12-2009

A Particle Swarm Optimization Using Random Keys For Flexible Flow Shop Scheduling Problem With Sequence Dependent Setup Times

Vinodh Sankaran
Clemson University, vsankar@clemson.edu

A PARTICLE SWARM OPTIMIZATION USING RANDOM KEYS FOR FLEXIBLE FLOW SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

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
Vinodh Sankaran
December 2009

Accepted by:
Dr. Mary E. Kurz, Committee Chair
Dr. Maria E. Mayorga
Dr. Rae Cho

ABSTRACT

In this research, a particle swarm optimization algorithm (PSO) using random keys is developed to schedule flexible flow lines with sequence dependent setup times to minimize makespan. The flexible flow line scheduling problem is a branch of production scheduling and is found in industries such as printed circuit board and automobile manufacturing. It is well known that this problem is NP-hard. For this reason, we approach the problem by implementing a particle swarm optimization (PSO), a metaheuristic which is inspired by the motion of a flock of birds or a school of fish searching for food. The proposed PSO has many features, such as the use of random keys for encoding the solution, "bounceback" of particles into the solution space and tuning of learning and weighting factors. The proposed PSO algorithm is implemented in C and tested on a large set of data found in the literature. Extensive computational experiments are facilitated through the use of high-throughput computing via Clemson's Condor grid. The solution qualities are compared and evaluated with the help of lower bound developed by Kurz and Askin [16]. Unfortunately, we conclude that the proposed PSO does not perform well for the problem examined. Areas for future work are identified to improve the overall performance of proposed PSO.

DEDICATION

I dedicate this thesis to my parents Sankaran Kutty and Vijayakumari Sankaran.

ACKNOWLEDGMENTS

I specially want to thank my advisor Dr. Mary Elizabeth Kurz for her guidance and support provided to me during my research work at Clemson University. She motivated and helped me from the first day of my research. As a result my thesis work went smoothly and was interesting.

I really want to thank to Dr. Maria E. Mayorga and Dr. Byung Rae Cho for their abundant help and cooperation with me during my research work at Clemson University.

Finally an honorable mention goes to my manager David Payne for his understanding and support in completing my thesis. I would also like to thank my friends Amit Arkashwera, Abinesh Rajagopal, Balaji Jayakumar, Oliver Johnson, Saravanan Kanan and Shobana Kumari for their moral and technical support provided during my thesis work.

My sincere gratitude goes to my family members Sankaran Kutty, Vijayakumari Sankaran, KP Padmavathi, and Lavanya Sankaran for their endless love and support towards me during my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

1. INTRODUCTION

In modern manufacturing, scheduling problems have become an interesting topic of research. Satisfying the daily demand of product with top quality and on-time delivery leads manufacturing industries to invest money and time in solving scheduling problems. Scheduling problems arise in different industries including chemical, food and discrete parts manufacturing. Among the many manufacturing settings, the flexible flow line is one of the more complicated, especially when compared to the well-researched single machine environment. The existence of multiple machines per stage and allowing job to skip stages make the flexible flow line environment more complicated than the standard flow line. The automobile and printed circuit board industries (Piramuthu et al. [20] and Agnetis et al. [1]) use flexible flow lines with an extra feature: sequence dependent setup times between jobs being processed on the same machine. The objective of the scheduling problem may vary according to industry needs. Potential objectives include minimizing the total weighted tardiness, total completion time or maximum completion time (also known as makespan). Minimizing the makespan is the objective for this research. Minimizing makespan in a flexible flow line with one stage and one machine in that stage is exactly the traditional traveling salesman problem. Based on the reduction, we see that minimizing makespan on a flexible flow line with an arbitrary number of stages and machines with sequence dependent setup times is NP-hard. This NP-hard problem is the focus of our research.

An example of the proposed flexible flow line is shown in Figure 1.1. A flexible flow line consists of several stages in series where each stage consists of (possibly) multiple parallel identical machines where at least one stage should have more than one machine. A job should not revisit a stage which it already visited but jobs can skip stages. This scenario is found in manufacturing industries where a job does not require all operations. We consider a flexible flow line similar to that developed by Wittrock [27] for a printed circuit board manufacturing line.

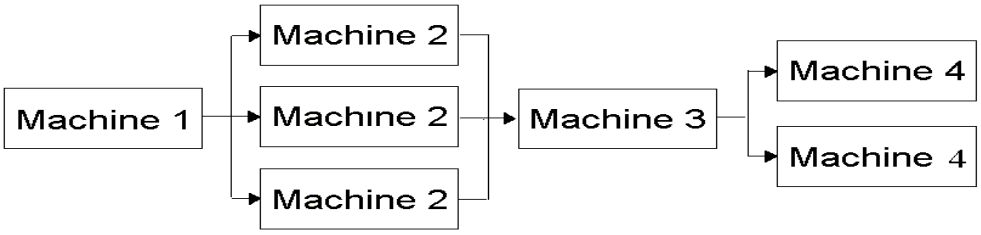


Figure 1.1: Flexible flow line with multiple machines at some stages

The proposed flexible flow line uses data which are known deterministically. No preemption is allowed between jobs. There is no priority value for jobs. Infinite buffers exist between stages. Machines are available at all time without any breakdown. Travel time between stages is zero and once the jobs are processed at the previous stage, they are immediately available for the next stage. Therefore, the ready time for the next stage is the completion time of the current stage. One of the distinguishing factors of our research is the existence of non-anticipatory sequence-dependent setup times between jobs at each stage. After one job is processed and before the next job starts processing, some kind of

setup is done. The time required is sequence dependent. We follow the setup time concept described by Rios-Mercado and Bard [24] for their flow line problem.

In order to shorten the discussion about the proposed flexible flow line we follow a modified version of the notation and makespan calculation introduced by Kurz and Askin [16].

n number of jobs to be processed

g number of stages

g_j last stage visited by job j

p_i^t processing time for job i at stage t

$s_{i,j}^t$ 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 $t = \{i:\ p_i^t > 0\}$

C_i^t completion time for job i at stage t

The makespan ($\max_i C_i^g$) is the maximum completion time and it is the objective criterion in our research. The completion time of the ith job (denoted $[i]$) at stage t can be calculated using equation (1)

$$C_{[i]}^t = p_{[i]}^t + \max\{C_{[i]}^{t-1}, C_{[i-1]}^t\} + s_{[i-1],[i]}^t \qquad (1)$$

The processing time of job 0, representing the initial state of the machines, is assumed to be 0 for all machines on all stages. Setup is non-anticipatory, meaning that the job to be setup must be available and the machine to be used must be idle. The completion times at stage t are the ready time at stage t+1.

One of the most commonly applied methods to solve NP-hard problems such as this is the application of heuristics. Generally heuristics are divided into two groups: constructive methods and improvement methods (Quan-Ke et al. [21]). We focus on a specific type of improvement method called metaheuristics, which include techniques like Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO). Particle swarm optimization (PSO) is a population-based search algorithm developed by Kennedy and Eberhart [14]. Each particle "flies" with a velocity which can be adjusted by flying experience. It can be applied to NP-hard scheduling problems such as ours, as detailed in this thesis.

The major advantage of PSO over other metaheuristic approaches is the simplicity in structure. PSO does not have mutation and evolution parameters like GA so it is easier to implement. We utilize a random keys solution encoding while applying PSO to our problem. We propose a novel updating strategy, called "bounceback" to ensure the particles remain in the feasible region. Computational experiments are facilitated through the use of high-throughput computing via Clemson's condor grid. In this thesis, we compare PSO-generated solutions to a strong lower bound, developed by Kurz and Askin [16].

Our Intent with this research is to investigate how an optimization method PSO has been developed for real valued decision variables, can be applied to combinatorial problems. We utilize a solution representation (Random Keys) which has been successfully used for this problem in Genetic Algorithm developed by Kurz and Askin [16].The notations, assumptions and equations of the proposed PSO are explained in the following chapter, along with a brief literature review in Chapter 2. The proposed PSO is described in Chapter 3. The experiments and results are shown in Chapter 4. Finally Chapter 5 concludes the research.

CHAPTER TWO

2. LITERATURE REVIEW

This brief literature review focuses on the following areas relevant for this thesis: general scheduling of flexible flow lines; literature that focuses on the makespan objective in flexible flow line scheduling; particle swarm optimization; and the use of the random keys encoding for flexible flow line problems.

2.1. Flexible Flow Line

We consider a flexible flow line to be an extension to the classic flow lines (with one machine per stage and all jobs visiting all stages) such that some stages may have multiple identical machines and jobs may not require processing on all stages. Salvador [25] considered a flexible flow line with multiple machines at several stages and with no buffers. They used branch and bound method to determine the optimal permutation schedule (in a permutation schedule, jobs enter the flow line according to one of the $n!$ permutation orderings and a first-in-first-out technique is used to assign jobs to machines at all stages in order to calculate the makespan). Gupta [12] applied Johnson's Rule to a specialized flexible flow line with one machine in the first stage and multiple machines in the second stage. A three stage flexible shop problem with setup on one machine was designed by Bellman et al. [4]. A dynamic program was developed to build the schedule. Cheng et al. [9] provides an overview of flexible flow lines. Cheng et al. [9] begins by describing flexible flow lines with sequence dependent setup times and flexible flow lines with two or multiple stages. Finally they conclude by giving suggestions to solve

complicated flexbile flow line problems. Campbell et al. [7] designed a heuristic to schedule flexible flow lines with a single machine per stage by placing jobs at the end of current sequence considering the idle time of the machine. As an extension to Campbell's single machine flexible flow line, a hybrid (multiple machines per stage, but no stage-skipping) flow line was designed by Ding and Kittichatphayak [10]. In 1991, a hybrid flow shop with an arbitrary number of stages and intermediate buffer was modeled by Brah and Hunsucker [6]. They used branch and bound to develop the schedule. In their work, they explained that a non-permutation schedule with inserted idle time can also be created and may be optimal for some problems.

2.2. Makespan Objective

Santos et al. [26] developed an algorithm to schedule a flexible flow line built on the idea of a permutation schedule to minimize the makespan. The optimal solution is evaluated with the use of a lower bound on the optimal makespan. Lee et al. [17] modeled a flexible flow line with sequence dependent setup times in which the buffers between stages were limited. They used a genetic algorithm to minimize the makespan. Bianco et al. [5] considered the flexible flow line with sequence dependent setup times, release dates and the requirement that jobs do not wait between stages (known as the "no-wait" requirement), to minimize makespan. They utilized branch and bound to minimize the makespan. Kurz and Askin [16] attacked the flexible flow line with sequence dependent setup times and minimized the makespan using heuristic and genetic algorithm approaches. They also developed and evaluated a strong lower bound on the makespan for flexible flow lines with sequence dependent setup times.

2.3. Particle Swarm Optimization for Scheduling

Particle swarm optimization (PSO) was introduced by Kennedy and Eberhart [14]. In recent years PSO has been implemented for combinatorial optimization problems like flow shop and job shop scheduling. Zhingnang et al. [29] consider the problem of minimizing the makespan in job shop scheduling. They provide a procedure for application of PSO for scheduling problems. As an extension to their work in 2008 they implemented the PSO for the job shop with makespan objective. They also compared the performance of PSO with the genetic algorithm (GA) and concluded with research motivation on mathematical validation of particle swarm theory. The proposed PSO worked effectively better than the genetic algorithm (GA) in their job shop scheduling problem. The job shop scheduling problem is sufficiently different that we cannot apply their PSO to our problem. Quan Ke et al. [21] proposed a PSO algorithm for the no-wait flow line scheduling problem with makespan objective and evaluated it by comparing to the heuristic developed by Rajendran [22] for the no-wait flexible flow shop. Cho-Tang and Ching-Jong [8] proposed a particle swarm optimization algorithm for hybrid flow line scheduling with multiprocessor tasks (tasks that can be processed by more than one machine simultaneously). No PSO for minimizing makespan in flexible flow lines with sequence dependent setup times is known in the open literature.

2.4. Encoding Using Random Keys

In 1994 Bean [3] proposed a new method to encode scheduling problem solutions using random numbers. He proposed an algorithm called random keys genetic algorithm which has been applied to many scheduling problems. Kurz and Askin [16] attacked the flexible flow line with sequence dependent setup times scheduling problem, to minimize makespan, with an adaptation of the random keys genetic algorithm. This research uses the same solution representation followed by Kurz and Askin [16]. No PSO implementation for scheduling problems in the open literature has used the random keys encoding.

2.5. Conclusion

From the literature review which we carried out throughout our research work we found that many journal articles have been written about scheduling flexible flow lines but many of them are restricted to special cases like no-wait or permutation schedules. There does not seem to be work focused on minimizing makespan in the flexible flow line with sequence dependent setup times using a random keys encoding in particle swarm optimization. This motivated the work in this thesis.

CHAPTER THREE

3. PROPOSED PARTICLE SWARM OPTIMIZATION

3.1. Introduction

In this section, a particle swarm optimization (PSO) algorithm for the flexible flowline scheduling problem is developed. We begin the chapter with the background and brief description of PSO followed by the proposed PSO. The main feature of our proposed PSO is the use of a random keys representation for sorting. In order to maintain feasible solutions, the random keys representation in a PSO requires one of two adaptations; this is evaluated in Chapter 4. Finally we conclude this chapter with an example illustrating random key PSO.

3.2. PSO Background

PSO was developed by Kennedy and Eberhart [14] in 1995. The motivating biological metaphor is the motion of flock of birds or fish searching for food. It is one of the swarm intelligence and optimization techniques that operate in real number spaces. The algorithm operates on a "swarm" of "particles", which represent potential solutions, and searches for an optimal solution by updating the velocities and positions of particles. The objective function value is used to determine the quality of the particle. Each particle has its own velocity used to update its position. A particle is composed of four pieces of information: position, velocity, current objective function value and position at which the particle has achieved its best-ever objective function value (called P_i for particle i, representing the personal best ever experienced). The swarm is composed of a set of particles and the position at which a particle achieved the best-ever objective function

value ever held by any particle in the swarm (called P_g representing the global best). Each particle *i*'s velocity is updated using P_i and P_g. The details of the particle and velocity updates are provided below.

This description of PSO is based on Kennedy and Eberhart [13] and Eberhart and Shi [11]. Let $Y_i^t = (y_{i1}^t, y_{i2}^t, y_{i3}^t .....y_{iD}^t)$, $y_{ij}^t \in \Upsilon \ \forall \ j$, be particle *i*'s position in *D*-dimensional space at iteration *t*. *S* is the number of particles in a swarm. Let V_i^t be the velocity of particle *i* denoted as $V_i^t = (v_{i1}^t, v_{i2}^t, v_{i3}^t .....v_{iD}^t)$, $v_{ij}^t \in \Upsilon \ \forall \ j$ at iteration *t*. Each coordinate's personal best position is $P_i^t = (p_{i1}^t, p_{i2}^t, p_{i3}^t .....p_{iD}^t)$, $p_{ij}^t \in \Upsilon \ \forall \ j$ at iteration *t*. The global best particle is $P_g^t = (p_{g1}^t, p_{g2}^t, p_{g3}^t .....p_{gD}^t)$, $p_{gj}^t \in \Upsilon \ \forall \ j$ at iteration *t*. All particles modify their position on a coordinate-by-coordinate basis with the help of velocities as shown in the following equation:

$$Y_{iD}^{t+1} = Y_{iD}^t + V_{iD}^t \tag{2}$$

Each element of the velocity vector is updated using a weighted combination of three factors: the current velocity; the difference between the current position and the particle's personal best position; and the difference between the current position and the swarm's global best position. The weight on the difference between the current position and the particle's personal best position is composed of a random number r_1, between (0, 1), and a tunable learning factor, c_1. The weight on the difference between the current position and the swarm's global best position is composed of a random number r_2, between (0, 1), and a tunable learning factor, c_2. Equation (3) illustrates the velocity updating equation, in which *w* is the weight on the current velocity. We discuss the tuning of these

parameters in Chapter 4. Eberhart and Shi [11] introduced the weighting parameter *w* to the velocity equation to balance local and global search.

$$V_{iD}^{t} = wV_{iD}^{t-1} + c_1 r_1 (P_{iD}^{t} - Y_{iD}^{t}) + c_2 r_2 (P_{gD}^{t} - Y_{iD}^{t}) \qquad (3)$$

Chao-Tang and Ching-Jong [8] describe the velocity update's intent as the adjustment of the searching direction of particles in *D*-dimensional space. One consideration in the generic PSO is that velocities may increase (or decrease) without bound, leading to particles making large steps in the solution space. Kennedy [15] used a constant V_{max} to limit the range of velocity, requiring that $v_{ij}^{t} \in (-V_{\max}, V_{\max})$. Figure 3.1 illustrates the idea of a swarm with 4 particles for a problem in which *D*=2.

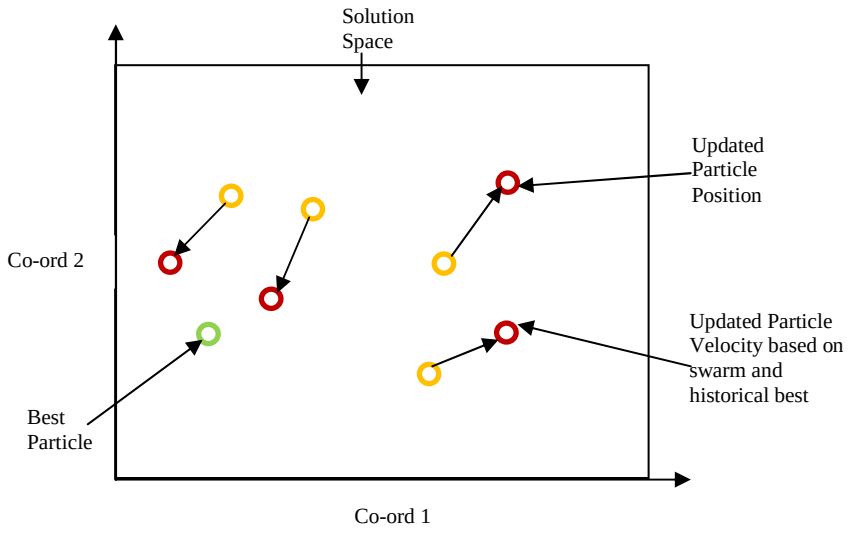


Figure 3.1: Example Swarm with Four Particles and Global Best – Two Coordinates

A generic description of the PSO algorithm is as follows:

Step 0: Initialize each particle position and velocity randomly. Go to Step 3.

Step 1: Update each particle's velocity according to Equation (3)

Step 2: Update each particle's position according to Equation (2)

Step 3: Evaluate the objective function for each particle.

Step 4: Update the personal best particle for each coordinate.

Step 5: Update the swarm's global best particle.

Step 6: If the stopping conditions are met, return the global best particle. Otherwise, go to Step 1.

Steps 0, 2 and 3 require further description based on our application of PSO to the FFL problem. First, we describe the solution representation, which impacts Steps 0 and 3. Then we describe how we initialize the particles (position and velocities) for Step 0. Two alternatives for updating velocities are then discussed. We conclude this chapter with an example problem.

3.2.1. Random Keys Representation

PSO operates in Υ^D and so cannot be directly applied to a permutation representation of a combinatorial optimization problem such as the traveling salesman problem or the problem of interest, flexible flow line scheduling. We represent solutions for the flexible flow shop with random keys in the same manner as Kurz and Askin [16]. In Random Keys PSO for FFL, each potential solution is represented by a particle with D equal to the number of jobs in the problem instance. For example, the particle Y_1 = (1.32, 1.22, 0.55, 0.35, 1.74, 0.65) is for a problem with 6 jobs. Each of the particle positions $y_{ij}^t \in [0, M)$,

where *M* is the number of machines in stage 1, serve as machine assignment and sort keys to decode the solution, following Bean [3] and Kurz and Askin [16]. 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.

Let us consider a problem with only a single stage and two machines. The positions for the six jobs are shown in Table 3.1. The particles tells us that in this solution, machine 0 has jobs 4,3 and 6, in that order and machine 1 has jobs 2, 1 and 5 in that order shown in Table 3.2. With this sequence and with job processing time and sequence-dependent setup times, we can determine when each job completes processing on the machines for each schedule.

Table 3.1: Positions for an Example Particle

Job	1	2	3	4	5	6
Position	1.32	1.22	0.55	0.35	1.74	0.65
Random Keys and Particle Representation						

Table 3.2: Sequences for an Example Particle

Machine 0	4	3	6	
Machine 1	2	1	5	
Job Sequence				

As described in Chapter 2, the makespan for a flexible flow line is computed using these sequences on stage 1 and the Best Completion Time algorithm. This computation of makespan (C_i^t) comprises Step 3.

3.2.2. Particle Initialization (Step 0)

Recall that *S* is the number of particles in a swarm. Position *j* of particle *i* is initialized randomly $y_{ij}^t \in [0, M)$, as described in the previous section. Kennedy et al. [15] introduced the constant V_{max} to limit the range of velocity. We must also limit the velocity because of the limited range of values that the positions can take on. Consider a problem with two machines in the first stage, in which each position must be in the range [0, 2). If the velocity in some coordinate is greater than 2, it is obvious that any position update will result in an infeasible solution. We introduce two potential mechanisms to deal with the issue of positions being outside the range of allowable values in the position update step, but first, we focus on setting a value for V_{max} for the purposes of initializing the particle velocities. We set V_{max} as 0.5 in this research so that jobs change machines approximately in 50% of position updates. We initialize and maintain the velocity *j* of particle *i* at iteration *t* as $V_{ij}^t \in \left(-V_{\max}, V_{\max}\right)$.

3.2.3. Position Update: Particle Bounceback

As described in the previous section, some velocity values may force a particle out of the range of allowable values for the position. For example, if $y_{ij}^t \in [0, 2)$, $y_{23} = 1.95$ and $v_{23} = 0.2$, the new value $y_{23} = 2.15$ will be found. We propose two potential solutions. First, we can fix the value of y_{ij}^t to a value close to its upper bound,

if it is too big, or at 0 if it is too small. Alternatively, we can ricochet the particle off the boundary back into the solution space. We call the ricochet "bounceback" and indicate the fixing of the position to near the boundary by the phrase "no bounceback". This is an innovative feature in the proposed PSO. Figure 3.2 shows the difference in potential positions in the case of bouncing back and not bouncing back.

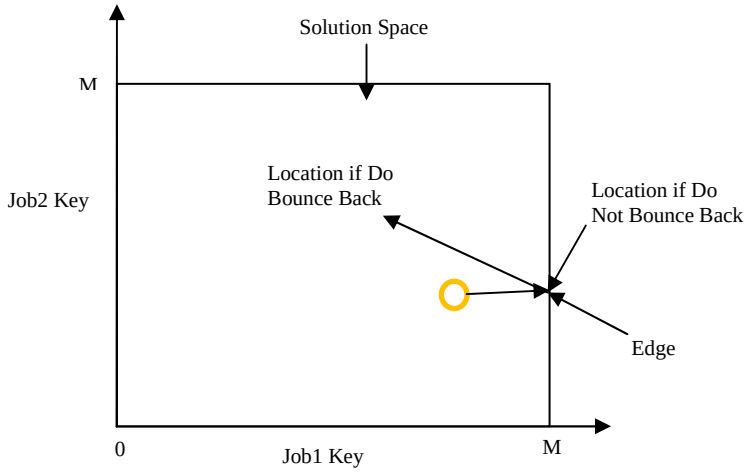


Figure 3.2: Graphical Representation Illustrating Bounce back of Random Keys PSO

We first update the positions according to Equation(2). Then we consider the potential for violation of the range of position values.

We implement the "no bounceback" option according to the following decision rules:

$$\text{If } y_{ij}^{t} \geq M,\ y_{ij}^{t} = M - \varepsilon \tag{4}$$

$$\text{If } y_{ij}^{t} < 0,\ y_{ij}^{t} = 0 \tag{5}$$

We implement the “bounceback” option according to the following decision rules:

$$\text{If } y_{ij}^t \geq M,\ y_{ij}^t = 2M - y_{ij}^t \tag{6}$$

$$\text{If } y_{ij}^t < 0,\ y_{ij}^t = -y_{ij}^t \tag{7}$$

3.2.4. Key Features of the Proposed PSO

The proposed PSO provides a contribution to the literature due to its use of the random keys representation, developed originally for use in genetic algorithms, and in its development of the “bounceback” mechanism to ensure that the particles remain feasible.

3.3. Sample Example

For the sake of clarity, we illustrate an example of our Random Keys PSO for a flexible flow shop problem with 2 jobs and 3 machines on stage 1 and 1 machine on stage 2. All time units used in our example are seconds. Let p_i^t be the processing time of job *i* on stage *t* shown in Table 3.3 and $s_{i,j}^t$ be the setup time from job *i* to job *j* on stage *t*.

Table 3.3: Example Processing Time Data

Stage *t*	Job *i*	p_i^t
1	1	3
	2	6
2	1	1
	2	7

We assume that no setup is required for the first job on each machine. The input data for the example problem is shown in Table 3.4. The PSO parameters are set as $c_1 = c_2 = w = 1$ and $V_{max} = 3.0$.

Table 3.4: Example Setup Time Data

Stage t	$s^t_{i,j}$	To job j	
1	From job i	1	2
	1	-	4
	2	2	-
2	From job i	1	2
	1	-	8
	2	5	-

Consider a 3 particle swarm, illustrated in Figure 3.3 and Table 3.5. The randomly generated positions and velocities of all three particles are shown in Table 3.5. Since the number of machines is 3, the positions are randomly selected such that $y_{ij} \in [0,3)$. The initial velocities are randomly generated as shown in Table 3.5. Now with the help of particle locations we can compute the makespan for each particle.

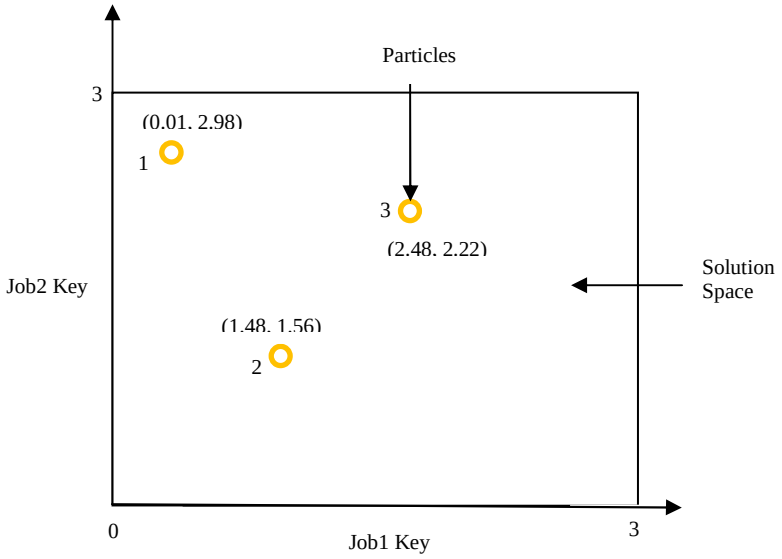


Figure 3.3: Example Initial Swarm Position with three particles

Consider the first particle. At stage 1, machine 0 has Job 1 and machine 2 has job 2. Job 1 will complete stage 1 at 3 and job 2 completes stage 1 at 6, as shown in Figure 3.4. Job 1 arrives at stage 2 at time 3, and then completes at time 4. Once job 2 arrives at stage 2, at time 6, the setup between jobs 1 and 2 can begin.

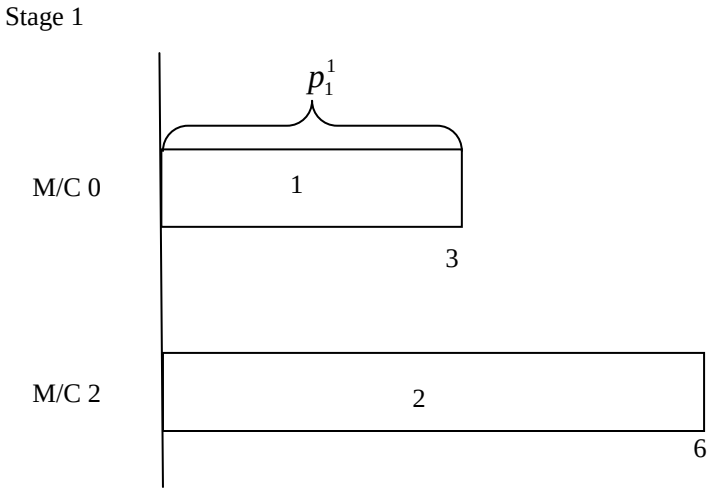


Figure 3.4: Processing time for job 1 and 2 at stage 1

Setup completes at time 14 (=6+8) and then job 2 completes at time 19, as shown in Figure 3.5. Following the same procedure, the makespans of particle 2 and particle 3 are determined and shown in Table 3.5.

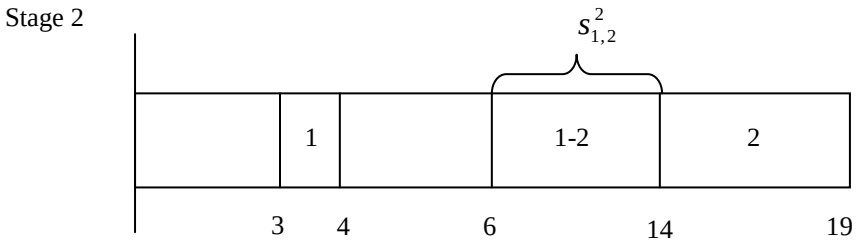


Figure 3.5: Total Completion time for job 1 and 2 at stage 2

Table 3.5: Example Problem Initial Swarm Data

Particle *i*	*Locations*		*Velocities*		Makespan
	Job 1 y_{i1}	Job 2 y_{i2}	Job 1 v_{i1}	Job 2 v_{i2}	C_i^t
1	0.01	2.98	0.19	0.80	19
2	1.48	1.56	-0.06	0.38	28
3	2.48	2.22	-0.52	0.09	17

Since this is the initialization step, $P_i = Y_i \ \forall i$ and $P_g = Y_3$. Next, we update all velocities. For particle 1, assume that r_1= r_2=0.5, c_1= c_2=w=1 and V_{max} = 3.0. Using Equation (3), the velocity of particle 1 for job 1 is calculated by v_{11} = 0.19 + (2.48-0.01) = 2.66, and the velocity of particle 1 for job 2 is calculated as v_{12}= 0.8 + (2.22-2.98) =0.04. Similarly, particle 2 and 3 velocities are determined and summarized in Table 3.5.

Now, we update all particle locations. Using Equation (2) for particle 1, job 1, we find y_{11} = 0.01 + 2.66 = 2.67. Similarly, y_{12} = 2.98 + 0.04 = 3.02. Since $y_{ij} \in [0,3)$ we have the situation that the particle y_{12} is going beyond the solution space as illustrated in Figure 3.6.

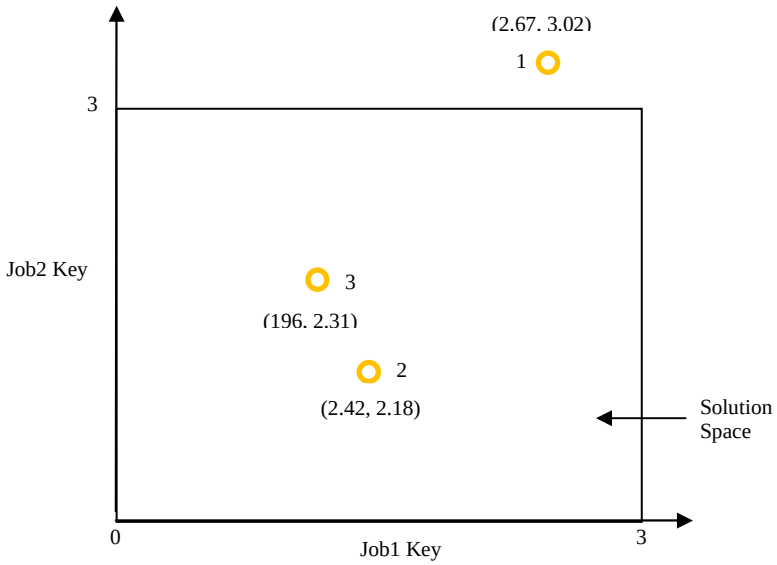


Figure 3.6: Example Swarm Position (Before Bounce Back)

If we do not utilize "bounceback", the new value of y_{12} will be set as 2.99, if we are using 2 decimals. If we utilize "bounceback", Equation (6) provides the new value of y_{12} as 2.98. Now the particle key is inside the solution space. Figure 3.7 shows the bounce back of particles into solution space.

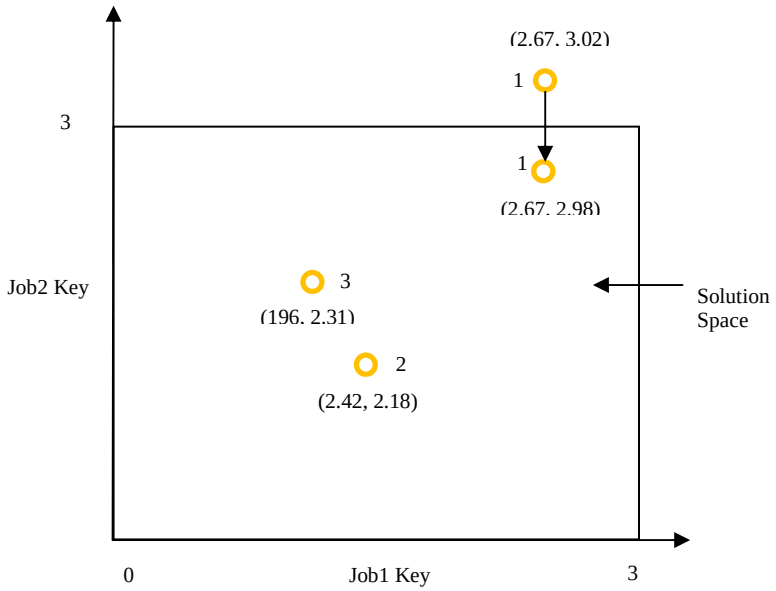


Figure 3.7: Example Swarm Position (After Bounce Back)

The makespans of all three particles are determined and shown in Table 3.6. Now particle 2 has the lowest makespan of 17.

Table 3.6: Example Problem Final Swarm Data

Particle i	*Locations*		*Velocities*		Bounce Back	Makespan C_{max}
	Job 1 y_{i1}	Job 2 y_{i2}	Job 1 v_{i1}	Job 2 v_{i2}		
1	2.67	2.98	2.66	0.04	Yes	28
2	2.42	2.18	0.94	0.62	No	17
3	1.96	2.31	-0.52	0.09	No	25

CHAPTER FOUR

4. EXPERIMENT AND RESULTS

In this chapter we elaborate about the experimental data setup for Random Keys PSO, tuning of learning and weighting factors, implementation of Random Keys PSO and finally discuss the results of our Random Keys PSO.

4.1. Generation of Experiment Test Data

Kurz and Askin [16] generated a large data set for the flexible flow line with sequence setup times. We used their experimental data for our flexible flow shop problem. Data required for our flexible flow shop with sequence dependent setup times consists of the range of processing times, number of jobs to be processed, number of stages with data explaining how many machines exist at each particular stage, processing times, ready times and sequence dependent setup times. The processing times are from one of two levels: uniformly distributed in the range of [50-70] or [20-100]. The distinguishing factor in our flexible flow line problem is the sequence dependent setup times. Our setup times are asymmetric; Kurz and Askin [16] generated them using the characteristics of setup time developed by Rios-Mercado and Bard [23]. The setup time matrices satisfy the triangle inequality and the setup times are uniformly distributed with the range of [12-24]. It is assumed that the largest number of machines in a stage should be less than the number of jobs to be processed at that stage. All jobs are assumed to be available for scheduling at time 0; at subsequent stages, the completion times at stage t are the ready times at stage t+1. In the proposed flexible flow line a job should not revisit a stage

which is already visited and jobs can skip some stages. Following Leon and Ramamoorthy [18], the probability of a job skipping a stage was fixed to be 0, 0.05 or 0.04.

Based upon the above discussion, we see that the experimental data depends up on the factors and levels described in Table 4.1. There are 3x2x3x5x2=180 test scenarios. For each scenario, Kurz and Askin [16] provide ten data sets. Therefore there are 1800 input data files. We subjected each of these 1800 input files to the Random Keys PSO multiple times.

Table 4.1: Experimental Data Setup

Factor	Levels	Values
Skipping Probability	1 2 3	0.00 0.05 0.40
Processing Times	1 2	Unif(50-70) Unif(20-100)
Number of Stages	1 2 3	2 4 8
Number of Machines	1 2 3 4 5	1 2 10 Unif(1,4) Unif(1,10)
Number of Jobs	1 2	30 100

4.2. Stopping Criteria and Other Parameters

In each replication, Random Keys PSO will run for 500 iterations or until the lower bound is achieved**.** Solutions are evaluated by their deviation from the lower bound. Each swarm has a population size of 100 particles. V_{max} is set at 0.5.

4.3. Lower Bound

Kurz and Askin [16] developed a strong lower bound for flexible flow line with sequence dependent setup times. Two lower bounds are actually computed for each of the input files; the higher of each is used as the lower bound for the input file. LB^1 assumes that every job must be processed at every stage while LB^2 is developed with the assumptions that every stage must process all of its jobs and we should also include the time for the first job to get to each stage and leave it as well. The solutions of our Random Keys PSO are evaluated with the help of their derivation from the lower bound.

$$LB^{(1)} = \max_{i=1,\ldots,n} \left\{ \sum_{t \in S_i} \left(p_i^t + \min_{j=0,\ldots,n} s_{ij}^t \right) \right\} \tag{8}$$

$$LB^{(2)} = \max_{t=1,\ldots,g} \left\{ \begin{array}{l} \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ij}^\tau \right) + \dfrac{\sum \left(p_i^t + \min_{j=0,\ldots,n} s_{ij}^t \right)}{m^t} + \min_{i \in S^t} \sum_{\tau=t+1}^{g} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ij}^\tau \right) + \\ \dfrac{1}{m^t} \sum_{k=1}^{m^t-1} \left[\min_{i \in S^t}{}_{[k]} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ij}^\tau \right) - \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ij}^\tau \right) \right] \end{array} \right\} \tag{9}$$

Following Kurz and Askin [16] we used the best lower bound for each test scenario. The solutions are compared and evaluated by the "Loss" where loss is the percentage deviation above the lower bound for the makespan and it is used as the key performance measure in our research. Loss is computed as shown in Equation (10) where C_{max} is the makespan determined by Random Keys PSO and *LB* is the lower bound. A loss of 0 indicates that the optimal solution is found.

$$loss = \frac{(C_{max} - LB)}{\mathrm{LB}} * 100 \qquad (10)$$

4.4. Random Numbers

The random numbers are generated using the Mersenne Twister random number generator. It is a pseudorandom number generator developed by Matsumoto and Nishimura [19]. Their algorithm generates random number uniformly in the range of [0, $2^{32} - 1$] for 32 bit integers, with a period of 2^{19937}-1. This pseudorandom number generator allows the coder to ensure that non-overlapping but reproducible pseudorandom number streams are used.

4.5. Computational Environment

The proposed Random Keys PSO is developed using C-language and compiled with Microsoft Visual Studio 2005. Computational experiments are facilitated through the use of high-throughput computing via Clemson's Condor grid. The quality of this research heavily depends on computing throughput. It is not uncommon to find problems that

require weeks or months of computation to solve. As described by Basney et al. [2], high-throughput computing (HTC) refers to environments in which large amounts of computing capacity are available over long periods of time. The initial set of 1800*54 runs used for Experiment 1 took less than one calendar day, using over 1500 CPU hours.

4.6. Experiment 1: Tuning of Parameters

The learning and weighting parameters play a major role in determining the velocity and position of the particles in the solution space. In order to evaluate the performance of Random Keys PSO, an experiment was conducted by tuning the learning and weighting factors in Equation (3). The initial population was generated randomly. The initial velocities are determined using Equation (3).

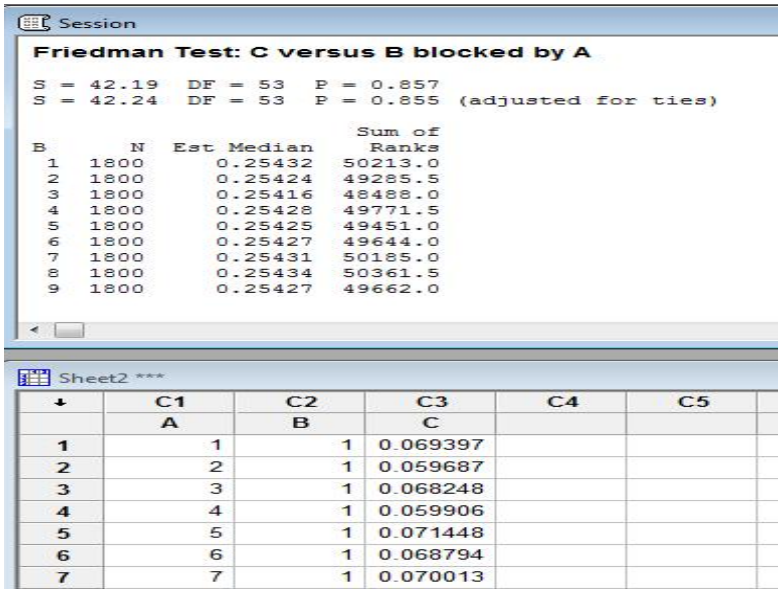
Following Kennedy et al [15], the learning parameters evaluated are $c_1 \in \{1, 1.5, 2\}$ and $c_2 \in \{1, 1.5, 2\}$. Following Chao-tang and Ching-Jong [8], the weighting factors evaluated are $w \in \{0.8, 1, 1.2\}$. One of the distinguishing characteristics of our Random Keys PSO is the proposed "bounce back" of particles into the solution space so we consider the PSO with and without bounceback to be tuned as well. We set the factor *b* to be either "Yes" (we use bounceback) or "No". These tunable Random Keys PSO parameters have the different levels summarized in Table 4.2. In Total there are 3x3x3x2=54 settings considered in our experiments. Since there are 1800 input problem data files, we consider 1800x54=97200 test scenarios with our Random Keys PSO.

Table 4.2: Tuning of Parameters

Parameter	Levels	Values
c_1	1 2 3	1 1.5 2
c_2	1 2 3	1 1.5 2
w	1 2 3	0.8 1 1.2
b	1 2	No Yes
Number of Settings	3*3*3*2=54	

Each of 54 settings of Random Keys PSO were tested for five replications; this low level of replications was used for the tuning experiment since tuning experiments should be smaller than the final experiment. Each of the 97200*5 makespans generated by this experiment was transformed into loss values using the lower bound. The range of loss values is found be between 1.5% and 95% above the lower bound. Since the number of replications is low, a non-parametric test was conducted in order to find the best

parameter setting for the Random Keys PSO. The average of the 5 replications for each of the 54 setting and 1800 file combinations were used as input into the test.



Session

Friedman Test: C versus B blocked by A

S = 42.19 DF = 53 P = 0.857
S = 42.24 DF = 53 P = 0.855 (adjusted for ties)

B	N	Est Median	Sum of Ranks
1	1800	0.25432	50213.0
2	1800	0.25424	49285.5
3	1800	0.25416	48488.0
4	1800	0.25428	49771.5
5	1800	0.25425	49451.0
6	1800	0.25427	49644.0
7	1800	0.25431	50185.0
8	1800	0.25434	50361.5
9	1800	0.25427	49662.0

Sheet2 ***

	C1	C2	C3	C4	C5
	A	B	C		
1	1	1	0.069397		
2	2	1	0.059687		
3	3	1	0.068248		
4	4	1	0.059906		
5	5	1	0.071448		
6	6	1	0.068794		
7	7	1	0.070013		

Figure 4.1: Friedman Test Results

The Friedman test was executed using MINITAB 15 and the results are shown in Figure 4.1 and Appendix A. The 54 settings are considered as the treatments. The treatment with lowest sum of ranks was the best treatment and taken as best parameter settings for Random Keys PSO.

The results shown in Appendix A were analyzed. Unfortunately, we cannot conclude that there is any significant difference between the settings. It was observed that the setting c_1=2, c_2=2, w=2, b=2, appears to yield the lowest makespan when compared to the other parameter settings. Therefore, these final parameter values are selected for use in Experiment 2 and are shown in Table 4.3.

Table 4.3: Final Parameter Levels and Values

Parameter	Level	Values
c_1	2	1.5
c_2	2	1.5
w	2	1
b	2	Yes

4.7. Experiment 2: Replication of Tuned RK PSO

With the final parameter values (c_1, c_2, w, b) = (1.5, 1.5, 1, yes), Random Keys PSO is applied to the same 1800 data set for 50 replications. In each replication Random Keys PSO will run for 500 iterations or until the lower bound is achieved**.** Computational experiments are facilitated through the use of high-throughput computing via Clemson's Condor grid. The 1800*50 runs took less than one calendar day, using over 1400 CPU hours.

The results are compared with the Genetic Algorithm (GA) developed by Kurz and Askin [16] for flexible flow line. It is found that GA works better than the Random Keys PSO even though the values sometimes appears to be far from the lower bound followed by Kurz and Askin [16]. The makespans of Random Keys PSO are compared and evaluated with the help of the loss figure of merit. The average loss over the 50 replications for each of the 1800 test scenarios is calculated. Here, we assume that the sample size allows us to use the Central Limit Theorem and the averages are assumed to be normally distributed. A single factor ANOVA test is done to demonstrate at 99% confidence level

whether there is any significant difference between each of the factors or not. The p-value from the ANOVA test is compared with α-value. Single factor ANOVA for all five factors are shown in Appendix B.

Table 4.4: Experimental Results

Factor	Levels	Minimum	Average	Maximum	Single Factor p-value	Does level matter? (conclusion)
Skipping Probability	1	0.03	0.09	0.27	<0.01	Yes
	2	0.08	0.21	0.47		
	3	0.12	0.29	0.55		
Processing Times	1	0.04	0.19	0.55	<0.01	Yes
	2	0.03	0.20	0.48		
Number of Stages	1	0.03	0.19	0.50	<0.01	Yes
	2	0.04	0.19	0.49		
	3	0.04	0.20	0.55		
Number of Machines	1	0.04	0.17	0.31	<0.01	Yes
	2	0.05	0.19	0.41		
	3	0.12	0.28	0.55		
	4	0.03	0.17	0.33		
	5	0.04	0.18	0.32		
Number of Jobs	1	0.03	0.17	0.50	<0.01	Yes
	2	0.06	0.22	0.55		

4.7.1. Impact of skipping probability on RK PSO

From Table 4.4, we see that the levels of the Skipping Probability factor are significant. From Figure 4.2 it is found that when all jobs visit all stages (level 1 of the Skipping Probability factor), Random Keys PSO performs significantly better with the average loss of makespan between 0.03 to 0.27.

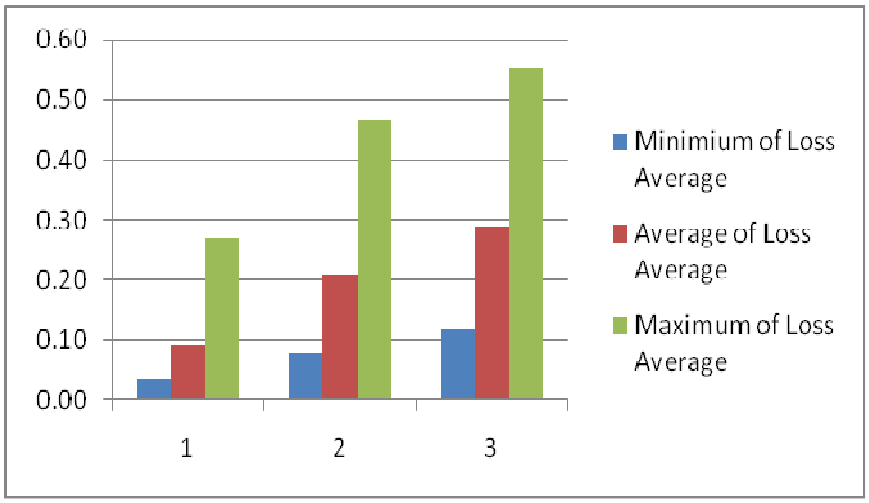


Figure 4.2: Difference in levels of Skipping Probability

We conjecture that the more "flexible" the flow line (as in a semiconductor industry in which all jobs do not undergo the same operations), the less appropriate PSO may be as a scheduling algorithm. In the proposed PSO with random keys encoding, a job that does not visit stage 1 will still have a key but the algorithm will spend time trying to find a good coordinate for that job. The PSO may make a lot of moves in the swarm space that don't actually move the job in the schedule in the solution space.

4.7.2. Impact of Processing Time on RK PSO

From Table 4.4, we see that the levels of the Processing Time factor are significant. Figure 4.3 indicates that level 1 has a better performance than level 2. The processing times are uniformly distributed with the same mean (60) but the level 1 range is 50 - 70 while the level 2 range is 20 - 100. Consider how makespan can be impacted by reversing the order of two jobs when at both ends of the processing time ranges. When the job processing times range from 50 to 70, the difference in makespan could be only as much as 20 time units, assuming one of these two jobs define the makespan. When the job processing times range from 20 to 100 but all other factors are identical, the difference in makespan could be much as 80 time units. We find that RK PSO performs better when the range of processing times is smaller, possibly because the order of jobs impacts makespan less in this case. We conjecture that this observation may hold true for any algorithm used to solve this problem.

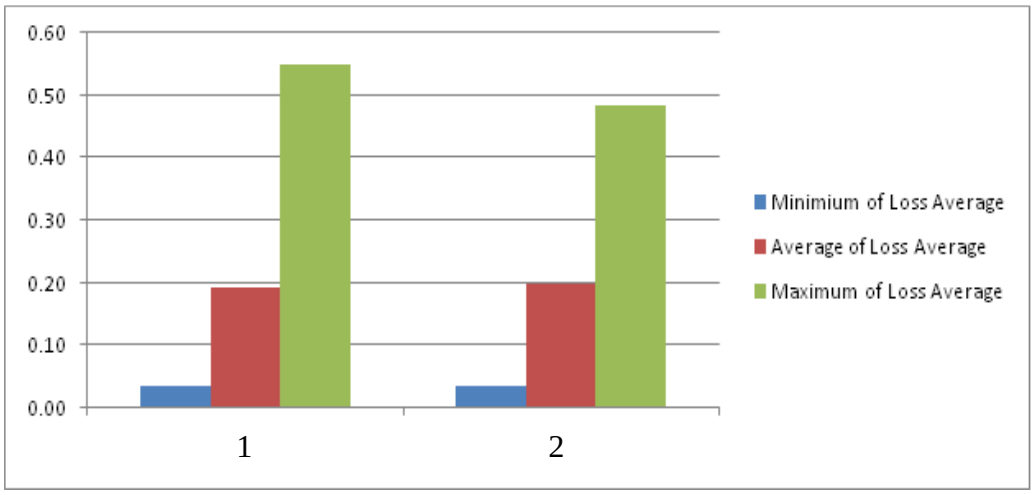


Figure 4.3: Difference in levels of Processing Times

4.7.3. Impact of Number of Stages on RK PSO

From Table 4.4, we see that the levels of the Number of Stages factor are significant. Figure 4.4 indicates that level 1 has a better performance than levels 2 or 3. Recall that level 1 corresponds to 2 stages, level 2 to 4 stages and level 3 to 8 stages. In general, the order induced by the stage 1 schedule persists strongly throughout the later stages due to the algorithm used to assign jobs to machines in later stages. The more stages the problem has, the less appropriate the initial order may be on later stages. This is insight can be understood by considering a regular flowline. Johnson's Rule tells us an optimal permutation schedule can be created for a 2 stage problem, and that a three stage problem can be solved optimally in some situations. Scheduling literature also tells us that in a regular flow line, the first two stages and the last two stages should have the same order of jobs, even if these orders are not the same. We can use this knowledge to conjecture that in a two stage flexible flow line problem, even with sequence dependent setup times, the order induced by the first stage may be reasonable for the second stage. However, the sequence induced by the first stage may be very poor for the later stages in an eight stage problem. A more detailed solution representation may allow a better solution for the problems with more stages, but with a concurrent increase in running time.

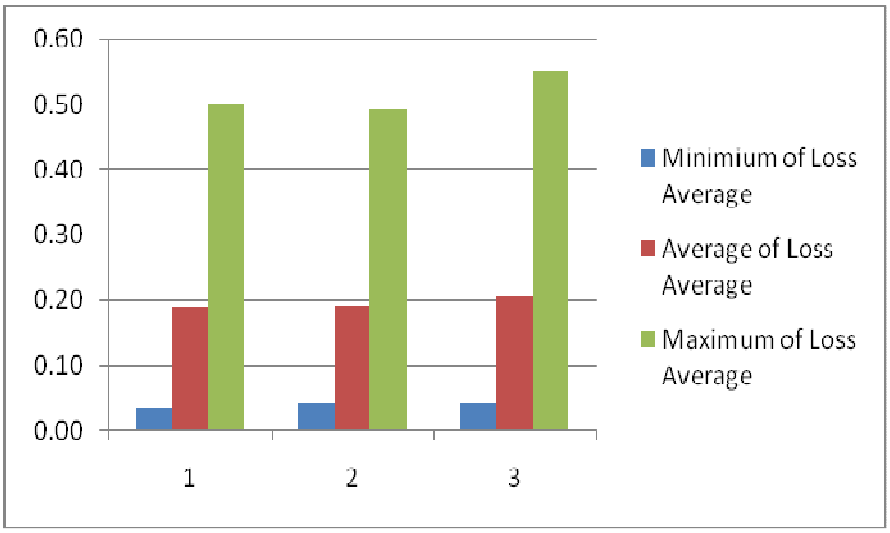


Figure 4.4: Difference in levels of Number of Stages

4.7.4. Impact of Number of Machines on RK PSO

From Table 4.4, we see that the levels of the Number of Machines are significant. Recall that level 1 corresponds to exactly 1 machine per stage, level 2 to exactly 2 machines per stage, level 3 to exactly 10 machines per stage. Levels 4 and 5 correspond to the cases of between 1 and 4 machines per stage (level 4) and between 1 and 10 machines per stage (level 5). From Table 4.4 and Figure 4.5 Random Keys PSO performs best when the number of machines is exactly 1 per stage. When the number of machines is exactly 10 per stage, random keys PSO performs very poorly, with the average loss increasing to 0.12 to 0.55.

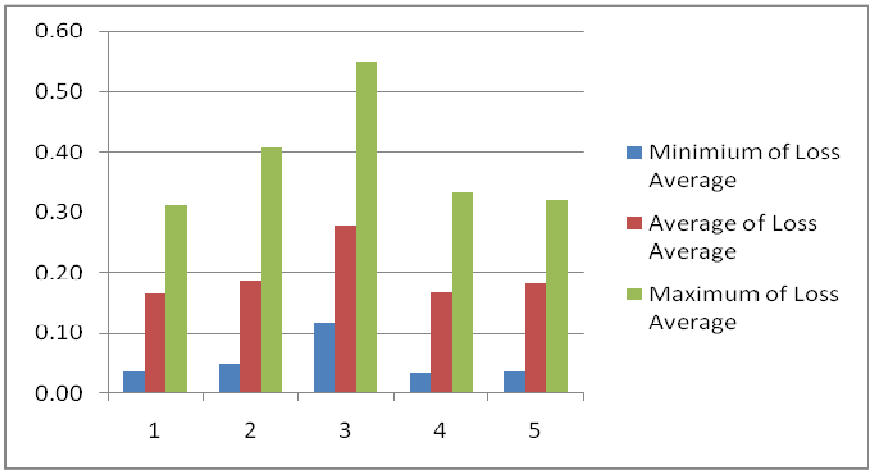


Figure 4.5: Difference in levels of Number of Machines

The insights from the previous factor are applicable to this case as well: in the level 1 cases, the permutation induced by the stage 1 order is preserved in later stages. On the other hand, when the number of machines per stage is exactly 10, the solution space is much larger and the solution representation cannot allow the changes that may be necessary in later stages. We believe this is also related to the interplay between the number of machines and the stopping criteria. Since the search space is so much larger when the number of machines is higher, the Random Keys PSO needs more time to find a good particle location; the current design is flawed since the number of iterations and particles is fixed regardless of the size of the solution space.

4.7.5. Impact of Number of Jobs on RK PSO

From Table 4.4, we see that the levels of the Number of Jobs are significant. As the number of jobs increases, makespan increases proportionally. Random keys are used to sort the jobs to suitable machines so increasing the number of jobs will leads the Random Keys PSO to take more time to search for the best particle in solution space.

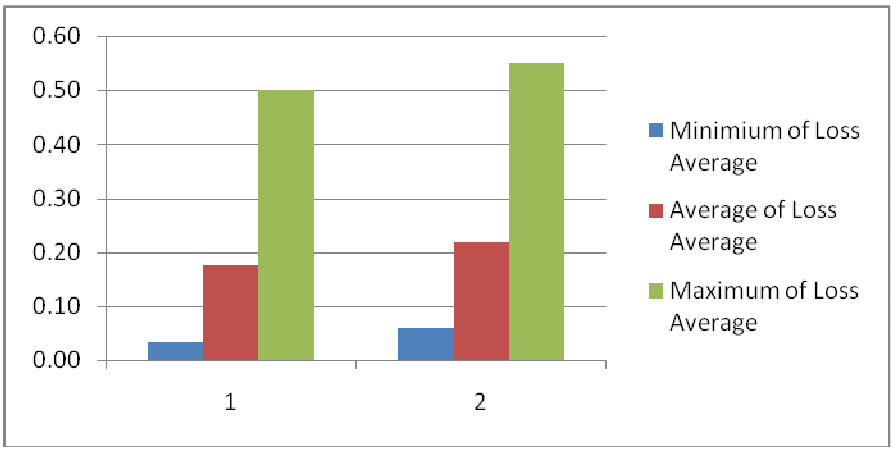


Figure 4.6: Difference in levels of Number of jobs

Figure 4.6 illustrates how the average loss increases when the number of jobs increases. We believe this performance is also directly related to the interplay between the number of jobs and the stopping criteria. Since the search space is so much larger when the number of jobs is higher, the Random Keys PSO needs more time to find a good particle location; the current design is flawed since the number of iterations and particles is fixed.

CHAPTER FIVE

5. CONLUSION AND FUTURE WORK

This research focused on one metaheuristic approach called Particle Swarm Optimization applied to find the makespan minimizing schedule in a flexible flow line with sequence-dependent setup times. PSO for scheduling problems is the vast area where there is an unlimited opportunity for researchers. The proposed PSO has been adapted for use in permutation problems in a novel fashion. The proposed PSO features the "bounceback" mechanism, using Random Keys as a solution representation and tuning of learning and weighting parameters. The experimental data come from Kurz and Askin [16]. There are 180 test scenarios with 10 files of each type. The Random Keys PSO is evaluated based on its performance on the 1800 data files. The computational experiments are facilitated through Clemson's high throughput machine via Clemson Condor grid. The results are compared with the lower bound developed by Kurz and Askin [16].

We find that the proposed PSO does not perform well in general to minimize makespan in a flexible flow line with sequence-dependent setup times. The makespan deviates from the lower bound more as the number of machines, jobs and stages increases. These problem characteristics increase the search space significantly, requiring the particles to explore more space before finding a good location. From the results it is also evident that Random Keys PSO performs significantly better when the all jobs visit all the stages.

We use this experience to provide the following suggestions for future research:

- The solution representation should allow for corrections to the ordering in later stages, perhaps by allowing the entire schedule's machine assignment and job ordering to be explicitly represented.
- The stopping criteria must consider the size of the solution space, which is a function of the number of stages, the number of machines at each stage and the number of jobs.

Using these insights, the performance of Random Keys PSO (and other heuristics and metaheuristics) can be improved.

APPENDIX A

FRIEDMAN TEST AND RESULTS

Friedman		Test: C versus	B blocked by A
S =42.19		DF = 53	P=0.857
S =42.24		DF = 53	P= 0.855 (adjusted for ties)

B	N	Est Median	Sum of Ranks
1	1800	0.25432	50213
2	1800	0.25424	49285.5
3	1800	0.25416	48488
4	1800	0.25428	49771.5
5	1800	0.25425	49451
6	1800	0.25427	49644
7	1800	0.25431	50185
8	1800	0.25434	50361.5
9	1800	0.25427	49662
10	1800	0.25434	50481
11	1800	0.2542	48923
12	1800	0.25423	49241
13	1800	0.2542	48888.5
14	1800	0.25424	49343.5
15	1800	0.25434	50457
16	1800	0.25425	49453.5
17	1800	0.25416	48454.5
18	1800	0.25426	49561
19	1800	0.25427	49690.5
20	1800	0.25426	49480.5
21	1800	0.25427	49571.5
22	1800	0.25429	49895.5
23	1800	0.25426	49507.5
24	1800	0.25425	49466.5
25	1800	0.2543	49988
26	1800	0.25422	49086.5
27	1800	0.25428	49751

Results	
Lowest Rank	48322.5
Treatment	28
Experiment	2222

Parameter	Values	Levels
c1	1.5	2
c2	1.5	2
w	1	2
b	TRUE	2

28	1800	0.25415	48322.5
29	1800	0.2542	48839
30	1800	0.25423	49215.5
31	1800	0.25417	48590.5
32	1800	0.25423	49283
33	1800	0.25424	49272.5
34	1800	0.25435	50466.5
35	1800	0.25424	49293
36	1800	0.25425	49528
37	1800	0.25429	49960
38	1800	0.25429	49837
39	1800	0.2542	48937
40	1800	0.25422	49120
41	1800	0.25431	50065.5
42	1800	0.25433	50347
43	1800	0.25425	49480.5
44	1800	0.2542	48872
45	1800	0.25427	49645.5
46	1800	0.25422	49089
47	1800	0.2543	50008
48	1800	0.25416	48561
49	1800	0.25435	50503.5
50	1800	0.25424	49324
51	1800	0.25426	49578.5
52	1800	0.25438	50746
53	1800	0.25423	49203.5
54	1800	0.25421	49085.5
Grand Median		0.25426	

APPENDIX B

ANOVA SINGLE FACTOR

1. Skipping Probability

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Skipping Prob	180	360	2	0.670391
opt 2 average	180	35.22483	0.195693	0.011833

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	292.997	1	292.997	858.9465	3.81E-97	6.706193
Within Groups	122.1181	358	0.341112			
Total	415.1151	359				

2.Processing Times

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Proc. Times	180	360	2	1.005587
opt 2 average	180	35.22483	0.195693	0.011833

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	292.9969761		292.997	575.961	1.53E-76	6.706193
Within Groups	182.118105	358	0.50871			
Total	475.11508	359				

3.Number of Stages

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Num of Stages	180	360	2	0.670391
opt 2 average	180	35.22483	0.195693	0.011833

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	292.997	1	292.997	858.9465	3.81E-97	6.706193
Within Groups	122.1181	358	0.341112			
Total	415.1151	359				

4.Number of Machines

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
mach factor - new	180	540	3	2.011173
opt 2 average	180	35.22483	0.195693	0.011833

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	707.7721	1	707.7721	699.7232	3.14E-86	6.706193
Within Groups	362.1181	358	1.011503			
Total	1069.89	359				

5. Number of Jobs

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Num of Jobs	180	270	1.5	0.251397
opt 2 average	180	35.22483	0.195693	0.011833

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	153.1094	1	153.1094	1163.314	1.6E-114	6.706193
Within Groups	47.1181	358	0.131615			
Total	200.2275	359				

REFERENCES

1. Agnetis, A., Pacifici, A., Rossi, F., Lucertini, M., Nicoletti, S., Nicolo, F., Oriolo, G., Pacciarelli, D., and Pesaro, E. “Scheduling of flexible flow lines in an automobile assembly plant”. *European Journal of Operational Research*, 1997, 28, 348-362.
2. Basney, J., Livny, M., and Tannenbaum, T. “High Throughput Computing with Condor”. *HPCU News*, 1997, Vol. 1(2).
3. Bean J.C. “Genetic algorithms and random keys for sequencing and optimization”. *ORSA Journal on Computing*, 1994, 6,154-160.
4. Bellman, R., A.O. Esogbue, and I. Nabeshima. “Mathematical Aspects of scheduling and Applications”. *Pergamon Press*, 1982, New York.
5. Bianco, L., P.Dell’ Olmo, and S.Giordani. “Flowshop No-wait scheduling with sequence dependent setup times and Release dates”. *INFORM*, 1999, 37, 1, 3-19.
6. Brah, S.A., and Hunsucker, J.L. “Branch and bound algorithm for the flow shop with multiple processors”. *European Journal of Operational Research*, 1991, 51, 88-99.
7. Campbell, H.G., Dudek, R.A., and Smith, M.L. “A heuristic algorithm for the *n* job, *m* machine sequencing problem”. *Management Science*, 1970, 16(10), B630-B637.
8. Chao-Tang, T., and Ching-Jong, L. “A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks”. *International Journal of Production Research*, 2008, 46:17, 4655-4670.

9. Cheng, T.C Edwin, Jatinder N. D. Gupta, and Guoquing Wang. “A Review of flowshop scheduling research with setup times”. *Production and operations management*, 2009, Vol.9, 262-282.
10. Ding, F-Y., and Kittichatphayak, D. “Heuristic for scheduling flexible flow lines”. *Computers and Industrial Engineering*, 1994, 26(1), 27-34.
11. Eberhart, R.C., and Shi, Y. “Comparing inertia weights and constriction factors in particle swarm optimization”. *Proceedings of Congress on Evolutionary Computing*, 2000, pp. 84-88.
12. Gupta, J.N.D. “A flowshop scheduling problem with two operations per job”. *International journal of production research*, 1997, 35(8), 2309-2325.
13. Kennedy, J and Eberhart, R.C. “A discrete binary version of the particle swarm algorithm, in proceedings of the World Multiconference on Sytemics, Cybernetics and Informatics”, 1997, pp. 4104-4109.
14. Kennedy, J and Eberhart, R.C. “Particle swarm optimization”. *Proceedings of IEEE International Conference on Neural Networks*, 1995, pp. 1942-1948.
15. Kennedy, J., Eberhart, R.C and Shi, Y. “Swarm Intelligence”. *Morgan Kaufmann: CA*, 2001.
16. Kurz, Mary E. and Ronald G. Askin. “Scheduling flexible flow lines with sequence-dependent setup times”. *European Journal of Operational Research*, 2004,159, 66-82.

17. Lee, I., Sikora, R., and Shaw, M.J. “A genetic algorithm –based approach to flexible flow-line scheduling with variable lot sizes”. *IEEE transactions on system, Man, and Cybernetics-Part B: Cybernetics*, 1997, 27(1), 36-54.
18. Leon, V.J., and Ramamoorthy, B. “An adaptable problem-spaced-based search method for flexible flow line scheduling”. *IIE Transactions*1997, 29, 115-125.
19. Matsumoto, M., and Nishimura, T. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". *ACM Transactions on Modeling and Computer Simulation*, 1998, Vol.8, 3-30.
20. Piramuthu, S., Raman, N., Shaw, M.J. “Learning-based scheduling in a flexible manufacturing flow line”. *IEEE Transactions on Engineering Management*, 1994, 41(2), 172-182.
21. Quan-Ke, P., M.Fatih Tasgetiren., and Yun-Chia Liang. “A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem”. *Computers & Operations Research*, 2008, 35, 2807-2839.
22. Rajendran C. “A no-wait flowshop scheduling heuristic to minimize makespan”. *Journal of the Operational Research Society*, 1994, 45:472
23. Rios-Mercado, R.Z., and Bard, J.F. “A branch- and- bound algorithm for permutation flow shops with sequence- dependent setup times”. *IIE Transactions*, 1999, 31, 721-731.
24. Rios-Mercado, R.Z., and Bard, J.F. “Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups”. *Computers and Operations Research*, 1998, 25(5), 351-366.

25. Salvador, M.S. “A solution to a special case of flow shop scheduling problems”. *In: Elmaghraby, S.E (Ed.), Symposium on the theory of scheduling and Its application. Springer-Verlag*, 1973, pp.83-91.
26. Santos, D.L., Hunsucker, J.L, and Deal D.E. “FLOWMULT: Permutation sequences for flow shops with multiple processors”. *Journal of Information and Optimization Sciences*, 1995, 16(2), 351-366.
27. Wittrock, R. “An adaptable scheduling algorithm for flexible flow lines”. *Operations Research*, 1988, 36(3), 445-453.
28. Zhigang, L., Gu, X., and Jiao, B. “A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan”. *Applied Mathematics and Computation*, 2006, 175, 773-785.
29. Zhigang, L., Gu, X., and Jiao, B. “A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan”. *Applied Mathematics and Computation*, 2006, 183, 1008-1017.