Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 409486, 9 pages
http://dx.doi.org/10.1155/2013/409486

Research Article

Particle Swarm Optimization Algorithm for Unrelated Parallel Machine Scheduling with Release Dates

Yang-Kuei Lin

Department of Industrial Engineering and Systems Management, Feng Chia University, P.O. Box 25-097, Taichung 40724, Taiwan

Correspondence should be addressed to Yang-Kuei Lin; yklin@mail.fcu.edu.tw

Received 6 September 2012; Revised 11 December 2012; Accepted 25 December 2012

Academic Editor: Baozhen Yao

We consider the NP-hard problem of minimizing makespan for n jobs on m unrelated parallel machines with release dates in this research. A heuristic and a very effective particle swarm optimization (PSO) algorithm have been proposed to tackle the problem. Two lower bounds have been proposed to serve as a basis for comparison for large problem instances. Computational results show that the proposed PSO is very accurate and that it outperforms the existing metaheuristic.

1. Introduction

This research considers the problem of scheduling n jobs on m unrelated parallel machines in the presence of release dates. The performance measure, makespan, is defined as $\max(C_1, \ldots, C_n)$, where C_j is the completion time of job j. Minimizing makespan not only completes all jobs as quickly as possible but also is a surrogate for maximizing the utilization of machines. Following the three-field notation of Graham et al. [1], we refer to this problem as $R|r_j|C_{\max}$. This problem is NP hard [2].

Chen and Vestjens [3] used the largest processing time (LPT) to minimize makespan for identical parallel machines with release dates ($P|r_j|C_{\max}$). The release date of a job is not known in advance, and its processing time becomes known at its arrival. Kellerer [4] proposed algorithms for the $P|r_j|C_{\max}$ problem and $P|r_j|C_{\min}$ problem. Koulamas and Kyparisis [5] considered uniform parallel machine scheduling problems ($Q|r_j|C_{\max}$). They proposed a heuristic and derived a tight worse-case ratio bound for this heuristic. Centeno and Armacost [6] showed that the LPT rule performed better than the least flexible job (LFJ) rule for the problem with machine eligibility restrictions ($P|r_j, M_j|C_{\max}$). Lancia [7] applied a branch-and-bound (b & b) procedure to solve scheduling problems with release dates and tails on two unrelated parallel machines ($R_2|r_j, q_j|C_{\max}$). Similarly, Gharbi and Haouari [8] also presented a b & b procedure to solve the $P|r_j, q_j|C_{\max}$ problem. Carlier and Pinson [9] reported new results on the structures of Jackson's pseudopreemptive scheduling applied to the $P|r_j, q_j|C_{\max}$ problem. Li et al. [10] used a polynomial time approximation scheme for scheduling identical parallel batch machines ($P|r_j, B|C_{\max}$). Li and Wang [11] proposed an efficient algorithm for scheduling with inclusive processing set restrictions and job release times ($P|r_j, incl.proc.sets|C_{\max}$).

To the best of our knowledge, no research has yet been published that develops an efficient algorithm to minimize makespan for unrelated parallel machines with release dates. The rest of this paper is organized as follows. Section 2 presents our proposed lower bounds. Section 3 presents the proposed PSO. In Section 4, the computational results are reported. Section 5 presents our conclusions and suggestions for future research.

2. Lower Bounds to $R|r_j|C_{\max}$

We propose two straightforward and easily implementable lower bounds for the studied problem. LB_1 is the maximum value of each job's release date plus the minimum processing time (across all machines). LB_2 is set to the minimum release date (among all jobs) plus the sum of all jobs' minimum processing times (across machines) divided by the number

Table 1: The matrix of processing times for example 1.

p_{ij}	j_1	j_2	j_3	j_4	j_5	j_6	j_7
m_1	15	29	40	32	46	8	44
m_2	41	29	40	32	31	24	49
r_j	27	4	3	9	13	17	1

of machines. We set lower bound LB equal to the maximum value of LB_1 and LB_2:

$$LB_1 = \max_{1\le j\le n}\left(r_j + \min_{1\le i\le m} p_{ij}\right)$$

$$LB_2 = \min_{1\le j\le n} r_j + \frac{\left(\sum_{j=1}^{n} \min_{1\le i\le m} p_{ij}\right)}{m} \tag{1}$$

$$\mathrm{LB} = \max\left\{LB_1, LB_2\right\}.$$

To illustrate the proposed lower bounds, we consider example 1, which has 2 machines and 7 jobs. The matrix of processing times for example 1 is given in Table 1:

$$\begin{aligned} LB_1 &= \max\{27+15, 4+29, 3+40, 9 \\ &\qquad +32, 13+31, 17+8, 1+44\} = 45 \\ LB_2 &= 1 + \frac{(15+29+40+32+31+8+44)}{2} \\ &= 1 + \frac{199}{2} = 100.5 \\ \mathrm{LB} &= \max\{45, 100.5\} = 100.5. \end{aligned} \tag{2}$$

3. The Proposed PSO Algorithm

PSO was first introduced by Kennedy and Eberhart [12] for solving continuous nonlinear function optimization problems. PSO is based on the metaphor of social interaction and communication in flocks of birds or schools of fish. In these groups, there is a leader (the one with the best performance) who guides the movement of the whole swarm. In a PSO, each individual is called a "particle," and each particle flies around the search space with some velocity. In each iteration, a particle moves from its previous location to a new location at its newly updated velocity, which is calculated based on the particle's own experience and the experience of the whole swarm.

A population of M particles are assumed to evolve in an N-dimensional vector search R^N such that each particle k is assigned the position vector $X_k^t = (x_{k1}^t, x_{k2}^t, \ldots, x_{kN}^t)$ and velocity vector $V_k^t = (v_{k1}^t, v_{k2}^t, \ldots, v_{kN}^t)$, where x_{kd}^t represents the location and v_{kd}^t represents the velocity of particle k in the dth dimension of the search space at the tth iteration and $k \in \{1, 2, \ldots, M\}$, $d \in \{1, 2, \ldots, N\}$. Each particle knows its position and the corresponding objective function. The local best position for each particle k is encoded in the variables $P_k^t = (P_{k1}^t, P_{k2}^t, \ldots, P_{kN}^t)$, and the global best position among all particles is encoded in the variable $P_g^t = (P_{g1}^t, P_{g2}^t, \ldots, P_{gN}^t)$. The standard PSO equations can be described as follows [12]:

$$\begin{aligned} v_{kd}^{t+1} &= w v_{kd}^t + c_1 r_1\left(p_{kd}^t - x_{kd}^t\right) + c_2 r_2\left(p_{gd}^t - x_{kd}^t\right) \\ x_{kd}^{t+1} &= x_{kd}^t + v_{kd}^{t+1}, \end{aligned} \tag{3}$$

where w is the weight that controls the impact of the previous velocities on the current velocity, c_1 is the cognition learning factor, c_2 is the social learning factor, and r_1 and r_2 are random numbers uniformly distributed in $[0, 1]$.

PSO has been successfully applied to a variety of continuous nonlinear optimization problems. In recent years, considerable effort has been expended on solving scheduling problems by PSO algorithms. Articles [13, 14] used PSO algorithms to solve scheduling problems similar to the problems in this paper. Reference [13] provided a PSO algorithm for scheduling identical parallel machines to minimize makespan ($P||C_{\max}$). Reference [14] presented a PSO algorithm for scheduling nonidentical parallel batch processing machines to minimize makespan ($P|batch, s_j, S_m|C_{\max}$). This research uses PSO for the $R|r_j|C_{\max}$ problem. The following five headings describe the PSO algorithm used in this research: particle representation, initial population generation, particle velocity and sequence metric operators, local search, stopping criteria, and parameter settings.

3.1. Particle Representation. A coding scheme developed in [15] is used to represent a solution to the problem at hand. The coding scheme uses a list of job symbols and partitioning symbols. A sequence of job symbols, denoted by integers, represents a possible sequence of jobs. The partitioning symbol, an asterisk, designates the partition of jobs to machines. Generally, for an m-machine n-job problem, a solution contains $m - 1$ partitioning symbols and n job symbols, resulting in a total size of $(m + n - 1)$. For example, for a schedule with 7 jobs and 2 machines, the particle can be represented as shown in Figure 1.

7	6	1	4	*	3	2	5

Figure 1: Representation of a particle's solution.

The completed schedule is thus jobs 7, 6, 1, and 4 on machine 1; jobs 3, 2, and 5 on machine 2. This coding scheme specifies not only which jobs are assigned to which machine but also the order of the jobs on each machine. These pieces of

information are important, since we are scheduling unrelated parallel machines with release dates.

3.2. Initial Population Generation. In order to give the PSO algorithm good initial solutions and to increase the chances of getting closer to regions that yield good objective functions, we propose a heuristic, named SRD_Reassign. The proposed heuristic SRD_Reassign is described as follows.

3.2.1. Heuristic SRD_Reassign

Step 1. Let U be the set of unscheduled jobs; let t_i be the sum of the processing times of the jobs that have already been scheduled on machine i, $i = 1, \ldots, m$. Initially, set $U = \{1, \ldots, n\}$ and $t_i = 0$, for $i = 1, \ldots, m$.

Step 2. Arrange the jobs in the order of the shortest release date (SRD) first, and then assign the job to machine i^* that has the minimum processing time, that is, $p_{i^*j} = \min_{1 \le i \le m} p_{ij}$. Repeat until all jobs have been scheduled to generate a complete schedule.

Step 3. Let A_i be the set of scheduled jobs on machine i, $i = 1, \ldots, m$; let $C_{\max} = \max_{i=1,\ldots,m}\{t_i\}$ represent the maximum completion time and denote the set of candidate jobs for reassignment as B. Initially, set $B = \{null\}$.

Step 4. Identify machine i for which $t_i = C_{\max}$. For every job j, $j \in A_i$, search for machine $h (h \neq i)$, such that if job j was reassigned to machine h and the jobs on machine h were sorted in SRD, the new calculated t_h would be smaller than $C_{\max}$, that is, $t_h = \max(t_h, r_j) + p_{hj, j \in A_h} < C_{\max}$. If job j and machine h can be found, update the candidate set by adding job j on machine h to the candidate set B by setting $B = B \cup \{(h, j)\}$. If $B = \{null\}$, then go to Step 6.

Step 5. Select machine h and job j from B for the reassignment that has maximum Gain, where Gain $= \max\{C_{\max} - t_h\}$. Reassign job j to machine h by setting $A_h = A_h \cup \{j\}$. Sort the jobs on machine h in SRD and update $t_h = \max(t_h, r_j) + p_{hj,\, j \in A_h}$. Remove job j from machine i by setting $A_i = A_i \setminus \{j\}$. Sort the jobs on machine i in SRD and update $t_i = \max(t_i, r_j) + p_{ij, j \in A_i}$. Set $C_{\max} = \max_{i=1,\ldots,m}\{t_i\}$ and $B = \{null\}$. Go to Step 4.

Step 6. Terminate the procedure.

The first two particles are generated by first-come, first-serve (FCFS) rule and SRD_Reassign. The remaining particles are generated by applying local search to the solution found by SRD_Reassign. For FCFS rule, we consider all unscheduled jobs and schedule each one on the first available machine according to FCFS. Local search is done by randomly choosing two jobs j_1 and j_2 from the solution found by SRD_Reassign and then interchanging jobs j_1 and j_2 to generate a new solution. From the initial population pool, we identify the best current solution $C^*_{\max}$ and update the global best location (P^t_g).

3.3. Particle Velocity and Sequence Metric Operators. Kashan and Karimi [13] worked on the classical PSO equations to provide a discrete PSO algorithm that maintained all major characteristics of the original continuous PSO equations when solving parallel machine scheduling problems. In this research, we use the two equations proposed in [13] to update the particle velocity and the sequence metric operators as shown in (4) and (5):

$$V_k^{t+1} = V_k^t \overset{+}{\mathrm{O}} \left(\left(R_1 \overset{\times}{\mathrm{O}} \left(P_k^t \overset{-}{\mathrm{O}} X_k^t \right) \right) \overset{+}{\mathrm{O}} \left(R_2 \overset{\times}{\mathrm{O}} \left(P_g^t \overset{-}{\mathrm{O}} X_k^t \right) \right) \right) \tag{4}$$

$$X_k^{t+1} = X_k^t \overset{+}{\mathrm{O}} V_k^{t+1}. \tag{5}$$

In (4), V_k^t and X_k^t represent the velocity and position arrays of particle k at the tth iteration, respectively. P_k^t and P_g^t represent the local best position for each particle k and the global best position among all particles visited so far. R_1 and R_2 are 1-by-$(m + n - 1)$ arrays in which each digit is 0 or 1. These random arrays are generated from a Bernoulli distribution. $\overset{-}{\mathrm{O}}$, $\overset{\times}{\mathrm{O}}$, and $\overset{+}{\mathrm{O}}$ are subtraction, multiplication, and addition operators, respectively. The definitions of the operators are as follows.

The subtraction operator ($\overset{-}{\mathrm{O}}$) defines the differences between the current position of the kth particle, X_k^t, and a desired position P_k^t(or P_g^t). It first finds elements that do not have the same content in X_k^t and P_k^t(or P_g^t). It schedules those elements based on their orders in P_k^t(or P_g^t). Next, it finds elements that have the same content in X_k^t and P_k^t(or P_g^t) and gives those elements zero values. It puts zero-valued elements in SRD and then schedules them on whatever machine that offers the earliest completion time (ECT). Figure 2 demonstrates the manner in which the $\overset{-}{\mathrm{O}}$ operator works for example 1.

The multiplication operator ($\overset{\times}{\mathrm{O}}$) can enhance our PSO algorithm's exploration. It first generates a 1-by-$(m + n - 1)$ binary vector for a solution vector, and then it does a multiplication process where the asterisk positions from solution are kept. These random binary arrays perform subroutines within PSO that use random numbers to enhance the exploration ability of PSO. Figure 3 demonstrates the manner in which the $\overset{\times}{\mathrm{O}}$ operator works for example 1. The nonzero-valued elements in $A \overset{\times}{\mathrm{O}} B$ are scheduled first. Then, all zero-valued elements in $A \overset{\times}{\mathrm{O}} B$ are sorted in SRD, and then each zero-valued element is assigned to the machine that offers the ECT.

The addition operator ($\overset{+}{\mathrm{O}}$) is a crossover operator that is commonly used in genetic algorithms. Here, we used a crossover that was proposed in [15]. The crossover scheme has three main steps: (1) it obtains asterisk positions from the first parent A; (2) it obtains a randomly selected subschedule from the first parent A; (3) it scans the second parent B from left to right and fills the gaps in the child's ($A \overset{+}{O} B$) schedule

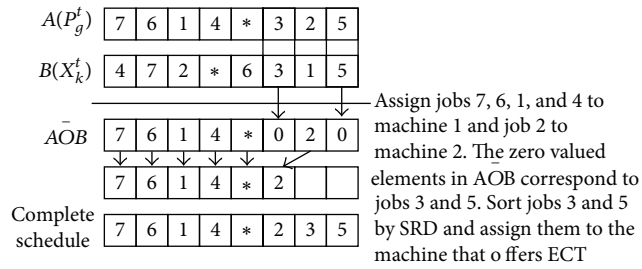

Figure 2: The subtraction operator ($\bar{O}$) applied to example 1.

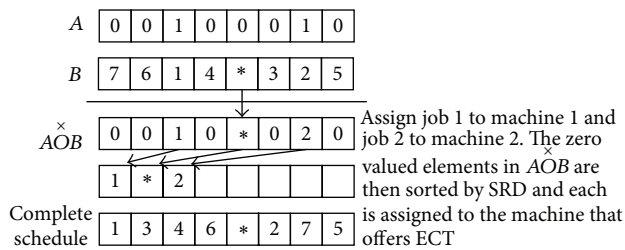

Figure 3: The multiplication operator ($\times$) applied to example 1.

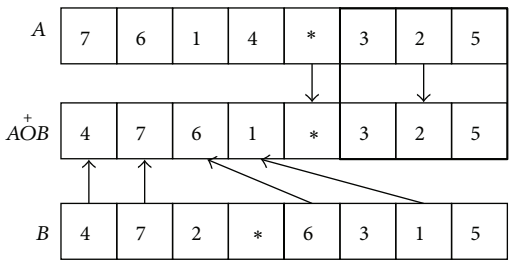

Figure 4: The addition operator ($+$) applied to example 1.

with jobs taken from the second parent B. Figure 4 shows an illustration of this crossover scheme.

During the execution of the $\bar{O}$, $\times$, and $+$ operators, whenever a complete schedule generates a better solution than the best current solution $C^*_{\max}$, we will update the best current solution $C^*_{\max}$ and the global best location (P_g^t).

3.4. Local Search. It is well known that evolutionary memetic algorithms can be improved by hybridization with local search. For each particle P_k^t, we do the following local search procedure (LSP) to further improve the current solution.

3.4.1. Local Search Procedure (LSP)

Step 1. Initially, set $l = 1$.

Step 2. Identify machine i that has maximum completion time ($C_{\max}$). Randomly choose one job j_1 from machine i and randomly choose one job j_2 from machine $h(h \neq i)$. Insert job j_1 after job j_2 on machine h. If a better $C_{\max}(P_k^t)$ is found, update P_k^t and go to Step 2; otherwise, go to Step 3.

Step 3. Randomly choose two jobs j_1 and j_2 from P_k^t; j_1 and j_2 can be on the same machine or on two different machines. Interchange jobs j_1 and j_2. If a better $C_{\max}(P_k^t)$ is found, update P_k^t and go to Step 2; otherwise, go to Step 4.

Step 4. If $l = L$, stop; otherwise set $l = l + 1$ and go to Step 2.

Again, during the execution of LSP, whenever a complete schedule generates a better solution than the best current solution $C^*_{\max}$, we will update the best current solution $C^*_{\max}$ and the global best location (P_g^t).

3.5. Stopping Criteria and Parameter Settings. We studied the effects of five important parameters (R_1, R_2, local search moves L, population size, and number of iterations) on the performance of our proposed PSO. The model was tested and parameterized through a factorial study. The selected PSO parameters were $R_1 = 0.9$, $R_2 = 0.1$, $L = 10$, population size = 73, and number of iterations = 4044. The appendix includes a detailed description of our parameterization study.

4. Computational Results

In this section, we present several computational results of the proposed PSO algorithm. We compare our proposed PSO algorithm with a mixed integer programming (MIP) model developed in our previous research [16] on the $R|r_j|C_{\max}$ problem. The MIP model [16] was coded in AMPL and implemented in CPLEX 11.2. The proposed heuristic, SRD_Reassign, and the proposed PSO algorithm were implemented in C. The MIP model, heuristic, and PSO algorithm were executed on a computer with a 2.5 GHz CPU and 2 GB of memory. Processing times p_{ij} were generated from the uniform distribution [1, 100]. Release dates were generated in a manner similar to that of Mönch et al. [17]. We generated release dates r_j from the uniform distribution $[0, (\alpha/m)((\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij})/m)]$. α controlled the range of release dates. High values of α tend to produce widely separated release dates. α values were set at 0.1, 0.25, and 0.5. We used 4 machines with 18 jobs ($4m18n$) to represent small problem instances and 10 machines with 100 jobs ($10m100n$) to represent large problem instances. For each α, 20 problem

Table 2: The performance of heuristic SRD_Reassign for small problem instances.

$4m18n$	MIP		SRD_Reassign		FCFS	
α	Mean	Avg. time	Mean	Avg. time	Mean	Avg. time
0.10	1	1176.32	1.11	0.003	1.46	0.000
0.25	1	239.19	1.09	0.002	1.43	0.000
0.50	1	58.02	1.04	0.002	1.34	0.000
Average	1	491.18	1.08	0.002	1.41	0.000

Mean = average ratios of heuristic/MIP obtained from 20 instances.

Table 3: The performance of heuristic SRD_Reassign for large problem instances.

$10m100n$	SRD_Reassign		FCFS	
α	Mean	Avg. time	Mean	Avg. time
0.10	1.43	0.005	2.03	0.001
0.25	1.17	0.003	1.62	0.000
0.50	1.05	0.005	1.36	0.000
Average	1.22	0.004	1.67	0.000

Mean = average ratios of heuristic/LB obtained from 20 instances.

instances were randomly generated. The effectiveness of each algorithm was evaluated by the mean performance and the required computation time (labeled as "Avg. time"). For small problem instances, a ratio was calculated by dividing the algorithm's makespan by the optimal MIP makespan. For large problem instances, a ratio was calculated by dividing the algorithm's makespan by the makespan from LB. The mean performance of the algorithm for each α was the average ratio obtained from 20 runs of the algorithm.

4.1. Comparison of Heuristics for $R|r_j|C_{\max}$ Problem. We compared the proposed heuristic SRD_Reassign with the optimal solutions obtained from the MIP model [16] and FCFS. FCFS is a dispatching rule that is commonly used for practical problems with release dates. Computational results for small and large problem instances are given in Tables 2 and 3, respectively. The results show that the proposed SRD_Reassign outperformed FCFS in terms of makespan. For small problem instances, the average SRD_Reassign makespan was 1.08 times the optimum, and the average FCFS makespan was 1.41 times the optimum. Both heuristics outperformed the MIP model in terms of computation time. When α was small, both heuristics had larger ratios to the optimal solutions than they had when α was large. Also, the MIP took more computation time to find optimal solutions when α was small. This probably indicates that problems with small release date ranges are harder to solve than problems with large release date ranges. For large problem instances, the average SRD_Reassign makespan was 1.22-times greater than the lower bound (LB), and the average FCFS makespan was 1.67 times the LB. Both heuristics were calculated very quickly (in less than 1 second) even for large problem instances.

4.2. Comparison of Metaheuristics for $R|r_j|C_{\max}$ Problem. We compared the proposed PSO with an existing metaheuristic, namely, the version of simulated annealing (SA) described by Lee et al. [18]. This SA variant was originally designed for solving the $P||C_{\max}$ problem. SA is a metaheuristic, and it can be used without any problem-dependent knowledge; therefore, it can be used to solve the $R|r_j|C_{\max}$ problem. In order to provide a fair comparison, we used the same initial solution (SRD_Reassign) for both PSO and SA. We also adjusted the SA parameters to ensure that both PSO and SA ran for similar computation times. The termination criterion for SA was set to run for 12 seconds for small problem instances and 83 seconds for large problem instances. If SA found a solution equal to LB, the program would terminate earlier. Computational results for small and large problem instances are given in Tables 4 and 5, respectively.

Computational results show that the proposed PSO outperformed the SA in terms of makespan. For small problem instances, the PSO found optimal solutions at all three α settings. The average SA makespan was 1.05 times the optimum. Both metaheuristics outperformed the MIP model in terms of computation time. The last column in Table 4 reports how many times a given algorithm produced a better makespan than the other algorithm. For instance, a value of c/d in column PSO/SA means that, out of 20 problems, there were c problems for which PSO yielded a better solution than SA, d problems for which SA performed better, and 20-c-d problems for which PSO and SA yielded the same makespan.

For large problem instances, the average PSO makespan was 1.08-times greater than the LB, and the average SA makespan was 1.15 times the LB. The last column in Table 5 shows how many times out of 20 the LB was obtained by LB_1 and how many times the LB was obtained by LB_2. When α was small, LB_2 provided a better lower bound than LB_1; however, when α was large, LB_1 provided a better lower bound than LB_2. This suggests that LB_2 performs better for problems with narrow release date ranges, and LB_1 performs better for problems with wide release date ranges.

Table 4: The performance of PSO for small problem instances.

$4m18n$	MIP		PSO		SA		PSO/SA
α	Mean	Avg. time	Mean	Avg. time	Mean	Avg. time	
0.10	1	1176.32	1.00	11.73	1.05	12.79	11/0
0.25	1	239.19	1.00	10.69	1.06	12.17	18/0
0.50	1	58.02	1.00	8.69	1.03	10.28	10/0
Average	1	491.18	1.00	10.37	1.05	11.75	13/0

Mean = average ratios of algorithm/MIP obtained in 20 instances.

Table 5: The performance of PSO for large problem instances.

$10m100n$	PSO		SA		PSO/SA	LB_1/LB_2
α	Mean	Avg. time	Mean	Avg. time		
0.10	1.20	82.61	1.32	84.1	20/0	0/20
0.25	1.03	49.37	1.11	85.54	20/0	10/10
0.50	1.00	25.34	1.01	55.18	10/0	20/0
Average	1.08	52.44	1.15	74.94	16.7/0	10/10

Mean = average ratios of algorithm/LB obtained in 20 instances.

4.3. The Effects of the Proposed PSO. Next, since the proposed PSO effectively incorporates a number of ideas (initial solutions, SRD, ECT, and LSP), we examine which parts are essential to its functionality. We examine these effects by disabling a single component, running the proposed PSO without that component, and observing performance. We choose to study large problems. These experiments are described as follows.

PSO-Initial Heuristics: instead of generating an initial population by heuristics, we randomly generated an initial set of solutions to make up the initial population.

PSO-SRD: instead of sorting elements by SRD and then scheduling them on whatever machine that offered the ECT within PSO operators ($\bar{O}$ and $\overset{\times}{O}$), we randomly chose unscheduled jobs and then scheduled them on whatever machine that offered the ECT.

PSO-ECT: instead of sorting elements by SRD and then scheduling them on whatever machine that offered the ECT within PSO operators ($\bar{O}$ and $\overset{\times}{O}$), we sorted elements by SRD and then scheduled them on the first available machine.

PSO-LSP: local search procedure was disabled.

PSO-LSP + LS [13]: local search procedure was disabled and a local search algorithm used in [13] was applied. Since the formulation $0 < p_a - p_b < FT_i - FT_j$ in step 4 [13] was not suitable for the unrelated parallel machines environment, we modified it to find two jobs from M_i and M_j such that an exchange of those two jobs was able to improve the current best makespan.

Table 6 lists the average makespan ratio of PSO-variant to standard PSO (which has initial heuristics, SRD, ECT, and LSP by default). Table 6 shows that the PSO performed poorly when the initial heuristics were not applied and when the initial population was randomly generated. The PSO also performed poorly when the ECT strategy was not applied within the PSO operators. The average ratio of PSO without initial heuristics to standard PSO was 1.019, the average of PSO without SRD to standard PSO was 1.006, the average of PSO without ECT to standard PSO was 1.020, and the average of PSO without LSP to standard PSO was 1.007. In all, all of the proposed PSO versions without any one of the parts (heuristic initial solutions, SRD, ECT, and LSP) performed worse than the PSO with all of them.

Moreover, we compared our proposed PSO with another existing PSO. The closest existing PSO that we were able to find was the hybridized discrete PSO (HDPSO) proposed in [13]. The HDPSO [13] was designed to minimize makespan for identical parallel machines ($P||C_{\max}$). The proposed PSO and the HDPSO both are designed to minimize makespan for parallel machines and they both use formulas (4)-(5) to update each particle's velocity and position. However, the HDPSO is still quite different from our PSO. The HDPSO considers problems without release dates; hence, its coding scheme does not consider the order of jobs on the same machine. Also, HDPSO considers an identical parallel machine environment; it uses the LPT rule to assign jobs with zero-valued elements within $\bar{O}$, $\overset{\times}{O}$, and $\overset{+}{O}$ operators in formulas (4)-(5). It is well known that the LPT rule does not perform well in unrelated parallel machine environments. If HDPSO is used to solve the $R|r_j|C_{\max}$ problem without modifications, it will not perform very well. Table 7 shows a comparison between HDPSO and our PSO. Since the $\bar{O}$, $\overset{\times}{O}$, and $\overset{+}{O}$ operators within formulas (4)-(5) are quite different in both PSOs, there is no point in comparing them. We focus our comparison on initial heuristics and local searches. The first column in Table 6 indicates that our PSO with an initial heuristic performs better than a version without an initial heuristic. HDPSO might exhibit similar performance differences. The last column in Table 6 indicates that our proposed LSP performs better than the local search algorithm used in [13]. The standard PSO (LSP is embedded)

Table 6: The effects of the proposed PSO.

$10m100n$	PSO-initial heuristics	PSO-SRD	PSO-ECT	PSO-LSP	PSO-LSP + LS [13]
α	Mean	Mean	Mean	Mean	Mean
0.10	1.054	1.017	1.055	1.019	1.023
0.25	1.001	1.001	1.004	1.002	1.003
0.50	1.001	1.001	1.001	1.001	1.001
Average	1.019	1.006	1.020	1.007	1.009
Ave. time	52.55	51.14	51.29	48.44	1624.31

Mean = average ratios of PSO-variant/standard PSO obtained in 20 instances.

Table 7: Comparison between two PSOs.

	Problem	Initial heuristic	$\bar{O}$, $\bar{\times}$, and $\bar{+}$ operators within formulas (4)-(5)	Local search
HDPSO	$P\|\|C_{\max}$	n/a	Coding scheme and LPT rule are not suitable for the $R\|r_j\|C_{\max}$ problem	Local search algorithm
PSO	$R\|r_j\|C_{\max}$	SRD_Reassign	Coding scheme, SRD, and ECT are designed for the $R\|r_j\|C_{\max}$ problem	LSP

Table 8: Factors and levels of PSO parameter study.

Factor	Design units	
	−1	+1
A: R_1	0.1	0.9
B: R_2	0.1	0.9
C: L	10	50
D: population size	10	100
E: iteration	1000	5000

versus PSO-LSP+LS [13] is 1.000 versus 1.009. Moreover, the proposed LSP outperforms the local search algorithm used in [13] in terms of average computation time. Therefore, we can conclude that our proposed PSO provides better and more efficient strategies for parallel machine makespan minimization problems than what HDPSO provides. Our PSO is more likely to provide promising results than HDPSO.

5. Conclusions and Future Work

We studied the problem of scheduling jobs on unrelated parallel machines with release dates to minimize makespan. In this research, we proposed two lower bounds for the studied problem. We also proposed a heuristic, SRD_Reassign, and a metaheuristic, PSO, to tackle the problem. Computational results showed that SRD_Reassign outperformed the commonly used heuristic, FCFS, in terms of makespan. The proposed PSO outperformed a comparable variant of SA in terms of makespan. Future work can extend our approach for other performance criteria or even for multiobjective parallel machine scheduling problems.

Appendix

We studied the effects of five important parameters (R_1, R_2, local search moves L, population size, and number of iterations) on the performance of our proposed PSO. In order to test the significance of each parameter, $10m100n$ was chosen as a representative problem instance; the objective was to minimize $C_{\max}$ in an unrelated parallel machine environment with release dates. Because problems with small ranges of release dates are harder to solve than problems with large ranges of release dates, the release date factor α was set to 0.1. In order to obtain information about the importance of each of the factors, we conducted an initial screening experiment. Each parameter was categorized as being at a high or low level, as shown in Table 8. We conducted a 2^{5-2}_{III} screening experiment to determine which factors were significant. The results of this experiment are shown in Table 9 where each $C_{\max}$ value is the average of 20 problem instances. The half normal plot provided by Design Expert indicates that R_1 (factor A), population size (factor D), and iterations (factor E) were significant to the $C_{\max}$ response in the screening experiment. The model in terms of coded factors is $\hat{y} = 116.43 - 1.13A - 0.45D - 0.59E$. The model shows that increasing the A, D, and E values could decrease makespan.

Next, we used the method of steepest descent (Myers et al. [19]) to provide information about the region of improved response. The path of steepest descent is shown in Table 10. The results show that a reduction in $C_{\max}$ was experienced after Run 2. Although Run 2 improved $C_{\max}$ by 2.4% compared with Run 0, its computation time was very long. Run 1 improved $C_{\max}$ by 2.1% relative to Run 0 and used less computation time. We chose the settings of Run 1 as our final set of parameters. Hence, PSO parameters were set to $R_1 = 0.9$, population size = 73, and iterations = 4044. Since

Table 9: Results of 2_{III}^{5-21} factorial design for parameters of PSO.

Run	A: R_1	B: R_2	C: L	D: Pop-size	E: Iteration	$C_{\max}$
1	0.1	0.9	50	10	1000	118.65
2	0.9	0.1	50	10	5000	115.45
3	0.9	0.9	10	100	1000	115.00
4	0.9	0.1	10	10	1000	116.30
5	0.1	0.1	10	100	5000	116.35
6	0.9	0.9	50	100	5000	114.45
7	0.1	0.9	10	10	5000	117.10
8	0.1	0.1	50	100	1000	118.10

Table 10: The path of steepest descent for PSO.

Run		Coded units			Natural units			$C_{\max}$	Improv.	Time
		A	D	E	R_1	Pop. size	Iteration			
0	Base	0	0	0	0.5	55	3000	117.30	—	50.68
	Increment = △	1	0.40	0.52	0.4	18	1044.4	—	—	—
1	**Base + △**	**1**	**0.40**	**0.52**	**0.90**	**73**	**4044**	**114.80**	**0.021**	**82.61**
2	Base + 2△	2	0.80	1.04	0.90	91	5089	114.40	0.024	129.06
3	Base + 3△	3	1.20	1.57	0.90	109	6133	114.95	0.002	193.28

the R_2 and L were not significant, they were kept at low levels ($R_2 = 0.1$ and $L = 10$) to save computation time.

References

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[2] M. L. Pinedo, *Scheduling Theory, Algorithms, and Systems*, Springer, New York, NY, USA, 4th edition, 2012.

[3] B. Chen and A. P. A. Vestjens, "Scheduling on identical machines: how good is LPT in an on-line setting?" *Operations Research Letters*, vol. 21, no. 4, pp. 165–169, 1997.

[4] H. Kellerer, "Algorithms for multiprocessor scheduling with machine release times," *IIE Transactions*, vol. 30, no. 11, pp. 991–999, 1998.

[5] C. Koulamas and G. J. Kyparisis, "Makespan minimization on uniform parallel machines with release times," *European Journal of Operational Research*, vol. 157, no. 1, pp. 262–266, 2004.

[6] G. Centeno and R. L. Armacost, "Minimizing makespan on parallel machines with release time and machine eligibility restrictions," *International Journal of Production Research*, vol. 42, no. 6, pp. 1243–1256, 2004.

[7] G. Lancia, "Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan," *European Journal of Operational Research*, vol. 120, no. 2, pp. 277–288, 2000.

[8] A. Gharbi and M. Haouari, "Minimizing makespan on parallel machines subject to release dates and delivery times," *Journal of Scheduling*, vol. 5, no. 4, pp. 329–355, 2002.

[9] J. Carlier and E. Pinson, "Jackson's pseudo-preemptive schedule and cumulative scheduling problems," *Discrete Applied Mathematics*, vol. 145, no. 1, pp. 80–94, 2004.

[10] S. Li, G. Li, and S. Zhang, "Minimizing makespan with release times on identical parallel batching machines," *Discrete Applied Mathematics*, vol. 148, no. 1, pp. 127–134, 2005.

[11] C.-L. Li and X. Wang, "Scheduling parallel machines with inclusive processing set restrictions and job release times," *European Journal of Operational Research*, vol. 200, no. 3, pp. 702–710, 2010.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pp. 1942–1948, Perth, Australia, December 1995.

[13] A. H. Kashan and B. Karimi, "A discrete particle swarm optimization algorithm for scheduling parallel machines," *Computers and Industrial Engineering*, vol. 56, no. 1, pp. 216–223, 2009.

[14] P. Damodaran, D. A. Diyadawagamage, O. Ghrayeb, and M. C. Vélez-Gallego, "A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines," *International Journal of Advanced Manufacturing Technology*, vol. 58, pp. 1131–1140, 2012.

[15] R. Cheng, M. Gen, and T. Tozawa, "Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms," *Computers and Industrial Engineering*, vol. 29, no. 1–4, pp. 513–517, 1995.

[16] Y. K. Lin and C. W. Lin, "Dispatching rules for unrelated parallel machine scheduling with release dates," *International Journal of Advanced Manufacturing Technology*, 2013.

[17] L. Mönch, H. Balasubramanian, J. W. Fowler, and M. E. Pfund, "Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times," *Computers and Operations Research*, vol. 32, no. 11, pp. 2731–2750, 2005.

[18] W. C. Lee, C. C. Wu, and P. Chen, "A simulated annealing approach to makespan minimization on identical parallel machines," *International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3-4, pp. 328–334, 2006.

[19] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, John Wiley & Sons, New York, NY, USA, 3rd edition, 2009.