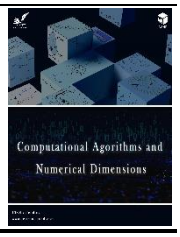
Computational Algorithms and Numerical Dimensions

www.journal-cand.com

Com. Alg. Num. Dim Vol. 2, No. 2 (2023) 74–86.



Paper Type: Original Article

A Hybrid Particle Swarm Optimization Algorithm for Single Machine Scheduling with Sequence-Dependent Setup Times and Learning Effects

Payam Chiniforooshan[1,*], Dragan Marinkovic[1]

[1] Department of Industrial Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran; p.chiniforooshan@gmail.com.

[2] Department of Structural Mechanics and Analysis, Technische University Berlin, Strasse des 17. Juni 135, 10623 Berlin, Germany; dragan.marinkovic@tu-berlin.de.

Citation:

Chiniforooshan, P., & Marinkovic, D. (2023). A hybrid particle swarm optimization algorithm for single machine scheduling with sequence-dependent setup times and learning effects. *Computational algorithms and numerical dimensions, 2*(2), 74-86.

Received: 12/02/2023 Reviewed: 12/03/2023 Revised: 09/04/2023 Accepted: 01/05/2023

Abstract

This paper deals with the single machine scheduling problem with sequence-dependent setup time and learning effect on processing time, where the objective is to minimize total earliness and tardiness of the jobs. A Mixed Integer Linear Programming (MILP) model capable of solving small-sized problems is proposed to formulate this problem. In view of the NP-hard nature of the problem, the Hybrid Particle Swarm Optimization (HPSO) algorithm is proposed to solve the large-sized problems. In order to utilize Particle Swarm Optimization (PSO) to solve the scheduling problems, the proposed HPSO approach uses a random key representation to encode solutions, which can convert the job sequences to continuous position values. Also, the local search procedure is included within the HPSO to enhance the exploitation of the algorithm. The performance of the proposed HPSO is verified for small and medium-sized problems by comparing its results with the best solution obtained by the LINGO. In order to test the applicability of the proposed algorithm to solve large-sized problems, 120 instances are generated, and the results are compared with a Random Key Genetic Algorithm (RKGA). The results show the effectiveness of the proposed model and algorithm.

Keywords: Single machine scheduling, Sequence-dependent setup time, Learning effect, Particle swarm optimization, Genetic algorithm.

1 | Introduction

This paper considers the single machine earliness-tardiness scheduling problem with sequence-dependent setup time and distinctive due date (SETSDSND) for jobs with varying processing times. This type of problem often arises in manufacturing systems such as plastic and glass manufacturing industries, where the setup times are significant [1]. Setups usually correspond to preparing the resources for the execution of the next job. When the duration of such operations depends on the type of last completed job, the setups are called sequence-dependent. A comprehensive review of sequence-dependent scheduling problems is provided in [2]. Literature reveals that the majority of scheduling research assumes that setup time is negligible or part of the job processing time. This assumption makes the model inapplicable in the real environment.

Corresponding Author: p.chiniforooshan@gmail.com

https://doi.org/10.22105/cand.2023.190873

The study of earliness and tardiness penalties in scheduling problems is a relatively new area of research [3] that received considerable attention in recent years due to its compliance with the new concepts of Just-in-Time (JIT). In the JIT manufacturing system, jobs are needed to arrive and processed just as they are needed, not earlier nor later. Earliness causes a high cost of inventory holding, and lateness delays other jobs. Thus, any schedule of a set of jobs should strive to minimize total earliness and tardiness, where tardiness reflects customer satisfaction and earliness measures inventory performance [4].

Traditionally, the processing time of jobs is assumed to be independent of their position or starting time. However, there are many practical situations in which the actual processing time of jobs may be subject to change due to the effect of learning or deterioration. In scheduling with the learning effect, the actual processing time of a job is modeled as a decreasing function if it is scheduled later in the sequence. This is due to the fact that workers can learn when they are processing similar jobs. So, the effect of worker learning on processing times is also considered in this research.

Single Machine Earliness-Tardiness Scheduling Problem (SMETP), even in the simplest formulation, is an NP-hard optimization problem [3]. Due to the complexity of this problem, recently, some researchers have addressed the problem using meta-heuristics such as Genetic Algorithm (GA) [5], Tabu Search (TS) [6], [7], or hybrid algorithms [8]. Lee and Kim [5] developed parallel GAs for SMETP with a common due date. Hao et al. [6] solved this problem by minimizing the sum of weighted earliness and tardiness values using the TS method. A hybrid algorithm that combines local search heuristics and GA is proposed by M'Hallah [4] to solve SMETP with distinct due dates in the JIT production environment.

Nearchou [9] considered a single-machine problem with a common due date that minimizes a summation of earliness and tardiness cost. A differential evolution approach was proposed to solve the given problem. Chang et al. [10] studied the problem of scheduling a single machine with a distinctive due date and learning effect with the objective of minimizing the weighted sum of earliness and tardiness. They presented a GA to solve their problem. Valente and Goncalves [11] studied the problem of scheduling jobs on a single machine where the objective was to minimize linear earliness and quadratic tardiness costs. They proposed a genetic approach based on a random key alphabet.

However, none of these research studies take into account the sequence-dependent setup time and assume that setup time is part of the job processing time. Moreover, most of these works consider the processing time of jobs to be independent of their position or starting time. The contribution of this paper is the development of a new Mixed Integer Linear Programming (MILP) model to formulate the SETSDSND problem with the effect of learning in processing time. Because of the complexity of this problem, a Hybrid Particle Swarm Optimization (HPSO) with new particle representation is proposed to solve the model. Standard particle optimization is generally used to solve continuous optimization problems and is rarely used to solve discrete problems such as scheduling problems. Therefore, a new particle representation based on a random key alphabet, which converts the sequence of jobs to continuous position values, is designed in this paper. This representation was first introduced by Bean [12] for sequencing and optimization problems.

The remaining sections of this paper are organized as follows. Section 2 presents an integer linear programming formulation for the SETSDSND problem. A brief overview of Particle Swarm Optimization (PSO) is explained in Section 3. Section 4 describes the proposed HPSO approach and highlights the new features introduced. Computational results are reported in Section 5. Finally, Section 6 summarizes the contribution of this paper and gives the conclusion.

2 | Problem Formulation

The problem under study is to schedule N jobs with distinct due dates and varying processing times on a single machine. All jobs are ready at time zero, the machine is continuously available, and it can handle only one job at a time. Preemption is not allowed, and there is no precedence relationship between jobs.

For each job j= 1, 2, …, N, the following quantities are given: a normal processing time pj and a due date dj. A sequence-dependent setup time Sij must be waited before starting the processing of job j if it is immediately following job i in sequence. Sii denotes the setup time for job i when it is the first in the sequence. Because of the learning effect, the processing time of a job depends on its position in the sequence. Hence, the actual processing time of the jobs when scheduled in position k is given by

$$P_{jk} = P_j k^a. \tag{1}$$

Where Pjr is the processing time of job j in position k, and a is the learning index (a < 0). The objective of the problem is to find a sequence of jobs that minimizes the total sum of earliness and tardiness. Let M be a large positive number. Moreover, let Cj, Ej, and Tj represent the completion time, earliness, and tardiness of job j, respectively. To formulate the above problem as MILP, define a binary variable Xjk that takes on value 1 if job j is assigned on position k and zero otherwise.

Based on the definition and notation described above, the SETSDSND problem with learning effect can be formulated as a MILP model, as shown below:

$$\text{Min } Z = \sum_{i=1}^{N} T_i + E_i), \quad i = 1,2,\dots,N, k = 1,2,\dots,N. \tag{2}$$

s. t.

$$\sum_{k=1}^{N} X_{jk} = 1, \quad i = 1,2,\dots,N. \tag{3}$$

$$\sum_{i=1}^{N} X_{ik} = 1, \quad k = 1,2,\dots,N. \tag{4}$$

$$C_i - C_j + M\left(2 - X_{ik} - X_{j,k-1}\right) \geq P_{ik} + S_{ij}, \quad k = 2,\dots,N, \quad i = 1,2,\dots,N, j \neq i = 1,2,\dots,N. \tag{5}$$

$$C_i + M\,1 - X_{i1}) \geq S_{ii} + P_{i1}, \quad i = 1,2,\dots,N. \tag{6}$$

$$C_i - d_i = T_j - E_j, \quad i = 1,2,\dots,N, \tag{7}$$

and boundary conditions are

$$X_{ik}, \in \{0,1\}, \quad i = 1,2,\dots,N, k = 1,2,\dots,N. \tag{8}$$

$$C_k, T_k, E_k \geq 0, \quad k = 1,2,\dots,N. \tag{9}$$

The objective of the model in *Eq. (2)* is to minimize the total tardiness and earliness of the problem. *Constraints (3)* and *(4)* ensure that each job can be assigned to one of the existing positions and each position can be occupied by only one job. *Constraint (5)* ensures that the completion time of a job in sequence will be at least equal to the sum of the completion time of the preceding job, the sequence-dependent setup time, and the processing time of the present job. *Constraint (6)* calculates the completion time of the job in the first position. *Constraint (7)* specifies the earliness and tardiness of jobs. *Constraints (8)* and *(9)* define the type of decision variables.

3 | Particle Swarm Optimization

PSO is a population-based stochastic optimization algorithm inspired by the behavior of a bird flock. The individuals in a PSO are denoted particles. The PSO algorithm represents each potential solution by the position of a particle in multi-dimensional hyperspace. Throughout the optimization process, velocity and displacement updates are applied to each particle to move it to a different position and, thus, a different solution in the search space. PSO refines its search by attracting the particles to positions with good solutions. PSO remembers the best position found by any particle (gbest). Also, each particle remembers its own previously best-found position (pbest). Suppose that the position of the particle i at the tth iteration is represented by $\vec{x}_i^t$. Then, the velocity vector of particle i at iteration t+1($\vec{v}_i^{t+1}$) is updated by the *Eq. (10)*.

$$\vec{v}_i^{t+1} = \omega \vec{v}_i^{t} + c_1 \times \text{rand}() \times \left(\vec{p}_i^{t} - \vec{x}_i^{t}\right) + c_2 \times \text{rand}() \times \left(\vec{p}_g^{t} - \vec{x}_i^{t}\right). \quad (10)$$

Where ω is the inertial weight, which is introduced to balance between the global and local search abilities, $\vec{p}_i^t$ is the best-found position of the ith particle at the tth iteration, and $\vec{p}_g^t$ is the best position known for all particles. c_1 and c_2 are the cognitive and social acceleration constants, and *rand*() is a random number generator with a uniform distribution over [0,1]. The position of each particle is updated in each iteration by adding the velocity vector to the position vector according to *Eq. (11)*.

$$\vec{x}_i^{t+1} = \vec{x}_i^{t} + \vec{v}_i^{t+1}. \quad (11)$$

This simultaneous movement of particles towards their own previous best solutions and the best solution found by the entire swarm results in the particles converging to one or more good solutions in the solution space.

4 | Proposed Hybrid PSO Approach

Even the simplest formulation of SMETP is an NP-hard optimization problem [3]. Therefore, the SETSDSND problem is also NP-hard because the incorporation of setup time will complicate the problem. Optimal solutions cannot be obtained for problems of reasonable size, and heuristics have to be utilized to obtain good near-optimal solutions. PSO, first introduced by Eberhart and Kennedy [13], is one of the most recent and hopeful heuristic techniques that has been applied to a wide range of applications such as power and voltage control [14], task assignment [15], project scheduling [16], cell formation problem [17], flow shop sequencing problem [18].

Because of the continuous nature of the position of particles in PSO, the standard PSO is rarely used to solve discrete problems such as scheduling problems. Thus, the most important issue in applying PSO to solve the SMS problem is to find a suitable method to represent the job sequence and the position of particles in PSO. Therefore, a new particle representation based on a random key alphabet, which converts the sequence of jobs to continuous position values, is designed in this paper. By this representation, the continuous version of PSO can be used.

In this research, HPSO algorithm is presented to solve the SETSDSND problem with a learning effect. The proposed HPSO contains the following parts: initialization, random key encoding scheme, local search procedure, and the PSO component. The following subsections describe how the implementation of the HPSO to solve the proposed model works out.

4.1 | Initialization

As mentioned before, each particle has its own position and velocity vector. Initial positions for particles that are the initial sequences are generated under some controls. In particular, two constructive heuristics, the Earliest Due Date (EDD) and the Shortest Processing Time (SPT), are used to generate two initial

positions. Then, the algorithm generates other initial positions randomly. The velocity vector of each particle is initialized randomly from a uniform distribution between 0 and 0.5.

4.2 | Solution Representation and Decoding

The standard PSO cannot be used directly to solve discrete problems since it was originally designed to solve continuous problems. Therefore, the proposed HPSO approach uses a random key scheme [12] to encode solutions. Based on this representation scheme, the sequence of jobs on the machine can be converted to continuous position values. Each position is a vector of uniform random numbers between 0 and 1. Thus, each particle is encoded as a vector of random keys.

In order to evaluate the fitness function of each particle, it is necessary to decode it into a sequence of jobs. The decoding of a chromosome into a sequence is accomplished by sorting the jobs. Then, the fitness of each particle is evaluated by *Eq. (2)*. *Fig. 1* illustrates the solution representation and decoding procedure.

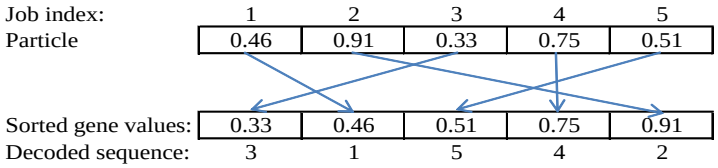


Fig. 1. Solution representation and decoding.

4.3 | Main PSO Procedure

The PSO procedure is the main component of the proposed HPSO algorithm. As mentioned before, a random key representation is used to represent each particle, which can convert the job sequence to continuous values. So, the standard PSO can be used to solve the presented problem. The standard PSO, described in Section 3, consists of a number of particles moving around in continuous search space. Each particle has its own position and velocity. In the proposed algorithm, the initial population is composed of Np particles. The size of the population is kept constant throughout the procedure. In each iteration, the velocity and position of each particle are updated according to *Eq. (10)* and *Eq. (11)*.

4.4 | Local Search Procedure

After HPSO generates new particles, the local search procedure is included within the algorithm to enhance the algorithm's exploitation ability. The Adjacent Pairwise Interchange (API) local search is used in this paper. In each iteration of the API procedure, all adjacent job positions are considered. A pair of adjacent jobs are then swapped if such an interchange improves the objective function value. This process is repeated until no improvement is found in a complete iteration. *Fig. 2* shows the framework of the HPSO for solving the SETSDSND problem.

5 | Computational Results

In this section, the effectiveness of the proposed HPSO algorithm in solving the SMETP with sequence-dependent setup time and learning effect is evaluated. Because no sample problems were found in the literature that could be used as a benchmark for testing the algorithm, a number of test problems are randomly generated in small and large sizes. To generate data, such as processing times, setup times, and due dates, the methods presented in [19] are used. The value of learning index (a) is considered -0.322, which corresponds to 80% of learning curves.

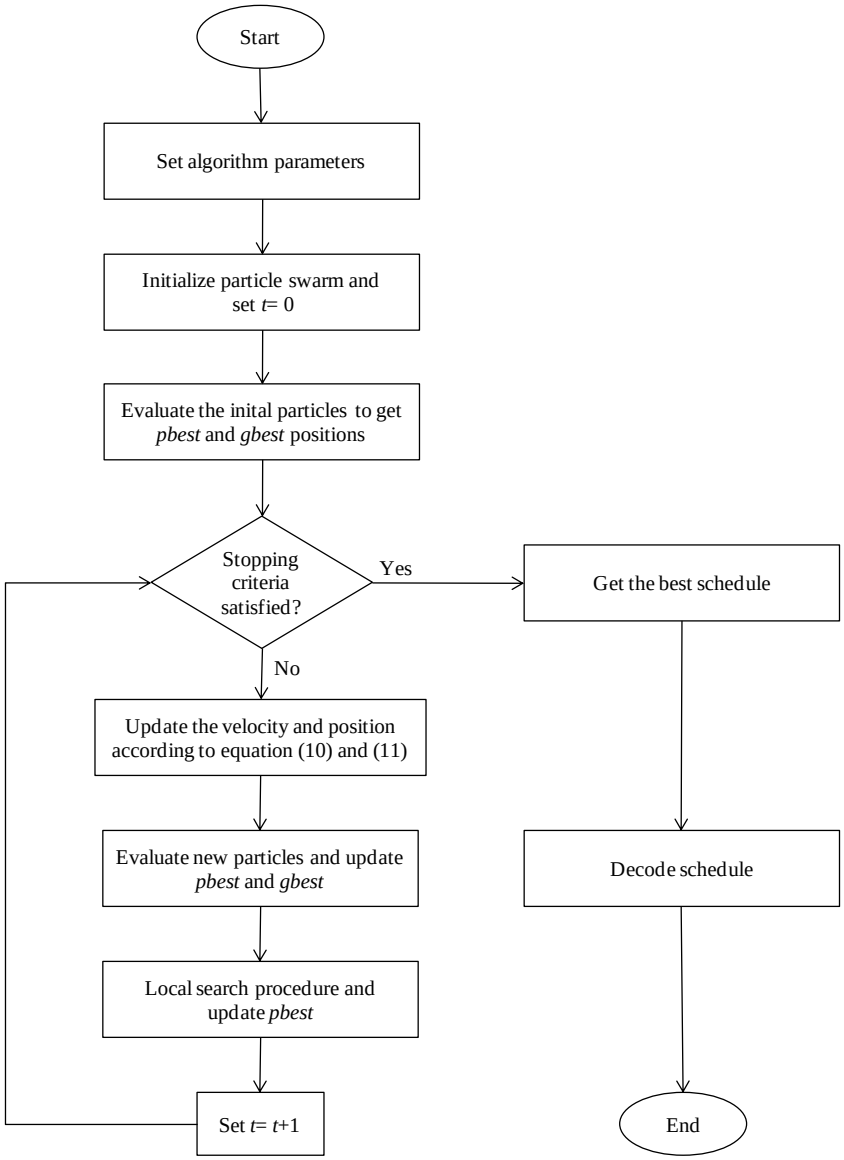


Fig. 2. The general framework of the proposed HPSO.

5.1 | Parameter Setting

The selection of suitable parameters for the algorithm's performance evaluation is one of the most important issues in PSO. Therefore, preliminary experiments are performed to determine the algorithm parameters. In this study, three parameters are examined. These parameters are inertia weight w and the acceleration constants c1 and c2. The inertia weight, w, is important in ensuring that a suitable trade-off is obtained between exploration of different areas of the search space and further exploitation of good areas. Large values of w thus encourage exploration, while smaller values encourage exploitation. The two acceleration coefficients (c1 and c2) control the influence of the cognitive and social components on particle velocity. In order to determine these parameters, four test problems with different sizes are tested with the following values:

$$w = \{0.6, 0.8, 1\},$$
$$c_1 = c_2 = \{0.5, 0.7, 1, 1.2, 1.5, 1.7, 2\}.$$

In each experiment, the population size (Np) is set to 100, and the number of generations (Ng) is set to 200. Due to the probabilistic nature of the proposed method, each problem is solved 10 times, and the best solution (Min), the average of the 10 runs (Avg), and the worst value (Max) among the ten runs are obtained. These results are summarized in *Tables 1-3*. According to the obtained results, with w= 0.8 and C1=C2= 1.2 or w= 1 and C1=C2= 1.5, HPSO can yield better solutions.

Table 1. Result analysis of HPSO with w= 0.6 and different values of C1 and C2.

C1= C2	Example 1 JobNum= 40			Example 2 JobNum= 60			Example 3 JobNum= 80			Example 4 JobNum= 100		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.5	1137	1279	1451	2580	2877	3162	5213	5715	6438	8929	9838	10641
0.7	1147	1222	1355	2378	2577	2888	4668	4956	5298	7730	9205	10290
0.9	1074	1155	1346	2264	2501	2768	4199	4605	5116	6966	7670	9208
1	1033	1110	1180	2232	2333	2469	4010	4376	5322	6184	6870	8478
1.2	1012	1084	1230	1977	2141	2619	3583	3882	4559	5625	6132	7792
1.5	1027	1100	1176	1938	2106	2410	3554	4120	5839	5844	6520	9784
1.7	1040	1094	1203	2010	2219	3186	3826	4151	5166	5962	6976	10700
2	1042	1209	1641	2131	2332	3036	3984	4499	7074	6292	7522	11295

Table 2. Result analysis of HPSO with w= 0.8 and different values of C1 and C2.

C1= C2	Example 1 JobNum= 40			Example 2 JobNum= 60			Example 3 JobNum= 80			Example 4 JobNum= 100		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.5	1154	1286	1624	2728	2974	3321	4900	5557	5931	8524	9873	11153
0.7	1049	1190	1305	2441	2632	2771	4583	5104	5694	7246	8449	9444
0.9	1041	1164	1330	2078	2370	2503	4118	4624	4989	6636	7609	8909
1	1035	1119	1280	2183	2291	2617	3872	4198	5438	6446	7032	9415
1.2	964.1	1097	1258	1997	2209	2799	3501	3996	5154	5709	6451	9062
1.5	1007	1094	1506	1962	2253	3074	3581	4148	5776	5724	6891	13063
1.7	1029	1123	1501	1990	2311	3493	3703	4281	6912	6029	6877	10561
2	1052	1123	1345	2092	2347	3028	3989	4476	6424	6371	7486	11487

Table 3. Result analysis of HPSO with w= 1 and different values of C1 and C2.

C1= C2	Example 1 JobNum= 40			Example 2 JobNum= 60			Example 3 JobNum= 80			Example 4 JobNum= 100		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.5	1092	1207	1315	2553	2971	3714	5108	5570	5972	8054	9702	11049
0.7	1065	1151	1262	2330	2610	2988	4601	5281	6308	7856	8824	9643
0.9	1047	1127	1274	2222	2716	4956	4093	4502	6119	7070	7980	10426
1	1039	1085	1153	2103	2322	3070	3881	4323	5719	6223	7267	11295
1.2	1056	1121	1338	2066	2245	3234	3541	4235	7130	5741	6545	9182
1.5	969.2	1115	1510	1969	2284	3434	3684	4246	6647	5592	6568	10130
1.7	969.4	1153	1510	1997	2290	3704	3665	4340	6840	5908	6980	11734
2	1040	1257	2050	2140	2510	3638	3819	4666	6849	6073	7657	11873

5.2 | Small and Medium-Sized Problems

In order to validate the performance of the HPSO algorithm, test problems are solved by using two approaches: the optimal solution approach (i.e., Branch and Bound) under the LINGO 8.0 software and the proposed HPSO. These problems are solved on a personal computer with an Intel Core2, 2GHz CPU, and 2 GB memory. The proposed algorithm is written in MATLAB 7.6 on the above-mentioned system. For small and medium-sized problems, the results of the HPSO algorithm are compared with the best solution obtained by the LINGO. Small-sized problems are optimally solved by a Branch and

Bound method. However, it is impossible to obtain an optimal solution for medium-sized problems. For the medium-sized examples, the run time of LINGO is limited to 3 hours. Thus, the best solution obtained after 3 hours is reported for medium-sized problems.

Table 4. Comparison of LINGO results with HPSO results for small and medium-sized problems.

		LINGO		HPSO	
No. Example	No. Jobs	OPT. Value	CPU Time (Sec)	Fitness Value	Mean CPU Time (Sec)
1	6	46.96	3	46.96	0.53
2	6	33.60	2	33.60	0.62
3	7	80.08	19	80.08	0.75
4	7	57.89	20	57.89	0.78
5	8	77.95	254	77.95	0.96
6	8	88.59	201	88.59	0.87
7	9	144.98	3193	144.98	1.52
8	9	84.73	3286	84.73	1.31
9	10	190.76	10800	187.43	2.22
10	10	141.71	10800	135.81	1.88
11	12	236.89	10800	226.29	2.52
12	14	247.48	10800	201.57	2.94
13	16	389.43	10800	314.14	2.64
14	18	378.27	10800	271.98	3.45

Table 4 provides a comparison of the solutions to the MILP model generated by the branch and bound method and the solution obtained by HPSO for small and medium-sized problems. From this table, it can be noticed that the solutions from the HPSO algorithm are also optimal for these problems. However, the CPU times of these two methods are not obviously comparable. The CPU time of the Branch and Bound method increases the size of test problems and has an exponential trend.

Table 5. Parameters used in HPSO.

Parameters	Description	Value
w	Inertia weight	0.8
c_1 and c_2	Acceleration constants	1.2
N_p	Population size	120
Ω	Constant coefficient	0.4

Table 6. Parameters used in RKGA.

Parameters	Description	Value
C_p	Crossover probability	0.7
M_p	Migration probability	0.2
E_p	Elite percent	0.1
N_p	Population size	120
Ω	Constant coefficient	0.4

5.3 | Large-Sized Problems

In order to test the applicability of the proposed HPSO to solve large-sized problems, 120 instances are generated with varied sizes from 60 jobs to 300 jobs. Because the problem under study is NP-hard, only small-sized problems can be solved optimally using LINGO. So, In order to evaluate the efficiency of the HPSO in large problem sizes, the results of the proposed HPSO are compared with the Random Key Genetic Algorithm (RKGA). The RKGA was introduced by Bean [12] and has been used in numerous

applications of combinatorial optimization [11], [20], [21]. Bean [12] shows the robustness of RKGA by using it to evaluate machine scheduling, vehicle routing, and quadratic assignment problems.

The RKGA begins by assigning a random number to each job. The jobs are then initially sorted according to their key value. Then, the crossover operator is applied to each job's value. In this algorithm, the migration operator replaces the traditional mutation operator. In the migration phase, new individuals are randomly generated and added to the new population. Migration is used to ensure diversity. The genetic operators used in this research are elitist reproduction, migration, and uniform crossover with tournament selection, similar to those in Bean's and Samanlioglu's research [22]. Also, Valente and Gonçalves [11] show that the RKGA with a local search procedure provides better results. So, the API local search procedure, as mentioned in section 4.2, is applied in the RKGA procedure.

Table 7. Comparison of the HPSO and RKGA for instances with 60 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	1696.45	1796.83	1906.07	1696.45	1748.78	1799.78
2	2086.90	2214.30	2438.68	2163.41	2249.48	2438.68
3	1938.80	2009.15	2144.98	1908.06	1961.83	2013.21
4	2266.45	2332.15	2432.15	2327.80	2371.07	2404.64
5	1889.41	1965.27	2037.46	1889.41	1942.50	2049.60
6	2499.57	2564.03	2723.57	2512.68	2629.68	2723.57
7	1928.95	1988.55	2119.93	1942.40	1988.46	2041.99
8	2216.81	2295.87	2379.87	2240.06	2277.52	2360.36
9	1504.97	1566.49	1713.50	1451.45	1548.32	1713.50
10	2295.71	2405.46	2510.45	2342.37	2388.82	2472.46
11	2367.73	2461.45	2569.44	2391.04	2451.97	2527.78
12	2017.78	2198.22	2287.30	2039.40	2114.98	2192.79
13	1739.38	1789.64	1861.78	1733.65	1767.13	1808.64
14	1477.96	1573.56	1694.00	1477.96	1560.84	1619.63
15	1797.02	1872.58	1926.74	1736.78	1834.37	1911.93
16	1809.76	1877.73	1964.49	1823.50	1895.71	1961.63
17	1798.80	1837.64	1891.84	1730.39	1838.34	1923.49
18	2105.85	2185.40	2278.14	2108.31	2220.34	2395.40
19	1958.97	2028.98	2111.84	1972.10	2054.51	2154.66
20	1870.99	1969.59	2100.31	1906.34	1973.61	2047.48

The comparison of HPSO and RKGA is made in terms of solution quality. For each instance, 10 independent runs are performed for two algorithms. Also, limited computational time is utilized as a stopping criterion in both algorithms to provide a fair opportunity for comparing algorithms. The computational time for each size of the problem is calculated according to $N \times \Omega$, where N is the number of jobs, and Ω is a constant coefficient. By different values of Ω, different computational times could be obtained. The algorithm parameters used in HPSO and RKGA in all experiments are given in *Table 5* and *Table 6*, respectively.

The comparison results obtained by HPSO and RKGA for test problems with different job numbers are summarized in *Tables 7-12*. Each table consists of 20 instances for each size. From the results presented, it can be observed that HPSO outperforms RKGA in solving all the instances.

Table 8. Comparison of the HPSO and RKGA for instances with 100 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	6040.35	6205.11	6299.94	6276.11	6455.95	6596.25
2	3553.99	3748.99	3890.76	3710.01	4022.21	4156.29
3	4540.22	4652.69	4763.22	4643.74	4863.00	5074.36
4	5372.94	5505.47	5795.70	5372.94	5821.67	6019.80
5	5752.33	5961.52	6241.24	5946.29	6129.01	6209.39
6	3943.60	4128.31	4230.38	3943.60	4201.77	4446.73
7	4067.33	4254.32	4385.10	4282.52	4450.93	4624.18
8	4672.27	4831.35	4973.94	4973.94	5252.04	5511.40
9	6926.95	7053.89	7236.57	6977.84	7175.06	7315.29
10	4397.10	4544.05	4741.06	4563.88	4780.42	4982.87
11	4453.74	4631.51	4919.81	4659.73	4790.45	4989.70
12	6047.29	6224.15	6411.15	6327.15	6490.57	6669.86
13	3971.60	4073.03	4186.21	3971.60	4221.13	4391.96
14	4553.92	4841.71	5112.16	4850.00	4963.72	5112.16
15	4712.05	4821.98	4883.02	4801.10	5230.76	5533.07
16	3935.68	4169.47	4411.23	4088.89	4388.73	4596.62
17	5763.28	5863.14	6067.70	5768.12	6141.77	6442.44
18	6033.60	6169.65	6275.35	6033.60	6337.28	6697.96
19	7040.12	7311.50	7481.35	7396.00	7523.18	7670.16
20	5720.35	5900.49	5996.65	5970.37	6199.58	6302.57

6 | Conclusions

This paper considered the single machine earliness-tardiness scheduling problem with sequence-dependent setup time and distinctive due date (SETSDSND) for jobs. Also, the processing time of jobs is dependent on their position due to the effect of worker learning. A MILP model was proposed and evaluated to solve this problem optimally using the LINGO solver. However, the CPU time required by this procedure increases exponentially as the problem size increases, and only small problems can be solved optimally using LINGO. Therefore, a HPSO method is proposed to find optimal or near-optimal solutions for large-sized problems.

The proposed HPSO approach uses a random key scheme to encode solutions, which can convert the job sequences to continuous position values. Also, in order to enhance the exploitation of the HPSO, the local search procedure was included within the algorithm. In addition, the parameter setting of three parameters of HPSO was investigated. The performance of the proposed HPSO was verified for small and medium-sized problems by comparing its results with the best solution obtained by the LINGO. The results showed that the solutions from the HPSO algorithm were also optimal for these problems. In order to test the applicability of the proposed algorithm to solve large-sized problems, 120 instances were generated, and the results of HPSO were compared with the RKGA. The results indicated that HPSO performs better than RKGA.

As an interesting future research, a further interesting issue is the consideration of realistic assumptions such as precedence constraints in the model. Another direction could be the extension of the proposed algorithm to more complex machine environments and other optimality criteria. Also, it is possible to develop additional meta-heuristic algorithms.

Table 9. Comparison of the HPSO and RKGA for instances with 120 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	7339.97	7496.91	7650.84	7370.69	7474.35	7598.71
2	7038.23	7177.74	7260.93	7178.72	7565.56	7875.07
3	6123.39	6383.86	6523.96	6523.96	6706.89	6956.93
4	6700.15	7062.86	7362.54	7178.39	7603.15	7969.23
5	5966.54	6178.06	6623.91	6044.98	6613.73	6942.27
6	5296.32	5512.49	5648.35	5563.87	5895.53	6126.99
7	6226.85	6348.91	6450.16	6449.50	6800.99	7197.93
8	7159.33	7225.06	7279.78	7221.57	7462.01	7599.62
9	6458.46	6692.57	7012.28	6521.85	7071.21	7264.73
10	7102.04	7269.11	7394.55	7102.04	7810.97	8166.19
11	7236.65	7425.66	7683.24	7591.10	7740.02	7931.65
12	9176.11	9463.88	9809.18	9454.03	9655.47	10076.39
13	5270.28	5584.15	5760.73	5308.60	5694.07	5892.89
14	8544.59	9025.37	9267.84	9033.71	9307.10	9546.05
15	7441.27	7573.07	7719.39	7470.12	7689.22	8194.90
16	6753.06	6991.43	7329.10	7020.78	7316.98	7583.99
17	8188.03	8437.08	8684.57	8374.03	8601.78	8845.32
18	6072.19	6202.64	6361.52	6361.52	6504.94	6732.88
19	6434.57	6801.77	7043.96	6985.53	7237.08	7439.80
20	7257.18	7425.29	7635.47	7257.18	7822.46	8187.19

Table 10. Comparison of the HPSO and RKGA for instances with 150 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	10935.18	11181.27	11459.47	11371.07	11926.64	12308.50
2	9509.90	9868.68	10477.48	10477.48	10722.21	10936.77
3	11708.69	11933.82	12284.87	11708.69	12535.29	13226.25
4	11184.28	11698.35	12132.78	11184.28	12244.84	12808.22
5	11712.38	11863.10	11961.66	11961.66	12829.64	13482.66
6	11495.30	11744.47	12059.78	12059.78	12654.30	13461.20
7	11855.33	12030.04	12231.21	12225.30	12849.70	13102.76
8	13081.39	13643.93	13982.09	13081.39	13988.45	14538.87
9	9854.45	10346.96	10786.37	9854.45	10913.36	11365.99
10	11057.99	11272.48	11515.50	11515.50	11934.15	12252.42
11	10494.81	11081.06	11602.27	11422.50	11817.02	12431.27
12	12828.78	13418.26	13693.45	13446.39	13675.86	13988.73
13	10707.86	10954.20	11248.91	11248.91	11698.40	12069.97
14	11434.81	11738.69	12096.13	11606.66	12330.25	12717.31
15	8822.69	9093.69	9510.09	8853.16	9783.35	10259.66
16	10169.14	10741.17	11137.27	10169.14	10922.02	11434.18
17	10201.22	10582.94	10845.80	10722.89	11183.42	11585.54
18	11671.88	11900.61	12256.48	11939.53	12397.75	12641.80
19	11300.43	11751.16	12276.04	12008.24	12324.43	12774.55
20	12693.45	13129.37	13753.32	13416.36	13641.44	13802.03

Table 11. Comparison of the HPSO and RKGA for instances with 200 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	18920.89	20090.86	21360.42	20703.62	21723.10	22339.36
2	19139.83	19767.12	20236.33	19949.53	21643.63	23000.44
3	19838.23	20407.85	21055.58	20547.19	22143.17	23202.73
4	16447.26	16964.18	17516.76	16679.12	19390.13	20349.63
5	16739.94	17712.80	19582.64	16739.94	19234.44	20453.77
6	19256.39	19541.13	19876.04	19781.07	21722.41	22778.22
7	20692.87	21138.40	21771.03	20885.63	22789.93	23879.88
8	19315.65	19718.71	20128.87	19924.69	21556.08	22716.95
9	20026.54	20759.85	21692.04	20734.69	22817.16	24382.69
10	22749.72	23227.89	23642.64	22878.15	24563.70	25480.14
11	23811.95	24565.32	25019.23	24809.12	25783.71	26688.98
12	18970.42	20261.12	21312.36	21312.36	22441.69	23366.37
13	20080.43	20638.66	21244.68	21244.68	22172.76	22766.17
14	22277.65	22686.96	23088.69	23088.69	23949.12	24568.77
15	23788.03	24882.74	25925.38	25500.51	26562.59	27349.82
16	23206.12	23540.56	24170.26	23284.12	24356.40	25129.52
17	19722.80	20061.42	20320.37	20320.37	22208.25	23294.25
18	20343.63	20913.75	21393.55	20343.63	23137.75	24854.58
19	26086.69	26493.25	26896.66	26896.66	27534.76	28035.63
20	20101.25	20386.03	20972.39	20972.39	22108.83	22777.78

Table 12. Comparison of the HPSO and RKGA for instances with 300 jobs.

No.	HPSO			RKGA		
	Min	Avg	Max	Min	Avg	Max
1	52166.03	55345.16	56781.85	56781.85	63975.88	66816.19
2	58178.52	59342.76	61483.40	58178.52	67904.17	71216.01
3	50222.12	52335.68	53878.71	53878.71	61245.71	63846.89
4	62111.60	63737.06	65611.04	65611.04	70100.40	72985.79
5	57679.87	58728.26	60290.04	57737.64	67292.35	70640.80
6	43089.56	45365.82	46749.71	46440.90	54167.64	57440.92
7	50085.65	51953.34	54472.25	54472.25	61140.46	65073.83
8	48908.39	50253.83	51180.12	51180.12	59369.62	62527.10
9	53926.41	56282.73	58788.09	58788.09	63970.56	66522.72
10	52592.79	54426.40	55434.99	52592.79	62289.00	66445.34
11	43392.42	44306.25	45731.31	43934.71	53085.40	56790.02
12	52812.47	54914.64	57420.94	57420.94	63952.04	66981.85
13	49773.51	51241.99	53876.50	50689.02	59589.32	63010.27
14	48315.93	49874.72	50905.12	50306.53	57951.63	61221.28
15	47996.14	48854.70	49743.01	48894.29	58142.07	61248.63
16	51834.83	53334.89	56465.71	53167.75	60235.36	63175.93
17	51470.87	53559.38	55622.32	54084.39	60991.78	64623.05
18	45576.03	46465.61	47246.28	46970.86	55055.70	58044.17
19	51218.54	52506.08	54097.55	52703.29	61795.82	64617.86
20	51946.97	53719.35	56053.00	51946.97	61827.26	65291.55

References

[1] Radhakrishnan, S., & Ventura, J. A. (2000). Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International journal of production research*, *38*(10), 2233–2252. DOI:10.1080/00207540050028070

[2] Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European journal of operational research*, *187*(3), 985–1032. DOI:10.1016/j.ejor.2006.06.060

[3] Baker, K. R., & Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties. A review. *Operations research*, *38*(1), 22–36. DOI:10.1287/opre.38.1.22

[4] M'Hallah, R. (2007). Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers and operations research*, *34*(10), 3126–3142. DOI:10.1016/j.cor.2005.11.021

[5] Lee, C. Y., & Kim, S. J. (1995). Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights. *Computers and industrial engineering*, *28*(2), 231–243. DOI:10.1016/0360-8352(94)00197-U

[6] Hao, Q., Yang, Z., Wang, D., & Li, Z. (1996). Common due-date determination and sequencing using Tabu Search. *Computers and operations research*, *23*(5), 409–417. DOI:10.1016/0305-0548(95)00051-8

[7] James, R. J. W. (1997). Using tabu search to solve the common due date early/tardy machine scheduling problem. *Computers and operations research*, *24*(3), 199–208. DOI:10.1016/S0305-0548(96)00052-4

[8] Almeida, M. T., & Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers and operations research*, *25*(7–8), 625–635. DOI:10.1016/S0305-0548(97)00097-X

[9] Nearchou, A. C. (2008). A differential evolution approach for the common due date early/tardy job scheduling problem. *Computers and operations research*, *35*(4), 1329–1343. DOI:10.1016/j.cor.2006.08.013

[10] Chang, P. C., Chen, S.-H., & Mani, V. (2007). Single machine scheduling for jobs with individual due dates and a learning effect: genetic algorithm approach single machine scheduling for jobs: genetic algorithm approach. *International journal of computational science*, *1*(2), 215–223.

[11] Valente, J. M. S., & Gonçalves, J. F. (2009). A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers & operations research*, *36*(10), 2707–2715. https://doi.org/10.1016/j.cor.2008.11.016

[12] Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, *6*(2), 154-160. DOI: 10.1287/ijoc.6.2.154

[13] Eberhart, R., & Kennedy, J. (1995). Particle swarm optimization. *Proceedings of the IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948). Citeseer.

[14] Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., & Nakanishi, Y. (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE transactions on power systems*, *15*(4), 1232–1239. https://ieeexplore.ieee.org/abstract/document/898095

[15] Salman, A., Ahmad, I., & Al-Madani, S. (2002). Particle swarm optimization for task assignment problem. *Microprocessors and microsystems*, *26*(8), 363–371. DOI:10.1016/S0141-9331(02)00053-4

[16] Zhang, H., Li, H., & Tam, C. M. (2006). Particle swarm optimization for resource-constrained project scheduling. *International journal of project management*, *24*(1), 83–92. https://doi.org/10.1016/j.ijproman.2005.06.006

[17] Andrés, C., & Lozano, S. (2006). A particle swarm optimization algorithm for part-machine grouping. *Robotics and computer-integrated manufacturing*, *22*(5–6), 468–474. DOI:10.1016/j.rcim.2005.11.013

[18] 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 flowshop sequencing problem. *European journal of operational research*, *177*(3), 1930–1947. DOI:10.1016/j.ejor.2005.12.024

[19] Choobineh, F. F., Mohebbi, E., & Khoo, H. (2006). A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. *European journal of operational research*, *175*(1), 318–337. DOI:10.1016/j.ejor.2005.04.038

[20] Norman, B. A., & Bean, J. C. (1999). A genetic algorithm methodology for complex scheduling problems. *Naval research logistics*, *46*(2), 199–211.

[21] Wang, C. S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers and operations research*, *29*(12), 1621–1640. DOI:10.1016/S0305-0548(01)00031-4

[22] Samanlioglu, F., Kurz, M. B., Ferrell, W. G., & Tangudu, S. (2007). A hybrid random-key genetic algorithm for a symmetric travelling salesman problem. *International journal of operational research*, *2*(1), 47–63.