Discrete Particle Swarm Optimization Algorithm for Flowshop Scheduling

Chapter · January 2009

DOI: 10.5772/6762 · Source: InTech

3 authors:

Ponnambalam S.G.
Vellore Institute of Technology (VIT)
231 PUBLICATIONS **3,840** CITATIONS
SEE PROFILE

Jawahar N
Thiagarajar College of Engineering
100 PUBLICATIONS **2,111** CITATIONS
SEE PROFILE

Srinivasan Chandrasekaran
Indian Institute of Science
341 PUBLICATIONS **5,534** CITATIONS
SEE PROFILE

Discrete Particle Swarm Optimization Algorithm for Flowshop Scheduling

S.G. Ponnambalam[1], N. Jawahar[2] and S. Chandrasekaran[3]
[1]Monash University, [2]Thiagarajar College of Engineering
[3]S R M V Polytechnic College
[1]Malaysia, [2,3]India

1. Introduction

In the context of manufacturing systems, scheduling refers to allocation of resources over time to perform a set of operations. Manufacturing systems scheduling has many applications ranging from manufacturing, computer processing, transportation, communication, health care, space exploration, education, distribution networks, etc. Scheduling is a process by which limited resources are allocated over time among parallel or sequential activities. Solving such a problem amounts to making discrete choices such that an optimal solution is found among a finite or a countably infinite number of alternatives. Such problems are called combinatorial optimization problems. Typically, the task is complex, limiting the practical utility of combinatorial, mathematical programming and other analytical methods in solving scheduling problems effectively. Manufacturing system entails the acquisition and allocation of limited resources to production activities so as to reduce the manufacturing cycle time and in-process inventory and to satisfy customer demand in specified time. Successful achievement of these objectives lies in efficient scheduling of the system. Scheduling plays an important role in shop floor planning. A schedule shows the planned time when processing of a specific job will start on a machine. It also indicates when a job will get completed on a machine. Scheduling is a decision-making process of sequencing a set of operations on different machines in a manufacturing unit. The objective of scheduling is generally to improve the utilization of resources and profitability of production lines. Scheduling problem is characterized by three components namely:

1. Number of machines, number of jobs and the processing time for each job using appropriate machine
2. A set of constraints such as operation precedence constraint for a given job and operation non-overlapping constraint for a given machine
3. A target function called objective function consisting of single or multiple criteria that must be optimized.

Traditionally, scheduling researchers has shown interest in optimizing a single-objective or performance measure while scheduling, which is not a reality. Practical scheduling problems acquire consideration of several objectives as desired by the scheduler. When multiple criteria are considered, scheduler may wish to generate a schedule which performs better with respect to all the measures under study, such solution does not exist. This chapter presents the application of Discrete Particle Swarm Optimisation Algorithm for solving flowshop scheduling problem (FSP) under single and multiple objective criteria.

2. Flowshop Scheduling

2.1 Description of FSP

In discrete parts manufacturing industries, jobs with multiple operations use machines in the same order. In such case, machines are installed in series. Raw materials initially enter the first machine and when a job has finished its processing on the first machine, it goes to the next machine. When the next machine is not immediately available, job has to wait till the machine becomes available for processing. Such a manufacturing system is called a flowshop, where the machines are arranged in the order in which operations are to be performed on jobs. A flowshop is a conventional manufacturing system where machines are arranged in the order of performing operations on jobs. The technological order, in which the jobs are processed on different machines, is unidirectional. In a flowshop, a job i with a set of m operations $i_1, i_2, i_3, \ldots, i_m$ is to be completed in a predetermined sequence. In short, each operation except the first has exactly one direct predecessor and each operation except the last one has exactly one direct successor as shown in Figure 1. Thus each job requires a specific immutable sequence of operations to be carried out for it to be complete. This type of structure is sometimes referred as linear precedence structure (Baker, 1974). Further, once started, an operation on a machine cannot be interrupted.

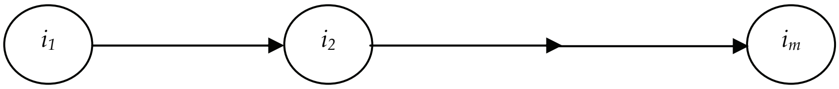

Figure 1. Work Flow in Flowshop

2.2 Characteristics of FSP

Flowshop consists of *m* machines and there are *n* different jobs to be optimally sequenced through these machines. The common assumptions used in modelling the flowshop problems are as follows:

- All *n* jobs are available for processing at time zero and each job follows identical routing through the machines.
- Unlimited storage exists between the machines. Each job requires *m* operations and each operation requires a different machine.
- Every machine processes only one job at a time and every job is processed by one machine at a time.
- Setup-times for the operations are sequence-independent and are included in processing times.
- The machines are continuously available.
- Individual operations cannot be pre-empted.

Further it is assumed that:

- Each job must be processed to completion.
- In-process inventory is allowed when necessary.

- There is only one machine of each type in the shop.
- Machines are available throughout the scheduling period.
- There is no randomness and the scheduling problem under study is a deterministic scheduling problem. In particular
  - The number of jobs is known and fixed.
  - The number of machines is known and fixed.
  - The processing times are known and fixed, and
  - All other quantities needed to define a particular problem are known and fixed.

The general structure of typical n job m machine FSP is shown in Figure 2.

Job	Processing order			
	M_1	M_2	M_3 ………	M_m
J_1	Pt_1	Pt_2	Pt_3 ………	Pt_m
J_2	Pt_1	Pt_2	Pt_3 ………	Pt_m
J_3	Pt_1	Pt_2	Pt_3 ………	Pt_m
..	..	..	..	..
..	..	..	..	..
J_n	Pt_1	Pt_2	Pt_3 ………	Pt_m

where Pt - processing time of job J in machine M

Figure 2. General Structure of Flowshop

2.3 Solution approaches for FSP

Computational complexity of a problem is the maximum number of computational steps needed to obtain an optimal solution. For example if there are n jobs and m available machines, the available number of schedule to be evaluated to get an optimal solution is $(n!)^m$. In a permutation flow based manufacturing system, the number of available schedules is $n!$. Based on the complexity of the problem, all problems can be classified into two classes, called P and NP in the literature. Class P consists of problems for which the execution time of the solution algorithms grows polynomially with the size of the problem. Thus, a problem of size m would be solvable in time proportional to m^k, when k is an exponent. The time taken to solve a problem belonging to the NP class grows exponentially, thus this time would grow in proportion to t^m, where t is some constant. In practice, algorithms for which the execution time grows polynomially are preferred. However, a widely held conjecture of modern mathematics is that there are problems in NP class for which algorithms with polynomial time complexity will never be found (French, 1982). These problems are classified as $NP-\text{hard}$ problems. Unfortunately, most of the practical scheduling problems belong to the $NP-\text{hard}$ class (Rinnooy Kan, 1976). Many scheduling problems are polynomially solvable, or NP-hard in that it is impossible to find an optimal solution here without the use of an essentially enumerative algorithm. FSP is a widely researched combinatorial optimization problem, for which the computational effort increases exponentially with problem size (Jiyin Liu & Colin Reeves, 2001; Brucker, 1998; Sridhar & Rajendran, 1996; French, 1982). In FSP, the computational complexity increases with increase in problem size due to increase in number of jobs and/or number of machines.

To find exact solution for such combinatorial problems, a branch and bound or dynamic programming algorithm is often used when the problem size is small. Exact solution methods are impractical for solving FSP with large number of jobs and/or machines. For the large-sized problems, application of heuristic procedures provides simple and quick method of finding best solutions for the FSP instead of finding optimal solutions. A heuristic is a technique which seeks (and hopefully finds) good solutions at a reasonable computational cost. A heuristic is approximate in the sense that it provides a good solution for relatively little effort, but it does not guarantee optimally. A heuristic can be a rule of thumb that is used to guide one's action. Heuristics for the FSP can be a constructive heuristics or improvement heuristics. Various constructive heuristics methods have been proposed by Johnson, 1954; Palmer, 1965; Campbell et al., 1970; Dannenbring 1977 and Nawaz et al. 1983. Literature shows that constructive heuristic methods give very good results for NP-hard combinatorial optimization problems. This builds a feasible schedule from scratch and the improvement heuristics try to improve a previously generated schedule by applying some form of specific improvement methods. An application of heuristics provides simple and quick method of finding best solutions for the FSPs instead of finding optimal solutions (Ruiz & Maroto, 2005; Dudek et al. 1992). Johnson's algorithm (1954) is the earliest known heuristic for the FSP, which provides an optimal solution for two-machine problem to minimize makespan. Palmer (1965) developed a very simple heuristic in which for every job a "slope index" is calculated and then the jobs are scheduled by non-increasing order of this index. Ignall & Schrage (1965) applied the branch and bound technique to the flowshop sequencing problem. Campbell et al. (1970) developed a heuristic algorithm known as CDS algorithm and it builds $m-1$ schedules by clustering the m original machines into two virtual machines and solving the generated two machine problem by repeatedly using Johnson's rule. Dannenbring's (1977) Rapid Access heuristic is a mixture of the previous ideas of Johnson's algorithm and Palmer's slope index. Nawaz et al.'s (1983) NEH heuristic is based on the idea that jobs with high processing times on all the machines should be scheduled as early in the sequence as possible. NEH heuristics seems to be the performing better compared to others. Heuristic algorithms are conspicuously preferable in practical applications. Among the most studied heuristics are those based on applying some sort of greediness or applying priority based procedures including, e.g., insertion and dispatching rules. The main drawback of these approaches, their inability to continue the search upon becoming trapped in a local optimum, leads to consideration of techniques for guiding known heuristics to overcome local optimality (Jose Framinan et al. 2003). And also the heuristics has the problems like

1. Lack of comprehensiveness
2. Little robustness of conclusions
3. Weak/partial experimental design

For these reasons, one can investigate the application of metaheuristic search methods for solving optimization problems. It is a set of algorithmic concepts that can be used to define heuristic methods applicable to wide set of varied problems. The use of metaheuristics has significantly produced good quality solutions to hard combinatorial problems in a reasonable time. It is defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions (Osman & Laporte, 1996). The fundamental properties which characterize metaheuristics are as follows (Christian Blum & Andrea Roli, 2003):

- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Metaheuristics or Improvement heuristics are extensively employed by researchers to solve scheduling problems (Chandrasekaran et al. 2006; Suresh & Mohanasundaram, 2004; Hisao Ishibuchi et al. 2003; Lixin Tang & Jiyin Liu, 2002; Eberhart & Kennedy, 1995). Improvement methods such as Genetic Algorithm (Chan et al. 2005; Ruiz et al. 2004; Sridhar & Rajendran, 1996), Simulated Annealing algorithm (Ogbu & Smith, 1990), Tabu Search algorithm (Moccellin & Nagamo, 1998) and Particle Swarm Optimization algorithm (Rameshkumar et al. 2005; Prabhaharan et al. 2005; Faith Tasgetiren et al. 2004) have been widely used by researchers to solve FSPs. Metaheuristic algorithms such as Simulated Annealing (SA) and Tabu Search (TS) methods are single point local search procedures where, a single solution is improved continuously by an improvement procedure. Algorithms such as Genetic Algorithm (GA), Ant Colony Optimization (ACO) algorithm and Particle Swarm Optimization (PSO) algorithm belongs to population based search algorithms. These are designed to maintain a set of solution transiting from a generation to the next. The family of metaheuristics includes, but is not limited to, GA, SA, ACO, TS, PSO, evolutionary methods, and their hybrids.

2.4 Performance measures considered

Measures of schedule performance are usually functions of the set of completion times in a schedule. Performance measures can be classified as regular and non-regular. A regular measure is one in which the penalty function is non-decreasing in terms of job completion times. Some examples of regular performance measures are makespan, mean flowtime, total flowtime, and number of tardy jobs. Performance measures, which are not regular, are termed non-regular. That is, such measures are not an increasing function with respect to job completion times. Some examples of non-regular measures are earliness, tardiness, and completion time variance. In this chapter, the performance measures namely minimization of makespan, total flowtime and completion time variance is considered for solving flowshop scheduling problems. Makespan (C_{max}) has been considered by many scheduling researchers (Ignall & Scharge, 1965; Campbell et al. 1970; Nawaz et al.1983; Framinan et al. 2002; Ruiz & Maroto, 2005). Makespan is defined as the time required for processing all the jobs or the maximum time required for completing a given set of jobs. Minimization of makespan ensures better utilization of the machines and leads to a high throughput (Framinan et al. 2002; Ruiz & Maroto, 2005). Makespan is computed using equation (1).

$$C_{max} = \max\{C_i, i = 1,2,........,n\} \tag{1}$$

The time spend by a job in the system has been defined as its flow time. Total flowtime is defined as the sum of completion time of every job or total time taken by all the jobs. Total flowtime (F_Σ) of the schedule is computed using equation (2). Minimizing total flowtime results in minimum work-in-process inventory (Chandrasekharan Rajendran & Hans Ziegler, 2005).

$$F_\Sigma = \sum_{i=1}^{n} C_i \tag{2}$$

Completion time variance is defined as the variance about the mean flowtime and is computed using equation (3). Minimizing completion time variance (V_T) serves to minimize variations in resource consumption and utilization (Gowrishankar et al. 2001; Gajpal & Rajendran, 2006; Viswanath Kumar Ganesan et al. 2006).

$$V_T = \frac{1}{n}\sum_{i=1}^{n}(C_i - \overline{F})^2 \tag{3}$$

where $\overline{F} = \frac{F_T}{n}$ is the mean flowtime.

3. Particle Swarm Optimization Algorithm

3.1 Features of PSO

Particle Swarm Optimization (PSO) algorithm is an evolutionary computation technique developed by Eberhart & Kennedy in 1995 inspired by social behavior of bird flocking or fish schooling. PSO is a stochastic, population-based approach for solving problems (Kennedy & Eberhart, 1995). It is a kind of swarm intelligence that is based on social-psychological principles and provides insights into social behavior, as well as contributing to engineering applications. PSO algorithm has been successfully used to solve many difficult combinatorial optimization problems. PSO algorithm is problem-independent, which means little specific knowledge relevant to a given problem is required. All we have to know is the fitness evaluation of each solution. This advantage makes PSO more robust than many search algorithms. In the last couple of years the particle swarm optimization algorithm has reached the level of maturity necessary to be interesting from an engineering point of view. It is a potent alternative optimizer for complex problems and possesses many attractive features such as:

- Ease of implementation: The PSO is implemented with just a few lines of code, using only basic mathematical operations.
- Flexibility: Often no major adjustments have to be made when adapting the PSO to a new problem.
- Robustness: The solutions of the PSO are almost independent of the initialization of the swarm. Additionally, very few parameters have to be tuned to obtain quality solutions.

- Possibility to combine discrete and continuous variables. Although some authors present this as a special feature of the PSO (Sensarma et al., 2002), others point out that there are potential dangers associated with the relaxation process necessary for handling the discrete variables (Abido, 2002). Simple round-off calculations may lead to significant errors.
- Possibility to easily tune the balance between local and global exploration.
- Parallelism: The PSO is inherently well suited for parallel computing. The swarm population can be divided between many processors to reduce computation time.

3.2 Applications of PSO

In recent years, PSO has been successfully applied in many areas. Currently, PSO has been implemented in a wide range of research areas such as functional optimization, pattern recognition, neural network training, fuzzy system control etc. and obtained significant success. PSO is widely applied and focused by researchers due to its profound intelligence background and simple algorithm structure. Many proposals indicate that PSO is relatively more capable for global exploration and converges more quickly than many other heuristic algorithms. It solves a variety of optimization problems in a faster and cheaper way than the evolutionary algorithms in the early iterations. One of the reasons that PSO is attractive is that there are very few parameters to adjust. One version, with very slight variation (or none at all) works well in a wide variety of applications. PSO has been used for approaches that can be used across a wide rage of applications, as well as for specific applications focused on a specific requirement. PSO has been applied to the analysis of human tremor. The diagnosis of human tremor, including Parkinson's disease and essential tremor, is a very challenging area. PSO has been used to evolve a neural network that distinguishes between normal subjects and those with tremor. Inputs to the network are normalized movement amplitudes obtained from an actigraph system. The method is fast and accurate (Eberhart & Hu, 1999). While development of computer numerically controlled machine tools has significantly improved productivity, there operation is far from optimized. None of the methods previously developed is sufficiently general to be applied in numerous situations with high accuracy. A new and successful approach involves using artificial neural networks for process simulation and PSO for multi-dimensional optimization. The application was implanted using computer-aided design and computer-aided manufacturing (CAD/CAM) and other standard engineering development tools as the platform (Tandon, 2000). Another application is the use of particle swarm optimization for reactive power and voltage control by a Japanese electric utility (Yoshida et al., 1999). PSO has also been used in conjunction with a back propagation algorithm to train a neural network as a state-of-charge estimator for a battery pack for electric vehicle use. Determination of the battery pack state of charge is an important issue in the development of electric and hybrid / electric vehicle technology. The state of charge is basically the fuel gauge of an electric vehicle. A strategy was developed to train the neural network based on a combination of particle swarm optimization and the back propagation algorithm. Finally, one of the most exciting applications of PSO is that by a major American corporation to ingredient mix optimization. In this work, "ingredient mix" refers to the mixture of ingredients that are used to grow production strains of microorganisms that naturally secrete of manufacture something of interest. Here, PSO was used in parallel with traditional industrial optimization methods. PSO provided an optimized ingredient mix that provided over twice the fitness as the mix found using traditional methods, at a very different location in ingredient space. PSO was shown to be robust: the occurrence of an ingredient becoming contaminated hampered the search for a few iterations but in the end did not result in poor final results. PSO, by its nature, searched a much larger portion of the problem space than the traditional method. Generally speaking, particle swarm optimization, like the other evolutionary computation algorithms, can be applied to solve most optimization problems and problems that can be converted to optimization problems. Among the application areas with the most potential are system design, multi-objective optimization, classification, pattern recognition, biological system modelling, scheduling (planning), signal processing, games, robotic applications, decision making, simulation and identification. Examples include fuzzy controller design, job shop scheduling, real time robot path planning, image segmentation, EEG signal simulation, speaker verification, time-frequency analysis, modelling of the spread of antibiotic resistance, burn diagnosing, gesture recognition and automatic target detection, to name a few (Eberhart & Shi, 2001).

3.3 Working of PSO

PSO is initialized with a swarm of random feasible solutions and searches for optima by updating velocities and positions. PSO algorithm is initialized with a set of several random particles called a swarm. A set of moving particles (the swarm) is initially thrown inside the multi-dimensional search space. Each particle is a potential solution, which has the ability to remember its previous best position and current position, and it survives from generation to generation. Each particle has the following features:

- It has a position and a velocity
- It knows its neighbours, best previous position and objective function value.
- It remembers its best previous position.

At each time step, the behavior of a given particle is a compromise between three possible choices

- To follow its own way
- To go towards its best previous position
- To go towards the best neighbour’s best previous position, or forwards the best neighbour.

The swarm is typically modelled by particles in multi-dimensional space that have a position and a velocity. These particles fly through hyperspace and have two essential reasoning capabilities: their memory of their own best position and knowledge of their neighborhood's best, "best" simply meaning the position with the smallest objective value. Members of a swarm communicate good positions to each other and adjust their own position and velocity based on these good positions. PSO shares many similarities with evolutionary computation techniques such as GA, SA, TS and ACO algorithms. The PSO system is initialized with a swarm of random solutions and searches for optima by updating generations. The advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied. Most of evolutionary techniques have the following procedure:

1. Random generation of an initial population
2. Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.

3. Reproduction of the population based on fitness values.
4. If requirements are met, then stop. Otherwise go back to step 2.

Evolutionary Algorithms use a population of potential solutions (points) of the search space. These solutions (initially randomly generated) are evolved using different specific operators which are inspired from biology. Through cooperation and competition among the potential solutions, these techniques often can find near-optimal solutions quickly when applied to complex optimization problems. There are some similarities between PSO and Evolutionary Algorithms:

1. Both techniques use a population (which is called *swarm* in the PSO case) of solutions from the search space which are initially random generated;
2. Solutions belonging to the same population interact with each other during the search process;
3. Solutions are evolved using techniques inspired from the real world.

PSO shares many common points with GA. Both algorithms start with a group of a randomly generated population; both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee success. However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. The information sharing mechanism in PSO is significantly different. In GA, chromosomes share information with each other. So the whole population moves like one group towards an optimal area. In PSO, only global or local best particle gives out the information to others. It is a one-way information sharing mechanism. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases/ PSO optimization algorithm uses a set of particles called a swarm, similar to chromosomes in a binary-coded Genetic Algorithm (GA). PSO and ACO are optimization algorithms based on the behavior of swarms (birds, fishes) and ants respectfully. However, the particles are multidimensional points in real space during the optimization. The PSO optimization run starts with a user-specified swarm size and objective function used to evaluate objection function values, called fitness in GA terminology. The particles are initialized randomly within the variable bounds and they search for the optimum (maximum or minimum) in the search space with some communication between particles. For a maximization (or minimization) problem, the particles will move towards the particle with the highest (or least) objective function value using a position update equation, that is stochastic. This is how randomness in introduced to PSO algorithm. This position update method is similar to the use of crossover and mutation operations used to generate new individuals in a new generation in the GA. However, the PSO differs in that, updates of particle position usually involve the best particles (global or in the neighborhood) of each particle. The position updating tends to always exploit the best solution found so far. While this may lead to premature convergence, when all particles positions become equal to that of the best particle (i.e., no diversity), there are schemes designed to prevent such premature convergence. In the PSO literature, several neighborhood schemes have been developed for the particle updating (Merkle and Middendorf, 2000). This chapter aims to develop a metaheuristic algorithm called PSO algorithm which is suitable for solving FSPs with the objective of minimising three performance measures namely makespan, total flowtime and completion time variance. Firstly, a single objective PSO is proposed and the above performance measures are considered individually. Performance of the proposed single objective PSO is tested by solving a large set of benchmark FSPs available in the literature having number of jobs varying from 5 to 500 and number of machines from 5 to 20.

3.4 Structure of PSO Algorithm

The pseudo-code of the simple PSO algorithm and its general framework are given in Figures 3 and 4 respectively.

The basic elements of PSO algorithm are summarized below:

Particle: X_i^t denotes the *ith* particle in the swarm at iteration *t* and is represented by *n* number of dimensions as $X_i^t = [x_{i1}^t, x_{i2}^t, .., x_{in}^t]$, where x_{ij}^t is the position value of the *ith* particle with respect to the *jth* dimension ($j = 1,2,...,n$).

Population: pop^t is the set of NP particles in the swarm at iteration *t*, i.e., $pop^t = [X_1^t, X_2^t, ..., X_{NP}^t]$.

Sequence: We introduce a new variable π_i^t, which is a permutation of jobs implied by the particle X_i^t. It can be described as $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, .., \pi_{in}^t]$, where π_{ij}^t is the assignment of job *j* of the particle *i* in the permutation at iteration *t*.

```
Initialize swarm
Initialize velocity
Initialize position
Initialize parameters
    Evaluate particles
    Find the local best
    Find the global best
Do
{
        Update velocity
        Update position
        Evaluate
        Update local best
        Update global best
}  ( until termination)
```

Figure 3. Pseudocode of the PSO Algorithm

Particle velocity: V_i^t is the velocity of particle *i* at iteration *t*. It can be defined as $V_i^t = [v_{i1}^t, v_{i2}^t, ..., v_{in}^t]$, where v_{ij}^t is the velocity of particle *i* at iteration *t* with respect to the *jth* dimension.

Local best: P_i^t represents the best position of the particle *i* with the best fitness until iteration *t*, so the best position associated with the best fitness value of the particle *i* obtained so far is called the *local best*. For each particle in the swarm, P_i^t can be determined and updated at each iteration *t*. In a minimization problem with the objective function $f(\pi_i^t)$ where π_i^t is the corresponding sequence of particle X_i^t, the local best P_i^t of the *ith* particle is obtained such that $f(\pi_i^t) \le f(\pi_i^{t-1})$ where π_i^t is the corresponding permutation of local best P_i^t

and π_i^{t-1} is the corresponding sequence of local best P_i^{t-1}. To simplify, we denote the fitness function of the local best as $f_i^{pb} = f(\pi_i^t)$. For each particle, the local best is defined as $P_i^t = [p_{i1}^t, p_{i2}^t, ..., p_{in}^t]$ where p_{ij}^t is the position value of the *ith* local best with respect to the *jth* dimension ($j = 1,2,...,n$).

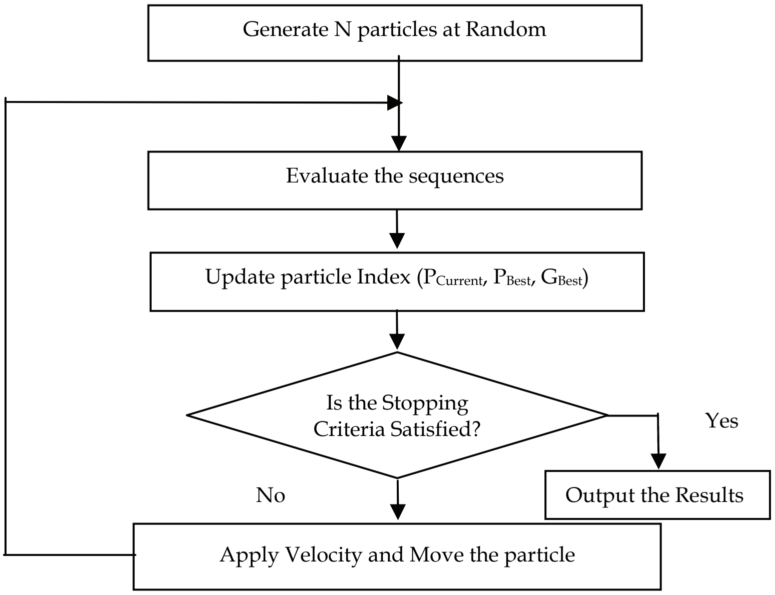
Generate N particles at Random
Evaluate the sequences
Update particle Index (PCurrent, PBest, GBest)
Is the Stopping Criteria Satisfied?
Yes
No
Output the Results
Apply Velocity and Move the particle

Figure 4. The Framework of PSO Algorithm

4. Discrete PSO Algorithm for Single-Objective FSP

4.1 Pseudocode of the proposed discrete PSO algorithm

Particle Swarm Optimization algorithm starts with a population of randomly generated initial solutions called particles (swarm). It is to be noted that the particle structure is taken as a string, which consists of job numbers in certain order. The order of jobs in the string represents a sequence. After the swarm is initialized, each potential solution is assigned a velocity randomly. The length of the velocity of each particle $\|v\|$ is generated randomly between *0* and *n* (Rameshkumar et al. 2005; Chandrasekaran et al. 2006) and the corresponding lists of transpositions $(i_q, j_q); q = 1, \|v_k\|$ are generated randomly for each particle. The above formulation permits exchange of jobs $(i_1, j_1),(i_2, j_2)......(i_{\|v\|}, j_{\|v\|})$ in the given order. Each particle keeps track of its improvement and the best objective function value achieved by the individual particles so far is stored as local best solution $({}^eP_k^t)$, and the overall best objective function achieved by all the particles together so far is stored as the global best solution (G_b^t). The particle velocity and position are updated continuously in all iterations. The iterative improvement process is continued afterwards to further improve the solution quality. The Pseudocode of the proposed discrete PSO algorithm is shown in Figure 5.

```
Initialize swarm $P$                    $t = 0$;
Initialize velocity $v_k^t$ and position $P_k^t$
Initialize parameters
    Evaluate particles
    Find the local best ${}^eP_k^t$ and global best $G_b^t$
Do
        {
        for $(k = 1, N)$
        Update Velocity $v_k^{t+1}$;
        Update Position $P_k^{t+1}$;
        Evaluate all particles;
        Update ${}^eP_k^{t+1}$ and $G^{t+1}$, $(k = 1, N)$;
        $t \to t+1$;
        } $(\text{while } t < t_{max})$
```

Figure 5. Pseudocode of the Proposed Discrete PSO Algorithm

The particle velocity and position are continuously updated using equation (4) and (5).

$$v_k^{t+1} = C_1 U_1 v_k^t + C_2 U_2 \,\text{rand}()({}^eP_k^{t+1} - P_k^{t+1}) + C_3 U_3 \,\text{rand}()(G_k - P_k^{t+1}) \tag{4}$$

$$P_k^{t+1} = P_k^t + v_k^{t+1} \tag{5}$$

where C_1, C_2 and C_3 is called acceleration constants. The acceleration constants C_1, C_2 and C_3 in equation (4) guide every particle toward local best and the global best solution during the search process. Low acceleration value results in walking far from the target, namely local best and the global best. High value results in premature convergence of the search process.

4.2 Procedural steps of the Discrete PSO Algorithm

The step by step procedure for implementing the proposed discrete PSO algorithm is as follows.

Step1: Initialize a swarm P_i with random positions and velocities in the problem space X.

Step2: For each particle, evaluate the desired optimization fitness function

Step3: Compare the fitness function with its previous best. If current value is better than previous best, then set previous best equal to current value and P_i equal to the current location X_i.

Step4: Identify the particle in the neighborhood with the best success so far, and assign its index to the variable G.

Step5: Apply local search algorithm to all the particles at the end of each iteration and evaluate for the objective function.

Step6: Change the velocity and position of the particle according to equation (4) and equation (5).

Step7: Loop to step (2) until a criterion is met (usually number of iterations).

4.3 Numerical Illustrations

An example illustrating the process of updating the velocity and the position of a sequence is explained as follows:

Velocity update: The procedure for updating the velocity of all the particles in each iteration is as follows: For example, let us assume

The sequence $P_k^t = \{2,3,4,1\}$; $C_1 = 1,\ C_2 = 2,\ C_3 = 2$; $U_1 = 0.2,\ U_2 = 0.4,\ U_3 = 0.3$; $\|V_k\| = 2$, $v = (1,4),(2,3)$; ${}^eP_k^t = (1,4,3,2)$ and $G_b^t = (3,1,4,2)$.

Velocity of the particle *k* at time step $t+1$ namely V_k^{t+1} is obtained using equation (4)

V_k^{t+1} = 1x 0.2 [(1,4),(2,3)] ⊕ 2 x 0.4 [(1,4,3,2) - (2,3,4,1)] ⊕ 2 x 0.3 [(3,1,4,2) - (2,3,4,1)]

where [(1,4,3,2) - (2,3,4,1)] represents a velocity such that applying the resulting velocity to the current particle (2,3,4,1) yields a position (1,4,3,2).

Thus, V_k^{t+1} = 0.2 [(1,4), (2,3)] ⊕ 0.8 [(2,3), (1,4)] ⊕ 0.6 [(1,2), (1, 4)]
= ((1, 4),(2, 3),(1, 2))

Position update: Position of the particle k at time step $t+1$ namely P_k^{t+1} is obtained using equation (5) by applying V_k^{t+1} over P_k^t as follows.

P_k^{t+1} =(2,3,4,1) + ((1,4), (2,3),(1,2));
= (1,3,4,2) + ((2,3),(1,2)); =(1,4,3,2) + (1,2);
= (4,1,3,2)

4.4 Performance Comparison

An extensive performance analysis using proposed discrete PSO algorithm is carried out by means of evaluating the performance measures by solving the benchmark FSPs of Taillard (1993). Extensive experiments are conducted to fix the parameters like number of particles, number of iterations, selection of learning coefficients and initial swarm generation. The evaluation of proposed discrete PSO algorithm is coded in Linux C and run on an Intel Pentium III 900MHz PC with 128 MB memory.

Number of iterations: Number of iterations or termination criterion is a condition that the search process will be terminated. It might be a maximum number of iteration or maximum CPU times are normally to terminate the search process (Liu & Reeves, 2001; Gowrishankar et al. 2001). In this chapter, for the single-objective optimization problems, an evaluation of *1000 x n x m* number of sequences or particles is taken as the termination criterion.

Number of particles: Experiments have been conducted to identify the optimal swarm size by solving a set of 30 different instances of Taillard (1993) for makespan objective with 20 jobs and machines varying from 5, 10 and 20 using discrete PSO algorithm. In experimentation, the performance of the algorithm is better with swarm size 80 and the same has been used throughout our evaluation.

Learning coefficients: The roll of learning coefficients or acceleration constants, namely C_1, C_2 and C_3 guide every particle towards the local best and the global best solutions during the search process. Low acceleration value results in walking far from the target, namely local best and the global best. High value results in premature convergence of the search process. Experiments have been conducted using different combinations of learning coefficients. To determine the best combinations of C_1, C_2 and C_3 values by solving a set of 30 FSPs for makespan objective with 20 jobs and machines varying from 5, 10 and 20 using the proposed PSO algorithm. The values $C_1 = 1, C_2 = 2$ and $C_3 = 2$ shows better performance and the same, has been used throughout our study.

Velocity coefficients: The velocity update is carried out after every iteration to improve the search process. The velocity coefficients, namely U_1, U_2 and U_3 guides the search to find the optimal solution quickly. As per the experiments, the values for U_1, U_2 and U_3 are generated randomly between 0 and 1.

Initial Swarm Generation: For the generation of initial swarm one particle is generated from the results obtained by certain algorithms for the desired optimization fitness function and remaining particles of the swarm is constructed in a way that a permutation is produced randomly. The particle generated from certain algorithms is added with randomly generated particles at the beginning of the search. This insertion of the particle in initial swarm is to find better sequences in each iteration of the search. And also it improves the performance of discrete PSO algorithm in terms of finding near-optimal solutions. The algorithms selected for generating the particle for different objective functions are listed below. For makespan objective, one particle is generated using NEH heuristic of Nawaz et al. (1983) and is added to the swarm. For total flowtime objective, one particle is generated based on the heuristic developed by Rajendran. (1993) and is added to the swarm. For completion time variance objective, a particle is generated based on the algorithm developed by Gajpal & Rajendran (2006), and is added to the swarm. These algorithms have better start with the respective objectives. Performance of the proposed discrete PSO with respect to makespan objective is carried out in comparison with the benchmark solutions given by Taillard (1993) and with the results published in the literature. The quality measure namely, "Average Relative Percent Deviation" $(\overline{\text{RPD}})$ is considered for the evaluation. During comparison, the corresponding better values reported in the literature are taken. The RPD is computed using equation (6).

$$\text{RPD} = [G - C^* / C^*] \times 100 \tag{6}$$

where, G represents the global best solution obtained by the proposed algorithm for a given problem and C^* represents the upper bound value reported in the literature for the corresponding objective function. Some sample results of problems ta001-ta010 of Taillard (1993) is presented in Table 1.

Instances	Problem	Results Reported	Results Obtained	RPD
20 x 5	ta001	1278	1278	0.0000
	ta002	1359	1360	0.0736
	ta003	1081	1088	0.6475
	ta004	1293	1293	0.0000
	ta005	1235	1235	0.0000
	ta006	1195	1195	0.0000
	ta007	1239	1239	0.0000
	ta008	1206	1206	0.0000
	ta009	1230	1237	0.5691
	ta010	1108	1108	0.0000
		$\overline{\text{RPD}}$		**0.1290**

Table 1. Sample Results for Makespan

In order to evaluate the performance of the proposed discrete PSO with respect to the total flowtime objective, the results are compared with the results of the popular performing heuristics developed by Liu & Reeves (2001), M-MMAS Algorithm and PACO Algorithm (Rajendran & Ziegler, 2004). Some sample results of problems ta001-ta010 for total flowtime criteria is presented in Table 2.

Instances	Problem	Results Reported	Results Obtained	RPD
20 x 5	ta001	14056 ^{3}	14033	-0.1636
	ta002	15151 ^{2}	15151	0.0000
	ta003	13403 ^{3}	13313	-0.6715
	ta004	15486 ^{2}	15459	-0.1744
	ta005	13529 ^{3}	13529	0.0000
	ta006	13123 ^{3}	13123	0.0000
	ta007	13559 ^{2}	13548	-0.0811
	ta008	13968 ^{1}	13948	-0.1432
	ta009	14317 ^{2}	14315	-0.0140
	ta010	12968 ^{2}	12943	-0.1928
	$\overline{\text{RPD}}$			**-0.1441**

Note: Superscript (1) refers to Heuristic Algorithm (Liu & Reeves, 2001) (2) M-MMAS Algorithm (Rajendran & Ziegler, 2004) (3) PACO Algorithm (Rajendran & Ziegler, 2004)

Table 2. Sample Results for Total Flowtime

Instances	Problem	Results Reported	Results Obtained	RPD
20 x 5	ta001	73040.55 ^{3}	72060.23	-1.3422
	ta002	90885.27 ^{2}	89238.17	-1.8123
	ta003	53894.49 ^{2}	53851.95	-0.0789
	ta004	89822.05 ^{4}	87104.42	-3.0256
	ta005	72350.55 ^{2}	72020.43	-0.4563
	ta006	71665.73 ^{2}	70817.64	-1.1834
	ta007	69088.45 ^{2}	68367.69	-1.0432
	ta008	70214.31 ^{2}	69793.85	-0.5988
	ta009	73329.22 ^{2}	72284.98	-1.4240
	ta010	52580.03 ^{1}	52015.34	-1.0740
	$\overline{\text{RPD}}$			**-1.2039**

Note: Superscript (1) refers to PACO Algorithm (Rajendran & Ziegler, 2004) (2) MMAS Ant Colony Algorithm (Stuetzle, 1998) (3) NACO Algorithm with position-job insertion local search (Gajpal & Rajendran, 2006) (4) NACO Algorithm with job-index based local search (Gajpal & Rajendran, 2006)

Table 3. Sample Results for Completion Time Variance

The performance of the proposed discrete PSO algorithm with respect to completion time variance criterion, the results are compared with the results of ant colony algorithm with random-job insertion local search by Gajpal & Rajendran (2006), M-MMAS Ant Colony Algorithm by Stuetzle(1998), PACO Algorithm by Rajendran & Ziegler(2004), and three

NACO Algorithm with position-job insertion and job-index based local searches by Rajendran & Ziegler (2004). To our knowledge, the results of completion time variance objective using PSO algorithm are not available in literature, the performance of the proposed algorithm is compared with other metaheuristic results. Some sample results of problems ta001-ta010 of Taillard (1993) for completion time variance objective are presented in Table 3.

The results show that the proposed single-objective discrete PSO algorithm performs better. The negative sign in RPD values shows that the proposed discrete PSO algorithm generates better results than the results reported in the literature considered. The summary of $\overline{\text{RPD}}$ values obtained for all the FSP instances of Taillard (1993) are presented in Table 4.

Instances	Number of problems	Makespan	Total Flowtime	Completion Time Variance
20 x 5	10	0.1290238	-0.1440537	-1.2038674
20 x10	10	0.5334462	-0.0164544	-1.7613968
20 x 20	10	0.5329960	-0.0260092	-0.8586390
50 x 5	10	0.0890855	-0.2925054	-0.9330275
50 x 10	10	1.7541958	-0.0108922	-0.2059756
50 x 20	10	2.9814187	0.2434647	1.7126618
100 x 5	10	0.1713382	-0.7238382	1.2988817
100 x 10	10	0.6882989	-0.1191928	0.9198400
100 x 20	10	2.8784086	0.1476830	3.4646301
200 x 10	10	0.5498368	1.8246721	0.0000000
200 x 20	10	2.7011408	1.4120018	0.0000000
500 x 20	10	1.8172343	1.4205378	0.0000000

Table 4. $\overline{\text{RPD}}$ Values Obtained for the Various FSP Instances

The proposed discrete PSO algorithm generates good results with reasonable CPU time. CPU time taken by the proposed discrete PSO algorithm for various FSPs are presented in Table 5.

Instances	Number of Problems	Makespan	Total Flowtime	Completion Time Variance
20x5	10	0m25.164s	0m5.201s	0m6.642s
20x10	10	1m36.844s	0m12.113s	0m33.619s
20x20	10	6m22.854s	0m35.139s	2m16.764s
50x5	10	13m44.973s	0m39.888s	1m10.433s
50x10	10	55m38.305s	1m45.854s	6m19.487s
50x20	10	110m32.087s	10m33.215s	32m41.970s
100x5	10	19m42.310s	4m17.995s	10m39.676s
100x10	10	26m3.295s	9m22.616s	45m1.041s
100x20	10	62m14.918s	33m57.255s	84m4.257s
200x10	10	143m25.161s	41m33.599s	50m27.703s
200x20	10	166m27.657s	79m22.342s	129m58.384s
500x20	10	543m32.695s	792m17.371s	410m50.485s

Table 5. CPU time taken for Various FSP Instances

5. Discrete PSO Algorithm for Multi-Objective FSP

5.1 Concept and terminology

The real-world scheduling problems are multi-objective in nature. In such cases, several objectives must be simultaneously considered when evaluating the quality of the proposed solution. In multi objective decision problems one desires to simultaneously optimize more than one performance objectives such as makespan, tardiness, mean flowtime of jobs, etc. multi-objective optimization usually results in a set of non-dominated solutions instead of a single solution. The goal of multi-objective scheduling is to find a set of compromising schedules satisfying different objectives under consideration. For a given finite set of schedules generated by using a suitable algorithm for a multi-objective scheduling problem, various objective functions $f(x)=\{f_1(x), f_2(x), ..., f_k(x)\}$ can be evaluated. These schedules are to be compared and a set of schedules called *non-dominated solutions* are to be identified. For those solutions, no improvement in any objective function is possible without scarifying at least one of the other objective functions. Some researchers have developed multi-objective metaheuristics for solving flowshop scheduling problems (Pasupathy et al. 2006; Prabhaharan et al. 2005; Loukil et al. 2005; Suresh & Mohanasundaram, 2004; Hisao Ishibuchi et al. 2003; Ishibuchi & Murata, 1998; Sridhar & Rajendran, 1996). A survey of multi-objective scheduling problems is given by T'kindt & Billaut (2001). A multi-objective PSO algorithm has been proposed for minimizing weighted sum of makespan and maximum earliness (Prabhaharan et al. 2005). A Pareto archived simulated annealing algorithm for multi-objective scheduling has been proposed (Suresh & Mohanasundaram, 2004). Hisao Ishibuchi et al. (2003) proposed a modified multi-objective genetic local search algorithm (MMOGLS) for multi-objective FSP. They showed that the performance of the evolutionary multi-objective optimization algorithm can be improved by hybridization with local search. They apply multi-objective GA for PFSP and the results are compared with results published in the literature. Pasupathy et al. (2005) proposed a pareto-ranking based multi-objective GA called Pareto genetic algorithm with local search (PGA-ACS) algorithm for multi-objective FSP with an objective of minimizing the makespan and total flowtime. Loukil et al. (2005) proposed multi-objective simulated annealing algorithm to tackle the multi-objective production scheduling problems.

Pareto dominance: Among a set of schedules P, a schedule $x^1 \in P$ is said to *dominate* the other schedule $x^2 \in P$, denoted as $\left(x^1 \phi x^2\right)$, if both the following conditions are true.

(i) The schedule $x^1 \in P$ is no worse than $x^2 \in P$ in all objectives.

(ii) The schedule $x^1 \in P$ is strictly better than $x^2 \in P$ in at least one objective.

When both the conditions are satisfied, x^2 is called as a dominated schedule and x^1 a non-dominated schedule. If any of the above condition is violated, the schedule x^1 does not dominate the schedule x^2. Among a set of schedules P, the non-dominated set P' are those that are not dominated by any member of the set (Deb, 2003).

Non-dominated front: The set of all non-dominated schedules.

***Pareto optimal set*:** When the set P is the entire search space X, the resulting non-dominated set is called the Pareto optimal set.

The primary objective is to find a set of non-dominated fronts for the FSPs with the consideration of performance measures.

5.2 Proposed Multi-objective Discrete PSO Algorithm

The discrete PSO algorithm proposed for single objective FSP has been suitably modified to generate non-dominated solution set considering three performance measures simultaneously. Before presenting the proposed algorithm, the non-dominated sorting procedure, Pareto search procedure and the parameters considered are discussed below.

***Non- Domination Sorting*:** Non-domination measures are used to find non-dominated set of solutions. The following procedure is used to generate non-dominated particle or solution set from the population of particles. Consider a swarm consisting of N solutions (particles).

Step 0: Begin with $i = 1$; $j = i + 1$, and repeat steps 1 and 2.

Step 1: Compare solutions x^i and x^j for domination using the two conditions mentioned.

Step 2: If x^j is dominated by x^i, mark x^j as 'dominated,' increment j, and go to step 1. Otherwise mark x^i as dominated, increment i, set $j = i + 1$ and go to step 1.

All solutions that are not marked 'dominated' forms a non-dominated solution set and these are stored separately in a memory called archive.

```
Initialize the parameters
Generate the swarm and velocity
t = 0:  // iteration counter
Evaluate all the particles
Perform non-dominated sorting to identify $G_b^t$
Open Archive to store $G_b^t$

Do    {
          Update position;
          $t = t + 1$
          Evaluate
          Do non-dominated sorting to identify $G_b^t$
          Archive update
          Update velocity
}     while $(t < t_{max})$ :  $t_{max} = 100$;
Output $G_b^t$
```

Figure 6. Iterative search loop of the multi-objective discrete PSO algorithm

***Pareto Search*:** In case of a single objective scheduling optimization, an optimal solution forms the Global best (G_b^t). Under multi-objective scheduling, with multiple objectives, G_b^t consist of a set of non-dominated solutions. Once the swarm is initialized, $G_b\,(t = o)$ is obtained after non-dominated sorting of the particles. During the subsequent iterations, position and velocity update of the particles are carried out using local best and global best. It is to be noted that one solution is randomly chosen from the archive as Global best set. During every iteration, non-dominated solution set is updated. This non-dominated solution set is added with the Archive and the combined set is sorted for non-dominance. Dominated solutions within the combined set are removed and the remaining non-dominated solutions forms $G_b\,(t = 1)$. This procedure is repeated to guide the non-dominated search process towards the Pareto region. Initially, a set of particles are generated randomly and evaluated. Then the non-dominated sorting of particles is done. Within the swarm, the non-dominated solution set i.e. G_b^t is identified and they are stored in an archive. Then the positions and velocities of the particles are updated iteratively. These current sets of non-dominated solutions are combined with the archive solutions. Non-dominated sorting of archive is done to identify the archive survival members. This process is called Archive update. During this, all dominated members of the combined set are removed. This procedure is repeated to guide the non-dominated search process towards generating a solution front close to the Pareto region. After the termination criterion is met, the solution set stored in the archive forms the result. The iterative improvement process of multi-objective PSO algorithm is presented in Figure 6.

5.3 Performance of Multi-objective Discrete PSO Algorithm

In this section, the performance measures namely minimization of makespan, total flowtime and completion time variance are considered simultaneously. It is to be noted that PSO algorithm has been very rarely studied by researchers for solving FSPs with multi-objective requirements.

***Parameter Selection*:** Using the proposed algorithm, experiments are conducted to redesign the algorithm with appropriate parameter settings. Parameters were identified by trial and error approach for the better performance. The swarm size is taken as 80. The values of acceleration constants are fixed by trial and error as $C_1 = 1; C_2 = 2$ and $C_3 = 2$. The values of velocity coefficients U_1, U_2 and U_3 are generated randomly between 0 and 1. Termination criterion is taken as 100 iterations. The benchmark instances of Taillard (1993) form a set of 120 problems of various sizes, having 20, 50, 100, 200 and 500 jobs and 5, 10 or 20 machines have been taken and solved. When the iterative search process is continued beyond 100 iterations, solution quality is expected to improve further and the non-dominated front will converge towards the Pareto front. Some samples of non-dominated solution sets obtained during 1st, 50th and 100th iterations of selected benchmark FSPs are presented in Table 6. to Table 10.

1st Iteration			50th Iteration			100th Iteration		
C_{max}	F_Σ	V_T	C_{max}	F_Σ	V_T	C_{max}	F_Σ	V_T
2372	37335	143185.03	2418	37282	163238.72	2380	37749	121770.54
2385	37379	134831.95	2450	37645	139131.23	2395	37465	130522.04
2410	36900	148013.59	2451	38181	137003.64	2458	37187	210477.03
2412	37674	129799.71	2495	36838	186985.58	2465	37341	187537.33
2412	36970	138733.95	2518	39668	127576.64	2488	36988	148466.05
2414	36786	157977.52	2544	36566	258462.42	2493	36787	244247.03
2425	36842	155318.20	2550	36352	180610.66	2518	36639	213526.66
2432	36071	225477.25	2633	37206	175815.11	2545	36177	189031.61
2437	36855	150071.23						
2448	37604	125025.85						
2451	36764	158552.28						
2451	36600	172676.80						
2464	37521	134748.27						
2468	37875	124452.44						
2480	39012	119837.64						
2491	36170	154730.75						
2523	38802	123177.59						

Table 6. Non-dominated fronts obtained for 20 x 20 FSP (Problem ta025 of Taillard,1993)

1st Iteration			50th Iteration			100th Iteration		
C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T
3840	127915	674615.63	4168	138549	801359.25	4192	143728	688058.06
3923	132364	655699.19	4170	139913	794893.25	4218	142073	835741.13
3979	130656	669600.50	4181	140250	769993.81	4226	136757	870648.81
3979	132435	633633.38	4188	138913	756248.50	4241	140962	788543.19
3982	132026	666358.94	4243	137007	882535.81	4245	138443	845496.63
4018	136354	604771.06	4254	141017	750998.31	4266	137938	828836.88
4023	132426	646723.94	4284	136183	929310.25	4298	137356	866164.31
4034	135781	631409.19	4290	137714	833303.44	4324	143038	776172.63
4058	131370	652795.69	4295	135927	845500.88	4329	143586	760850.94
4081	137607	586079.44	4319	142649	731565.19	4334	141675	780154.75
4084	136148	601373.06	4320	140119	747898.00	4343	136398	868004.75

Table 7. Non-dominated fronts obtained for 50 x 20 FSP (Problem ta055 of Taillard,1993)

1st Iteration			50th Iteration			100th Iteration		
C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T
6719	414626	2332780.00	7079	442243	2714971.00	6977	429237	2643600.00
6736	407661	2339133.25	7122	431015	2619110.50	7187	429079	2992237.75
6754	407217	2426269.50	7125	430238	2888681.25	7222	423655	3181877.50
6759	414920	2322475.00	7279	427670	3036344.25	7266	427705	3032460.25
6772	421227	2319961.50	7307	426737	3014873.00	7287	426588	3061585.25
6776	420444	2215965.00						
6780	406735	2308902.00						
6785	417764	2299484.50						
6804	417373	2165440.25						
6934	402802	2477583.00						

Table 8. Non-dominated fronts obtained for 100 x 20 FSP (Problem ta085 of Taillard,1993)

1st Iteration			50th Iteration			100th Iteration		
C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T
11883	1341992	8681969.00	12169	1370395	8968974.00	12213	1382492	9226709.00
11922	1378165	8301979.00				12246	1418388	8839896.00
11935	1361242	8654574.00				12304	1390924	9191086.00
11938	1365058	8581394.00				12361	1380781	9530417.00
11964	1363602	8492216.00				12445	1379004	9589141.00
11995	1355612	8551758.00						
12020	1371423	8237680.50						
12051	1369441	8470111.00						
12115	1354810	8405068.00						

Table 9. Non-dominated fronts obtained for 200 x 20 FSP (Problem ta105 of Taillard,1993)

1st Iteration			50th Iteration			100th Iteration		
C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T	C_{max}	F_{Σ}	V_T
27361	7380460	53524660.00	27802	7498389	54440864.00	27612	7421436	53180528.00
27417	7405289	51892856.00	27811	7402468	53268448.00	27765	7458248	53042776.00
27448	7419382	51504108.00	27999	7543786	53059836.00	27870	7440681	53140668.00
27465	7394286	52016468.00	28091	7529455	52754652.00	27891	7374759	53306856.00
27534	7392887	51930096.00						
27593	7458730	51066888.00						
27603	7373445	51681608.00						
27638	7439401	51390116.00						
27680	7445450	51262332.00						
27700	7418177	51122680.00						
27729	7492150	51039416.00						

Table 10.Non-dominated fronts obtained for 500 x 20 FSP (Problem ta115 of Taillard,1993)

Normalized values of the performance measures are plotted for better visualization. Some samples of non-dominated front obtained during 1st, 50th and 100th iterations of selected benchmark FSPs are presented in Fig. 7. to Fig. 11.

6. Conclusion

Literature survey indicates that very few authors have studied the applications of multi-objective scheduling in flowshop scheduling using particle swarm optimization algorithm is scarce. This Chapter presents a discrete PSO algorithm to solve FSPs. This work has been conducted in two phases. In the first phase, a discrete PSO is proposed to solve the single-objective FSPs. In the second phase, a multi-objective discrete PSO algorithm is proposed to solve the FSPs with three objectives. The performance of the proposed single-objective discrete PSO is tested by solving a large set of benchmark FSPs. The quality measure namely "Average Relative Percent Deviation" ($\overline{\text{RPD}}$) is used to compare the solution quality obtained with the results available in the literature. It shows that the proposed discrete PSO algorithm performs better in terms of quality of results. Using the proposed algorithm, experiments are conducted to redesign the algorithm with appropriate parameter settings. The $\overline{\text{RPD}}$ for each set of instances are also shown in an efficient way. The parameters selected for solving the problems are holds good. The proposed multi-objective discrete PSO algorithm performs better in terms of yielding more number of non-dominated solutions close to Pareto front during the search. It is seen that, when the number of iterations is more, the non-dominated solution set generated is close to the Pareto front.

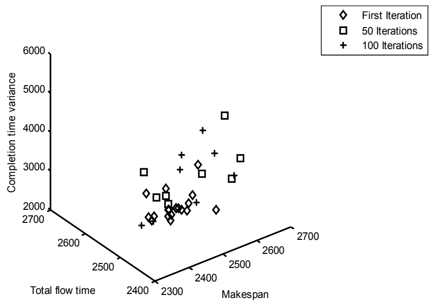

Figure 7. Non-dominated solution set obtained for 20 x 20 FSP (Problem ta025 of Taillard,1993)

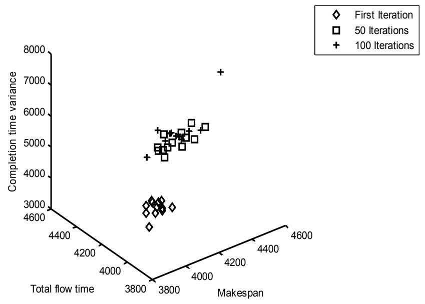

Figure 8. Non-dominated solution set obtained for 50 x 20 FSP (Problem ta055 of Taillard,1993)

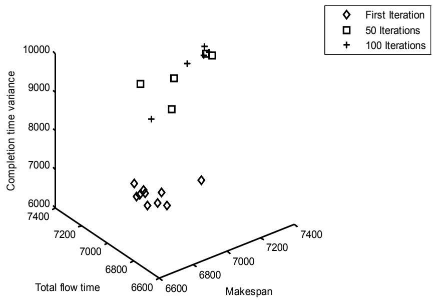

Figure 9. Non-dominated solution set obtained for 100x20 FSP (Problem ta085 of Taillard,1993)

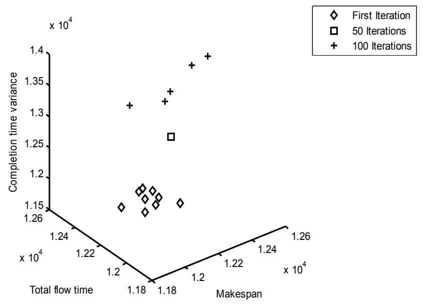

Figure 10.Non-dominated solution set obtained for 200x20 FSP (Problem ta105 of Taillard,1993)

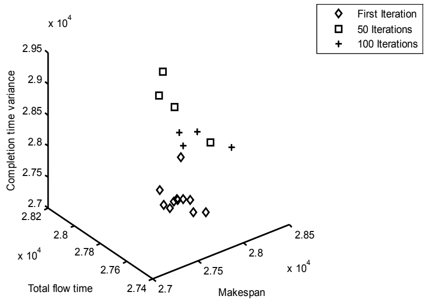

Figure 11.Non-dominated solution set obtained for 500x20 FSP (Problem ta115 of Taillard,1993)

7. References

Abido, M.A. (2002). Optimal power flow using particle swarm optimization. *Electrical Power and Energy Systems*, Vol.24, 563-571

Bagchi, T.P. (1999). *Multi-objective scheduling by Genetic Algorithms*, Kluwer Academic Publishers, Boston, Massachusetts

Baker, K.R. (1974). *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York

Brucker, P. (1998). *Scheduling Algorithms*, Springer-Verlag, Berlin

Campbell, H.G.; Dudek, R.A. & Smith, M.L. (1970). A heuristic algorithm for the n job, m machine sequencing problem, *Management Science*, Vol.16, No: 10, B630-B637

Chan, F.T.S.; Wong, T.C. & Chan, L.Y. (2005). A genetic algorithm based approach to machine assignment problem, *International Journal of Production Research*, Vol.43, No: 12, 2451-2472

Chandrasekaran, S.; Ponnambalam, S.G.; Suresh, R.K. & Vijayakumar N. (2006). An Application of Particle Swarm Optimization Algorithm to Permutation Flowshop Scheduling Problems to Minimize Makespan, Total Flowtime and Completion Time Variance, *Proceedings of the IEEE International Conference on Automation Science and Engineering, 2006 (CASE '06.)*, pp-513-518, ISBN: 1-4244-0311-1, Shanghai, China,

Chandrasekharan Rajendran. & Hans Ziegler. (2005). Two Ant-colony algorithms for minimizing total flowtime in permutation flowshops, *Computers & Industrial engineering*, Vol.48, 789-797

Christian Blum. & Andrea Roli. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, Vol. 35, No. 3, 268-309

Dannenbring, D.G. (1977). An evaluation of flowshop sequencing heuristics, *Management Science*, Vol.23, No: 11, 1174-1182

Dudek, R.A.; Panwalkar, S.S. & Smith, M.L. (1992). The lessons of flowshop scheduling research, *Operations Research*, Vol.40, No: 1, 7-13

Eberhart, R.C. & Hu, X. (1999). Human tremor analysis using particle swarm optimization. *Proceedings of the Congress on Evolutionary Computation,* pp-1927-1930, IEEE Service Center, Washington, DC, Piscataway, NJ

Eberhart, R.C. & Kennedy J. (1995). A New Optimizer Using Particles Swarm Theory, *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp-39-43, IEEE Service Center, Nagoya, Japan

Eberhart, R.C. & Shi, Y. (2001). Particle swarm optimization: developments, applications and resources, *Proceedings of IEEE Congress on Evolutionary Computation 2001*, Seoul, Korea

Faith Tasgetiren, S.; Mehmet Sevkli.; Yen-Chia Liang. & Gunes Gencyilmaz. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem, *IEEE Transaction on Power and Energy Systems*, 1412-1419

Framinan, J.M. & Leisten, R. (2003). An efficient constructive heuristic for flowtime minimization in permutation flowshops, *Omega,* Vol.31, 311-317

French, S. (1982) *Sequencing and Scheduling: An introduction to the mathematics of the jobshop*, Ellis Horword Limited, Chichester, England

Gowrishankar, K.; Rajendran, C. & Srinivasan, G. (2001). Flowshop scheduling algorithms for minimizing the completion time variance and the sum of squares of completion time deviation from the common due date, *European Journal of Operational Research*, vol.132, No: 31, 643-665

Ignall, E. & Scharge, L. (1965). Application of the branch and bound technique to some flowshop-scheduling problems, *Operations Research*, Vol.13, 400-412

Ishibuchi, H.; Yoshida, T. & Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multi-objective permutation flowshop scheduling, *IEEE Transaction on Evolutionary Computation,* Vol.7 No.2, 204-223

Johnson, S.M. (1954). Optimal two-stage and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, Vol.1 61-68

Kalyanmoy Deb. (2003). *Multi-objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, First Edition.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks-IV,* pp-1942-1948, Piscataway, NJ: IEEE service center, Perth, Australia

Kennedy, J.; Eberhart, R. & Shi, Y. (2001). *Swarm Intelligence*, Morgan Kaufmann, San Mateo,CA,USA

Liu, J. & Reeves, C.R. (2001). Constructive and composite heuristic solutions to the $P//\sum C_i$ scheduling problem, *European Journal of Operational Research.*, Vol.132, 439-452

Lixin Tang. & Jiyin Liu. (2002). A modified genetic algorithm for the flowshop sequencing problem to minimize mean flowtime, *Journal of Intelligent Manufacturing*, Vol.13, 61-67

Loukil, T.; Teghem, J. & Tuyttens, D. (2005). Solving multi-objective production scheduling problems using metaheuristics, *European Jour. of Operational Research*, Vol.161, 42-61

Merkle, D. & Middendorf, M. (2000). An ant algorithm with new pheromone evaluation rule for total tardiness problems, *Proceedings of the Evolutionary Workshops 2000*, pp-287-296, vol.1803, Lecture Notes in Computer Science, Springer

Moccellin, J.V. & Nagano, M.S. (1998). Evaluating the performance of tabu search procedures for flowshop sequencing, *Journal of the Operational Research Society*, Vol.49, 1296-1302

Nawaz, M.; Enscore Jr, E.E. & Ham, I. (1983). A Heuristic algorithm for the m-machine, n-job scquencing problem, *Omega*, Vol.11, 91-98

Ogbu, F.A. & Smith, D.K. (1990). The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem, *Computers and Operations Research*, Vol.17, No: 3, 243-253

Osman, I.H. & Laporte, G. (1996). Metaheuristics: A bibliography. *Operations Research,* Vol.63, 513–623

Palmer, D. (1965). Sequencing jobs through a multi-stage process in the minimum total time- a quick method of obtaining a near optimum, *Opn. Research*, Vol.16, No: 1, 101-107

Pasupathy, T.; Chandrasekharan Rajendran. & Suresh, R.K. (2006). A multi-objective genetic algorithm for scheduling in flowshops to minimize makespan and total flowtime, *International Journal of Advanced Manufacturing Technology*, Springer-Verlag London Ltd, Vol.27, 804-815

Pinedo, M. (2002). *Scheduling: Theory, Algorithms and Systems*, Second edition,. Prentice-Hall, Englewood Cliffs, New Jersey

Prabhaharan, G.; Shahul Hamid Khan, B.; Asokan, P. & Thiyagu M. (2005). A Particle swarm optimization algorithm for permutation flowshop scheduling with regular and non-regular measures, *International Journal of Applied Management and Technology*, Vol.3, No: 1, 171-182

Rajendran, C., (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime, *International Journal of Production Economics*, Vol.29, 65-73

Rameshkumar, K.; Suresh, R.K. & Mohanasundaram, K.M. (2005). Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan, *Lecture Notes in Comp. Science,* Springer Verlag-GMBH.0302-9743. Vol.3612

Rinnooy Kan, A.H.G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations,* Nijhoff, The Hague

Ruben Ruiz. & Concepcion Maroto. (2005). A comprehensive review and evaluation of permutation flowshop heuristics, *European Journal of operational Research*, Vol.165, 479-494

Ruben Ruiz.; Concepcion Maroto. & Javier Alcaraz. (2004). Two new robust genetic algorithms for the flowshop scheduling problem, *OMEGA*, 2-16

Sensarma, P. S. ; Rahmani, M. & Carvalho, A. (2002). A comprehensive method for optimal expansion planning using particle swarm optimization, *IEEE Power Engineering Society Winter Meeting*, Vol. 2, .1317-1322

Sridhar, J. & Rajendran, C. (1996). Scheduling in flowshop and cellular manufacturing system with multiple objectives - A genetic algorithmic approach, *Production Planning and Control*, Vol.74, 374-382

Stuetzle, T. (1998). An ant approach for the flowshop problem, *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing* (EUFIT '98), pp-1560-1564, Vol.3, Verlag Mainz, Aachen, Germany

Suresh, R.K, & Mohanasundaram, K.M, (2004). Pareto archived simulated annealing for permutation flowshop scheduling with multiple objectives, *Proceedings of the IEEE Conference on Cybermatics and Intelligent Systems*, pp-1-3, Singapore

Taillard, E. (1993). Benchmarks for basic scheduling problem, *European Journal of Operational Research*, Vol.64, 278-285

Tandon, V. (2000). Closing the gap between CAD/CAM and optimized CNC end milling. *Master's thesis*, Purdue School of Engineering and Technology, Indiana University ,Purdue University, Indianapolis.

Yoshida, H.; Kawata, K.; Fukuyama, Y. & Nakanishi, Y. (1999). A particle swarm optimization for reactive power and voltage control considering voltage stability. *Proceedings of the International Conference on Intelligent System Application to Power Systems,* pp-117-121, Rio de Janeiro, Brazil

Yuhui Shi. (2004). Particle Swarm Optimization, *IEEE Neural Networks Society*, 8-13

Yuvraj Gajpal. & Chandrasekharan Rajendran. (2006). An ant-colony optimization algorithm for minimizing the completion time variance of jobs in flowshops, *International Journal of Production Economics*, Vol. 101, No: 2, 259-272