1					Process 2				
P1	1.41	2.13	1.23	1.65	2.01	3.27	1.19	1.87	2.61	3.05
P2	2.12	1.30	2.91	1.04	2.76	1.12	3.42	1.67	2.55	2.10

Exchanging the whole number of the sequences before the cut point, and the integer part of the remained sequences

C1	1.41	2.xx	1.xx	1.xx	2.xx	3.27	1.19	1.87	2.xx	3.xx
C2	2.12	1.xx	2.xx	1.xx	2.xx	1.12	3.42	1.67	2.xx	2.xx

Copying the corresponding fractional parts from one parent to another

C_1	1.41	2.30	1.91	1.04	2.76	3.27	1.19	1.87	2.55	3.10
C_2	2.12	1.13	2.23	1.65	2.01	1.12	3.42	1.67	2.61	2.05

Fig.4.OX crossover operator example.

Mutation operators:

One-segment move insert mutation (M_1): A moving segment and one inserting point are randomly selected along the length of the predecessor chromosome. The jobs inside segment are moved and inserted in the inserting point. This operator is illustrated in Fig. 5 with moving segment at 2–5 and inserting point at 11 of the string.

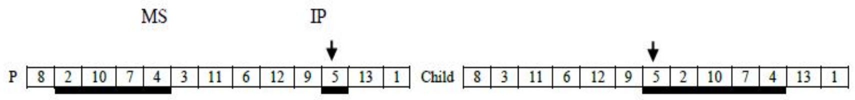
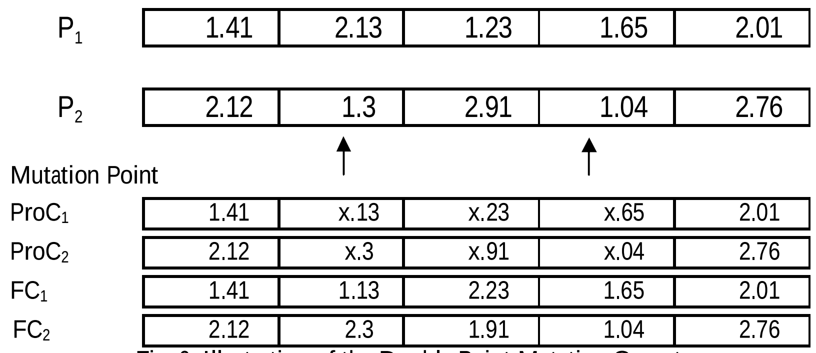


Fig. 5. Illustration of the shift mutation operator

Double point mutation operator (M_2): This mutation operator is used to alter the machine assignment for a job in a particular process. Basically, mutation begins when two random mutation points in the sequences are selected. Then, the integer part of real number of the genes between two mutation points is/are changed (Fig. 6).



P_1	1.41	2.13	1.23	1.65	2.01
P_2	2.12	1.3	2.91	1.04	2.76
Mutation Point		↑		↑	
$ProC_1$	1.41	x.13	x.23	x.65	2.01
$ProC_2$	2.12	x.3	x.91	x.04	2.76
FC_1	1.41	1.13	2.23	1.65	2.01
FC_2	2.12	2.3	1.91	1.04	2.76

Fig. 6. Illustration of the Double Point Mutation Operator

5. Experimental design

5.1. Data generation and settings

An experiment was conducted to test the performance of the NPSO algorithm. Following Kurz and Askin (2003,2004), data required for a problem consists of the number of jobs, number of stages, number of machines in each stage, range of processing times, and the range of sequence dependent setup times. The ready times for stage 1 are set to 0 for all jobs. The ready times at stage t+1 are the completion times at stage t, so there is no need this data to be generated. Processing times are distributed uniformly over two ranges with a mean of 60: [50–70] and [20–100]. Flexible flow lines are considered by allowing some jobs to skip some stages. Following Leon and Ramamoorthy (1997), the probability of skipping a stage is set at 0, 0.05, or 0.40. The setup times are uniformly distributed from 12 to 24 which are 20–40% of the mean of the processing time. The setup time matrices are asymmetric and satisfy the triangle inequality. The setup time characteristics follow Rios-Mercado and Bard (1998).

Table 1. Factor levels

Factors		Levels
Number of jobs		6-30-100
Number of stages		2-4-8
Number of Machine	Constant:	1-2-10
	Variable:	U[1-4] to U[1-10]
Processing time		U[50-70] to U[20-100]
Skipping probability		0.00-0.05-0.40

The problem data can be characterized by six factors, and each of these factors can have at least two levels. These levels are shown in Table 1.

In general, all combinations of these levels will be tested. However, some further restrictions are introduced. The variable machine distribution factor requires that at least one stage have a different number of machines than the others. Also, the largest number of machines in a stage must be less than the number of jobs. Thus, the combination with 10 machines at each stage and six jobs will be skipped and the combination of 1–10 machines per stage with six jobs will be changed to 1–6 machines per stage with six jobs. There are 252 test scenarios and five data sets are generated for each one.

5.2. NPSO algorithm parameters tuning

It is known that the different levels of the parameters clearly affect the quality of the solutions obtained by a NPSO algorithm. A number of different NPSO algorithms can be obtained with the different combinations of the parameters. We have applied parameters tuning only for the crossover (C_1, C_2), mutation (M_1, M_2), *Maxgen* value, *psize* value and *N* (mutating particle number in every generation), considering in Table 2.

Table2. NPSO parameter levels

Parameters	Number of levels	Levels
Mutation Operator	2	M_1, M_2
Crossover Operator	2	C_1, C_2
(*Maxgen*, *psize*, *N*)	6	(100, 100, 25)
		(100, 100, 50)
		(50,200,50)
		(50,200,100)
		(200,50,12)
		(200,50,25)

24 different NPSOs are obtained by these levels. In this paper, we have run the NPSO algorithm five times for five medium problems and five large ones. The metaheuristics were implemented in MATLAB 7 and run on a PC with a Pentium IV 3.00 GHz processor with 1 GB of RAM.

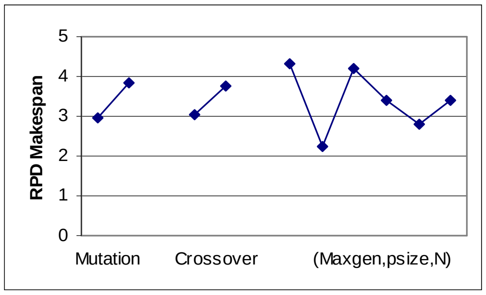


Fig. 7. Average of makespan

The results are analyzed by the means of multi-factor Analysis of Variance (ANOVA) technique. It is necessary to notice that for using ANOVA, three main hypotheses, normality, homogeneity of variance and independence of residuals, must be check. We did that and found no bias for questioning the validity of the experiment.

Fig. 7 is the Average of makespan in parameter tuning process. It is obtained from Fig. 7 that using mutation operator M_1, crossover type C_1, value 200 as *Maxgen*, value 50 as *psize* and value 12 as *N*, is the best set of parameters.

5.3. Experimental results

In this section we are going to compare the proposed NPSO algorithm with the RKGA which proposed by Kurz and Askin (2003, 2004) for the SDST flexible flow lines. The heuristics were implemented in MATLAB 7 and run on a PC with a Pentium IV 3.00 GHz processor with 1 GB of RAM.

5.3.1. Analysis of makespan and computational time

The convergence of NPSO algorithm for one middle size problem and one big problem is shown. Fig. 8 shows the convergence of NPSO algorithm of $30/4/1,1,1,1/HF/C_{max}$ and Fig. 9 shows the convergence of NPSO algorithm of $100/8/10,9,7,6,5,1,1,1/\ HF/C_{max}$.

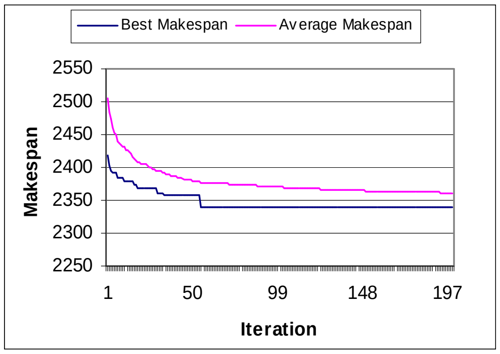


Fig. 8. Convergence of NPSO for instance ($30/4/1,1,1,1/HF/C_{max}$)

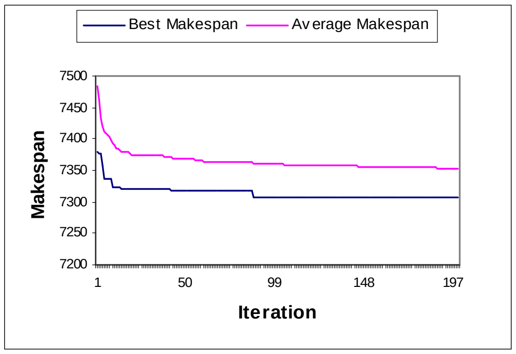


Fig. 9. Convergence of NPSO for instance ($100/8/10,9,7,6,5,1,1,1/\ HF/C_{max}$)

To evaluate the performance of each heuristic, we applied the lower bounds proposed by Kurz and Askin (2001, 2004). They proposed two lower bounds $LB^{(1)}$ and $LB^{(2)}$. For any feasible solution of SDST hybrid flow shop, they presented $LB^{(1)}$ and $LB^{(2)}$ in the following form:

$$LB^{(1)} = \max_{i=1,\dots,n}\left\{\sum_{t\in S_i}\left(\mathbf{p}_i^t + \min_{j=0,\dots,n} s_{ji}^t\right)\right\}$$

$$LB^{(2)} = \max_{t=1,\dots,g}\left\{\min_{i\in S_t}\sum_{\tau=1}^{t-1}\left(\mathbf{p}_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau\right) + \sum_{i\in S_t}\frac{1}{\mathbf{m}^t}\left(\mathbf{p}_i^t + \min_{j=0,\dots,n} s_{ji}^t\right) + \min_{i\in S_t}\sum_{\tau=t+1}^{g}\left(\mathbf{p}_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau\right) + \right.$$
$$\left.\frac{1}{m^t}\sum_{k=1}^{m^t-1}\left[\min_{i\in S_t}{}_{[k]}\sum_{\tau=1}^{t-1}\left(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau\right) - \min_{i\in S_t}\sum_{\tau=1}^{t-1}\left(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau\right)\right]\right\}$$

The notation "$\min_{[k]}$" is used to indicate the (k+1)st from the lowest value and $\min_{[0]} = \min$. For example, given a list of values {2, 5, 7, 8, 9}, $\min_{[1]} = 5$.

The first above mentioned equation is job-based bound and the second is machine-based bound. We defined "RPD" as: (makespan-lower bound)/lower bound. To calculate the RPD value, the best lower bound was used for each problem. The running times were found using the tic-toc function.

Every heuristic considered here was run on the same 1260 data sets. NPSO and RKGA were run 5 times and the minimum and average RPD over the 5 runs was found for each of the 1260 data sets.

NPSO achieves the lowest values for the RPD statistics and finds the minimum RPD schedules more frequently than the RKGA heuristic. The results within the 5 runs of NPSO and RKGA are described in Table 3.

Table 3. Average RPD Results grouped by *n* and *g* for NPSO versus RKGA

Instance	NPSO	RKGA
6×2	15.75	20
6×4	11.69	13.1
6×8	0.007	0.026
6 jobs	9.149	11.042
30×2	3.35	12.02
30×4	0.62	0.69
30×8	2.07	2.06
30 jobs	2.01	4.92
100×2	0.55	6.93
100×4	0.64	0.39
100×8	6.47	9.1
100 jobs	2.55	5.47
Average	4.57	7.15

The results indicate that there is a clearly significant difference between two algorithms. As it can be seen, NPSO provides better results in medium and large problems.

Fig. 10 shows the average RPD of both NPSO and RKGA algorithms.

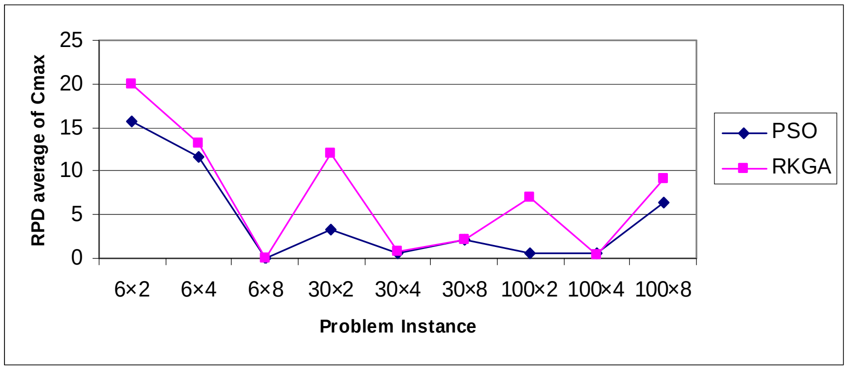


Fig. 10. Average RPD Results grouped by *n* and *g* for NPSO versus RKGA

Another aspect of comparison is to subtract the computational time of RKGA from NPSO for all of 252 problem instances, then calculate the number of decrease, similar and increase in each of the problem sizes, and divide it to total number of problems in the related size. The results are shown in Table 4, Fig. 11 and Fig. 12. This process can be applied for makespan value as shown in Table 5, Fig. 13 and Fig. 14.

Table 4. Computational time values of NPSO versus RKGA

Size of problem	*Decrease*		*Similar*		*Increase*	
	Percentage of problems	Number of problems	Percentage of problems	Number of problems	Percentage of problems	Number of problems
Small (job:6)	50%	39	0.01%	1	49.99%	38
Medium (job:30)	60%	54	-	-	40%	36
Large (job:100)	50%	45	-	-	50%	45

According to Table 4, in 50% of large problems, 60% of medium problems and 50% of small problems, the computational time of NPSO is less than the computational time of RKGAGA

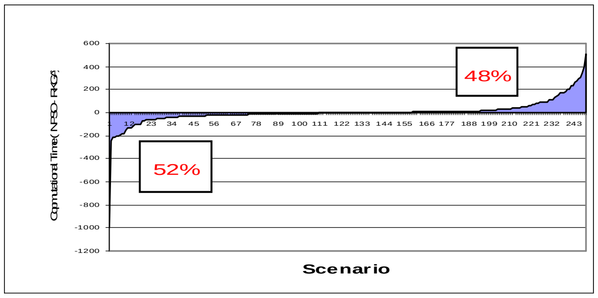


Fig. 11. Computational time differences (NPSO- RKGA) for all problems

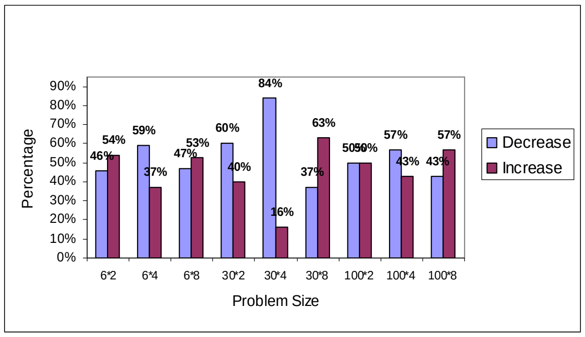


Fig. 12. Increase/Decrease of NPSO computational time versus RKGA

Table 5. Makespan values of NPSO versus RKGA

Size of problem	*Decrease*		*Similar*		*Increase*	
	Percentage of problems	Number of problems	Percentage of problems	Number of problems	Percentage of problems	Number of problems
Small(job:6)	56.7%	41	26.3%	19	17%	12
Medium(job:30)	61.3%	55	-	-	38.7%	35
Large(job:100)	46%	41	-	-	54%	49

According to Table 6, in 46% of large problems, 61.3% of medium problems and 56.7% of small problems, the makespan of NPSO is less than the makespan of RKGA.

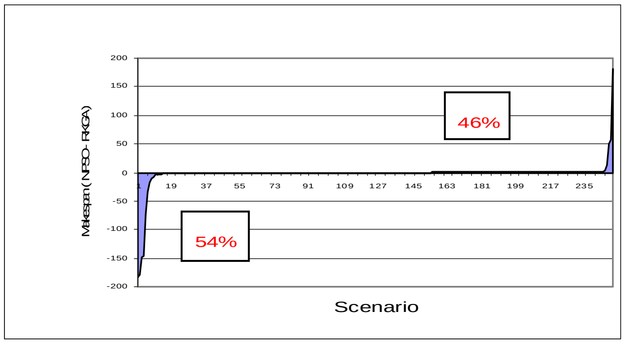


Fig. 13. Makespan differences (NPSO-RKGA) for all Problems

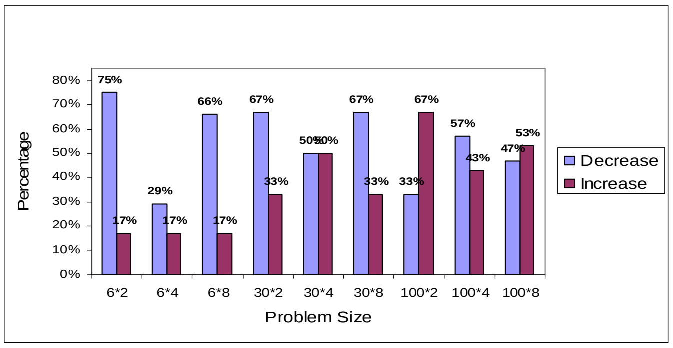


Fig. 14. Increase/Decrease of RKGA C_{max} versus NPSO C_{max}

5.3.2. Analysis of controlled factors in computational time

In order to see the effects of number of stages and number of jobs on computational time of algorithms, another ANOVA is applied. Means plot for the interaction between the factors type of algorithm and each combination of stage and job are shown in Fig. 15.

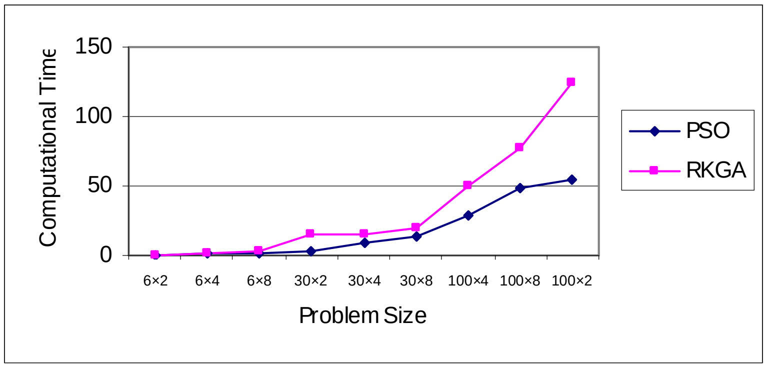


Fig. 15. Means plot for the interaction between the factors type of algorithm and each combination of stage and job

Also an ANOVA test is applied to see the effect of number of jobs on computational time of algorithms. The results are shown in Fig.16. The problem instances are classified into three categories: small problems (with 6 jobs), medium problems (with 30 jobs) and large problems (with 100 jobs). In each category, the average value of computational time is calculated.

According to Fig. 16, it is clear that by increase in number of jobs, computational time of RKGA increases by greater gradient in contrast with NPSO algorithm, i.e. in large problems, NPSO has a better performance than RKGA for purposes of computational time.

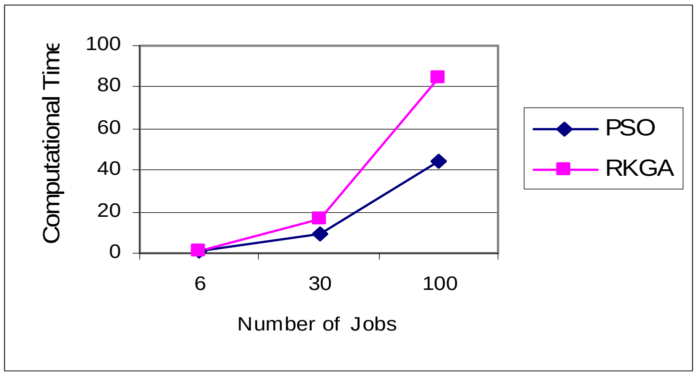


Fig. 16.Computational time of NPSO versus RKGA

6. Conclusions and future work

NPSO approach for the scheduling of a hybrid flow shop has been successfully adjusted. An experiment was carried out to illustrate the effectiveness of NPSO algorithm in scheduling. The makespan values of NPSO in many test problems, especially in medium and large problems, are less than the makespan values of GA.

There are potentially unlimited opportunities for research in scheduling to minimize makespan in hybrid flow shops with sequence-dependent setup times. In this paper, we have addressed only a few areas.

In many researches, the lower bounds are typically used to evaluate the performance of heuristics for solving combinatorial optimization problems. In the absence of tight analytical lower bounds, optimal objective function values may be estimated statistically. Extreme value theory can be used to construct confidence-interval estimates of the minimum makespan.

Also by creating a general permutation schedule definition, we may be able to find a class of schedules that contains the optimal makespan schedule for some special cases, such as two stages with one machine at the first stage and two machines at the second, with sequence-dependent setup times at both.

References

Arthanary T.S., Ramaswamy K.G. (1971). An extension of two machine sequencing problems, *Operations Research, 8*, 10–22.

Clerc M, Kennedy J. (2002). The particle swarm: explosion stability and convergence in a multi-dimensional complex space. *IEEE Trans Evolutionary Compute, 6*,1,58–73

Eberhart RC, Shi Y. (1998). Comparison between genetic algorithms and particle swarm optimization. Evolutionary programming VII: *proceedings of the seventh annual conference on evolutionary programming*. Berlin San Diego, CA: Springer-verlag, 611–6.

Elmaghraby S.E., Karnoub R.E., (1995). Production control in flexible flowshops: an example from textile manufacturing, OR Report No. 305 OR and IE Department, North Carolina State University, USA.

Fan HY. A. (2002). modification to particle swarm optimization algorithm. Eng Comput, 19(8):970–89.

Gupta J.N.D., Tunc E.A. (1994). Scheduling a two-stage hybrid flowshop with separable setup and removal times, *European Journal of Operational. Research, 77*, 415–428.

Hung T.S.L. , Ching J.L. (2003). A case study in a two-stage hybrid flow shop with setup time and dedicated machines, *International Journal of Production Economics, 86*, 133–143.

Johnson S.M. (1954). Optimal two and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly 1*, 61–67.

Kennedy J, Eberhart R. (1995). Particle swarm optimization. *In: IEEE Int'l conf on neural networks*, Perth, Australia, p. 1942–8.

Kennedy J, Mendes R. (2002). Population structure and particle swarm performance. *Proceedings of the 2002 congress on evolutionary computation CEC2002. IEEE Press*, 1671–6.

Khoo L.P. , Lee S.G. , Yin X.F. , (2000). A prototype genetic algorithm-enhanced multi-objective scheduler for manufacturing systems, *International Journal of Advanced Manufacturing Technology, 16,* 131–138.

Kochhar S., Morris R.J.T. (1987. Heuristic methods for flexible flow line scheduling, *Journal of Manufacturing Systems 6* ,4, 299– 314.

Kurz M.E. , Askin R.G. (2001). Note on "an adaptable problem-space-based search method for flexible flow line scheduling", *IIE Transactions, 33,* 8, 691–693.

Kurz M.E. , Askin R.G. (2003). Comparing scheduling rules for flexible flow lines, *International Journal of Production Economics, 85,* 371–388.

Kurz M.E. , Askin R.G. , (2004). Scheduling flexible flow lines with sequence-dependent setup times, *European Journal of Operational Research, 159,* 1,66–82.

Lee I. , Sikora R. , Shaw M.J. (1997). A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes, *IEEE Transactions on Systems, Manufacturing, and Cybernetics, Part B,* 27, 1, 36–54.

Leon V.J. & Ramamoorthy B. (1997). An adaptable problem-space-based search method for flexible flow line scheduling, *IIE Transactions, 29*, 115–125.

Luh P.B. , Gou L., Zhang Y. , Nagahora T. , Tsuji M. , Yoneda K., Hasegawa T. , Kyoya Y. , Kano T. (1998). Job shop scheduling with group dependent setups, finite buffers, and long time horizon, *Annual of Operation Research, 7*6, 233–259.

Michalewicz Z. (1994). Genetic Algorithms + Data Structure = Evolution Programs, New York: Springer.

Norman B.A., Bean J.C. (1999). A genetic algorithm methodology for complex scheduling problems, *Naval Research Logistics, 46*, 199–211.

Pinedo M., (1995). *Scheduling Theory, Algorithms, and Systems,* Englewood Cliffs, NJ: Prentice-Hall.

Rios-Mercado R.Z., Bard J.F. (1998). Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups, *Computers and Operations Research, 25*,5, 351–366.

Sawik T.J., A (1992). Scheduling algorithm for flexible flow lines with limited intermediate buffers, in: *Proceedings of the Eighth International Conference on CAD/CAM, Robotics and Factories of the Future*, Metz, France, 2, 1711–1722.

Sawik T.J., (1994). New algorithms for scheduling flexible flow lines, in: Proceedings of the 1994 Japan–USA Symposium on Flexible Automation, Kobe, Japan, vol. 3, pp. 1091–1094.

Sawik T.J. (1995). Scheduling flexible flow lines with no in-process buffers, *International Journal of Production Research, 33*,5, 1357– 1367.

Shi XH,Liang YC,Lee HP,Lu C, Wang LM. (2005). An improved GA and a novel PSO-GA-based hybrid algorithm. *Inform Process Lett; 93*,255–61.

Srikar B.N. , Ghosh S. (1986). A MILP model for the N-job, M-stage flowshop with sequence dependent set-up times, *International Journal of Production Research, 24*,6, 1459–1474.

Stafford E.F., Tseng F.T. (1990). On the Srikar–Ghosh MILP model for the N *M SDST flowshop problem, *International Journal of Production Research, 28*,10, 1817–1830.

Storer R.H. , Wu S.D., Vaccari R. (1992). New search spaces for sequencing problems with application to job shop scheduling, *Management Science, 38,* 10, 1495–1509.

Sule D.R., (1982). Sequencing n jobs on two machines with setup, processing and removal times separated, *Naval Research Logistics Quarterly, 29*, 517–519.

Szwarc W., Gupta J.N.D. (1987). A flow-shop with sequence dependent additive setup times, *Naval Research Logistics, 34*, 619–627.

Trelea IC. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Inform Process Lett.85*,6,317–25.

Wittrock R. , (1985). Scheduling algorithms for flexible flow lines, *IBM Journal of Research and Development, 29*, 24, 401–412.

Wittrock R., (1988). An adaptable scheduling algorithm for flexible flow lines, *Operations Research, 36*, 3, 445–453.

Zandieh M., Fatemi S.M.T., (2003). A framework and a classification scheme for modelling production systems, in: *Proceedings of the Second National Industrial Engineering Conference*, Yazd University, Yazd, Iran, 308–315.

Zandieh M, Fatemi S.M.T. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup timesApplied. *Mathematics and Computation, 180*, 111–127.