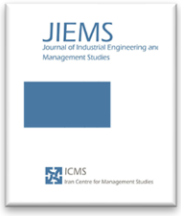
Vol. 9, No. 2, 2022, Pages 1-12

JIEMS
Journal of Industrial Engineering and Management Studies

journal homepage: www.jiems.icms.ac.ir



Just-in-time parallel jobs scheduling: A novel algorithm

Javad Behnamian *[1]

[1] Department of Industrial Engineering, Faculty of Engineering, Bu-Ali Sina University, Hamedan, Iran.

Received: Jun 2022-08/ Revised: Aug 2022-08/ Accepted: Aug 2022-01

Abstract

This research extends a two-phase algorithm for parallel job scheduling problem by considering earliness and tardiness as multi-objective functions. Here, it is also assumed that the jobs may use more than one machine at the same time, which is known as parallel job scheduling. In the first phase, jobs are grouped into job sets according to their machine requirements. For this, here, a heuristic algorithm is proposed for coloring the associated graph. In the second phase, job sets will be sequenced as a single machine scheduling problem. In this stage, for sequencing the job sets which are obtained from the first phase, a discrete algorithm is proposed, which comprises two well-known metaheuristics. In the proposed hybrid algorithm, the genetic algorithm operators are used to discretize the particle swarm optimization algorithm. An extensive numerical study shows that the algorithm is very efficient for the instances which have different structures so that the proposed algorithm could balance exploration and exploitation and improve the quality of the solutions, especially for large-sized test problems.

Keywords: parallel job scheduling; parallel machine; earliness and tardiness; graph coloring; particle swarm optimization.

Paper Type: Original Research

1. Introduction

The scheduling problem is the allocation of limited resources to perform a set of activities in a period of time (Ebrahimi Zade et al. 2016). A parallel job scheduling consists of a production stage that has a parallel machine, and it is assumed that at the same time, a job may use more than one machine for its processing. Many applications of parallelism, such as bandwidth and storage management, are reported (Zhang et al. 2020). Furthermore, many real-world problems such as semiconductors and aircraft manufacturing involve simultaneous optimization of several objective functions. In this paper, to reflect real-world situations adequately, the earliness and tardiness (ET) of jobs are concerned, in which both early and tardy deliveries of a job with respect to its due date are penalized. Another assumption considered in this scheduling model is parallel job scheduling in which a job may use more than one machine simultaneously. Loom scheduling problem in the textile industry (Serafini 1996) and berth allocation problem (Guan et al. 2002) are two applications of the parallel job scheduling problem.

This paper considered the scheduling of parallel jobs on the parallel machine with the sum of the earliness and tardiness objective function. The problem of parallel job scheduling with earliness and tardiness is one of the most up-to-date problems. These problems are significant because of the importance of simultaneous reducing earliness and tardiness, considering the parallel resources, in which the demand is met in a shorter time. Reviewing various research in this field in recent years, it was observed that there are several applications, among them CPU scheduling in computer science (as parallel tasks scheduling) and operating room scheduling (where the patient may simultaneously need several doctors during surgery) mentioned. Here, two decisions are made in two-phase: dividing jobs into independent sets using a graph coloring-based heuristic algorithm and sequencing the independent sets as a single machine scheduling problem using metaheuristic. As proved in Theorem 3.2. in Hoogeveen et al. (1994), a parallel job scheduling problem with only two parallel machine and total completion times objective function is NP-hard. Also, according to complexity hierarchies of deterministic scheduling problems in objective functions in Pinedo (2008), the problem with total tardiness objective function is harder than the other one with total completion time objective function. For these reasons, we can easily conclude that our problem with m parallel machine and the sum of the earliness and tardiness objective function is also NP-hard. Since the problem of parallel machine scheduling with the sum of the earliness and tardiness objective function is NP-hard (Pinedo 2008), in phase 2, an efficient metaheuristic is proposed to achieve a good solution.

*Corresponding Author: behnamian@basu.ac.ir

DOI: 10.22116/jiems.2022.346243.1491

The paper has the following structure. Section 2 reviews the literature of parallel job scheduling. The problem defines in Section 3. Section 4 introduces the proposed graph-based algorithm for dividing the jobs into independent sets. Section 5 introduces an algorithm for sequencing the job sets. Section 6 presents the computational design. Section 7 states conclusions and further research.

2. Literature review

For optimizing the performance and energy efficiency in parallel job scheduling on homogeneous clusters, Zong et al. (2011) proposed performance-energy balanced-based algorithms. Hao et al. (2017) considered a parallel job scheduling problem under multi-clouds in which jobs are in different lists according to the waiting time of jobs and every job has different parallelism. For this problem, they proposed parallel job scheduling based on ZERO-ONE scheduling with multiple targets algorithm. For the parallel job scheduling on multiple manycore machines, Li (2018) proposed the lower bounds for the problems of energy and time-constrained scheduling with precedence constrained in a cloud computing environment. To minimize the maximum completion time, Jansen and Trystram (2016) proposed an approximation algorithm with the absolute ratio for parallel job scheduling on heterogeneous platforms.

For parallel workload consolidation, Liu et al. (2015) introduced a prioritized two-tier virtual machines architecture and proposed a consolidation-based parallel job scheduling algorithm. For scheduling the modular non-linear parallel jobs, Hao et al. (2016) proposed an adaptive algorithm. At the same time, four characteristics of the jobs were taken into account, including the deadlines of jobs, average execution time, the overall system loads and the number of assigned resources. To minimize makespan in the multi-factory scheduling problem with the parallel job, Behnamian (2016) proposed semidefinite programming. In this paper, it is assumed that some factories join together to form a production network. Due to various applications of parallel jobs, this environment has been studied extensively. Table 1 summarizes in chronological order.

Table 1. Summarized literature review

Year	Author/s	Comments
1989	Du and Leung	complexity of scheduling
1992	Wang and Cheng	heuristic of scheduling
1994	Babbar and Krueger	online hard real-time scheduling, partitionable multiprocessors
1994	Turek et al.	scheduling parallelizable tasks, minimizing average response time
1996	Drozdowski	real-time scheduling of linear speedup
1996	Sgall	randomized online scheduling
1997	Glasgow and Shachnai	channel-based scheduling
1998	Rapine et al.	online scheduling of parallelizable jobs
1998	Feitelson and Rudolph	metrics and benchmarking for parallel job scheduling
1998	Feldmann et al.	optimal online scheduling, jobs arrive dynamically according to the dependencies
1999	Krishnamurti and Gaur	approximation algorithm, hypercube parallel task
1999	Kwon and Chwa.	parallel tasks with individual deadlines
1999a	Li	approximation algorithm, independent parallel tasks
1999b	Li	list scheduling algorithm, precedence constrained parallel tasks
2000	Deng et al.	preemptive scheduling on multiprocessors
2000	Jansen and Porkolab	preemptive parallel task
2000	Li and Pan	probabilistic analysis, precedence constrained parallel tasks, multi-computers with contiguous processor allocation
2001	Bischof and Mayr	online scheduling of parallel jobs with runtime restrictions
2002	Jansen	malleable parallel tasks, asymptotic fully polynomial-time approximation scheme
2002	Jansen and Porkolab	linear-time approximation schemes
2002	Srinivasan et al.	selection of partition sizes for moldable scheduling
2003	Jansen and Porkolab	optimal preemptive schedules, Linear programming approaches
2003	Ye and Zhang	online scheduling, parallel job with dependencies
2004	Dutot et al.	approximation algorithms
2007	Ye and Zhang	a 7-competitive online algorithm, improves the previous upper bound
2010	Guo and Kang	malleable parallel job, online algorithm with competitive ratio, optimal for two machines
2011	Barbosa and Moreira	Batch of jobs with non-deterministic arrival times, minimizing the scheduling makespan, Using direct acyclic graph for list scheduling
2012	Ebrahimi Moghaddam and Bonyadi	Multiprocessor task scheduling, immune-based Genetic algorithm, new coding scheme to reduce search space
2013	Brelsford et al.	parallel job scheduling for extreme scale computing, hybrid centralized and distributed approach, improves the scaling behavior of scheduling time
2014	Sun et al.	Online adaptive scheduling for multiple sets of parallel job, two-level algorithm scenario with a feedback-driven adaptive scheduler, minimizing the scheduling total response time and makespan
2018	Parida et al.	designing and modeling the Petri Net for parallel task scheduling for deadline-based task by resolving the conflicts
2020	Behnamian	a semi-definite relaxation-based algorithm for parallel job scheduling with a specific due date
2021	Zheng et al.	online service scheduling of parallel jobs with variable resources in clouds

According to the reviewed literature, the main novelty of this paper is the scheduling of parallel jobs using a novel hybrid algorithm composed of the graph coloring concept and heuristic algorithm. In this regard, after grouping the jobs by a graph coloring heuristic algorithm, we proposed a discrete hybrid metaheuristic approach for sequencing the obtained job sets in order to minimize the sum of earliness and tardiness. In this regard, crossover and mutation operators are embedded in the particle swarm optimization algorithm (PSO). To the best of our knowledge, this study is novel research that combined a graph model and metaheuristic to scheduling problems.

3. Problem statement

This paper considered the parallel job scheduling on a parallel machine. The objective is to find a feasible schedule in which earliness and tardiness are penalized at the same rate for all jobs.

3.1. Notations

The notations used in the rest of the paper are as follows:

m	The number of identical parallel machines,
n	The number of independent parallel jobs
$T = \{T_1, T_2 \ldots T_n\}$	A set of n independent jobs
$P = \{P_1, P_2 \ldots P_m\}$	A set of m machines
$m_j \leq m$	The number of machines simultaneously required at any point in time to process job j (is known as the width of the job and is part of the input)
C_j	The completion time of job j
d_j	The due date of job j
$E_j = \max(0, d_j - C_j),$	The earliness of job j
$T_j = \max(0, C_j - d_j),$	The tardiness of job j

3.2. Assumptions

The characters of the considered problem in this paper are as follows.

- All jobs have unit processing time which we also call its length
- All the problem parameters are known deterministically when scheduling is undertaken.
- Workstation has a set of identical machines.
- A job once started on the machine must be completed on it without interruption.
- A machine can process only one job at a time.
- The machines are available at all times if they are not busy.
- There are no breakdowns or scheduled or unscheduled maintenance.

4. Grouping the jobs

The correspondence between scheduling problems and associated graphs is described in this section. Here it is assumed all jobs have unit processing times and the processing of each job requires the simultaneous availability of a set of machines $P(T_j) \subseteq P$ during its processing time.

4.1. Parallel job scheduling and related graph

The parallel job scheduling, as shown in Figure 1, can intercommunicate with the graph in which a vertex is used as a job and an edge between two vertices is used when the corresponding jobs are in conflict (i.e. $P(T_i) \cap P(T_j) \neq \phi$).

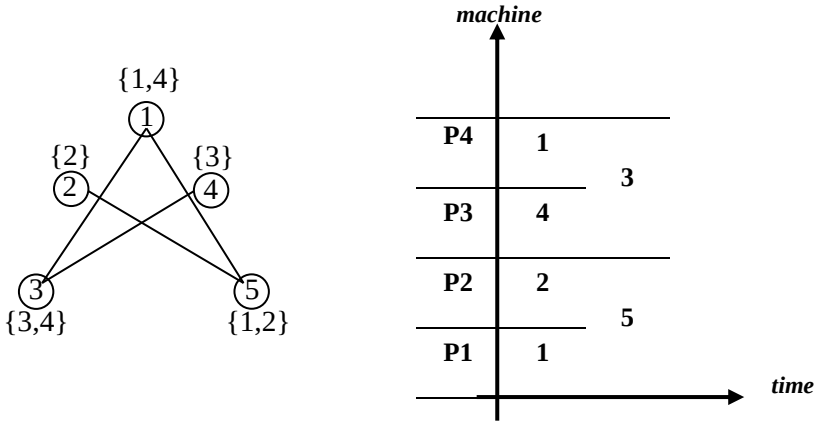


Figure 1. A scheduling system and the associated constraint graph

In this example, the scheduling problem has four machines $\{P_1, P_2, P_3, P_4\}$ and five jobs $\{T_1, T_2, T_3, T_4, T_5\}$ such that $P(T_1) = \{P_1, P_4\}$, $P(T_2) = \{P_2\}$, $P(T_3) = \{P_3, P_4\}$, $P(T_4) = \{P_3\}$ and $P(T_5) = \{P_1, P_2\}$. Note that with unit processing times, since a minimum coloring on the graph corresponds to scheduling with minimum length (Dell'Olmo & Gentili 2006), we are interested in the graph with a coloring that implies a schedule without idle times. In the following subsection, correspondence between graph and scheduling systems is described more formally. To do that, we need to introduce a method for coloring the associated graph.

4.2. Graph coloring

In the first phase, the jobs must be grouped into job sets according to their machine requirements using the vertex coloring concept. In general, the grouping representation consists of a job part and a group part. The job part consists of n genes that must be colored and the color part consists of a permutation of the k color labels. A job gene can take any of k colors as an allele, indicating that the job in question belongs to a color of the given label. An example of a chromosome with $n = 7$ and $k = 3$ is shown in Figure 2.

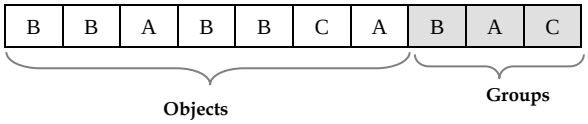


Figure 2. Example of a chromosome in grouping representation

The group part shows that three colors A, B and C are used for coloring the graph. The job part discloses that nodes 3 and 7 are colored with color A, nodes 1, 2, 4 and 5 are colored with color B and node 6 is colored with color C.

```
for i = 1 to n
   assign to i color 1
end for
for i = 1 to n
  for j = 1 to n
    if i is adjacent to j and have the same color
    (Search the new color for j)
      k = color of j+1;
      Assign color k to j
      for l = 1 to n
        if color of l is k and l is adjacent to j
          k = k + 1
          Next l
          Assign color k to j
        end if
      end for
    end if
  end for
end for
```

Since finding the minimum vertex coloring of an arbitrary graph is a known NP-hard problem (Garey & Johnson1979), in Algorithm 1, we propose a heuristic algorithm that can generate legal coloring for n nodes. To evaluate the colored graph, the objective of the corresponding machine scheduling system is defined as the fitness function. It is obvious to minimize the corresponding scheduling problem, the proposed heuristic algorithm must minimize the number of colors used for coloring the graph.

4.3. Sequencing the job sets

In phase 2, the job sets that are obtained by classification of jobs according to their need to machines in the independent set, must be sequenced at the group of the machine. In this phase, we have a single machine scheduling problem, which in it the job sets and parallel machine are considered as jobs and single machine, respectively.

If C_j and d_j be the completion time and due date for the job j, respectively, the earliness and tardiness of job j are defined as

$$E_j = \max(0, d_j - C_j), \tag{1}$$

$$T_j = \max(0, C_j - d_j), \tag{2}$$

It is clear in the scheduling problem with $\sum(E+T)$ objective function, earliness and tardiness are penalized at the same rate for all jobs (Su 2009). This form of objective function perfectly fits a Just-in-time (JIT) manufacturing policy where an early or late delivery of a job results in an increase in production costs.

In this study, as shown in Equation (3), the sum of the earliness and tardiness of jobs are combined as a single scalar value.

$$Z = \sum_{j=1}^{n} (E_j + T_j) = \sum_{j=1}^{n} |d_j - C_j|. \tag{3}$$

For solving our problem with this objective function, a discrete particle swarm optimization algorithm is presented.

5. A discrete hybrid metaheuristic approach

In applying the particle swarm optimization algorithm (PSO) to generate a movement based on swarm in continuous problems, there are several approaches, but in discrete ones, it needs more innovation. In this regard, for example, by coding a solution as a chromosome, crossover and mutation operators can be used as a movement policy. Such coding allows the possible integration of different features from other metaheuristic algorithms, such as the genetic algorithm (GA) in the PSO. The following subsection discusses which aspects have been borrowed from the particle swarm optimization algorithm and genetic algorithm.

5.1. Particle swarm optimization algorithm

This algorithm is based on modeling and simulating the behavior of a group flight of birds or a mass movement of fishes. Each particle has a coordinate that specifies what the particle coordinates are in the multidimensional search space. As the particle moves over time, the location of the particle changes. $x_{id}(k)$ denotes the location of the particle i in dimension d at time k. Each particle also needs speed to move through space, with $v_{id}(k)$ being the particle speed i in dimension d at time k. Whether or not the location of a particle in the search space is a suitable location is evaluated by a fitness function. The particles are capable of remembering the best location they have been in their lifetime. It is called the best individual experience of a particle or the best location met by a swarm, which is shown by Pbest. Particles can also be aware of the best location met by the whole group, called gbest. The particle speed involved in the optimization process reflects the experimental knowledge of the particle and the swarm information. The new velocity of each particle is calculated as follows:

$$v_{id}(k+1) = \omega v_{id}(k) + \lambda_1 r_1 [pn_{id}(k) - x_{id}(k)] + \lambda_2 r_2 [pn_{gd}(k) - x_{id}(k)], \quad (4)$$

where λ_1 and λ_2 are acceleration coefficients, ω is inertia factor, and r_1 and r_2 are two independent random numbers uniformly distributed in the range [0, 1].
Thus, in each generation, the position of each particle is updated according to Equation (5).

$$x_{id}(k+1) = x_{id}(k) + v_{id}(k+1) \quad (5)$$

Algorithm 2 shows the structure of the proposed algorithm.

Algorithm 2: Basic PSO structure

Step 1: Initialize a population of particles with random positions and velocities, where each particle contains *d* variables.
Step 2: Evaluate the objective values of all particles. Let p_{best} of each particle and its objective value be equal to its current position and objective value, and let g_{best} and its objective value be equal to the position and objective value of the best initial particle.
Step 3: Update the velocity and position of each particle according to Equations (4) and (5).
Step 4: Evaluate the objective values of all particles.
Step 5: For each particle, compare its current objective value with the objective value of its p_{best}. If the current value is better, then update p_{best} and its objective value with the current position and objective value.
Step 6: Determine the best particle of the current swarm with the best objective value. If the objective value is better than the objective value of g_{best}, update g_{best} and its objective value with the position and objective value of the current best particle.
Step 7: If a stopping criterion is met, output g_{best} and its objective value; otherwise, go to Step 3.

5.2. Genetic algorithm

In our proposed algorithm, we make use of GA operators as the most well-known mechanisms to generate new individuals (Yazdani & Jolai 2015). In the subsequent section, a discrete PSO algorithm that hybrids with crossover and mutation operators is introduced.

5.3. The proposed algorithm

In applying the PSO algorithm to our problem, the important issue is to find a method to redefine subtraction and addition operations. In this paper, the genetic operators as a useful method (Niu et al. 2008) are used to update the particle position. The structure of the proposed algorithm is shown in Algorithm 3 and the implementation details are mentioned after that.

Algorithm 3: DPSO algorithm structure

Step 1: Randomly initialize a population with 25 particles
Step 2: Evaluate the objective values of all particles for determining p_{best} and g_{best}.
Step 3: Calculate the velocity elements of each particle according to the following procedures.
- Through the mutation process, we make a slight change in the sequence, i.e. generating a new but similar sequence as a solution based on inertia (first part of Equation 4).
- Use crossover for generating the solution based on p_{best} and based on g_{best} (second and third parts of Equation 4) for each particle.

Step 4: Update the position of each particle: After the calculating three components of velocity (solution based on inertia (S_i), solution based on p_{best}(S_p), solution based on social (S_s)), elitism among { S_i , S_p , S_s } is used to update the new population and enforce the search ability.
Step 5: Evaluate each particle by its corresponding objective value
Step 6: Determine the best particle of the current swarm with the best objective value. If the objective value is better than the objective value of g_{best}, update g_{best} and its objective value with the position and objective value of the current best particle.
Step 7: If a stopping criterion is met, output g_{best} and its objective value; otherwise, go to Step 3.

5.3.1. Encoding scheme

The proposed representation of our proposed algorithm to solve scheduling is based on coding all job sets as genes in a $1\text{-by-}(n')$ string where n' is a number of jobs sets. In this type of representation, the sequence of job sets is represented by the number of genes from the left side to the right side. Figure 3 shows an example of representation. In this example, there are five job sets with order 5→1→2→4→3.

5	1	2	4	3

Figure 3. Solution representation

5.3.2. Initial solution

In this paper, to generate a feasible solution, a random initial solution is used.

5.3.3. Crossover operator

In this paper, a two-point crossover operator is used to generate a new position (NP) to the p_{best} or g_{best} (S_p or S_s).

P	4	8	1	2	3	9	10	5	7	6
S	2	7	8	1	3	10	4	6	5	9
NP	4	8	2	1	3	10	5	9	7	6

Figure 4. Two-point crossover operator

As shown in the example, after selecting two parents (P and S) randomly from the population, two points are chosen from the first parent and the genes outside these points are transmitted exactly to the offspring's chromosome (NP). Then these genes are removed from the second parent and the remaining genes, in order of presentation in the second parent, are copied into the empty cells of the offspring's chromosome.

5.3.4. Mutation operators

In this paper, the insertion mutation is used. As shown in the following example, a randomly chosen cell is inserted into a randomly chosen position (the 2nd locus marked by a blue block).

Before	4	8	1	2	3	9	10	5	7	6
After	4	8	5	1	2	3	9	10	7	6

Figure 5. Insertion mutation

6. Computational results

Considering earliness/tardiness objective, parallel job, and parallel machine, to the best of our knowledge, the most related and newest study to our research is Leung et al. (Leung et al. 2002), which proposed a neighborhood search (NS) algorithm. In order to evaluate the effectiveness of the proposed algorithm, at first, the neighborhood search is adapted to the ET objective, and then, for the randomly generated test problems, our proposed algorithm was compared with it. The algorithms were implemented in MATLAB 7 under a Microsoft Windows 7 environment.

6.1. Data generation and settings

The test problems used in this paper were generated using the level(s) of factors are shown in Table 2.

Table 2. Factor levels

Factor	Levels			
Number of jobs (n)	50	100	50	1000
Number of machines (m)	10	20	50	
Width of job (m_j)	$(0,\ m/2)$			

Another important issue is the due dates of the jobs. In this study, the due dates are uniformly distributed from 1 to 3, which are 100% to 300% of processing time.

6.2. Stopping rule

The stopping condition for sequencing the job sets is set to a number of iterations to 25, 50, 100 and 200 repetitions for the problem with 50,100, 500 and 1000 jobs, respectively.

6.3. Numerical Results

In this paper, to evaluate the performance of the proposed algorithm, as described in subsection 6.1, 120 instances are generated and each of them is solved ten times and the average results of them are used in Equation (6). The comparison results are reported in Table 3.

Table 3. Computational results

Instance (Job × Machine)	RPD of Algorithm	
	DPSO	NS
50×10	0.000000	0.021253
50×20	0.001033	0.029671
50×50	0.000134	0.029417
50 Jobs	0.000389	0.026780
100×10	0.000188	0.017626
100×20	0.000564	0.024644
100×50	0.000060	0.045902
100 Jobs	0.000271	0.029391
500×10	0.000621	0.027224
500×20	0.001058	0.029846
500×50	0.000316	0.097436
500 Jobs	0.000665	0.051502
1000×10	0.000047	0.015282
1000×20	0.001053	0.021110
1000×50	0.000627	0.079693
1000 Jobs	0.000576	0.038695
Average	0.000475	0.036592

In this table, after computation of $\sum(E+T)$ of each instance of each algorithm, considering the result of two algorithms, Min_{sol} as the best solution is calculated, relative percentage deviation (RPD) is obtained by

$$RPD = \frac{A\lg_{sol} - Min_{sol}}{Min_{sol}}, \tag{6}$$

where $A\lg_{sol}$ is the $\sum(E+T)$ obtained for a given algorithm and instance.

For verifying the statistical validity of the results shown in Table 3, in this subsection, the analysis of variance (ANOVA) is used in which the different algorithms are factors and the response variable are RPDs. For the algorithms, single factor ANOVA results are shown in Table 4.

Table 4. ANOVA results for the method

Source	Df	Sum of square	Mean square	F	$F_{0.01,n1,n2}$	P-value
Method	1	0.073062	0.073062	162.0662	$6.743^{***}=F_{1,238}$	1.16E-28
Error	238	0.107295	0.000451			
Total	239	0.180357				

These results indicate that there is a method that is different in mean response. The results show that there is a significant difference between the performances of the algorithms. This chart is a Least Significant Difference (LSD) chart, which not only shows the difference in averages, but also shows the standard deviation of the results. As you can see, in addition to the lower mean of the proposed algorithm, there is a lower standard deviation for the DPSO algorithm compared to the NS algorithm in the results.

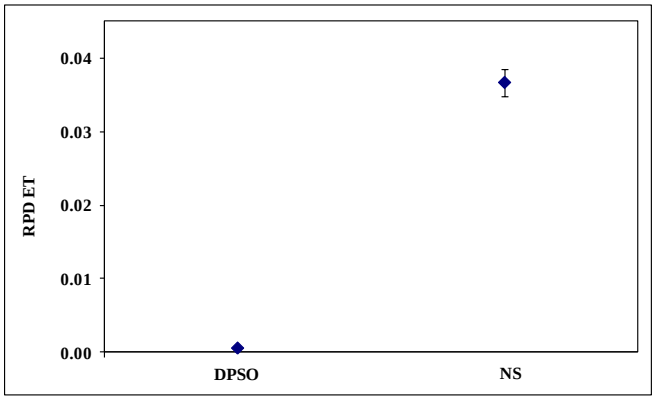


Figure 6. Plot of $\overline{RPD}$ for the type of algorithm factor

6.4. Analysis of controlled factors

Table 5 shows the interaction between the method and the number of jobs. As it can be seen in this table, the effect of a number of jobs, main and interaction effects are significant. Also, as shown in the $\overline{RPD}$ plot of interaction in Figure 7, in all cases, the DPSO algorithm works better than NS.

Table 5. ANOVA results for method and the number of jobs

Source	Df	Sum of square	Mean square	F	F0.01,n1,n2	P-value
A: Method	1	0.073062	0.073062	175.7286	$6.74^{***}=F1,232$	3.11E-30
B: Number of jobs	3	0.005574	0.001858	4.468871	$3.87^{***}=F3,232$	0.004506
A×B	3	0.083899	0.027966	67.2641	$3.87^{***}=F3,232$	2.48E-31
Error	232	0.096458	0.000416			
Total	239	0.180357				

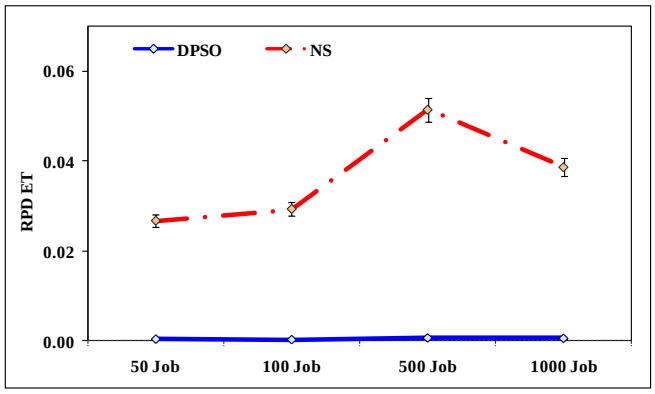


Figure 7. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and the number of jobs

As shown in Table 6, all sources of variations, including main effects and interaction between the method and the number of machines, are also significant. The interaction plot, as shown in Figure 8, in general, demonstrates that with increasing the number of machines, the performance of the algorithms is declined.

Table 6. ANOVA results for method and the number of machines

Source	Df	Sum of square	Mean square	F	F0.01,n1,n2	P-value
A: method	1	0.073062	0.073062	258.325	6.74***=F1,234	1.15E-39
D: number of machines	2	0.020376	0.010188	36.0219	4.69***=F2,234	2.30E-14
A×D	2	0.020736	0.010368	36.6581	4.69***=F2,234	1.42E-14
Error	234	0.066183	0.000283			
Total	239	0.180357				

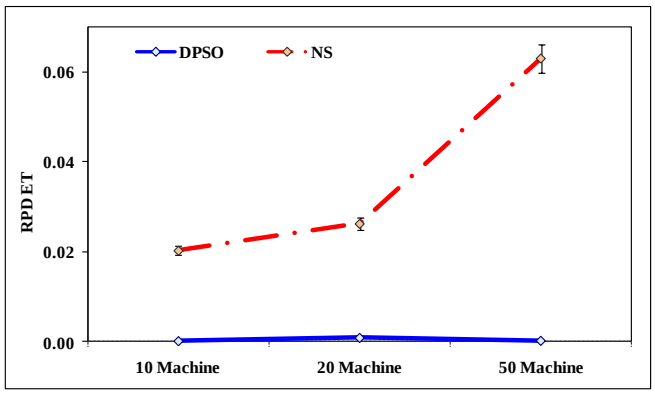


Figure 8. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and magnitude of machines

6.5. Discussion

A just-in-time parallel job scheduling consists of several machines in parallel in where a job may be proceed on more than one machine at the same time. In this paper, it is assumed that the early and tardy deliveries of a job with respect to its due date are penalized. Given the needs of the market, the existence of some industries that require JIT, and the need for research on parallel job scheduling problem is felt. Furthermore, in most of the classic scheduling research, it is assumed that a job is needed one machine in its processing, but as real problems become more complex such as the textile industry (Serafini 1996) and berth allocation problem (Lee and Cai 1999) in which such parallelism is required, the classic approach is not effective enough in them. In order to maintain competition in such markets, manufacturers need to keep their customers satisfied by meeting their demands on time. For such complex problem, the aim is to give a combinatorial characterization of the properties of the system in which these schedules are allowed. In this paper, at first, we presented the problem by analyzing some simple cases. Then, we used a graph model approach for a multi-machine task scheduling model with prespecified machine allocation. Our main decisions were:

- Dividing jobs into independent sets: in this phase, we propose a heuristic in which we use a graph coloring concept, and
- Scheduling the independent sets in which job must be scheduled to minimize bi objectives.

Despite the many applications that exist for the investigated problem and the acceptable results were obtained here, there were limitations in this research that by solving them, it is expected that the problem will be closer to the real world. For example, as a completely new trend in production planning problems, changes in manufacturing factories and the interest of smaller companies to enter the global arena have created a new challenge in structuring efficient operations management in geographically distributed production networks. As a result, these small organizations merge and form a distributed production network to overcome the following problems:

- Response time to large stochastic disturbances in the market is not satisfactory due to the slowness of traditional production systems,
- Insufficient, inaccurate, and unreliable information due to the geographical size of the customer distribution has led decision-makers to make decisions based on conjecture or very little information, and
- The organizational structure of traditional systems is predetermined, which makes it inflexible to the emergence of new markets and changes.

In such multi-factory mode, factories are distributed in different geographical locations to save transfer costs and time. Also, they provide a better level of service to customers by placing the factory close to the customer (Behnamian and Fatemi Ghomi 2015). In the mentioned network, the decision is delegated to lower levels of the organization hierarchy and resolved locally in different system institutions. The solutions are then coordinated together under a global objective function. Multi-factory production takes place in multi-factories, which may be geographically distributed in different places to satisfy the demand and adapt to the globalization trend. Not paying attention to more recent objective functions such as considering environmental issues and green scheduling as well as paying attention to social criteria is another limitation of this research.

7. Conclusions and future works

This paper presents an algorithm for a parallel job scheduling problem with the sum of the earliness and tardiness objective function. The algorithm has two-phase. The first phase applied a heuristic algorithm, which concentrates on the coloring of the associated scheduling system. In fact, in this phase, we make use of the graph coloring concept to divide the jobs into independent sets according to their machine requirements. Then, in order to sequence the job sets obtained from phase 1, in the second phase, we developed a new discrete version of the particle swarm optimization algorithm that is hybridized with the genetic algorithm. Our experimental results indicated that the proposed algorithm could find better solutions compared to the competing algorithm, especially for large-sized instances. Furthermore, the analysis of controlled factors showed that the effect of a number of jobs and machines are significant, so that in the both cases, the absolute superiority is with the proposed algorithm. The future work is to change the objectives that we used in this paper and change the assumptions of this problem. For example, this study assumes that all data is deterministic. There are different uncertainties in the real world's parameters, and considering them will make the problem more attractive. Furthermore, due to the inherent complexity of the problem, it is suggested that heuristic methods be used to solve the problem in future studies.

References

Barbosa, J.G. & Moreira, B. (2011). Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters, Parallel Computing, 37(8), 428–438.

Behnamian, J. (2016). Graph coloring-based algorithm to parallel job scheduling on parallel factories. International Journal of Computer Integrated Manufacturing, 29(6), 622-635.

Behnamian, J. (2020). Parallel Jobs Scheduling with a Specific Due Date: Asemi-definite Relaxation-based Algorithm, Journal of Optimization in Industrial Engineering, 13(2), 199-210.

Behnamian, J., Fatemi Ghomi, S.M.T. (2015), Minimizing cost-related objective in synchronous scheduling of parallel factories in virtual production network, Applied soft computing, 29, 221-232.

Bischof, S. & Mayr, E.W. (2001). On-line scheduling of parallel job with runtime restrictions. Theoretical Computer Science, 268, 1, 67–90.

Brelsford, D., Chochia, G., Falk, N., Marthi, K., Sure, R., Bobroff, N., ... & Seelam, S. (2012, May). Partitioned parallel job scheduling for extreme scale computing. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 157-177). Springer, Berlin, Heidelberg.

Dell'Olmo, P., & Gentili, M. (2006). Graph models for scheduling systems with machine saturation property. Mathematical Methods of Operations Research, 63(2), 329-340.

Dutot, P. F., Mounié, G., & Trystram, D. (2004). Scheduling parallel tasks: Approximation algorithms.

Ebrahimi Moghaddam, M., & Bonyadi, M. R. (2012). An immune-based genetic algorithm with reduced search space coding for multiprocessor task scheduling problem. International Journal of Parallel Programming, 40(2), 225-257.

Ebrahimi Zade, A., Fakhrzad, M., Hasaninezhad, M. (2016). A heuristic algorithm for solving single machine scheduling problem with periodic maintenance. Journal of System Management, 2(4), 1-12.

Garey, M. R. (1979). computers and intractqbility. A Guide to the Theory of NP-Completeness.

Guan, Y. Xiao, W-Q. Cheung, R. K. & Li, C-L. (2002). A multiprocessor task scheduling model for berth allocation: Heuristic and worst-case analysis, Operations Research Letters, 30(5), 343-350.

Guo, S., & Kang, L. (2010). Online scheduling of malleable parallel jobs with setup times on two identical machines. European Journal of Operational Research, 206(3), 555-561.

Hao, Y., Wang, L., Zheng, M. (2016). An adaptive algorithm for scheduling parallel job in meteorological Cloud, Knowledge-Based Systems, 98, 226-240.

Hao, Y., Xia, M., Wen, N., Hou, R., Deng, H., Wang, L., Wang, Q. (2017). Parallel task scheduling under multi-Clouds, KSII Transactions on Internet and Information Systems, 11, 1, 39-60.

Hoogeveen, J. A., van de Velde, S. L., & Veltman, B. (1994). Complexity of scheduling multiprocessor tasks with prespecified processor allocations. Discrete Applied Mathematics, 55(3), 259-272.

Jansen, K., & Porkolab, L. (2002). Linear-time approximation schemes for scheduling malleable parallel tasks. Algorithmica, 32(3), 507-520.

Jansen, K. & Porkolab, L. (2003). Computing optimal preemptive schedules for parallel tasks: Linear programming approaches. Mathematical Programming, 95, 3, 617–630.

Jansen, K., Trystram, D. (2016). Scheduling parallel job on heterogeneous platforms, Electronic Notes in Discrete Mathematics, 55, 9-12.

K. Jansen (2002). Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme. In R. Möhring and R. Raman, editors, Proceedings of ESA 2002. LNCS, 2461, 562–574, Springer, Berlin.

Lee, C. Y., & Cai, X. I. A. O. Q. I. A. N. G. (1999). Scheduling one and two-processor tasks on two parallel processors. IIE transactions, 31(5), 445-455.

Leung, K.K. Yung N.H.C. & Cheung, P.Y.S. (2002). Novel neighborhood search for multiprocessor scheduling with pipelining, Journal of Parallel and Distributed Computing 62, 85–110.

Li, K., & Pan, Y. (2000). Probabilistic analysis of scheduling precedence constrained parallel tasks on multicomputers with contiguous processor allocation. IEEE transactions on computers, 49(10), 1021-1030.

Li, K. (2018). Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment, Future Generation Computer Systems, 82, 591-605.

Liu, X., Zha, Y., Yin, Q., Peng, Y., Qin, L. (2015). Scheduling parallel job with tentative runs and consolidation in the cloud, Journal of Systems and Software, 104, 141-151.

Niu, Q. Jiao, B. Gu, X. (2008). Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time, Applied Mathematics and Computation, 205(1), 148-158.

Parida, S., Nayak, S. C., Priyadarshi, P., Pattnaik, P. K., & Ray, G. (2018). Petri net: Design and analysis of parallel task scheduling algorithm. In Advances in Electronics, Communication and Computing (pp. 765-776). Springer, Singapore.

Pinedo, M. L. (2012). Scheduling (Vol. 29). New York: Springer.

Serafini, P. (1996). Scheduling jobs on several machines with the job splitting property. Operations Research, 44(4), 617-628.

Srinivasan, S., Subramani, V., Kettimuthu, R., Holenarsipur, P., & Sadayappan, P. (2002, December). Effective selection of partition sizes for moldable scheduling of parallel jobs. In International Conference on High-Performance Computing (pp. 174-183). Springer, Berlin, Heidelberg.

Su, L. H. (2009). Minimizing earliness and tardiness subject to total completion time in an identical parallel machine system. Computers & operations research, 36(2), 461-471.

Sun, H. Hsu, W-J. & Cao, Y. (2014). Competitive online adaptive scheduling for sets of parallel job with fairness and efficiency, Journal of Parallel and Distributed Computing, 74(3), 2180–2192.

Yazdani, M., Jolai, F. (2015). A genetic algorithm with modified crossover operator for a two-agent scheduling problem. Journal of System Management, 1(3), 1-13.

Ye, D., & Zhang, G. (2003). On-line scheduling of parallel jobs with dependencies on 2-dimensional meshes. In in proceedings of the 14th annual international symposium on algorithms and computation (ISAAC).

Ye, D. & Zhang, G. (2007). On-line scheduling of parallel job in a list, Journal of scheduling, 10, 407–413.

Zhang, L., Zhou, L., & Salah, A. (2020). Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. Information Sciences, 531, 31-46.

Zheng, B., Pan, L., & Liu, S. (2021). Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments. Journal of Systems and Software, 176, 110934.

Zong, Z., Manzanares, A., Ruan, X., & Qin, X. (2010). EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. IEEE Transactions on Computers, 60(3), 360-374.

This article can be cited: Behnamian, J., (2022). Just-in-time parallel jobs scheduling: A novel algorithm. Journal of Industrial Engineering and Management Studies, Vol 9, No.2, pp. 1-12.