4-2015

Discrete Particle Swarm Optimization for Flexible Flow Line Scheduling

Parastoo Amiri
Clemson University

DISCRETE PARTICLE SWARM OPTIMIZATION FOR FLEXIBLE FLOW LINE SCHEDULING

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Industrial Engineering

---

by
Parastoo Amiri
April 2015

Accepted by:
Dr. Mary E. Kurz, Committee Chair
Dr. Scott J. Mason
Dr. Amin Khademi

ABSTRACT

Previous research on scheduling flexible flow lines (FFL) to minimize makespan has utilized approaches such as branch and bound, integer programming, or heuristics. Metaheuristic methods have attracted increasing interest for solving scheduling problems in the past few years. Particle swarm optimization (PSO) is a population-based metaheuristic method which finds a solution based on the analogy of sharing useful information among individuals. In the previous literature different PSO algorithms have been introduced for various applications. In this research we study some of the PSO algorithms, continuous and discrete, to identify a strong PSO algorithm in scheduling flexible flow line to minimize the makespan. Then the effectiveness of this PSO algorithm in FFL scheduling is compared to genetic algorithms.

Experimental results suggest that discrete particle swarm performs better in scheduling of flexible flow line with makespan criteria compared to continuous particle swarm. Moreover, combining discrete particle swarm with a local search improves the performance of the algorithm significantly and makes it competitive with the genetic algorithm (GA).

DEDICATION

I would like to dedicate this thesis to my mother, Fahimeh Farid, my father, Kioumars Amiri and my uncle, Dr. Mohammad Ali Farid for all their love and support.

ACKNOWLEDGMENTS

This project would not have been possible without the guidance and help of several individuals who in one way or another contributed and extended their valuable assistance in preparation and achievement of this research. First, my utmost gratitude to Dr. Mary Elizabeth Kurz, my project advisor, whose encouragement, guidance and support from the initial to the final level enabled me to complete this study. I would also like to thank Dr. Scott Mason and Dr. Amin Khademi, the members of the project committee, who have provided useful feedback and without their knowledge and assistance this study would not have been successful. Lastly, I would like to thank Clemson Computing team, especially, Dr. Marcin Ziolkowski for providing the support and equipment I have needed to develop and complete my project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued) Page

CHAPTER 1

INTRODUCTION

Scheduling problems have been the subject of much research for many years. They can be found wherever there are some tasks which should be assigned to some resources. In a manufacturing environment, scheduling is done with regard to different objectives such as minimizing flow time, tardiness, lateness or makespan.

Makespan is the maximum completion time of all jobs. Minimizing makespan is important because it tends to increase the facility utilization. Minimizing makespan has been of great interest in both job shops and flow shops.

In a flow shop, jobs follow the same path from one machine to another (Figure 1.1) while in a job shop there is no common pattern of movement from machine to machine (Figure 1.2). Each job is processed by at most one machine in each stage. The machines available at each stage are identical.

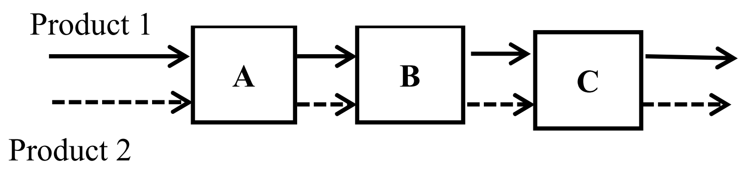


Figure 1.1. Flow shop

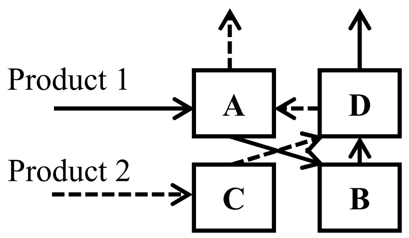


Figure 1.2. Job shop

More than one machine might be available in each stage of a flow line, in which case it is called a hybrid flow line. One special case of a hybrid flow line is when the jobs are allowed to skip some stages, which is called a flexible flow line (Figure 1.3). It can be seen in industries such as automobile or printed circuit boards where there is no need for some jobs to visit all stages, but they keep the same linear path.

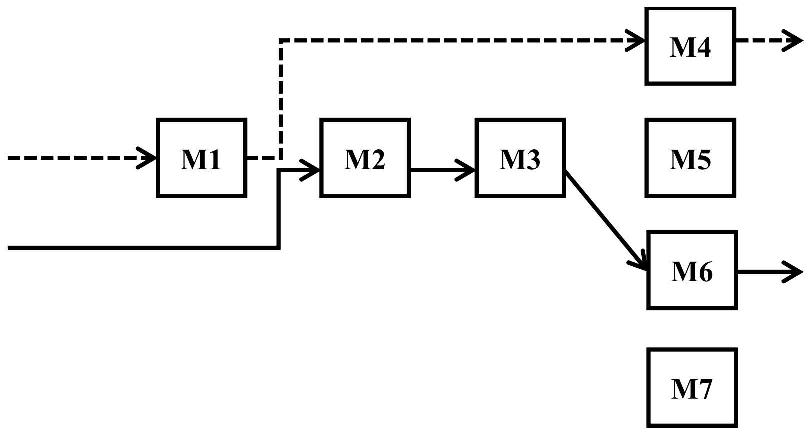


Figure 1.3. Flexible flow line

There are different approaches to scheduling flexible flow lines, such as branch and bound (Salvador, 1973; Brah and Hunsucker, 1991; Carlier and Neron, 2000), or integer programming (Sawik, 2002; Kurz and Askin, 2004). It is proved that even the two-stage flow shop scheduling problem with parallel machines to minimize makespan is a NP-hard problem (Gupta, 1988). Accordingly heuristics have been used to solve these kinds of problems. Heuristics are experienced-based techniques which try to find a solution which is not guaranteed to be optimal. They are divided into two main categories: constructive heuristics which produce initial solution and improvement heuristics which improve the solution by using search techniques. In scheduling, a constructive heuristic starts without a schedule or job sequence and then adds one job at a time to find the solution, while improvement heuristics, such as metaheuristics, use an initial schedule, and then try to find a better "similar" schedule, referred to as improved solution. Metaheuristics such as tabu search (TS), simulated annealing (SA), genetic algorithms (GA) and particle swarm optimization (PSO) are based on local search techniques. Particle swarm optimization is a population-based metaheuristic method introduced by Kennedy and Eberhart (1995) which has been recently the focus of some articles dealing with scheduling problems. It has also been applied to different NP-hard problems such as traveling salesman problem, lot sizing, etc. The potential merit of PSO over other metaheuristics is its ability to find solutions based on social behavior of sharing useful information among individuals.

In this chapter, after reviewing some literatures which are available in this area, the problem considered in this study and the objective of this research will be discussed.

1.1 Literature Review

This literature review is focused on the following topics:

- Particle Swarm Optimization
- Flow shop scheduling
- Applying particle swarm optimization in scheduling problems

1.1.1 Standard Particle Swarm Optimization

Kennedy and Eberhart (1995) introduced standard PSO for continuous optimization problems. Particle swarm optimization is a population based metaheuristic which is inspired from bird flocking and fish schooling, searching for food. Various PSO algorithms are introduced for continuous and discrete solution spaces. In this algorithm, particles fly through the solution space by learning from the historical information that they gain from the swarm population.

Each particle has its own velocity and it has a memory of the best solution which has been found by itself (*pbest*) and by the swarm (*gbest*). Figure 1.4 shows how particle *i* changes its position from time *t* (x_i^t) to *t+1* (x_i^{t+1}) based on its trust in its own experience at time *t* (v_i^t), its neighbor experience (p_i^t) and the whole swarm experience (g_i^t).

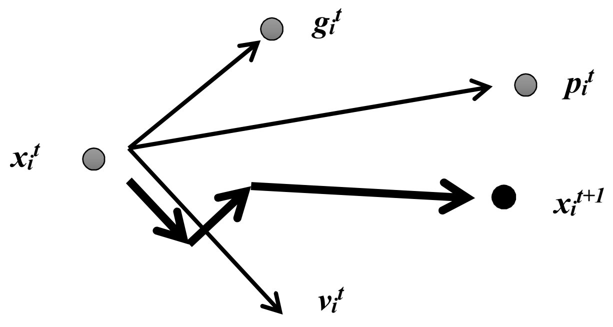


Figure 1.4. Particle Motion (Clerc, 2004)

1.1.2 Discrete PSO

In a standard PSO positions are real valued, so it cannot be applied directly to binary/discrete space. Efforts have been made to adapt this algorithm for discrete solution space. Kennedy and Eberhart introduced the discrete binary version of PSO with a stochastic velocity model in 1997, which was the first PSO algorithm to be used in discrete space. They were motivated by the idea that any problem, discrete or continuous, can be expressed in a binary notation so an optimizer with binary representation can be advantageous. In the discrete binary version of PSO, the position x_i^t can only be zero or one and v_i is the probability that x_i changes to state 0 or one. This probability is computed as:

$$s(v_i^t) = \frac{1}{1 + \exp(v_i^t)} \tag{1.1}$$

Then x_i^t can be defined as:

$$\begin{cases} 1 & if\ random\ number\ < s(v_i^t), \\ 0 & Otherwise, \end{cases} \tag{1.2}$$

1.1.3 Discretization Methods

An appropriate representation of particle position is needed in order to use PSO for discrete problems. The sets of real variables in original PSO which represent particle position have to be discretized in order to be applied to discrete problems. Krause et al. (2013) characterize the codification of candidate solutions in three encoding schemes:

1- Binary codification (BC) for candidate solutions.
2- Integer codification (IC) for candidate solutions.
3- Using transformation methods to transform real values into a BC (real-to-binary: RTB) or an IC (real-to-integer: RTI), where RTI represents a combination of integer values. These transformations have to be done at each iteration loop.

They also categorize the discretization methods which are used in literature as follow:

- *Sigmoid Function*

  The Sigmoid function transforms a continuous space value into a binary one. The transformation is applied to each dimension of the position vector:

$$\begin{cases} 1 & if\ random\ number\ < \dfrac{1}{1+\exp(v_i^t)}, \\ 0 & Otherwise, \end{cases}$$

  where *i* is the index of population size. The random number is drawn uniformly from [0,1].

- *Random-Key*

  The random-key (RK) transforms a continuous space value into an integer/combinatorial value. To decode the position the nodes are visited in ascending order for each dimension. e.g: *x*=(0.90, 0.35, 0.03, 0.21, 0.1) → *x*=(3, 5, 4, 2, 1). (Bean et al., 1994; Kurz and Askin, 2004). This method will be discussed in detail in Chapter 2.

- *Smallest Position Value*

  The smallest position value (SPV) transforms a continuous space value into an integer value. The smallest position value method maps the positions of the solution vector by placing the index of the lowest valued component as the first item on a permutated solution, the next lowest as the second, and so on. This method creates an integer vector solution by indexing the position of all the particles (Tasgetiren et al., 2004).

- *Modified Position Equation*

  Pan et al. (2008) and Tasgetiren et al. (2007) use the modified position equation (MPE) method to update the positions of particles in PSO algorithm. The details of this method is available in Chapter 2.

- *Great Value Priority*

  Congying et al. (2011) use the great value priority (GVP) method to transform a continuous space into a binary space. First, the position of the solution x_i^t with the largest element is selected, where i=1,.., N and N is the population size. This position is set on the first position of a new vector named as permutation vector p. Next, the position of the second largest element of x_i^t is selected and placed in the next position of p. This procedure is repeated successively for all dimensions of x_i^t and once permutation vector p is fulfilled, following equation is applied to transform it into binary, where j=1, . . . , D and D is the dimension size:

$$x_{ij}^t = \begin{cases} 1, & if\ p_j > p_{j+1}, \\ 0, & otherwise. \end{cases}$$

- *Nearest Integer*

  In this method, a real value is converted to the nearest integer (NI) by rounding or truncating up or down (Burnwal and Deb, 2012).

1.1.4 Modified PSO

In order to use PSO in a permutation problem, the standard PSO should be modified. In permutation problem elements of a position are not independent, while in standard PSO elements are independent so two elements can have the same value. This conflict is not accepted in permutation problems.

Hu et al. (2003) introduced a new velocity and particle update to handle the permutation parameter set. They use this algorithm to solve the n-queen problem**.** The velocity is defined as the possibility that a position changes, in other words, the probability that each particle swaps is equal to the value of the velocity. The mutation factor is also used to update the positions when they are identical to *gbest*. Their objective is minimizing the number of diagonal conflicts. The results are compared to the result of the same problem which was solved using a GA.

One of the main problems of original PSO on strongly multi-modal test problems is its premature convergence due to loss of diversity in search space. Convergence happens when the system or process reaches a stable state. Based on the definition by Van den Bergh (2002), in flow shop scheduling problem, convergence is written as:

$$\lim_{t\to\infty} gbest(t) = gbest^*$$

where *gbest*(*t*) is best position found in time *t* or in *t*th generation, *gbest** is a fixed position in the solution space. It implies that, if *gbest* does not change after some point in time, then convergence is achieved. If *gbest* is the global best position, then the algorithm attains the global best convergence. Otherwise, the algorithm is stuck in a local optima. Of course, the true optimal solution is not known, so if *gbest* is not the optimal solution, premature convergence has occurred.

Premature convergence may happen because of fast information flow between particles. In this case, the swarm may converge to a local solution and may not be able to explore the search space thoroughly. Figure 1.6 shows a local minimum x_1, if the algorithm converges at this point, the better solution x_2 will be screened out. So there is a need to improve the exploration of this algorithm in order to avoid sub-optimal solutions more frequently.

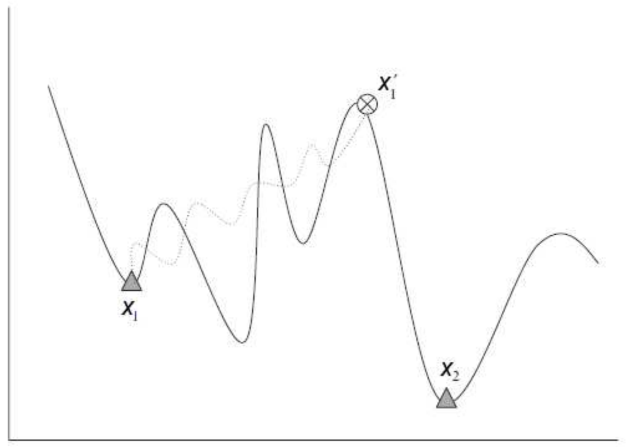
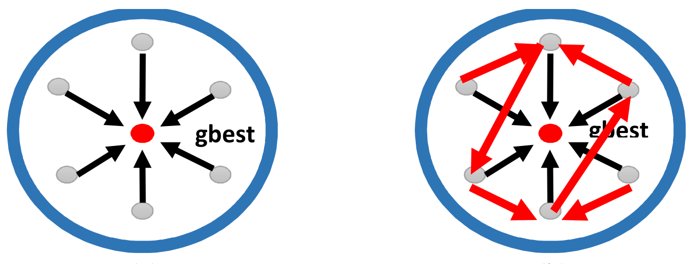


Figure 1.6. x_1 is the current solution, x_2 is another solution which is better than x_1, in order to avoid premature convergence, x_1 should change into an intermediate solution x_1'(Liao et al, 2007).

Riget and Vesterstrøm (2002) propose an algorithm based on attraction and repulsion between particles. They define a critical value for diversity ($d_{critical}$). When the diversity is less than $d_{critical}$, particles repel each other and when the calculated diversity is above this value, they attract each other.

They test this new algorithm on four standard multi-modal objective functions. The results are competitive with the GA algorithm and better than original PSO. Dallard et al. (2007) use this algorithm to solve an orienteering problem successfully.

He et al. (2004) propose a new algorithm by considering passive congregation in the velocity update. The idea is each particle in an aggregation has lots of potential useful information that may help them to reach to optimal solutions. Figure 1.7 shows the interaction between particles in SPSO (1.7a) and passive congregation algorithm (1.7b).



(a) (b)

Figure 1.7. (a) Interaction of particles in Standard PSO, (b) Interaction of particles with passive congregation (He et al., 2004)

Ho et al. (2005) claim that cognitive and social behavior of SPSO are not completely independent; as in human decision making, the personal best may overcome the social best. They modify the velocity equation in order to improve the exploration and exploitation behavior of the swarm (Eq. 1.3). A random number (r_1) is used to control these two parts and random number r_2 is used to balance between global and local searches. The value s_3 is used in order to increase the diversity. The values c_1 and c_2 are cognitive and social parameters, respectively.

$$v_i^{t+1} = s_3 r_2 v_i^t + (1 - r_2) c_1 r_1 \left(p_i^t - x_i^t\right) + (1 - r_2) c_2 (1 - r_1) \left(g_i^t - x_i^t\right) \quad (1.3)$$

$$s_3 = \begin{cases} 1 & random\ number > 0.05, \\ -1 & Otherwise. \end{cases}$$

Zhang et al. (2010) suggest a discrete algorithm called circular discrete particle swarm optimization (CDPSO). They address the premature convergence of PSO algorithm by considering the swarm activity and the similarity of particles in each iteration. The swarm activity is small when the algorithm is trapped into local optimum, in order to escape from this situation, the mutation is used to send the particles to a new search area. Decreasing the diversity of the swarm leads to increase of the similarity of particles, so by calculation the similarity, it is possible to prevent the premature convergence. They use this algorithm to obtain the minimum makespan in flow shop scheduling.

Chen and Yangmin Li (2007) propose a modified PSO with controllable random exploration velocity (PSO-CREV) added to the velocity updating in order to balance exploration behavior and convergence rate with respect to different optimization problems. They use various benchmarks to evaluate this algorithm.

Sevkli and Sevilgen (2010) improve the PSO algorithm by considering both exploration and exploitation. They propose a new algorithm by modifying the update method of the best particle in the swarm (the pioneering particle). They strengthen the exploitation mechanism by using Reduced Variable Neighborhood Search (RVNS) and at the same iteration random velocity is used to improve the exploration mechanism. The proposed algorithm is successfully tested on discrete and continuous problems. The results in both cases were competitive or even better than previous results, e.g. their results for orienteering problem were better than the published results by Dallard et al. (2007).

M. R. Singh et al. (2013) uses a chaotic mutation operator in order to overcome the problem of trapping at local minima in standard PSO algorithm. The Chaotic sequence using logistic mapping is used instead of random numbers to improve the diversity in solution space.

1.1.5 Flexible Flow Line

A flexible flow line is a manufacturing system where multiple machines can exist in each stage, each job must be processed by at most one machine at each stage and jobs can skip some stages. If there is only one machine at each stage and jobs have to meet all the stages, this line is the flow shop line.

Many works have been done in scheduling of flexible flow shops. Kurz and Askin (2003) compare various scheduling rules in flexible flow line scheduling. They categorize the previous works based on their approaches, such as branch-and-bound, extensions of previous techniques

(Johnson's rule etc.), applying metaheuristics and development of new techniques. They apply eight constructive heuristics to minimize the makespan in the systems with more than two stages and various configurations of machines. These heuristics are being compared using the value of (makespan-lower bound)/lower bound.

There might also some setup times involved in scheduling of flexible flow line. The setup usually corresponds to preparing the machines for the execution of the next job and when the setup time depends on the previous job which has completed on the machine, the setups is sequence dependent. Kurz and Askin (2004) tackle the scheduling of flexible flow line with sequence dependent setup times by applying random keys genetic algorithms in order to minimize the makespan. They also develop a strong lower bound for this problem. This lower bound shows that the makespan is at least as large as the longest completion time, considering the setup time which is assumed to be the shortest setup possible. This research shows that in the case where more than two stages are considered, the genetic algorithm outperformed the procedures presented by their previous work (Kurz and Askin, 2003).

Tavakkoli-Moghaddam et al. (2007) consider a flexible flow line problem with blocking processor (FFLB). They proposed a queen-bee-based genetic algorithm to schedule flexible flow lines. They also apply memetic algorithm (MA) along with using a local search (Tavakoli Moghadam, 2009), namely, nested variable neighborhood search (NVNS), to minimize the makespan. It is claimed that this algorithm outperforms the classical genetic algorithm. Kia et al. (2009) use simulation to investigate dynamic scheduling of flexible flow lines with sequence dependent setup times. Scheduling of flexible flow lines with unrelated parallel machine is addressed in Zandieh et al (2010). They apply GA in order to solve this problem and also try to consider the constraints which exist in real world scheduling. Karmakar and Mahanty (2010) apply genetic algorithm and the theory of constraints to solve a mixed integer linear program for a flexible flow line in a paint factory with makespan criteria. Shahvari et al. (2011) develop a mixed-integer linear programming model for the flexible flow shop sequence dependent group scheduling problem, then they apply six metaheuristics based on tabu search (TS) to solve this problem. Sawik (2011) address the deterministic cyclic and batch scheduling problem in flexible flow lines with continuous and limited machine availability to schedule the jobs so that they are completed in the shortest possible time. He develops a mixed-integer programming model for these problems and compares the computational results of the models.

Particle swarm optimization (PSO) is also applied in scheduling of flexible flow lines to minimize the makespan (Sankaran, 2009; Singh et al., 2013). Sankaran (2009) applies the original PSO algorithm with random keys as a representation, but the results indicate that this algorithm does not outperform GA in minimizing the makespan. Singh et al. (2013) modify the original algorithm by using chaotic numbers and mutation operator to increase the diversity of the solution space. They claim that their algorithm outperforms GA for the same problem.

1.1.6 PSO in scheduling

There are papers which address the PSO algorithm in scheduling problems. This algorithm has been applied to some scheduling environments such as no-wait flow shop scheduling (Pan et al., 2008a and 2008b), permutation flow shop (Tasgetiren et al., 2007; Lian et al., 2006), parallel batch processing machines (Damodaran et al, 2012), sequence dependent disassembly line (Kalayci and Gupta, 2013), single machine (Tasgetiren et al, 2004; Anghinolfi and Paolucci, 2009), flow shop (Liao et al, 2007; Zhang et al., 2010), hybrid flow shop (Tseng and Liao, 2008) and flexible flow shop (Sankaran, 2009; Singh et al., 2013)

Tasgetiren et al. (2004) use continuous PSO to minimize total weighted tardiness on single machine. They use each dimension to represent the number of jobs and Smallest Position Value (SPV) rule and random keys representation, are used to sort the dimensions and transform the particle positions into job permutations. Later they minimize makespan and maximum lateness of jobs using the same technique and utilize variable neighborhood search (VNS) as a local search method in permutation flow shop sequencing problem (Tasgetiren et al., 2007).

The research by Tasgetiren et al. is an extension of continuous PSO. Pan et al. (2008a) reported the first DPSO algorithm to solve no-wait flow shop scheduling using a new position update method based on discrete job permutation. They use a new crossover (PTL crossover) to produce a pair of different permutation even from two identical parents. In this method, a block of jobs is chosen by two-cut points randomly and then it is moved to one side of solution vector and at the end the new permutation is filled with the remaining jobs from the other particle. They also use several speed-up methods for the Swap and Insert neighborhood structures. Finally they use variable neighborhood (VND) local search to improve the DPSO algorithm.

Anghinolfi and Paolucci (2009) proposed new particle swarm optimization approach to solve the total weighted tardiness scheduling problem with sequence dependent setup times on a single machine. As Tasgetiren et al. (2004), they use permutation solution-particle as a representation and they create a list of moves to update the particles' positions. Results from implementing the proposed DPSO to Cicirello's benchmark were satisfactory.

Lian et al. (2006) propose a new approach called similar particle swarm optimization algorithm (SPSOA) inspired from mutation and crossover which are used in genetic algorithm. They use these operators to update the velocity and position in order to minimize the makespan in permutation flow shop problem. The results demonstrate that SPSOA performs better than GA.

Liao et al. (2007) extend the binary PSO algorithm which is proposed by Kennedy and Eberhart (1997) by using similar approaches as Tasgetiren et al. (2004), in order to solve flow shop scheduling problems. Velocity update equation is the same as original PSO but they redefine the velocity as how likely job j is to be placed in the k^{th} position. They compare their algorithm to the continuous PSO algorithm proposed by Tasgetiren et al. (2004) and two genetic algorithms, results show that the proposed algorithm can be very competitive. They also use local search in their algorithm in order to improve their algorithm.

In another work, Tseng and Liao, (2008) apply particle swarm optimization algorithm for hybrid flow-shop scheduling. "Absolute" solution encoding is only used for encoding of the first stage because it is the only stage that all jobs are available at time zero. For other stages the list scheduling (LS) algorithm is used to determine the start times of the jobs at other stages. According to this algorithm, jobs are processed as soon as possible based on their completion times from the previous stage in such a way that job completion times will be minimal. They employ various velocity update methods and neighborhood topologies (*gbest*, pbest and time-delay models). They claim that their algorithm outperforms GA and ACS.

M. R. Singh et al. (2013) uses chaotic mutation operator in order to overcome the problem of trapping at local minima in standard PSO algorithm. The chaotic sequence using logistic mapping is used instead of random numbers to improve the diversity in solution space. They use random keys as a representation and employ mutation strategy to increase the diversity. Mutation is performed each time the number of iterations without diversity exceeds an exact number.

Sankaran (2009) applies PSO in scheduling flexible flow line with sequence dependent setup times. Random keys are used as a solution representation and the algorithm is evaluated by the lower bound which is proposed by Kurz and Askin (2004). The results indicate that the PSO algorithm does not perform well in minimizing makespan in this problem. Several potential weaknesses of this research are: (1) Use of standard PSO algorithm without considering the premature convergence of this algorithm, (2) Not modifying position and velocity update equations, (3) Not considering other encoding methods.

1.2. Problem Description

The problem considered in this study involves scheduling jobs for a flexible flow line with the objective of minimizing the makespan. This problem consists of a set J of n jobs that need to be processed in a flexible flow line. Each job $j\in J$ is associated with processing time (p_j) and setup time (s_{ij}) where i is the job processed before job j on the same machine. The problem under study is NP-hard (Gupta, 1988). Therefore, various algorithms of the particle swarm optimization are used to minimize the maximum completion time.

1.3. Research Motivations and Objectives

This study is motivated by the importance of flexible flow line scheduling, the effectiveness of PSO in various applications and the lack of any publication, to the authors' knowledge, which addresses this scheduling problem by using discrete particle swarm optimization in a flexible flow line. Flexible flow lines (FFL) are used in various industries such as automotive, printed circuit board and textile. Finding the optimal assignment of limited resources to a number of jobs to obtain minimum flow time, makespan, lateness and tardiness or other objectives is very important.

According to the literature, PSO is an effective algorithm which can reach high quality solutions in a reasonable computational time. It also has fewer numbers of parameters than other evolutionary metaheuristics such as genetic algorithm (GA). GA uses mutation and crossover to update the solution space. As discussed earlier in a permutation problem, every job should appear just once in the sequence. Since the crossover operator does not consider this fact, there is a need to recheck the solution which is created by crossover. So using crossover in GA can make the algorithm more complicated in scheduling problems. However it does not mean that crossover operator is not an efficient operator because it may be worthwhile when applied to PSO.

There are many papers available which use heuristic approaches to solve scheduling problems, but few use PSO. More specifically there are no papers available with the focus in using DPSO in order to solve flexible flow line scheduling problem with sequence dependent setup times.

Kurz and Askin (2004) address this problem with makespan criteria using a random keys genetic algorithm approach and obtained a strong lower bound for this problem, However Sankaran and Kurz (2009) applied a continuous version of PSO to solve this problem but the results were not satisfactory. Since the scheduling is a discrete problem we propose to apply the discrete version of PSO using the same data set as in these two works and compare the results.

In this work we plan to use DPSO algorithms. The functionality of PSO is based on how it updates the position and velocity of the particles. Our main focus is on applying different methods to update velocity and positions. Additionally we will examine the impact of using an appropriate encoding to represent the sequence of *n* jobs.

The primary goal of this research is finding an alternative PSO method for GA in scheduling a flexible flow line to minimize the makespan. This goal is achieved through the following objectives:

1. Develop various PSO algorithms
2. Compare the solution quality of the proposed algorithms against GA

CHAPTER 2

METHODOLOGIES

Some of the PSO methodologies have been discussed in the previous chapter. In this chapter some of the methods which are used in this research are discussed in detail. These methods are as follows:

1. Standard Particle Swarm Optimization (SPSO)
2. Passive Congregation Particle Swarm Optimization (PCPSO)
3. Attraction Repulsion Particle Swarm Optimization (ARPSO)
4. Discrete Particle Swarm Optimization (DPSO)
5. Hybrid Discrete Particle Swarm Optimization with a Local Search (DPSO-LS)

The performance of each of these algorithms are evaluated by using a method which is presented at the end of this chapter.

2.1 Standard Particle Swarm

In this method, the position of ith particle of the swarm in the continuous n-dimensional search space at iteration t is $x_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t)$ with the objective value of $f(x)$ (fitness). The best previous position (*pbest*) of each particle (best personal position of particle i) is shown by $p_i^t = (p_{i1}^t, p_{i2}^t, \ldots, p_{in}^t)$ and the last particle position change (velocity) is represented by $v_i^t = (v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t)$. The position with the best function value found so far is the global best (*gbest*) position and is represented by $g_i^t = (g_{i1}^t, g_{i2}^t, \ldots, g_{in}^t)$. Each particle adjusts its position during time based on its own experience and also the experience of other particles. The SPSO algorithm is given in Figure 2.1. The position and velocity of particle i at iteration t of the SPSO algorithm, x_i^t and v_i^t respectively, are updated by following equations:

$$v_i^{t+1} = w\, v_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t) \tag{2.1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.2}$$

where:

- w is the inertia parameter that weights the previous velocity of a particle (how much a particle trusts its own experience).
- c_1 and c_2 are cognitive and social parameters, respectively.
- r_1 and r_2 are uniform random numbers between [0, 1] which are used to weight ($p_i^t - x_i^t$) and ($g_i^t - x_i^t$).

```
Initialize parameters and particles random positions and velocities on n-dimensions in the search space
Do
    For each particle i with position $x_i^t$ do
        If ($x_i^t$ is better than $p_i^{t-1}$) then
            $p_i^t \leftarrow x_i^t$
        End-if
    End-for
    Update $g_i^t$
    For each particle i do
        $v_i^{t+1} \leftarrow wv_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t)$
        $x_i^{t+1} \leftarrow x_i^t + v_i^{t+1}$
    End-for
While (a stop criterion is not satisfied)
```

Figure 2.1. SPSO Algorithm

2.1.1 Solution Representation

The solution representation in this method is random keys (Bean et al. 1994; Kurz and Askin, 2004). In this encoding, each solution is represented by a particle with an n dimension vector, where n is the number of jobs. Each dimension is a random number between [0, M) with two decimals, where M is the number of machines in the stage. For example, for a problem with 5 jobs and 3 machines in a stage, a particle position can be defined as: x_i^t= (1.78, 1.65, 2.23, 3.45, 2.49). The integer part is the machine number to which the job is assigned and fractional part serves as the sort key to sort the jobs assigned to each machine. This particle represents jobs 2 and 1 will be assigned to machine one, respectively, jobs 3 and 5 will be assigned to machine two and job 4 will be processed at machine 3.

2.1.2 Initialization and Position Update

The parameters used for c_1 and c_2 are set as c_1=c_2=2 as recommended by Kennedy and Eberhart (2001). The inertia parameter is a critical parameter for the convergence behavior of this algorithm. It can be constant or decreasing over time (similar to the β parameter in simulated annealing which decreases in each iteration).

The position of particle *i* is initialized randomly between [0, *M*) and the velocity of particle *i* is randomly chosen from [0,1]. After every updating process, all the positions should be in range of [0, *M*), which might be violated in some iterations. In this research, two mechanisms are used to deal with the issue of positions being outside the range of allowable values:

1- $x_i^t = min\,(M\text{-}0.01, max(0, x_i^t))$
2- Bounce Back (Sankaran, 2009):
   if $x_i^t \geq M$ then $x_i^t = 2M - x_i^t$
   if $x_i^t < 0$ then $x_i^t = -x_i^t$

To better understand how these mechanisms help to maintain the position in the allowable range, consider a stage consisting of two machines which is supposed to process 4 jobs. The allowable range for each position is between [0,2) but after updating the velocity and adding it to the current position, a position is obtained with some out of range dimensions. Table 2.1 illustrates how the two mechanisms bring those values back in range.

	Position
$x_i^{t+1} = x_i^t + v_i^{t+1}$	(**-1.78**, 1.65, **2.23**, 0.45)
Mechanism1	(**0**, 1.65, **1.99**, 0.45)
Mechanism2	(**1.78**, 1.65, **1.77**, 0.45)

Table 2.1. Mechanisms to bring the positions back into the allowable range.

2.2 Passive Congregation Particle Swarm

The algorithm of PCPSO is similar to SPSO except for the velocity update process. In this method each particle gets information from personal best, global best and one other random particle in the swarm in order to update its position. They enter this kind of information into Eq. (2.1) and rewrite the equation as follow:

$$v_i^{t+1} = w\, v_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t) + c_3 r_3 (p_c^t - x_i^t) \tag{2.3}$$

where p_c^t is a particle selected randomly from the swarm, c_3 is the passive congregation coefficient and r_3 is a random uniform number in [0, 1].

The PCPSO algorithm is given in Figure 2.2. The solution representation in this method is also random keys.

```
Initialize parameters and particles random positions and velocities on $n$-dimensions in the search space
Do
    For each particle $i$ with position $x_i^t$ do
        If ($x_i^t$ is better than $p_i^{t-1}$) then
            $p_i^t \leftarrow x_i^t$
        End-if
    End-for
    Update $g_i^t$
    For each particle $i$ do
        $v_i^{t+1} \leftarrow w v_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t) + c_3 r_3 (p_c^t - x_i^t)$
        $x_i^{t+1} \leftarrow x_i^t + v_i^{t+1}$
    End-for
While (a stop criterion is not satisfied)
```

Figure 2.2. PCPSO Algorithm

2.3 Attraction Repulsion Particle Swarm

As mentioned in the previous chapter, in this method the position update depends on the diversity of the swarm. If the diversity is less than a critical value, the particles will repel each other; otherwise, they will attract each other. In the proposed algorithm, the velocity equation is:

$$v_i^{t+1} = wv_i^t + dir\left(c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t)\right) \quad (2.4)$$

Where variable *dir* directs the velocity of the swarm being updated by attraction (*dir* = 1) or repulsion (*dir* = -1).

$$\begin{cases} dir = -1 & if\ (dir > 0\ and\ diversity < dcritical), \\ dir = 1 & if\ (dir < 0\ and\ diversity \geq dcritical) \end{cases} \quad (2.5)$$

As with the previous algorithms, the random keys are used for encoding the solution space. Figure 2.3 shows the ARPSO algorithm.

```
Initialize parameters and particles random positions and velocities
Do
    For each particle $i$ with position $x_i^t$ do
        If ($x_i^t$ is better than $p_i^{t-1}$) then
            $p_i^t \leftarrow x_i^t$
        End-if
    End-for
    Update $g_i^t$
    Calculate $diversity\ (N)$
    For each particle $i$ do
        If $diversity < critical\ value$
            $v_i^{t+1} \leftarrow wv_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g_i^t - x_i^t)$
        Else
            $v_i^{t+1} \leftarrow wv_i^t - c_1 r_1 (p_i^t - x_i^t) - c_2 r_2 (g_i^t - x_i^t)$
        End-if
        $x_i^{t+1} \leftarrow x_i^t + v_i^{t+1}$
    End-for
While (a stop criterion is not satisfied)
```

Figure 2.3. ARPSO Algorithm

There are different methods to calculate the diversity of a swarm. Here we use the average distance around the swarm center (Olorunda and Engelbrecht, 2008):

$$diversity(N) = \frac{1}{|N|}\sum_{i=1}^{|N|}\sqrt{\sum_{j=1}^{n}(x_{ij}^{t} - \bar{x}_j)^2} \qquad (2.6)$$

where N is the swarm, $|N|$ is the swarm size, $|L|$ is the length of the longest diagonal in the search space, n is the dimensionality of the problem (number of jobs), x_{ij}^{t} is the jth value of the ith particle at iteration t and $\bar{x}_j$ is the average of jth value of all the particles:

$$\bar{x}_j = \frac{\sum_{i=1}^{|N|} x_{ij}}{|N|} \qquad (2.7)$$

2.4 Discrete Particle Swarm

In order to be able to use the job permutation based encoding scheme, Pan et al. (2008) introduced a method for updating the position. In this method the position is updated in one step, meaning that the particle has no velocity. The position update equation is as follows:

$$x_i^t = c_2 \otimes F_3(c_1 \otimes F_2(w \otimes F_1(x_i^{t-1}), p_i^{t-1}), g_i^{t-1}) \qquad (2.8)$$

As mentioned before, $\lambda_i^t = w \otimes F_1(x_i^{t-1})$ is the velocity of the particle. F_1 is the mutation operator which is applied with probability w. If a random number $r \in [0,1]$ is less than w then mutation will be performed.

$\delta_i^t = c_1 \otimes F_2(\lambda_i^t, p_i^{t-1})$ is the cognitive part of the particle and F_2 is the crossover operator which occurs with probability c_1. $x_i^t = c_2 \otimes F_3(\delta_i^t, g_i^{t-1})$ is the social part of the particle and F_3 is the crossover operator which occurs with probability c_2. DPSO algorithm is given in Figure 2.4.

```
Initialize parameters and particles random positions on n-dimensions in the search space
Do
    For each particle i with position $x_i^t$ do
        If ($x_i^t$ is better than $p_i^{t-1}$) then
            $p_i^t \leftarrow x_i^t$
        End-if
    End-for
    Update $g_i^t$

    For each particle i do
        If randomNum<w
            $\lambda_i^{t+1} = mutation(x_i^t)$
        End-if

        If randomNum <$c_1$
            $\delta_i^{t+1} = crossover(\lambda_i^t, p_i^t)$
        End-if
        If randomNum <$c_2$
            $x_i^{t+1} = crossover(\delta_i^t, g_i^t)$
        End-if
    End-for
While (a stop criterion is not satisfied)
```

Figure 2.4. DPSO Algorithm

2.4.1 Mutation

The insert mutation is used in DPSO algorithm. Figure 2.5 illustrates how insert mutation works. In this example two random numbers are generated. These random numbers represent the position of the jobs. The job associated with the bigger random number is inserted after the job associated with the smaller random number. In the following example job 4 is inserted after job 3.

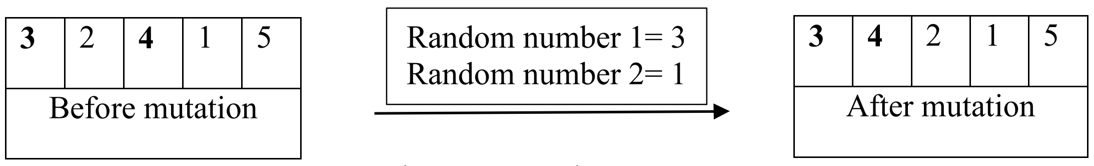


Figure 2.5. Mutation process

2.4.2 Crossover

A two cut crossover introduced by Pan et al. (2008) is used for position update (PTL crossover). In this method, a block of jobs is chosen by two-cut points randomly and then it is moved to one side of solution vector and at the end the new permutation is filled with the remaining jobs from the other particle. Figure 2.6 illustrates PTL crossover method.

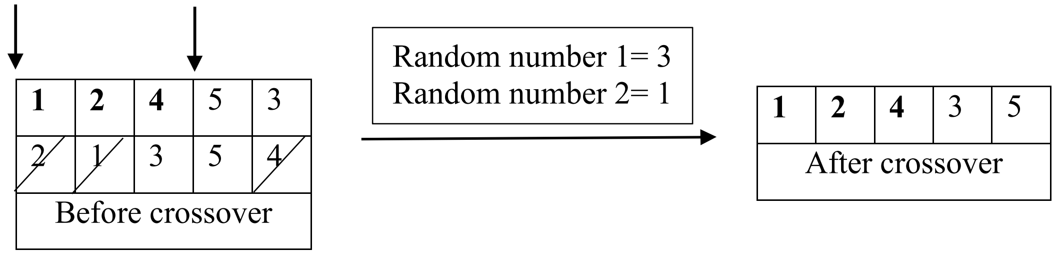


Figure 2.6. PTL crossover process

The following example illustrates the position update in this algorithm. The current position of a particle, its personal best and the global best is shown in Table 2.2. This particle might mutate with probability *w*. Then it can be recombined with the personal best with the probability c_1 and finally it might recombined with the global best with the probability c_2.

		Insert Mutation (Inertia)		PTL Crossover (Cognitive)		PTL Crossover (Social)	
x_i^t	(3 5 1 2 4)			λ_i^t	(3 **1 5 2** 4)	δ_i^t	(1 **5 2** 3 4)
p_i^t	(1 2 3 4 5)	x_i^t	(**3** 5 **1** 2 4)	p_i^t	(1 2 3 4 5)	g_i^t	(1 4 5 3 2)
g_i^t	(1 4 5 3 2)	λ_i^t	(**3 1** 5 2 4)	δ_i^t	(**1 5 2** 3 4)	x_i^t	(**5 2** 1 4 3)
	(a)		(b)		(c)		(d)

Table 2.2. (a) Particle Info (b) Mutation (c) Recombined with the personal best (d) Recombined with the global best.

2.5 Hybrid Discrete Particle Swarm with a Local Search (DPSO-LS)

In order to improve the DPSO algorithm, Pan et al. (2008) apply a local search based on the insert neighborhood on the global best of each iteration which helps the exploitation (Figure 2.7). The algorithm of the local search which is applied in this research is given in Figure 2.8. The new neighbor (*U*) is found by using an insert mutation. A simulated annealing type of acceptance criterion is used in this algorithm. The local search runs for 500 iterations or until a value less than the global best is found.

```
Initialize parameters and particles random positions on $n$-dimensions in the search space
Do
    For each particle $i$ with position $x_i^t$ do
        If ($x_i^t$ is better than $p_i^{t-1}$) then
            $p_i^t \leftarrow x_i^t$
        End-if
    End-for
    Update $g_i^t$
    For each particle $i$ do
        If $randomNum < w$
            $\lambda_i^t = mutation(x_i^{t-1})$
        End-if
        If $randomNum < c_1$
            $\delta_i^t = crossover(\lambda_i^t, p_i^{t-1})$
        End-if
        If $randomNum < c_2$
            $x_i^t = crossover(\delta_i^t, g_i^{t-1})$
        End-if
    End-for
    Apply Local search to $g_i^t$
While (a stop criterion is not satisfied)
```

Figure 2.7. DPSO-LS Algorithm

```
Do
        $U$=mutation ($g_i^t$)
        Evaluate
        If $f(U) < f(g_i^t)$
            $g_i^t = U$
        End-if
While (a stop criterion is not satisfied)
```

Figure 2.8. Local search Algorithm

2.6 Evaluation

To describe the makespan and lower bounds equations, the following notations are used (Kurz and Askin, 2004):

2.6.1 Notations

n	Number of jobs
k	Number of stages
k_j	Last stage visited by job *j*
p_i^t	Processing time for job *i* at stage *t*
m^t	Number of machines at stage *t*
s_{ij}^t	Setup time from job *i* to job *j* at stage *t*
S_i	Set of stages visited by job *i*
S_t	Set of jobs that visit stage $t = \{i\text{: } p_i^t > 0\}$
C_i^t	Completion time for job *i* at stage *t*

2.6.2 Makespan

The makespan which is the maximum completion time among all the jobs is the objective used in this research. When job *j* is processing on a machine and job *i* is the next job to be processed, the completion time of job *i* is calculated using Eq. (2.9).

$$C_i^t = P_i^t + \max\{C_i^{t-1}, C_j^t\} + S_{j,i}^t \tag{2.9}$$

2.6.3 Lower Bound

Kurz and Askin, 2001, developed a lower bound for flexible flow line with sequence dependent setup times:

$$LB^1 = \max_{i=1,\dots,n}\left\{\sum_{t\in S_i}(p_i^t + \min_{j=0,\dots,n} S_{ji}^t)\right\} \tag{2.10}$$

$$LB^2 = \max_{t=1,\ldots,k} \left\{ \begin{array}{l} \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} S_{ji}^\tau \right) + \frac{\sum_{i \in S^t} \left(p_i^t + \min_{j=0,\ldots,n} S_{ji}^\tau \right)}{m^t} + \min_{i \in S^t} \sum_{\tau=t+1}^{k} \left(p_i^\tau + \min_{j=0,\ldots,n} S_{ji}^\tau \right) \\ + \frac{1}{m^t} \sum_{q=1}^{m^t-1} \left[\min_{i \in S^t[q]} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} S_{ji}^\tau \right) - \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} S_{ji}^\tau \right) \right] \end{array} \right\} \tag{2.11}$$

LB^1 is developed with the assumption that every job must be processed at every stage while *LB^2* assumes that every stage must process all of its jobs. The time for the first job to get to each stage and leave it as well and the idle time for parallel machines at each stage waiting for the first available job are also included in *LB^2*. These two lower bounds are calculated for each of the datasets and the higher LB is used as the lower bound for that test scenario.

2.6.4 Loss

The measure to evaluate the solutions is "%Loss" which is the percentage of deviation of the makespan from the lower bound Eq. 2.12. where C_{max} is the makespan for each test scenario and *LB* is the lower bound for that dataset.

$$\%Loss = \frac{C_{\max} - LB}{LB} \times 100\,. \tag{2.12}$$

CHAPTER 3

EXPERIMENTATION AND COMPUTATIONAL RESULTS

The algorithms which were introduced in the previous chapter are tested using 180 problem instances. In this chapter, the datasets on which the various experiments have been conducted are explained. The computational results are also discussed in this chapter.

3.1 Test Data

The problem instances are obtained from the work presented by Kurz and Askin (2004). Table 3.1 shows the different levels of each factor. The factors are skipping probability, processing time, number of stages, number of machines and number of jobs, which leads to 3×2×3×5×2=180 test scenarios. Kurz and Askin also provide 10 datasets for each of these test scenarios. These 1800 datasets are available at http://people.clemson.edu/~mkurz/ffl.html.

The setup times in the datasets were generated randomly from a Unif (12-14) distribution. The setup time matrices satisfy the triangle inequality (Rios-Mercado and Bard, 1998). As mentioned before, in a flexible flow line jobs are allowed to skip stages as long as they are processed at least at one stage. The skipping probability are chosen to be 0%, 5% and 40% (Leon and Ramamoorthy, 1997).

Table 3.1 Characteristics of the problem instances

Factor	**Level**	
Skipping probability	0.00	
	0.05	
	0.40	
Processing times	Unif (50-70)	
	Unif (20-100)	
Number of stages	2	
	4	
	8	
Machine distribution & Number of Machines	Constant	1
		2
		10
	Variable	Unif (1,4)
		Unif (1,10)
Number of jobs	30	
	100	

3.2 Assumptions

It is assumed that machines are available at all times, all jobs are available at time 0 (the ready time for stage 1 are set to 0 for all jobs), the ready times at stage $t+1$ are the completion times at stage t (no travel time between stages). Preemption is not allowed and jobs have the same priorities. Infinite buffers exist before each machine. Parallel machines are identical in capability and processing rate. The number of machines in each stage should be less than the number of jobs to be processed at that stage.

3.3 Experimental Environment

The algorithms which are introduced in Chapter 2 are coded in MATLAB 2013. For each setting, 50 replications have been run and in each replication the program runs for certain number of iterations (300 or 500) or until the lower bound is achieved. The percentage loss is computed as (makespan – lower bound)/lower bound for each result, and we report the average loss over the 50 replications for each dataset for each algorithm.

All computational experiments are performed using the Palmetto Cluster, Clemson University's primary high performance computing (HPC) resource. The amount of time used to run a program is highly dependent on the available resources at any given time and the number of jobs run on that resource at the same time. SPSO, PCPSO and ARPSO programs required about a calendar day to run for all data sets and all replication (1800*50 executions), less than two calendar days for DPSO and a little more than two days for DPSO-LS using 32 CPUs at a time.

3.4 Generating Random Numbers

The set of random numbers which is generated for a replication is unique to that replication and it is replicable. In generating the random numbers, we utilize the Mersenne Twister pseudorandom number generator, seeded with *seed* which is calculated as follows:

$$seed = 1800 \times (repNum - 1) + FileNum \qquad (3.1)$$

Where *repNum* is the replication number which changes from 1 to 50 and *FileNum* is the file number which changes from 1 to 1800.

3.5 Algorithms

In this section the results of experiments on the mentioned algorithms are provided.

3.5.1 Tuning SPSO

This algorithm, which is the standard particle swarm algorithm, has several parameters which need to be set initially, such as the inertia weight w, cognitive parameter c_1 and social parameter c_2. The parameters c_1 and c_2 are set at 2 following Kennedy and Eberhart (1995). The inertia parameter w is critical for the convergence behavior of PSO algorithms. There should be a balance between exploration and exploitation ability in this algorithm. This parameter can help the algorithm to better explore the solution space.

We experimentally evaluated two methods for setting the inertia parameter: a constant value of 1 or a dampened value, set initially to 1 but decreasing by a damping weight at each iteration. Decreasing the value at each iteration is intended to help the exploitation ability. The damping weight is 0.99. The population size is set at 100 for all algorithms. Table 3.2 shows the result for these experiments. As it is observed from this table and Kruskal-Wallis test result (Figure 3.1), decreasing the inertia weight w in each iteration improves the performance of the algorithm.

Table 3.2. %Loss results of SPSO

w %Loss	Constant	Decreasing
Average	22.26651	19.1006
Standard deviation	14.54169	11.80097
Min	3.087108	2.357537
Max	86.34629	66.69318

```
Kruskal-Wallis Test on %Loss

Treatment            N   Median  Ave Rank     Z
SPSO-Constant        1800   22.33    1908.8   6.25
SPSO-Decreasing       1800   19.07    1692.2  -6.25
Overall              3600            1800.5

H = 39.06  DF = 1  P = 0.000
H = 39.06  DF = 1  P = 0.000  (adjusted for ties
```

Figure 3.1. Kruskal-Wallis test at 95% confidence level

3.5.2 Tuning PCPSO

As mentioned in Chapter 2, this algorithm introduces a new parameter, the passive congregation coefficient c_3. The effect of parameter c_3 is studied by setting it at 0.1, 0.3 and 0.6 following He et al. (2004). The program runs for 50 replications, each contains 300 iterations. The parameters c_1 and c_2 are set at 2 and the inertia parameter w is set at 1 initially and decreases with the damping weight of 0.99, as determined in the previous section. Using the same figure of merit (average loss across all replications and all 1800 data sets), Table 3.3 and Figure 3.2 indicate that increasing the passive congregation coefficient (thereby increasing the effect of the selected random particle), has a negative impact on the average loss percentage. Therefore, including a random particle in updating the velocity does not help the performance of the algorithm. We hypothesize that the negative impact of the increased weight for the passive congregation parameter may be explained as follows: since 100 particles are available at each iteration, getting the information from a random particle can move the particle in a non-desired direction when the particle are already moving in a desired direction.

Table 3.3. % Loss results of PCPSO

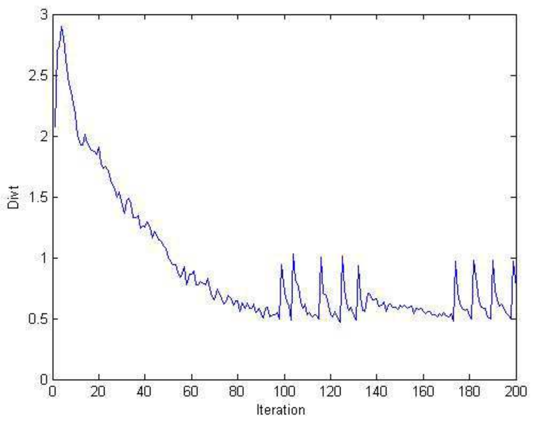
c_3 / % Loss	0.1	0.3	0.6
Average	19.84147	20.549	21.30217
Standard deviation	12.38769	12.844	13.48009
Min	1.912021	2.2779	2.653979
Max	71.21025	76.422	79.72438

```
Kruskal-Wallis Test on %Loss

Treatments     N  Median  Ave Rank      Z
0.1         1800   20.11    2618.8  -2.72
0.3         1800   20.81    2701.4   0.03
0.6         1800   21.48    2781.2   2.69
Overall     5400            2700.5

H = 9.76  DF = 2  P = 0.008
H = 9.76  DF = 2  P = 0.008  (adjusted for ties)
```

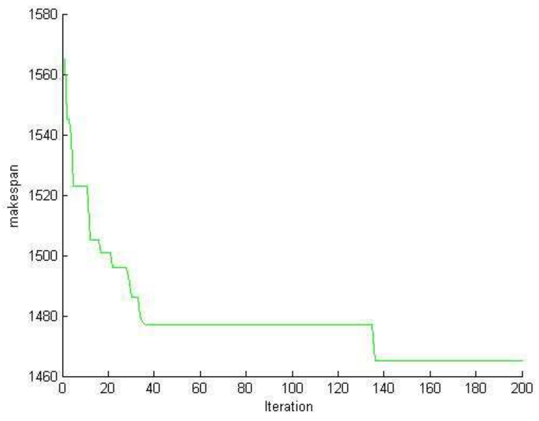
Figure 3.2. Kruskal-Wallis test at 95% confidence level on c_3

3.5.3 Exploring ARPSO

The parameter in this algorithm which plays an important role is the critical value, as described in the previous chapter. In this experiment this value is set at 0.5. Figure 3.3 shows a typical relation between diversity in the swarm (3.3a) and the makespan of the best particle (3.3b) in the swarm at each iteration. As you can see from Figure 3.3a the diversity decreases over time until it hits the critical value, at which point the diversity in the swarm is forced to increase. By increasing the diversity, it is expected that the chance of finding a better solution increases but as illustrated in Figure 3.3b, it is not very effective. The bounce back method was also applied to this variant of the PSO (ARPSO-BB) and compared to ARPSO (see Table 3.4). ARPSO and ARPSO-BB do not evidence very different behavior (Figure 3.4). Since increasing the diversity decreases the exploitation ability, this algorithm is highly sensitive to the critical value. It might be possible to obtain a better result from ARPSO by tuning the critical value.



(a)



(b)

Figure 3.3. (a) Changing diversity over time (b) Changing in makespan over time

Table 3.4. %Loss results

%Loss \ Algorithm	ARPSO	ARPSO-BB
Average	19.49828	19.49344
Standard deviation	11.71283	11.71868
Min	2.88688	2.940846
Max	66.9113	67.07617

```
Kruskal-Wallis Test on Loss

treatment           N  Median  Ave Rank      Z
ARPSO            1800   19.51    1800.8   0.02
ARPSO-BB            1800   19.53    1800.2  -0.02
Overall          3600            1800.5

H = 0.00  DF = 1  P = 0.987
H = 0.00  DF = 1  P = 0.987  (adjusted for ties)
```

Figure 3.4. Kruskal-Wallis test at 95% confidence level

3.5.4 DPSO

As mentioned before, the solution representation in this algorithm is based on job permutation. At the first stage, jobs are assigned based on the order in the sequence and the available machine. The parameters in this algorithm are different from the continuous algorithms introduced earlier. The parameters w, c_1 and c_2 are the probabilities of mutation, crossover with *pbest* and crossover with *gbest* respectively. In order to find the best values for these parameters, parameter tuning is done. Following Pan et al. (2008), the learning parameters $c_1 \in \{0.2, 0.3, 0.8\}$, $c_2 \in \{0.2, 0.3, 0.8\}$ and the weighting factors are $w \in \{0.1, 0.2, 0.6\}$ (Table 3.5).

Table 3.5 Levels of different parameters

Parameter	w	c_1	c_2
Level	0.1	0.2	0.2
	0.2	0.3	0.3
	0.6	0.8	0.8

Each of the 27 settings are tested for 20 replications, 300 iterations on 1800 datasets (1800*27*20 makespan values). The Friedman test is used to find the best setting to minimize the percentage Loss. This test is done using MINITAB 17 and the results are shown in Appendix A. The setting with the lowest sum of ranks is chosen to be the best setting. The results show that the setting w=0.6 and c_1=c_2=0.8 leads to the lowest makespan comparing to other settings. Therefore, this setting is selected for all the experiments using DPSO.

The effect of number of iterations is also studied by changing the iterations from 300 to 500. It is observed that at 95% confidence level the results are not significantly different (Figure 3.5). As it is shown in Table 3.6 by increasing the number of iterations a better result is obtained.

```
Kruskal-Wallis Test on Loss

treatment       N  Median  Ave Rank      Z
DPSO 300     1800   17.91    1812.7   0.70
DPSO 500     1800   17.67    1788.3  -0.70
Overall      3600            1800.5

H = 0.49  DF = 1  P = 0.483
H = 0.49  DF = 1  P = 0.483  (adjusted for ties)
```

Figure 3.5 Kruskal-Wallis test at 95% confidence level

To further improve the results the local search is implemented on the global best of each iteration of DPSO (both 300 and 500 iterations). Table 3.6 illustrates that the local search significantly improve the results (Figure 3.6 & 3.7), from average loss percentages of 17.73 to 16.87 for DPSO-500 and from 17.97 to 17.14 for DPSO-300. By using the local search, the probability of leaving the local optima and finding a better solution increases. Figure 3.8 compares the average loss of the various DPSO algorithms. It is observed that DPSO-LS-500 has the lowest average Loss.

Table 3.6. DPSO results

Algorithm / %Loss	DPSO 300	DPSO 500	DPSO-LS-300	DPSO-LS-500
Average	17.97066	17.73969	17.14973	16.878
Standard deviation	11.27256	11.23991	11.29628	11.2607
Min	0.648955	0.550353	0.438936	0.37703
Max	60.84259	60.76667	60.0463	59.79905

```
Kruskal-Wallis Test on Loss

treatment           N  Median  Ave Rank      Z
DPSO-300         1800   17.91    1846.3   2.64
DPSO-LS-300      1800   17.00    1754.7  -2.64
Overall          3600            1800.5

H = 6.98  DF = 1  P = 0.008
H = 6.98  DF = 1  P = 0.008  (adjusted for ties)
```

Figure 3.6 Kruskal-Wallis test at 95% confidence level

```
Kruskal-Wallis Test on Loss

treatment           N  Median  Ave Rank      Z
DPSO-500         1800   17.67    1848.2   2.76
DPSO-LS-500      1800   16.74    1752.8  -2.76
Overall          3600            1800.5

H = 7.60  DF = 1  P = 0.006
H = 7.60  DF = 1  P = 0.006  (adjusted for ties)
```

Figure 3.7 Kruskal-Wallis test at 95% confidence level

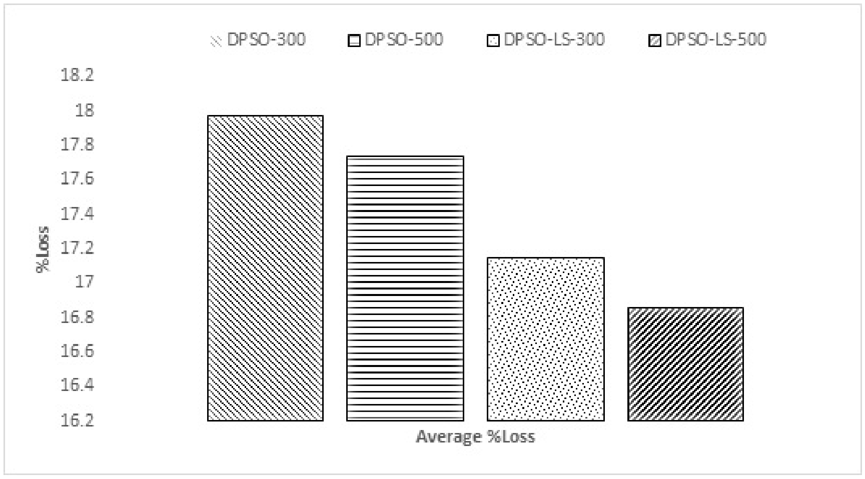


Figure 3.8 Comparison between discrete algorithms (300 and 500 iterations)

3.6 Comparing the Algorithms with GA

The results of SPSO (decreasing *w*), PCPSO, ARPSO and DPSO-LS-500 are compared with the results of the GA developed by Kurz and Askin (2004) (Table 3.7 & Figure 3.9). Figure (3.10) shows that there is a significant difference between these algorithms.

Table 3.7. % Loss results

%Loss \ Algorithm	SPSO	ARPSO	PCPSO	DPSO-LS-500	GA
Average	19.1006	19.49828	19.84147	16.878	17.55725
Standard deviation	11.80097	11.71283	12.38769	11.2607	11.1508
Min	2.357537	2.88688	1.912021	0.37703	1.328397
Max	66.69318	66.9113	71.21025	59.79905	59.62358

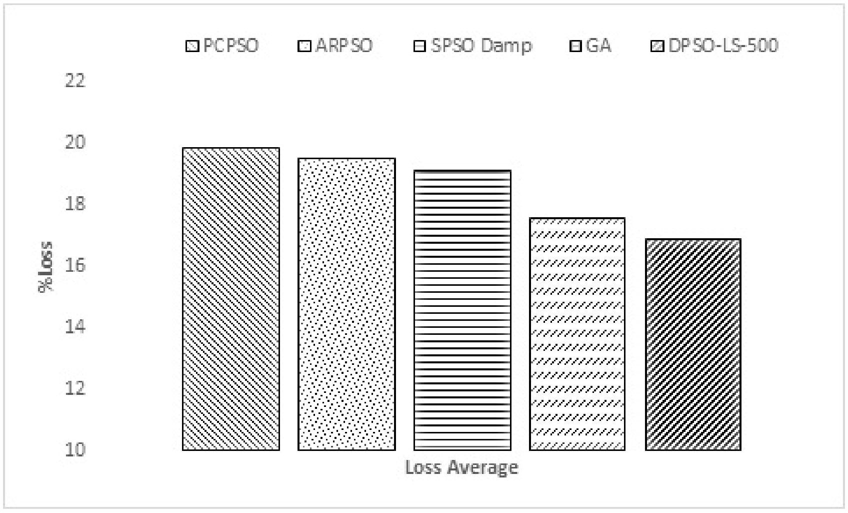


Figure 3. 9. Comparison between all algorithms

```
Kruskal-Wallis Test on Loss

treatment        N  Median  Ave Rank      Z
SPSO          1800   19.07    4619.8   2.18
ARPSO         1800   19.51    4738.5   4.34
PCPSO         1800   20.11    4760.7   4.75
DPSO-LS-500   1800   16.74    4106.8  -7.19
GA            1800   17.65    4276.7  -4.09
Overall       9000            4500.5

H = 91.62  DF = 4  P = 0.000
H = 91.62  DF = 4  P = 0.000  (adjusted for ties)
```

Figure 3.10. Kruskal-Wallis test at 95% confidence level

The Kruskal-Wallis test on SPSO, ARPSO and PCPSO indicates that these algorithms do not have any significant difference at 95% confidence level (Figure 3.11). The tuned SPSO with the decreasing inertia weight factor is compared to the GA using the non-parametric test of Kruskal-Wallis (Figure 3.12), it is observed that there is a significant difference between the average percentage loss of these two algorithms at the 95% confidence level. The scatter plot of these two algorithms illustrates the degree to which the GA performs better (Figure 3.13).

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank      Z
SPSO       1800   19.07    2647.4  -1.77
ARPSO      1800   19.51    2719.5   0.63
PCPSO      1800   20.11    2734.5   1.13
Overall    5400            2700.5

H = 3.21  DF = 2  P = 0.201
H = 3.21  DF = 2  P = 0.201  (adjusted for ties
```

Figure 3.11. Kruskal-Wallis test at 95% confidence level

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank      Z
SPSO       1800   19.07    1869.9   4.01
GA         1800   17.65    1731.1  -4.01
Overall    3600            1800.5

H = 16.06  DF = 1  P = 0.000
H = 16.06  DF = 1  P = 0.000  (adjusted for ties)
```

Figure 3.12. Kruskal-Wallis test at 95% confidence level

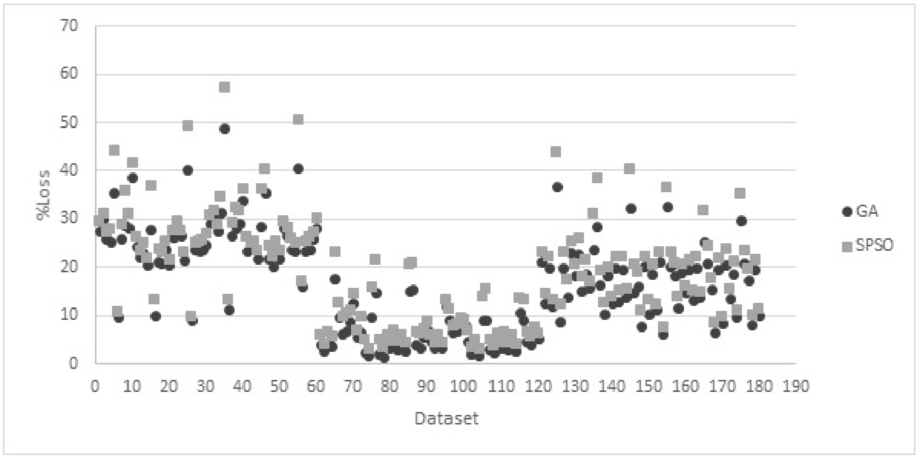


Figure 3.13. Scatter plot of SPSO (decreasing *w*) vs GA

The results of DPSO-LS-500 is also compared with GA, it is observed that there is a significant difference between the results at 95% confidence level (Figure 3.14) and DPSO-LS-500 can hit a lower average loss. It indicates that by changing the solution representation, the performance of a PSO algorithm can improve significantly and become competitive with the GA (Figure 3.15)

```
Kruskal-Wallis Test on Loss

treatment        N  Median  Ave Rank      Z
DPSO-LS-500   1800   16.74    1765.1  -2.04
GA            1800   17.65    1835.9   2.04
Overall       3600            1800.5

H = 4.17  DF = 1  P = 0.041
H = 4.17  DF = 1  P = 0.041  (adjusted for ties)
```

Figure 3.14 Kruskal-Wallis test at 95% confidence level

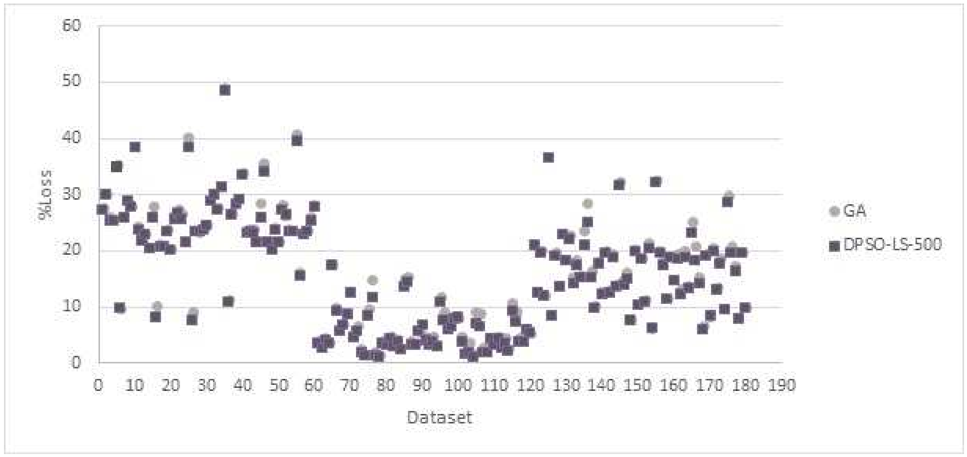


Figure 3.15 Scatter plot of DPSO-LS-500 vs GA

3.7 Effect of Different Factors in the Data Sets

The Kruskal-Wallis test is performed on DPSO-LS-500 results to illustrate whether there is any significant difference between each of the factors at 95% confidence level. Table 3.8 shows the *p*-values obtained from these tests. All p-values are less than 0.01 except for the processing time factor, which means almost all the factors have significant effect at 95% confidence level.

Factor	***P* value**
Number of jobs	< 0.01
Skipping probability	< 0.01
Process time	> 0.01
Number of stages	< 0.01
Number of machines per stage	< 0.01

Table 3.8 *p*-values obtained from Kruskal-Wallis test

3.7.1 Skipping Factor

It is observed that when all jobs visit all stages (0% Skip), DPSO-LS performs significantly better with the average loss of 6.12 % (Figure 3.16). By changing the skipping probability from 0% to 5% and 40%, the average Loss increases from 6.12% to 17.75% and 26.76%, respectively. Figure 3.17 illustrates how the skipping probability divided the datasets into three distinct groups. It can be concluded that DPSO is more effective when the skipping probability is low.

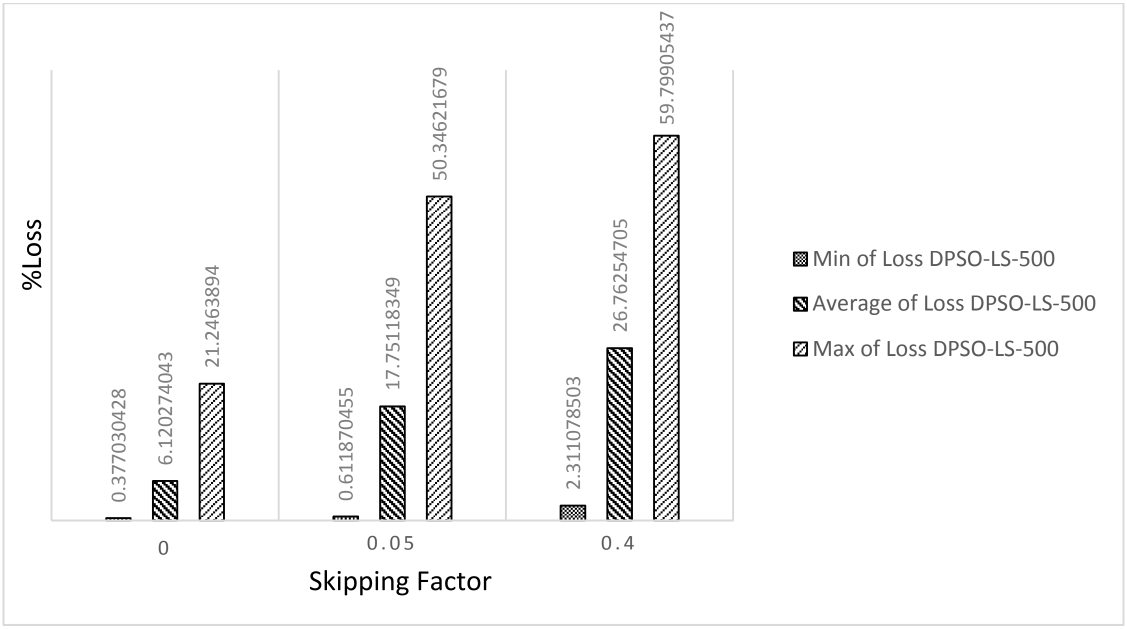


Figure 3.16 Effect of skipping factor on %Loss

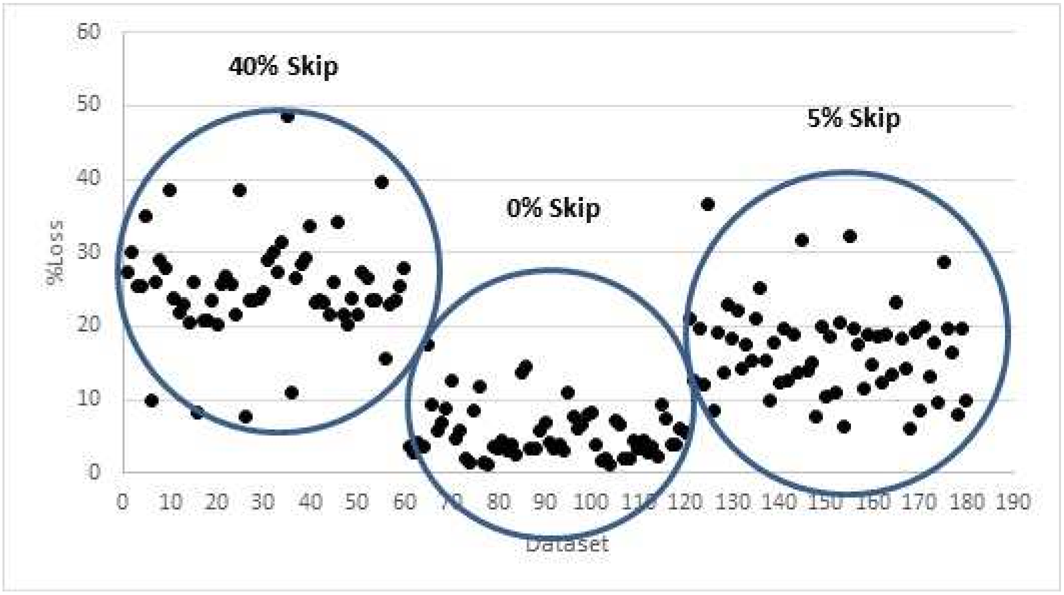


Figure 3.17 Effect of skipping factor on %Loss

3.7.2 Number of Jobs

Figure 3.18 shows that when the number of jobs increases from 30 to 100 the average Loss increases from 15.00 to 18.74. By increasing the number of jobs, the solution space becomes larger and exploring the whole solution space to obtain a better result will take more time.

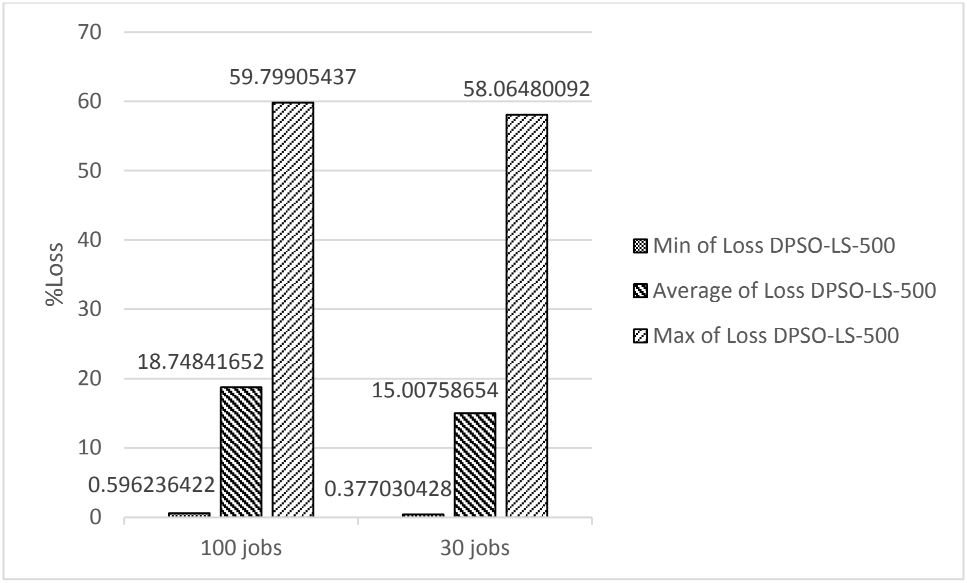


Figure 3.18 Effect of number of jobs on %Loss of DPSO-LS-500

3.7.3 Number of Stages

Figure 3.19 indicates that DPSO performs better when the number of stages is low. The order which is selected for stage 1 may not be the optimal order for all stages and by moving through the flexible flow line the effect of choosing poor order for later stages becomes more apparent. Another possible reason is the effect of skipping probability on the results. By increasing the number of stages, jobs have more stages to skip, therefore average Loss will increase (Figure 3.20).

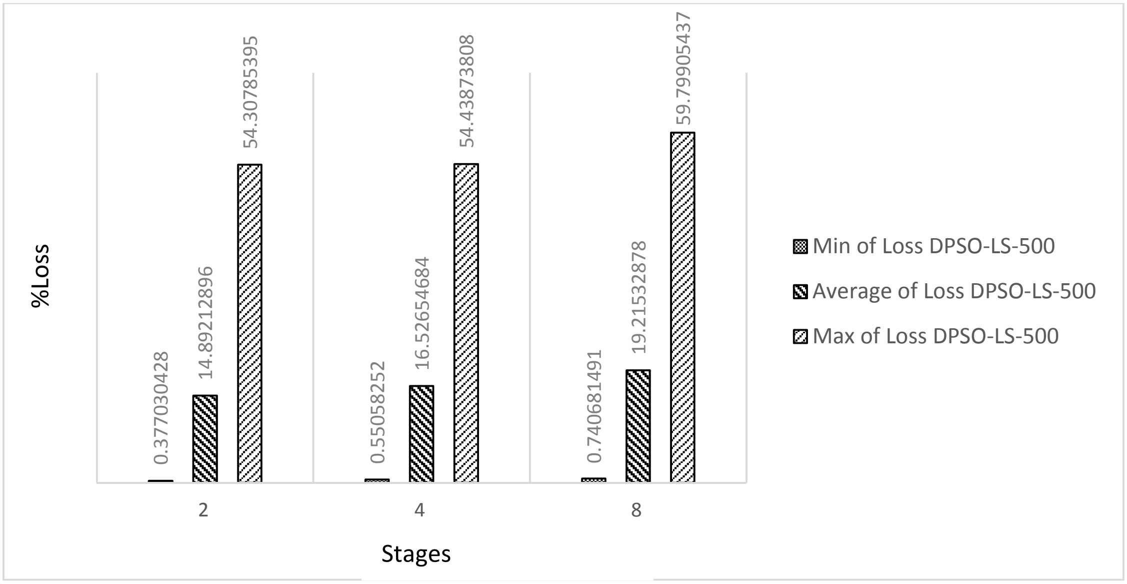


Figure 3.19 Effect of number of stages on %Loss

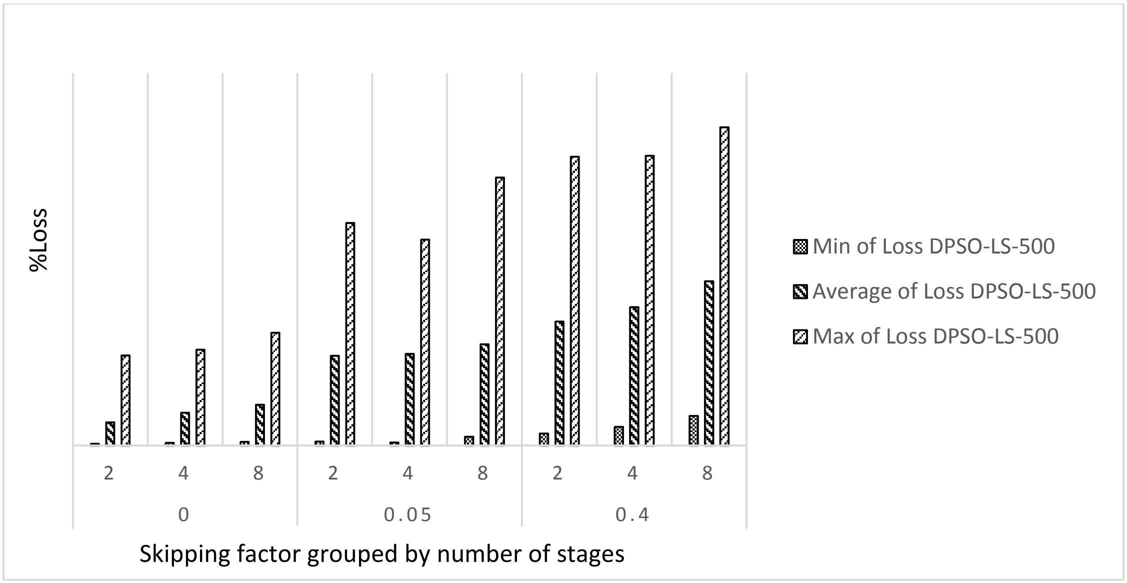


Figure 3.20 Effect of skipping factor on %Loss grouped by number of stages

3.7.4 Number of Machines

From Figure 3.21, the DPSO algorithm performs very poorly when there are 10 machines in each stage. Generally by increasing the number of machines, the solution space becomes larger and the average loss increases.

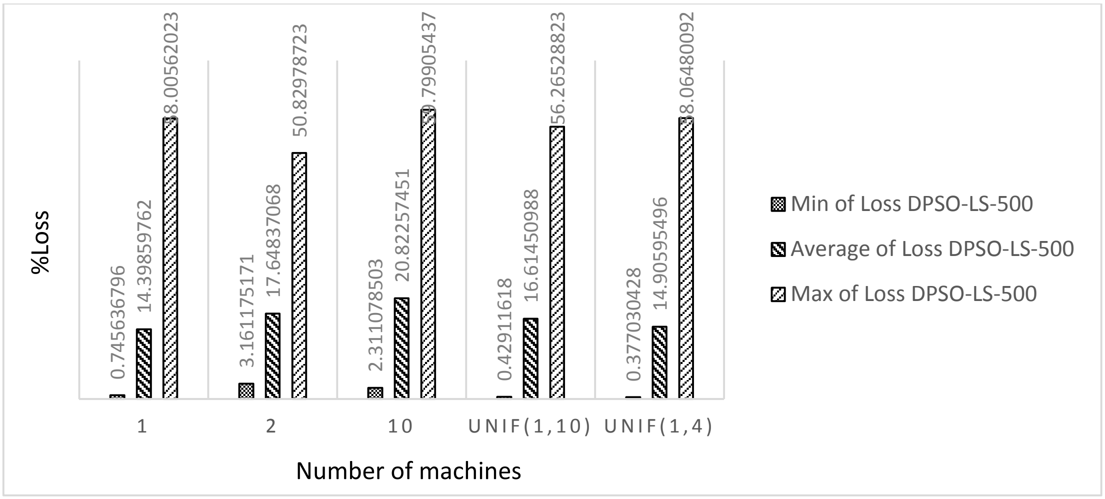


Figure 3.21 Effect of number of machines on %Loss

3.7.5 Processing Time

Figure 3.22 indicates that the average Loss changes when the distribution of the processing times changes, but the amount of this change is not statistically significant. In this case, when the distribution of the processing time changes from Unif (50-70) to Unif (20-100), average Loss increases from 16.75 to 17.00. It can be concluded that the performance of DPSO algorithm does not change significantly, practically or statistically, by this factor.

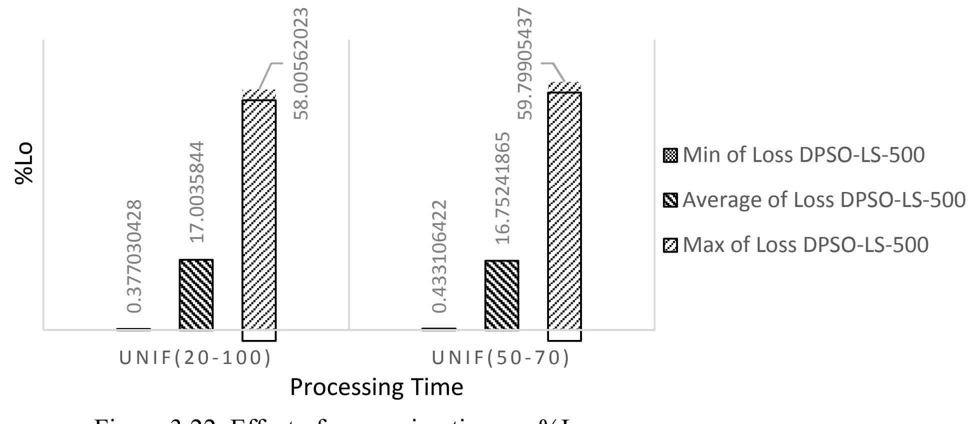


Figure 3.22 Effect of processing time on %Loss

CHAPTER 4

CONCLUSIONS AND FUTURE WORKS

The presented study analyzes the performance of continuous and discrete particle swarm optimization methods to minimize the makespan for a flexible flow line.

4.1 Conclusions

Various particle swarm optimization algorithms were developed to solve the problem of scheduling jobs for a flexible flow line. First SPSO, with constant and also decreasing inertia weight parameter, was implemented, then ARPSO and PCPSO which are some extensions of SPSO were also developed to improve the SPSO result using random keys as a solution representation. Results indicate that SPSO with decreasing inertia parameter has the best performance among these algorithms.

A discrete particle swarm optimization was also developed using permutation-based encoding. Moreover, a local search was used to improve the performance of DPSO. The proposed DPSO-LS performs significantly better than DPSO.

The performance of the proposed SPSO with decreasing inertia weight and DPSO-LS were evaluated against the result of GA. The results illustrate that there is a significant difference between the performances of these algorithms. GA performs better than SPSO and DPSO-LS outperformed the results from both of these algorithms.

The presented DPSO-LS algorithm and the program developed in MATLAB can help individuals in charge of scheduling jobs on flexible flow line with sequence dependent setup time, make informed decisions and effectively schedule the line so that the makespan will be minimized. Satisfactory performance in minimizing the makespan can have a significant impact on facility utilization.

The analysis performed also illustrates how the characteristics of the problem setting (number of stages, for example) may influence the quality of the makespan found using these algorithms, providing a scheduler with information about the appropriateness of these methods.

4.2 Future Research

There are several extensions of the proposed algorithm which may be beneficial to investigate. Applying other local search methods is a good area to research. One of these methods can be swapping the job with the longest completion time with other jobs in the schedule. Moreover, applying a local search on SPSO can lead to improvements in the algorithm and improve the quality of the final solution. Feeding the particle swarm algorithm with the result of other metaheuristic methods such as simulated annealing can be another interesting area to study. Finally, future research may analyze other methods of assigning jobs to machines, other than list scheduling.

Appendix A

Friedman Test-Tuning DPSO

```
Friedman Test: Response versus Treatment blocked by
Block
 S = 15203.25    DF = 25    P = 0.000
 S = 15330.02    DF = 25    P = 0.000 (adjusted for ties)


                              Est
   Treatment       N       Median    Sum of Ranks
       1         1800      18.157       31817.5
       2         1800      18.036       19974.0
       3         1800      18.060       23433.5
       4         1800      18.169       32853.0
       5         1800      18.133       29752.0
       6         1800      18.184       34261.0
       7         1800      17.939       13371.0
       8         1800      18.094       25992.0
       9         1800      18.008       18383.5
      10         1800      17.988       15623.5
      11         1800      18.091       25229.5
      12         1800      18.128       28682.5
      13         1800      18.031       19548.5
      14         1800      18.040       20594.5
      15         1800      17.934       12929.5
      16         1800      18.031       20435.5
      17         1800      18.249       38997.5
      18         1800      18.224       37027.5
      19         1800      18.022       20048.5
      21         1800      18.046       22209.5
      22         1800      17.985       15722.5
      23         1800      18.140       29606.0
      24         1800      17.940       12726.0
      25         1800      17.984       15390.5
      26         1800      18.182       32788.0
      27         1800      18.187       34403.0

Grand median = 18.076
```

The lowest rank is 12726.0 for treatment 24 with following parameters:

C1	0.8
C2	0.8
W	0.6

Appendix B

Kruskal-Wallis Test on Different Factors
DPSO-LS-500

1. Number of Jobs

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank      Z
h           900   12.73     806.9  -7.64
a           900   20.46     994.1   7.64
Overall    1800             900.5

H = 58.35  DF = 1  P = 0.000
H = 58.35  DF = 1  P = 0.000  (adjusted for ties)
```

h=30 jobs, a=100 jobs

2. Number of Machines

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank      Z
l           720   15.16     796.1  -6.96
h           720   17.39     975.6   5.00
m           360   17.70     959.2   2.40
Overall    1800             900.5

H = 48.68  DF = 2  P = 0.000
H = 48.68  DF = 2  P = 0.000  (adjusted for ties)
```

l= 1, h=2, m=10

3. Number of Stages

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank      Z
l           600   16.54     810.1  -5.22
h           600   18.06     998.2   5.64
m           600   15.84     893.2  -0.42
Overall    1800             900.5

H = 39.47  DF = 2  P = 0.000
H = 39.47  DF = 2  P = 0.000  (adjusted for ties)
```

l=2, m=4, h=8

4. Processing Time

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank       Z
l           900   16.52     890.8  -0.79
h           900   16.93     910.2   0.79
Overall    1800             900.5

H = 0.62  DF = 1  P = 0.431
H = 0.62  DF = 1  P = 0.431  (adjusted for ties)
```

l= Unif(50-70), h=Unif(20-100)

5. Skipping Probability

```
Kruskal-Wallis Test on Loss

treatment     N  Median  Ave Rank        Z
l           600   5.254     372.3  -30.49
h           600  25.976    1363.0   26.69
m           600  17.989     966.2    3.79
Overall    1800             900.5

H = 1104.18  DF = 2  P = 0.000
H = 1104.18  DF = 2  P = 0.000  (adjusted for ties)
```

l= 0.00, h=0.05, m= 0.40

REFERENCES

Anghinolfi, D., & Paolucci, M. (2009). “A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence dependent setup times.” *European Journal of Operational Research*, 193(1), 73-85.

Bean, J. C. (1994). “Genetic algorithms and random keys for sequencing and optimization.” *ORSA journal on computing*, 6(2), 154-160.

Clerc, M. (2004). “Discrete particle swarm optimization, illustrated by the traveling salesman problem.” *New optimization techniques in engineering*, Springer Berlin Heidelberg, 219-239.

Congying, Lv, Zhao Huanping, and Yang Xinfeng.(2011) "Particle swarm optimization algorithm for quadratic assignment problem." *Computer Science and Network Technology (ICCSNT)*, *2011 International Conference on*., Vol. 3, 1728-1731, IEEE.

Dallard, H., Lam, S. S., & Kulturel-Konak, S. (2007). “Solving the orienteering problem using attractive and repulsive particle swarm optimization.” *In Information Reuse and Integration*, *2007. IRI 2007. IEEE International Conference on,* pp. 12-17. IEEE.

Damodaran, P., Diyadawagamage, D. A., Ghrayeb, O., & Vélez-Gallego, M. C. (2012). “A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines.” *The International Journal of Advanced Manufacturing Technology*, 58(9-12), 1131-1140.

Gupta, J.N.D. (1988). “Two stage hybrid flow shop scheduling problem.” *Journal of Operational Research Society*, 39(4), 359–364.

He, S., Wu, Q. H., Wen, J. Y., Saunders, J. R., & Paton, R. C. (2004). “A particle swarm optimizer with passive congregation.” *Biosystems*, 78(1), 135-147.

Ho, S. L., Yang, S., Ni, G., Lo, E. W., & Wong, H. C. C. (2005). “A particle swarm optimization-based method for multiobjective design optimizations.Magnetics.” *IEEE Transactions on*, 41(5), 1756-1759.

Hu, X., Eberhart, R. C., & Shi, Y. (2003). “Swarm intelligence for permutation optimization: a case study of n-queens problem.” *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pp. 243-246. IEEE.

Kalayci, C. B., & Gupta, S. M. (2013). “A particle swarm optimization algorithm with neighborhood-based mutation for sequence dependent disassembly line balancing problem.” *The International Journal of Advanced Manufacturing Technology*, 69(1-4), 197-209.

Karmakar, S., & Mahanty, B. (2010). “Minimizing Makespan for a Flexible Flow Shop Scheduling Problem in a Paint Company.” *Industrial Engineering and Operations Management*.

Kennedy, J., & Eberhart, R. (1995). “Particle swarm optimization.” *In Proceedings of IEEE international conference on neural networks*, 4(2), 1942-1948.

Kennedy, J., & Eberhart, R. C. (1997). “A discrete binary version of the particle swarm algorithm.” *In Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 5, pp. 4104-4108. IEEE.

Kia, H. R., Davoudpour, H., & Zandieh, M. (2010). “Scheduling a dynamic flexible flow line with sequence dependent setup times: a simulation analysis.” *International Journal of Production Research*, *48*(14), 4019-4042.

Krause, J., Cordeiro, J., Parpinelli, R. S., & Lopes, H. S. (2013). “A survey of swarm algorithms applied to discrete optimization problems.” *Swarm Intelligence and Bio-inspired Computation: Theory and Applications.* Elsevier Science & Technology Books, 169-191.

Kurz, M. E., & Askin, R. G. (2003). “Comparing scheduling rules for flexible flow lines.” *International Journal of Production Economics*, *85*(3), 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.

Lian, Z., Gu, X., Jiao, B., (2006a). “A similar particle swarm optimization algorithm for permutation flow shop scheduling to minimize makespan.” *Applied Mathematics and Computation,* 175(1), 773–785.

Liao, C. J., Tseng, C. T., & Luarn, P. (2007). “A discrete version of particle swarm optimization for flow shop scheduling problems.” *Computers & Operations Research*, 34(10), 3099-3111.

Fatih Tasgetiren, M., Sevkli, M., Liang, Y. C., & Gencyilmaz, G. (2004). “Particle swarm optimization algorithm for single machine total weighted tardiness problem.” *In Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, pp. 1412-1419. IEEE.

Oĝuz, C., & Ercan, M. F. (2005). “A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks.” *Journal of Scheduling*, *8*(4), 323-351.

Pan, Q. K., Fatih Tasgetiren, M., & Liang, Y. C. (2008). “A discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem.” *Computers & Operations Research*, 35(9), 2807-2839.

Pan, Q. K., Wang, L., Tasgetiren, M. F., & Zhao, B. H. (2008b). “A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion.” *The International Journal of Advanced Manufacturing Technology*, 38(3-4), 337-347.

Riget, J., & Vesterstrøm, J. S. (2002). “A diversity-guided particle swarm optimizer-the ARPSO.” *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, 2, 2002.

Salvador, M. S. (1973). “A solution to a special class of flow shop scheduling problems.” *In Symposium on the theory of scheduling and its applications*, pp. 83-91. Springer Berlin Heidelberg.

Sankaran, V., Kurz, M. E., (2009), “A particle swarm optimization using random keys for flexible flow shop scheduling problem with sequence dependent setup times.” *Master thesis, Clemson University, Clemson, SC.*

Sevkli, A. Z., & Sevilgen, F. E. (2010). “StPSO: Strengthened particle swarm optimization.” *Turkish Journal of Electrical Engineering & Computer Sciences*, 18, 1095-1114.

Shahvari, O., Salmasi, N., Logendran, R., & Abbasi, B. (2012). “An efficient tabu search algorithm for flexible flow shop sequence dependent group scheduling problems.” *International Journal of Production Research*, *50*(15), 4237-4254.

Singh, M. R., Mahapatra, S. S., & Mishra, K. (2013). “A novel swarm optimiser for flexible flow shop scheduling.” *International Journal of Swarm Intelligence*, 1(1), 51-69.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2007). “A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flow shop sequencing problem.” *European Journal of Operational Research*, 177(3), 1930-1947.

Tavakkoli-Moghaddam, R., & Safaei, N. (2007). “A New Mathematical Model for Flexible Flow Lines with Blocking Processor and Sequence dependent Setup Time.” *Multiprocessor Scheduling*, 255.

Tavakkoli-Moghaddam, R., Safaei, N., & Sassani, F. (2009). “A memetic algorithm for the flexible flow line scheduling problem with processor blocking.” *Computers & Operations Research*, *36*(2), 402-414.

Tseng, C. T., & Liao, C. J. (2008). “A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks.” *International Journal of Production Research*, 46(17), 4655-4670.

Zandieh, M., Mozaffari, E., & Gholami, M. (2010). “A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems.” *Journal of Intelligent Manufacturing*, *21*(6), 731-743.

Zhang, Jindong, Changsheng Zhang, and Shubin Liang.(2010) "The circular discrete particle swarm optimization algorithm for flow shop scheduling problem." *Expert Systems with Applications*, 37(8), 5827-5834.