Performance Evaluation of Continuous and Discrete Particle Swarm Optimization in Job-Shop Scheduling Problems

IOP Conf. Series: Materials Science and Engineering **530** (2019) 012044 doi:10.1088/1757-899X/530/1/012044

Performance Evaluation of Continuous and Discrete Particle Swarm Optimization in Job-Shop Scheduling Problems

N I Anuar[1*], M H F M Fauadi[2] and A Saptari[3]

[1] Faculty of Engineering & Technology, Multimedia University, Jalan Ayer Keroh Lama, 75450 Ayer Keroh, Melaka, Malaysia

[2] Faculty of Manufacturing Engineering, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

[3] Department of Industrial Engineering, President University, Jl. Ki Hajar Dewantara, Kota Jababeka, Cikarang Baru, Bekasi 17550 - Indonesia

Corresponding author *: nurulizah.anuar@mmu.edu.my

Abstract. The Particle Swarm Optimization (PSO) is an optimization method that was modeled based on the social behavior of organisms, such as bird flocks or swarms of bees. It was initially applied for cases defined over continuous spaces, but it can also be modified to solve problems in discrete spaces. Such problems include scheduling problems, where the Job-shop Scheduling Problem (JSP) is among the hardest combinatorial optimization problems. Although the JSP is a discrete problem, the continuous version of PSO has been able to handle the problem through a suitable mapping. Subsequently, its modified model, namely the discrete PSO, has also been proposed to solve it. In this paper, the performance of continuous and discrete PSO in solving JSP are evaluated and compared. The benchmark tests used are FT06 and FT10 problems available in the OR-library, where the goal is to minimize the maximum completion time of all jobs, i.e. the makespan. The experimental results show that the discrete PSO outperforms the continuous PSO for both benchmark problems.

1. Introduction

The Job-shop Scheduling Problem (JSP) is described as consisting of a set of jobs and a set of machines, where each job consists of a set of operations. Each operation of a job needs to be processed on a certain machine following a specified order, where this machine order is unique for each job. Each operation also has a processing time, specific on each machine. The order of machines that a job must visit or the specific order of operations is the precedence constraints for that job. These precedence constraints impose some difficulty to the JSP [1].

The Particle Swarm Optimization (PSO) is a population-based heuristics that is used to optimize the solution of many engineering problems. It is one of the evolutionary computation search algorithms, developed by Kennedy and Eberhart in 1995, which refers to a family of algorithms used to find optimal or near optimal solutions to numerical or qualitative problems. The original PSO algorithm was discovered through simulations of social behaviour; it is related to the bird flocking, fish schooling and swarm theory. PSO simulates this behaviour, learns from the scenario and uses it to solve the optimization problems [2].

PSO was first introduced to solve continuous optimization cases, and it has been extended to solve problems in discrete spaces as well. This is done by implementing a particle representation procedure that can encode the particle's position in the continuous PSO with the solution of discrete problems [3].

For scheduling problems like JSP, the particle representation procedure specifies the mapping between the position of the particle in PSO and the scheduling solution in JSP. Each particle in PSO represents its position in the continuous search space, but not the actual potential solution of the discrete problems. Thus, the mapping is established so that each particle in PSO can represent a schedule in JSP.

On the other hand, in the discrete PSO, each particle directly represents the candidate solution. For scheduling problems like JSP, this means each particle represent the sequence of operations, instead of its position in the search space. Thus, it eliminates the need to encode the particles to the potential scheduling solutions as compared to the continuous PSO.

In this paper, the continuous and discrete PSO are employed separately to find the values of the minimum makespan for FT06 and FT10 instances. Subsequently, the performances of these two versions of PSO are evaluated and compared to distinguish which version produces a better performance. The rest of this paper is organized in the following manner: first, the standard PSO is described along with a brief review of the discrete PSO. This is followed by a description of the continuous and discrete PSO implemented in this research. Lastly, the experimental results and conclusions are presented.

2. Particle Swarm Optimization

In a PSO algorithm, each agent is called a 'particle' and each particle corresponds to a position in the continuous, multi-dimensional search space, where it flies around with a velocity. It can be described as having an initial of randomized swarm of particles (a population of agents) and then searches for optima by updating each generation. The candidate solutions travel through the problem space by following the current optimum particles. All of the particles have fitness values which are evaluated by the fitness function to be optimized. All of these particles also have velocities which direct the flying of the particles [2].

This velocity is constantly updated by the particle's own experience and the experience of the particle's neighbours or the experience of the whole swarm. Thus, in every iteration, each particle is updated by following two 'best' values. The first 'best' value is the best solution (fitness) each particle has achieved so far. This value is called pBest. Another 'best' value is the best value obtained so far by any neighbour of the particle in the population. This value is called nBest. When a particle takes all the population as its neighbours, the best location is a global best. This value is called gBest.

After finding the two best values, the particle updates its velocity and position using the following equations:

$$v_{id} = w * v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

for $i = 1, 2, 3, \ldots, N_s$ and $d = 1, 2, 3, \ldots, D$
N_s = the total number of particles in the swarm
D = the problem dimension, i.e. the number of parameters of the function being optimized
w = inertia weight, to control the impact of the previous history of velocity on the current velocity
c_1 and c_2 = positive constants, called acceleration coefficients or learning factors
r_1 and r_2 = independently generated random numbers in the range [0, 1]
$x_{id} = [x_{i1}, x_{i2}, \ldots, x_{iD}]$, represents the position of the i^{th} particle
$v_{id} = [v_{i1}, v_{i2}, \ldots, v_{iD}]$, represents the velocity of the i^{th} particle
$p_{id} = [p_{i1}, p_{i2}, \ldots, p_{iD}]$, represents the best position of the i^{th} particle
g represents the index of the best particle among all particles in the population
$p_{gd} = [p_{g1}, p_{g2}, \ldots, p_{gD}]$ is the best position found among all particles in the population

The two equations above are the equations describing the flying trajectory of a population of particles. Equation (1) is applied to compute the new velocity of each particle based on its previous velocity and the distances of its current position from its own best experience and the neighbourhood's best experience. The particle will then fly towards a new position as per equation (2). Each particle's performance is assessed corresponding to a pre-defined fitness function, which is associated to the problem being solved.

PSO has been presented as an optimization method in the continuous domain. However, many optimization cases are established in the discrete domain. Typical examples in the discrete domain include problems that involve sequencing or permutation, such as in JSP. Consequently, the continuous PSO has been modified to operate on discrete problems; there are already several variants of discrete PSO developed to achieve this purpose. Based on the literature, discrete PSO techniques can be categorized into either binary numbers or positive integers [4].

In 1997, Kennedy and Eberhart extends their work of the continuous PSO to a discrete binary version of PSO for solving De Jong's suite of test functions [5]. In this binary version, the velocity equation remains the same as equation (1) except for p_{id} and x_{id} are integers in {0, 1}. v_{id} is constrained to the interval [0.0, 1.0] using the sigmoid function and it then represents the probability of bit x_{id} taking the value of 1. Thus, the position update for each particle does not make use of equation (2). Instead, a random number is generated within the interval of [0.0, 1.0] and if its value is less than the value of the transformed v_{id} (using the sigmoid function), x_{id} is updated to bit 1. Otherwise, it is updated to bit 0.

For the second category of discrete PSO approaches, Hu, Eberhart and Shi in 2003 proposed a PSO for permutation optimization in solving n-queens problems [6]. Each particle represents the permutation of numbers of D dimensions, i.e. it is made up of positive integers. The velocity equation also remains the same as equation (1), like in the binary version, but it now represents the possibility that the particle changes, i.e. a larger velocity will cause the particle to more likely change to a new permutation sequence. The velocity is normalized to the range of 0 to 1 by dividing it by the maximum range of the particle. According to Wang, Lin and Yang [7], a random number is set between 0 and 1. If the normalized velocity is less than or equal to the random number, a position will be swapped, i.e. the position will be set to the value of the same position in the global best position by swapping the operations. Otherwise, the position is maintained. Thus, the position update for each particle also does not make use of equation (2), similar to the binary version of PSO discussed previously.

Zhang, Li, Li and Huang proposed in 2005 a method to solve the resource-constrained project scheduling problem, where they considered two forms of solution representations, i.e. priority-based representation and permutation-based representation [8]. The first form operates on the continuous domain, where each particle represent the priority values of activities and the values can be ranged from between 0 and 1. Each particle is also randomly initialized with the velocity of continuous values. The update of velocity and position are similar to equation (1) and (2) as well. On the other hand, the second form of solution representations operates on the discrete domain, where each particle represents the permutation or sequence of activities and its value must be in the form of positive integers. Each particle is also randomly initialized with the velocity of continuous values, but later rounded up to integers and limited to the number of dimensions or activities in a project. The velocity equation also remains the same as equation (1), but it now corresponds to the distance or gap between the current permutation or sequence and its own local or global best permutation found so far. It represents the probability that the permutation will be changed, i.e. a larger gap will cause the particle to more likely change to a new permutation sequence.

Lian, Jiao and Gu in 2006 proposed a similar particle swarm optimization algorithm (SPSOA) to solve JSP [9]. In their research, each particle represents a sequence of working procedures of jobs. Then, a coding scheme is used to make every particle to have a valid scheduling. Instead of using equation (1) and (2), various new mutation and crossover operators are introduced in order to perform the velocity update and the position update afterwards.

Zhang, Wang and Ji provided a comprehensive survey on PSO and its applications which also includes the field of binary optimization and discrete optimization in [10]. In this research, the second category of discrete PSO which consists of positive integers will be used, instead of binary numbers, since it is more suitable for the permutation optimization problems like JSP.

3. Continuous and Discrete PSO

For the two versions of PSO, the main differences lie in the initialization and the update of both position and velocity of the particles. The equation for velocity is the same for both versions of PSO. However, in discrete PSO, the velocity does not represent the change in position; it represents the probability that the positions are swapped to produce a new permutation sequence.

For the continuous PSO, the position and velocity are initialized in a continuous search space. Then, the velocity and position are updated according to equation (1) and (2), respectively. For the discrete PSO, the position is initialized as a sequence of operations of jobs, while the velocity is initialized in a continuous search space. Then, the velocity is updated according to equation (1), while the position is updated by swapping the operations. The details of the position update are as follows:

First, the velocity is normalized to the range of 0 to 1. In this paper, the sigmoid function is used to compress the velocity value to be between 0.0 and 1.0. Second, a random number between 0 and 1 is generated such that, if the normalized velocity is greater than or equal to the random number, the position will be set to the value of the same position in the global best position by swapping the operations.

For instance, given a JSP of 3 jobs being processed on 2 machines, the total operations will amount to 6, which will be the number of dimensions of the particle. For the continuous PSO, the position and velocity are initialized randomly in continuous values from x_{min} to x_{max} and $-v_{min}$ to v_{max}, respectively as shown in table 1:

Table 1. Example of position and velocity initialization for continuous PSO.

Dimension	Initial Position	Initial Velocity
1	3.4299	−0.2464
2	0.3782	−2.0930
3	5.8050	3.7116
4	4.6567	1.7110
5	2.4170	4.8089
6	1.6693	−5.1933

For the discrete PSO, the position is initialized randomly in discrete values as a sequence of operations of jobs from 1 to the number of operations. The velocity is initialized randomly in continuous values from $-v_{min}$ to v_{max} as shown in table 2:

Table 2. Example of position and velocity initialization for discrete PSO.

Dimension	Initial Position	Initial Velocity
1	4	0.9142
2	3	−4.9549
3	1	−3.2301
4	5	5.9768
5	6	−1.4690
6	2	2.6561

For the continuous PSO, the velocity and position are updated according to equation (1) and (2), respectively. For the discrete PSO, after the velocity has been updated according to equation (1), it will be normalized by using the sigmoid function to compress the velocity value to be between 0.0 and 1.0, as shown in figure 1. A sigmoid function is given by the following equation:

$$s(v_{id}) = \frac{1}{1 + \exp(-v_{id})} \tag{3}$$

Dimension	1	2	3	4	5	6
Current Velocity	−4.4162	−1.6928	5.7734	−0.3526	3.1054	−2.6846
Normalized Velocity	0.0119	0.1554	0.9969	0.4128	0.9571	0.0639

Figure 1. Example of velocity normalization for discrete PSO.

For the position, in this paper, it will not be updated according to equation (2) since the meaning behind the update in position now refers to the swapping of the operations in each particle. This is done by comparing the position with the global best position and swapping the operation to match the operation in the global best position.

A random number is generated between 0 and 1 and each element in the normalized velocity is compared against the random number. If the normalized velocity is greater than the random number, that element in the current position will be swapped with another element so that it matches the element in the global best position. For example, in figure 2, assume that the normalized velocity is greater than the random number in Dimension 4, thus the value 5 in the current position will be swapped. Since the global best position in Dimension 4 contains the value 6, thus the value 5 will be swapped with the value 6 in Dimension 5.

Dimension	1	2	3	4	5	6
Global best position	5	1	2	6	3	4
Current Position	4	3	1	5	6	2
Global best position	5	1	2	6	3	4
Next position	4	3	1	6	5	2

Figure 2. Example of position update for discrete PSO.

4. Experimental Results

In the experiments, the continuous and discrete PSO for JSP is programmed using MATLAB on an Intel Core notebook with 4GB of RAM running Windows 10. The performance of both versions of PSO are evaluated using two benchmark problems which are accessible from the OR-Library website [11]. The first problem, called FT06, consists of 6 machines and 6 jobs with a total of 36 operations. While the second problem, called FT10, consists of 10 machines and 10 jobs with a total of 100 operations. During the experimental process, the parameters used are given as follows: The size of population is set to 45 and 50 for FT06 and FT10 problems, respectively. The inertia weight is linearly decreased from 0.9 at the start of the run and 0.4 at the end of the run. Both acceleration constants are set to 2.0. The maximum of iterative generations is set to 3000 for FT06 and 6000 for FT10, where each instance is performed randomly 20 times.

The computational results for continuous and discrete PSO are presented in table 3 to be compared. The quality of solution is evaluated in terms of the minimum makespan, the percentage of deviation from the optimal solution and the computational time in seconds. In table 3, best solution indicates the best minimum makespan obtained and mean deviation indicates the average percent relative increase in minimum makespan with respect to the optimal solution after 20 runs. The mean computational time denotes average computational time in seconds which is obtained after 20 runs.

The optimal solution to be compared for FT06 problem is 55, which is obtained from the paper by Klemmt, Horn, Weigert, and Wolter [12] who used exact methods to solve the problem. The best solution attained by both versions is 55 as well; therefore, both versions are able to achieve the optimal solution. However, in terms of mean deviation, the discrete PSO performs better with a mean deviation of 0%; in other words, the best solution is equal to the optimal solution in all 20 runs. For the mean computational time, the discrete PSO performs faster with 20 seconds.

The optimal solution to be compared for FT10 problem is 930, which is obtained from the paper by Carlier and Pinson [13] who used exact methods to solve the problem. The best solution attained by the discrete PSO is 965 which outperforms the continuous PSO; hence, the discrete PSO manages to attain a near-optimal solution with 3.76% deviation from the optimal solution. In terms of mean deviation, the discrete PSO performs better with a mean deviation of 9.62%. For the mean computational time, the discrete PSO performs faster with 536 seconds.

Table 3. Computational results for continuous and discrete PSO in solving FT06 and FT10 problems.

Problem instance	Optimal solution	Continuous PSO			Discrete PSO		
		Best solution	Mean deviation (%)	Mean computational time (sec)	Best solution	Mean deviation (%)	Mean computational time (sec)
FT06	55	55	1.36	57	55	0	20
FT10	930	982	12.47	724	965	9.62	536

The significant differences of these experimental results could be due to the fact that the continuous PSO needs to implement the particle representation procedure to establish the mapping between the position of the particle in the continuous search space of PSO and the scheduling solution in the discrete space of JSP. Thus, this extra step leads to additional processing effort and computational time. On another hand, for the discrete PSO, the particle's position directly represents its scheduling solution in JSP. This approach is more straightforward since it eliminates the need to encode each particle to a schedule. Therefore, in contrast to the continuous PSO, it spends less effort and time to find the best solution.

5. Conclusion

In this paper, two versions of PSO in the case of solving JSP are studied. In the continuous PSO, the initialization and the update of both position and velocity of the particles are carried out using the

standard PSO algorithm. For the discrete PSO proposed in this paper, the position is initialized as a sequence of operations of jobs, while the velocity is initialized in a continuous search space. Then, the velocity is updated according to the velocity equation of the standard PSO algorithm, while the position is updated by swapping the operations.

The performance of continuous and discrete PSO are tested on FT06 and FT10 instances, which is taken from the OR-library. The performance measure considered in this study is the makespan minimization with the use of MATLAB software. Based on the experimental results, it is found that the discrete PSO produces better performance in solving both problem instances, as compared to the continuous PSO.

During the course of evaluation and comparison of these two versions of PSO, the authors are able to obtain some insight into different PSO versions in terms of how it can considerably have some bearing on the performance of PSO in solving JSP.

The versions are shown to yield substantially contrasting degree of performances for similar benchmark problems, where the discrete PSO is able to outperform the continuous PSO with a significant improvement in attaining the best solutions of the two problem instances as well as the mean computational times. Thus, the types of problem space should be contemplated as one of the main factors when working with PSO in solving JSP.

Future efforts will include further investigation of the mechanism of each PSO version in order to thoroughly examine what makes one version better than the other. Since only two problem instances are experimented on, more problem instances in the OR library will be applied to assess the continuous and discrete PSO in order to verify whether the performance is subject to the problems being solved.

References

[1] Błażewicz J, Domschke W and Pesch E 1996 The job shop scheduling problem: Conventional and new solution techniques *Eur. J. Oper. Res.* 93 pp 1–33

[2] Kennedy J and Eberhart R 1995 Particle swarm optimization *Neural Networks, 1995. Proceedings., IEEE International Conference on* vol 4 pp 1942–8

[3] Liao C J, Chao-Tang Tseng and Luarn P 2007 A discrete version of particle swarm optimization for flowshop scheduling problems *Comput. Oper. Res.* 34 pp 3099–111

[4] Anghinolfi D and Paolucci M 2009 A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times *Eur. J. Oper. Res.* 193 pp 73–85

[5] Kennedy J and Eberhart R C 1997 A discrete binary version of the particle swarm algorithm *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* vol 5 pp 4104–8

[6] Hu X, Eberhart R C and Shi Y 2003 Swarm intelligence for permutation optimization: A case study of n-queens problem *2003 IEEE Swarm Intelligence Symposium, SIS 2003 - Proceedings* pp 243–6

[7] Wang Y R, Lin H L and Yang L 2012 Swarm refinement PSO for solving n-queens problem *Proceedings - 3rd International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2012* pp 29–33

[8] Zhang H, Li X, Li H and Huang F 2005 Particle swarm optimization-based schemes for resource-constrained project scheduling *Autom. Constr.* 14 pp 393–404

[9] Lian Z, Jiao B and Gu X 2006 A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan *Appl. Math. Comput.* 183 pp 1008–17

[10] Zhang Y, Wang S and Ji G 2015 A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications *Math. Probl. Eng.* 2015 pp 1–38

[11] Beasley J E 1990 OR-Library

[12] Klemmt A, Horn S, Weigert G and Wolter K-J 2009 Simulation-based optimization vs. mathematical programming: A hybrid approach for optimizing scheduling problems *Robot.*

Comput. Integr. Manuf. 25 pp 917–25
[13] Carlier J and Pinson E 1989 An algorithm for solving the job-shop problem *Manage. Sci.* 35 pp 164–76