Journal of Optimization in Industrial Engineering
Vol.13, Issue 2, Summer & Autumn 2020, 199- 210
DOI: 10.22094/JOIE.2017.599.1385

Research Article

Parallel Jobs Scheduling with a Specific Due Date: a Semi-Definite Relaxation-Based Algorithm

Javad Behnamian*

Department of Industrial Engineering, Faculty of Engineering, Bu-Ali Sina University, Hamedan, Iran
Received 13 March 2016; Revised 12 October 2016; Accepted 20 November 2017

Abstract

One of the assumptions made in classical scheduling theory is that a job is always executed by one machine at a time. Since this assumption is not always true, in this paper, a relatively new concept of job scheduling, namely parallel jobs, is considered, in which a job can be executed by one or more machine at the same time. While the analytical conditions can be easily stated for some simple models, a graph model approach is required when conflicts of processor usage are present. The main decisions and solving steps are: (*i*) converting the scheduling problem to graph model; (*ii*) dividing jobs into independent sets: in this phase, we propose a semi-definite relaxation-based algorithm in which we use graph coloring concept; (*iii*) sequencing the independent sets as a single-machine scheduling in which jobs in such a system are job sets formed by using a semi-definite relaxation solution and determining the problem as a schedule that minimizes the sum of the tardiness of jobs. In this regard, after grouping the jobs by a semi-definite programming relaxation-based algorithm, we used the rounding algorithm for graph coloring. We also proposed a variable neighborhood search algorithm for sequencing the obtained job sets in order to minimize the sum of the tardiness. Experimental results show that this methodology is interesting by obtaining good results.

Keywords: Parallel jobs scheduling; Semidefinite relaxation; Tardiness; Graph coloring.

1. Introduction

The scheduling problem frequently arises in the manufacturing systems. This problem is the allocation of limited resources to perform a set of activities in a period of time. In this paper, we study the scheduling model of parallel jobs. It is assumed that a parallel job may use more than one machine at the same time. This relaxation departs from one of the classic scheduling assumptions. Parallel job scheduling has recently gained considerable attention (Sun et al., 2014). Many applications of parallelism are reported. This parallelism is expressed in a certain programming environment, and finally a parallel application is executed on some computing platform. In many applications, a network topology is considered, in which jobs can only be executed on particular machines. Only connected machines may execute a job together simultaneously for its processing time. Parallel machines with a specific network topology can be viewed as a graph where each node represents a machine and each edge represents the communication link between the two nodes. Another area of application for parallel job model is bandwidth and storage management. In fact, the study of computer architectures with parallel machines has prompted the design and analysis of good algorithms for parallel jobs scheduling.

In order to clarify our specific goal in this paper, let us consider the following well-known problem. We are given m identical parallel machines and a set of n independent, parallel jobs $j = 1 \ldots n$. All jobs have an equal positive integer processing time, which we also call its *length*. Let $T = \{T_1, T_2 \ldots T_n\}$ be a set of n independent tasks and $P = \{P_1, P_2 \ldots P_m\}$ a set of m machines. During each time instant, each machine can be used by a single task at most. A schedule for each task is an allocation of one or more time intervals to one or more machines. Job j simultaneously requires $m_j \leq m$ machines at each point in time for its processing. Positive integer m_j is also known as the *width* of job j. Note that we assume that m_j is part of the input. Any machine can process at most one job at a time. The objective is to find a feasible schedule; that is, the sum of the tardiness is to be minimized. We consider the non-preemptive variant of this problem in which tasks are independent.

Our objective of this study is to give a combinatorial characterization of the properties of the system in which these schedules are allowed. At first, we present the problem by analyzing some simple cases. Then, we will use a graph model approach for a multi-machine task scheduling model with pre-specified machine allocation. The main decisions and solving steps are as follows, respectively.

(i) Converting the scheduling problem to graph model,
(ii) Dividing jobs into independent sets: in this phase, we propose a semi-definite relaxation in which we use graph coloring concept, and
(iii) Sequencing the independent sets as a single-machine scheduling in which jobs in such a system are job sets formed by using a semi-definite relaxation solution and determining the problem as a schedule that minimizes the sum of the tardiness of jobs.

*Corresponding author Email address: Behnamian@Basu.ac.ir

Since problem 1 || $\sum T_j$ is NP-hard (Pinedo, 2008), so the use of a metaheuristic algorithm can be appropriate, and in this phase, for (*ii*), we propose an efficient metaheuristic algorithm.
The remainder of this paper is organized as follows: Section 2 gives the brief literature review of parallel jobs scheduling. Section 3 introduces the proposed graph-based algorithm for dividing the jobs into independent sets. Section 4 presents a VNS-based algorithm used in the sequencing the job sets. Section 5 presents experimental design, and finally, Section 6 states our conclusions and further research studies.

2. Literature Review

In the last two decades, many results have appeared in the parallel machines scheduling literature. More recent results of this model include, among others, Emmons (1987), Kubiak et al. (1990), Herrmann and Lee (1993), Chen (1996), Almeida and Centeno (1998), Chen and Powell (1999), Birman and Mosheiov (2004), Esteve et al. (2006), and Bülbül et al. (2007). A last survey of these problems was published by Gordon et al. (2002), Lauff and Werner (2004), and Hoogeveen (2005).
Due to a different application of parallel jobs in various contexts, recently, the model of parallel job scheduling has been also studied extensively, see e.g., (Drozdowski, 1996; Du and Leung, 1989; Feldmann et al., 1998; Ludwig and Tiwari, 1994, Naroska and Schwiegelshohn, 2002), and (Amoura et al., 1997; Blazewicz, et al., 1986; Chen and Miranda, 1999; Du and Leung, 1989; Feldmann et al., 1994; Jansen and Porkolab, 1999; Ludwig and Tiwari, 1994; Mu'alem and Feitelson, 1999; Turek et al., 1992).
In this regard, for the online scheduling of parallel jobs on *m* identical machines to minimize the makespan where jobs arrive over time, Chen and Vestjens (1997) proved a lower bound 1.347 on the case without preemption, and Johannes (2006) showed that 6/5 is a lower bound on the case where preemption is allowed, and he proved that a list scheduling algorithm has a competitive ratio of 2 for both cases. An online algorithm with competitive ratio 2 - 1/*m* was raised by Naroska and Schwiegelshohn (2002) for the model where the processing times of jobs are not known until they are finished. This algorithm is optimal since Shmoys et al. (1995) showed a lower bound 2 - 1/*m* on the competitive ratio of any online algorithm.
These papers are just small samples of work in this area. Table1 contains, in a chronological order, other papers.

Table1
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, minimize 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, multicomputers 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 jobs with dependencies
2004	Dutot et al.	approximation algorithms
2005	Frachtenberg et al.	adaptive parallel job scheduling with flexible co-scheduling
2006	Johannes	parallel jobs with release dates, approximation algorithms for both the preemptive and the non-preemptive
2007	Ye and Zhang	7-competitive online algorithm, improving the previous upper bound
2010	Guo and Kang	malleable parallel jobs, 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, a new coding scheme to reduce the search space
2012	Damodaran and Vélez-Gallego	parallel batch processing machines with unequal job ready times, simulated annealing algorithm, makespan
2013	Brelsford et al.	parallel job scheduling for extreme scale computing, hybrid centralized and distributed approach, improving the scaling behavior of scheduling time
2013	Cheng et al.	Parallel batching machines with arbitrary job sizes, improved ant colony optimization
2014	Sun et al.	online adaptive scheduling for multiple sets of parallel jobs, two-level algorithm scenario with a feedback-driven adaptive scheduler, minimizing the scheduling total response time and makespan
2015	Bougeret et al.	5/2-approximation to multiple cluster scheduling problem corresponds to minimizing the makespan and 2-approximation to a restricted problem

According to the reviewed literature, the main novelty of this paper is the scheduling of parallel jobs using a novel hybrid algorithm composed of exact and heuristic algorithms. In this regard, after grouping the jobs by a semi-definite programming relaxation algorithm, we used the rounding algorithm for graph coloring. We also proposed a variable neighborhood search algorithm for sequencing the obtained job sets in order to minimize the sum of the tardiness. To the best of our knowledge, this study is the first research that combined graph model, semi-definite programming, rounding algorithm, and metaheuristic to scheduling problems.

3. Grouping the Jobs

3.1. Parallel jobs scheduling and related graph

We now define the correspondence between scheduling systems and associated graphs. We focus, in particular, on the scheduling system in which job does not allow preemption and all tasks have a unit processing time, and the processing of each task requires the simultaneous

availability of a set of machines $P(T_j) \subseteq P$ during p_j time units. We say that tasks T_i and T_j are in conflict if $P(T_i) \cap P(T_j) \neq \emptyset$. A schedule is a set of values of starting times $\{t_j, j = 1, 2, \ldots, n\}$ to be associated with each task T_j, such that no two tasks use the same machine in the same interval time. Given a multi-machine task scheduling system, we can associate it with the so-called constraint graph as shown in the example in Figure 1: there is one vertex for each task and one edge between two vertices if and only if the corresponding tasks are in conflict. Note that if a processing time $p_j \in Z^+$ is associated with each task T_j, the resulting constraint graph has a weight $w_j = p_j$ for each vertex j associated with task T_j.

On the contrary, given a graph $G = (V(G), E(G))$, where $V(G)$ denotes the node set and $E(G)$ denotes the edge set, a multi-machine task scheduling system can be associated with it in the following way: the set of tasks corresponds to the set of vertices, the set of machines corresponds to the cliques of a minimum edge clique cover *(ECC)*, and set P_i of machines associated with task T_i corresponds to the cliques of the minimum ECC covering vertex v_i. We recall that clique C of G is a complete subgraph of G, and a minimum ECC of G_i is a collection of cliques of minimum size $\theta(G)$ that covers all edges of G.

In this example, as it can be seen, a minimum ECC is composed of the five cliques corresponding to the five edges of the graph. We can associate with this graph the scheduling system with five machines $\{P_1, P_2, P_3, P_4, P_5\}$ (corresponding to the cliques of the minimum ECC) and five tasks $\{T_1, T_2, T_3, T_4, T_5\}$ (corresponding to the five vertices), such that $P(T_1) = \{P_1, P_2\}$, $P(T_2) = \{P_2, P_3\}$, $P(T_3) = \{P_1, P_4\}$, $P(T_4) = \{P_4\}$, and $P(T_5) = \{P_3, P_5\}$.

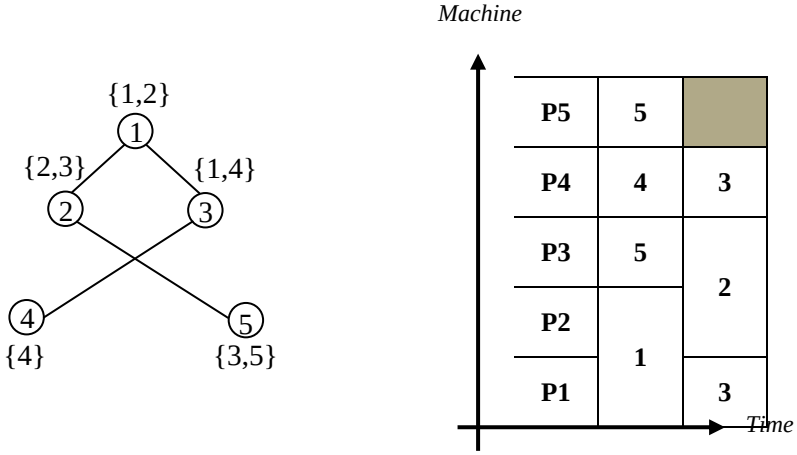

Fig. 1. A scheduling system and the associated constraint graph

An independent set of G is a subset $S \subseteq V(G)$ of vertices, no two of which are adjacent. A *coloring* of the vertices of G is a partition $\{S_j\}$ of the vertices of G, such that each set S_j is an independent set. In the particular case of unit processing times, a minimum coloring on the graph corresponds to a schedule of minimum length where the set of tasks executed in the same time instant is associated with vertices in the same color class (Dell'Olmo and Gentili 2006). We are interested in classes of graphs for which such a coloring implies a scheduling without idle times. We now state more formally this correspondence between graph and scheduling systems.

3.2. Graph coloring

In graph theory, a k-coloring for G is a function $f: V \rightarrow [k]$, such that $f(u) \neq f(v)$ for all $(u, v) \in E$. Clearly, finding a k-coloring of a graph is equivalent to the problem of partitioning the vertices into k or fewer independent sets. Note that for a given k-colorable graph $G = (V, E)$, finding a k-coloring for G is NP-hard for $k \geq 3$. In this paper, for graph coloring, a semi-definite programming relaxation approach is used.

3.3. Semi-definite programming relaxation

In our problem, for a given k-coloring of the graph, the semi-definite programming (SDP) relaxation assigns a unit vector to each vertex of a graph $G = (V,E)$ to color it, such that certain separation properties are satisfied for vectors corresponding to each pair of adjacent vertices. Vector $v_i \in \Re^n$ assigned to vortex $i \in V$ is the vector corresponding to the color class of i. Recall that the dot product between the two vectors increases as they get closer to each other. Therefore, to maximize the distance between the edges, we want to minimize quantity $max_{(i,j) \in E} v_i.v_j$.

It is known that for any integer $k < n + 1$, there exist k unit vectors in R^n, such that their pairwise inner products are $-1/(k-1)$. Moreover, given that k-coloring of graph $G=(V,E)$, we can assign one of these k vectors to each color class, such that $\langle v_i, v_j \rangle = t = -1/(k\text{-}1)$, $\forall (i, j) \in E$ where v_i is the vector assigned to vertex $i \in V$ (see Lemma 4.1 in Karger et al. (1998) for details). The relaxation considered by Karger et al. (1998) is as in the following SDP.

$$
\begin{aligned}
&(P1) \quad \textit{Minimize} \;\; t \\
&S.t: \quad \langle v_i, v_j \rangle \le t \quad \forall (i,j) \in E, \\
&\qquad \langle v_i, v_i \rangle = 1 \quad \forall i \in V, \\
&\qquad v_i \in R^n, \qquad \forall i \in V
\end{aligned} \tag{1}
$$

In fact, this problem is an SDP and the optimum value of k is called the vector chromatic number of G. To see this, consider matrix $W=(w_{ij}) \in S^{(n+1)(n+1)}$ with $w_{ij}=v_i.v_j$ and $w_{i,n+1}=w_{n+1,i}=0$ for all $i, j \in \{1, .. . ,n\}$, $w_{n+1,n+1}=t$. Then, problem (P1) has the linear constraints and linear objective function of the elements of W, which is positive semi-definite.

Lemma 1: Let t^* be the optimal value of (P1). If G is k-colorable, then $t^* \le -1/(k-1)$.

Lemma 2: Let t^* be the optimal value of (P1). If G contains a k-clique, then $t^* \ge -1/(k-1)$.

Note that if $t^* = -1/(k-1)$, function $\theta(G) = 1- (1/t^*) = k^*$ (this function is known as the Lovasz theta function). Problem (P1) is essentially equivalent to the below semi-definite programming:

$$
\begin{aligned}
&(P2) \quad \textit{Minimize} \;\; t \\
&S.t: \quad A_{ij} \bullet X \le t \quad \forall (i,j) \in E, \\
&\qquad A_{ii} \bullet X = 1 \quad \forall i \in V, \\
&\qquad X \in S_+^n, \qquad \forall i \in V
\end{aligned} \tag{2}
$$

where $A_{ii} = e_i e_i^T$, $A_{ij} = e_i e_j^T$, and e_i is the unit vector with its ith element equal to 1 (a unit vector is a vector of length 1, sometimes also called a direction vector). Clearly, a solution to (P1) can be obtained via Cholesky decomposition of any solution to (P2).

Suppose that we found a feasible solution to the SDP. How do we convert it to a valid coloring? Any rounding scheme will probably miscolor some edges. We use rounding that gives us a large independent set. We aim for an algorithm that colors the graph almost properly. This is exactly the motivation of the semi-colorings of a graph presented in Karger et al. (1998), and we will describe it in the following sub-section.

3.4. Rounding algorithm

A k-semicoloring of graph $G = (V, E)$ is an assignment of k colors to at least half of its vertices, such that no two adjacent vertices share the same color. It is clear that at least $n/2$ vertices are properly colored in any k-semicoloring of graph (Dell'Olmo and Gentili 2006). Therefore, if we can semicolor a graph with k colors, we can color graph with $k\ 124343log\ n$ colors where $n = |V|$ (Li and Liu, 2008). Here, we can semicolor the remaining half vertices of the graph with k new colors at each iteration. We used the idea of Li and Liu (2008) as a rounding algorithm.

Algorithm KMS (proposed by Karger, Motwani and Sudan (1998))

1: Solve the semi-definite programming relaxation and obtain a vector 3-coloring $\{v_i\}_{i=1}^n$.

2: Let $c = (2(k-2)/k \cdot \ln\Delta)^{1/2}$ for sufficiently large Δ. Set $j = 1$.

3: Create a random vector r_j .

4: Denote $R_j = \{i \in V \mid \langle v_i, r_j \rangle \ge c\}$.

5: Assign color j to R_j and remove the vertices in R_j. Set $j = j + 1$.

6: Repeat Steps 3 to 5 until all vertices are colored.

4. Sequencing the Job Sets

After classification according to the fact that jobs need machines in the independent set, we will have sets of jobs that must be sequenced at group of machine. In this phase, we have a single machine scheduling problem whose job sets and group of machine in parallel can be considered as jobs and single machine, respectively. In a machine scheduling, a late delivery implies a penalty in the form of loss of good will and the magnitude of the penalty depends on the importance of the order or the client and the tardiness of the delivery. One of the most important objectives in the scheduling is to minimize the sum of the tardiness (Lushchakova, 2012). The *tardiness* of job j is defined as follows:

$$T_j = \max (C_j - d_j , 0). \tag{3}$$

In this study, we consider the scheduling problem where the sum of tardiness of jobs must be minimized. For this, in the following sub-section, we will present an advanced local search, namely variable neighborhood search algorithm (VNS).

4.1. VNS-based metaheuristic

For sequencing the independent job sets, we propose a VNS-based metaheuristic algorithm. The construction of VNS-based metaheuristic is motivated by the need to achieve a good trade-off between the global exploration and local exploitation during the search. Let us now discuss the aspects of the proposed algorithm.

4.2. Encoding scheme

The proposed representation for our proposed algorithm to solve scheduling is based on coding all job sets as genes in a 1-by-(*n'*) string where *n'* is a number of job 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 2 shows an example for representation. In this example, there are five job sets with order 5→1→2→4→3.

5	1	2	4	3

Fig. 2. Solution representation

4.3. Initial solution

Any available method would be sufficient to generate a feasible solution to our algorithm. For this algorithm, we use a random initial solution.

4.4. Variable neighborhood search

Variable neighborhood algorithm is based on the principle of systematic change of the neighborhood during the search. The motivations behind the concept of VNSs are: (*i*) A local minimum in one neighborhood structure is not necessarily locally minimal with respect to another neighborhood structure; (*ii*) A global optimum is locally optimal with respect to all neighborhood structures.

4.5. Our proposed algorithm

In this algorithm, every time a neighborhood is selected, a random procedure is called. This procedure selects a random solution from the selected neighborhood structure (Rocha et al., 2007). Therefore, neighborhoods $N_1(S)$ and $N_2(S)$ are created in the following manner, one for each l, respectively:

Neighborhood structure 1

1: Choose randomly two different job sets i_1 and i_2
2: Swap job sets i_1 and i_2.

Neighborhood structure 2

1: Choose randomly one job set i_1;
2: Choose randomly a valid position *pos*;
3: Transfer job set i_1 to a position *pos*.

In this algorithm, if no improvement is made in first neighborhood after an iteration, then the other one (l = 2) is applied, and every time a new solution is found, the first and fastest local search is used (l = 1).
The steps of the proposed algorithm are as follows:

Algorithm 2: Basic VNS structure

1: Find an initial solution S^* randomly;
2: $l \leftarrow 1$;
3: **for** *iterations* ← 1 to a maximum number of *iterations* **do**
4: $S \leftarrow S^*$;
5: Shake procedure: find a random solution $S' \in N_l(S)$;
6: Perform a local search on $N_l(S')$ to find a solution S'';
7: **if** $f(S'') \leq f(S^*)$ **then**
8: $S^* \leftarrow S''$;
9: $l \leftarrow 1$;
10: **end if**
11: $l \leftarrow l+1$;
12: **end for**

The algorithm proposed here has three specifications:

(*i*) *Generating the initial solution via a random initial solution*: The method starts with generating initial chromosomes randomly.

(*ii*) *Intensification phase using VNS local search*. In the proposed algorithm, varying the neighborhood structure during the search process could facilitate the avoidance of traps and enlarge the search scope. VNS works by performing movements that upgrade the solutions. So, the solution that has the minimal *objective value* is selected as a move.

(*iii*) *Diversification phase shaking function in VNS*. In VNS algorithm, the shaking procedure in the basic VNS approach is a diversification factor, whilst the local search will intensify the search to lead to its converging to a local optimum.

5. Experimental Design

To show the performances of the metaheuristic algorithms suggested in this paper, a series of computational experiments were done on randomly generated test problems, and the results are reported in this section.
We have thoroughly reviewed the literature, and according to the literature review, we conclude that the most related and best algorithms are simulated annealing (SA) proposed by Damodaran and Vélez-Gallego (2012) and improved ant colony optimization (IACO) proposed by Cheng et al. (2013). So, after adapting these algorithms to our problem assumptions, we compare the proposed algorithm (namely, HMH) with them.
The algorithms were implemented in MATLAB 7 and run on an Intel Pentium IV dual core 2.00 GHz PC at 1022 MB RAM under a Microsoft Windows Vista environment.

5.1. Data generation and settings

The lack of other works dealing with this problem forces us to use artificial instances to test our algorithms. To analyze the algorithms and models developed for this problem, several classes of instances are defined. In each class, there is a change in one of the inputs. The problem data can be characterized by three factors. These levels are shown in Table 2.

Table 2
Factor levels

Factor	Level(s)			
Number of jobs (n)	10	15	20	25
Number of machines (m)	2	3	5	10
Width of jobs (w)	2	(1,3)		

For the test on our proposed algorithms, 320 problems were randomly generated, i.e., 10 problems for each of 32 combinations. 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 the processing time. The stopping criterion for VNS-based metaheuristic is also set to a number of iterations to 10 repetitions.

5.2. Experimental results

In order to evaluate the efficacy and performance of the algorithms proposed in this paper, 320 instances are solved.
Table 3 represents the computational results. In this table, each instance is solved using 10 different seeds, and the average cases are considered. The results demonstrate that there is a clear, statistically significant difference between the performances of the algorithms. The means plot and least significant difference (LSD) intervals (at 95% confidence level) for three algorithms are shown in Figure 3.

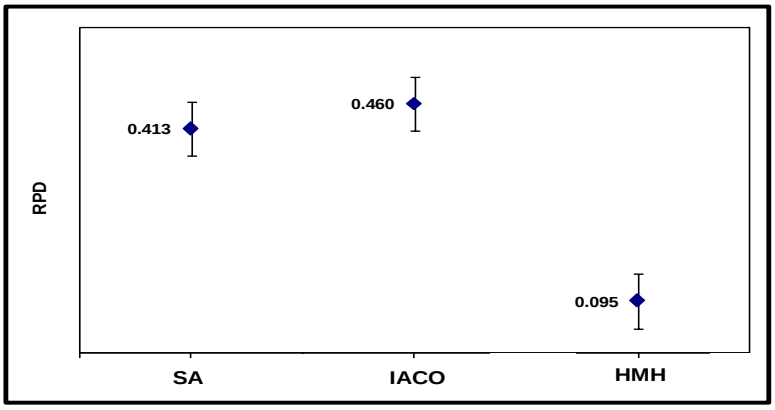

Fig. 3. Plot of $\overline{RPD}$ for the type of algorithm factor

Table 3
Computational results (sum of the tardiness)

	Instance			Algorithm		
No.	***n***	***m***	***w***	**SA**	**IACO**	**HMH**
1	10	2	2	0.759	0.147	0.000
2	10	3	2	0.176	0.000	0.176
3	10	5	2	1.400	0.900	0.000
4	10	10	2	0.243	0.526	0.000
5	10	2	(1,3)	0.000	1.182	0.469
6	10	3	(1,3)	0.505	0.437	0.000
7	10	5	(1,3)	0.786	0.000	0.044
8	10	10	(1,3)	0.064	0.418	0.000
Average				**0.492**	**0.451**	**0.086**
9	15	2	2	0.424	0.238	0.000
10	15	3	2	0.818	0.000	0.538
11	15	5	2	0.248	0.404	0.000
12	15	10	2	0.000	1.154	0.243
13	15	2	(1,3)	0.000	1.091	0.189
14	15	3	(1,3)	0.000	0.350	0.154
15	15	5	(1,3)	0.505	0.574	0.000

16	15	10	(1,3)	0.238	0.486	0.000
Average				**0.279**	**0.537**	**0.140**
17	20	2	2	0.300	0.950	0.000
18	20	3	2	0.529	0.300	0.000
19	20	5	2	0.118	0.000	0.267
20	20	10	2	0.130	0.130	0.000
21	20	2	(1,3)	0.353	0.000	0.086
22	20	3	(1,3)	1.200	1.100	0.000
23	20	5	(1,3)	0.050	0.100	0.000
24	20	10	(1,3)	0.885	0.885	0.000
Average				**0.446**	**0.433**	**0.044**
25	25	2	2	0.231	0.000	0.598
26	25	3	2	1.065	0.835	0.000
27	25	5	2	0.661	0.228	0.000
28	25	10	2	0.300	0.300	0.000
29	25	2	(1,3)	0.059	0.059	0.000
30	25	3	(1,3)	0.417	0.000	0.282
31	25	5	(1,3)	0.083	1.094	0.000
32	25	10	(1,3)	0.682	0.835	0.000
Average				**0.437**	**0.419**	**0.110**

To compare the algorithms, in this table, the relative percentage deviation (RPD) is used that is obtained by formula (4):

$$RPD = \frac{\mathrm{Alg}_{sol} - Min_{sol}}{Min_{sol}} \times 100 \qquad (4)$$

where Min_{sol} is the best solutions obtained for each instance, and Alg_{sol} is an objective function obtained by a given algorithm.

5.3. Analysis of controlled factors

5.3.1. Analysis of problem size factor (number of jobs)

In order to see the effects of the number of jobs on three algorithms, a two-way ANOVA is applied. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and number of jobs is shown in Figure 4. As it can be seen, in all cases, the HMH works better than others.

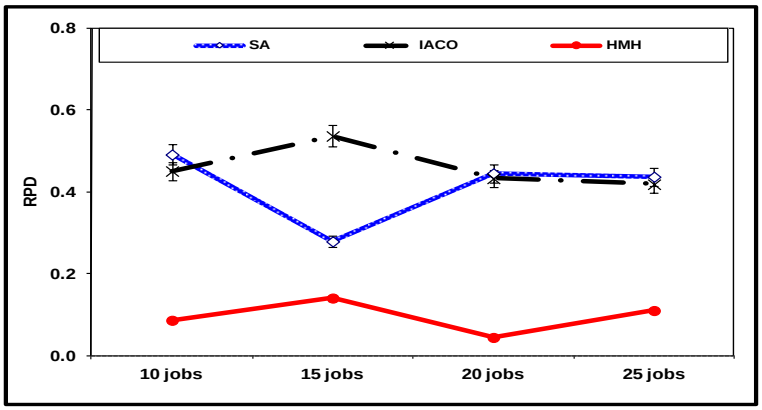

Fig. 4. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and the number of jobs

5.3.2. Analysis of number of parallel machine factor

Another two-way ANOVA and LSD test is applied to see the effect of the magnitude of machines on the quality of the algorithms. The results are illustrated in Figure 5. Figure 5 shows that HMH has good performance compared to those of IACO and SA.

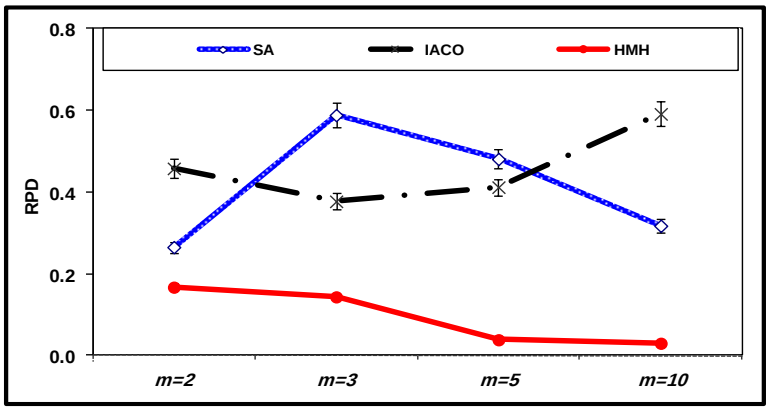

Fig. 5. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and number of parallel machine

5.3.3. Analysis of width of jobs factor

Another two-way ANOVA and LSD test is applied to see the effect of the width of jobs on the quality of the algorithms. The results are illustrated in Figure 6. You can see that this factor has no significant effect on the performance of HMH. Also, in all cases, i.e., w=2 and w=(1,3), HMH works better than other algorithms

.

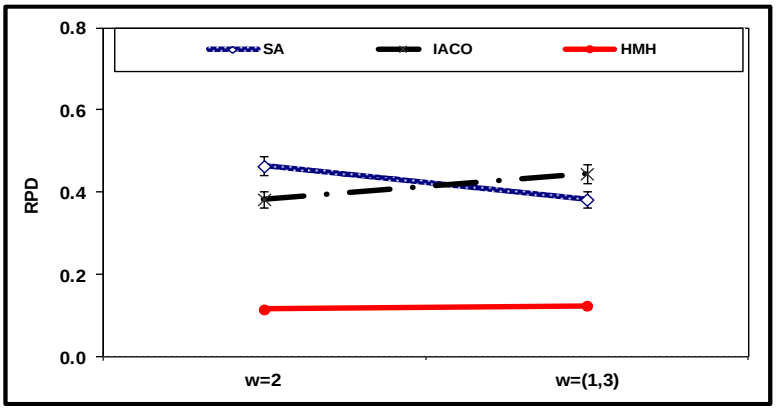

Fig. 6. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and width of jobs

6. Conclusions and Future Work

In this paper, we investigated the identical parallel machines scheduling problems in which the jobs may use more than one machine at the same time; we desired to minimize the sum of the tardiness of jobs. The problem involves job set with specific due date and it is assumed that the tardiness penalty will occur if the job is completed after its due date.

In this research, to deal with parallel job scheduling, first, after converting scheduling to the graph model, we make use of the semi-definite programming relaxation method for coloring it. In this phase, jobs are grouped in the independent sets according to their requirement of machines. The core of our method is the novel idea of using the non-negative matrix instead of variable in linear programming. Then, a VNS-based metaheuristic was proposed for sequencing the job sets. In the proposed algorithm, the balance between the global exploration and the local exploitation was stressed. Experiments demonstrated that our proposed algorithm generates an efficient set of solutions. The future work is to change the objectives in this paper and develop an effective proposed algorithm for other types of scheduling problems.

References

Almeida, M.T., & Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers andOperations Research*, 25, 625-635.

Babbar, D., & Krueger, P. (1994). On-line hard real-time scheduling of parallel tasks on partitionable multiprocessors. *In Proceedings of International Conference on Parallel Processing (ICPP'94),* pages II–29–II–38. IEEE Computer Society, Los Alamitos, CA, USA.

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

Birman, M., & Mosheiov, G. (2004). A note on a due-date assignment on a two-machine flow-shop. *Computers and Operations Research*, 31, 473–480.

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

Bougeret, M., Dutot, P-F., Trystram, D., Jansen, K., & Robenek, C. (2015). Improved approximation algorithms for scheduling parallel jobs on identical clusters, *Theoretical Computer Science*, 600(4), 70-85.

Brelsford, D., Chochia, G., Falk, N., Marthi, K., Sure, R., Bobroff, N., Fong, L., & Seelam, S. (2013). Partitioned parallel job scheduling for extreme scale computing, In: Job Scheduling Strategies for Parallel Processing, *Lecture Notes in Computer Science*, 7698, 157-177.

Bülbül, K., Kaminsky, P., & Yano, C. (2007). Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling*, 10, 271–292.

Chen, B., & Vestjens, A.P.A. (1997). Scheduling on identical machines: How good is LPT in an on-line setting?. *Operations Research Letters*, 21, 165–169.

Chen, Z.L. (1996). Scheduling and common due date assignment with earliness-tardiness penalties and batch delivery costs. *European Journal of Operational Research*, 93, 49-60.

Chen, Z.L., & Powell, W.B. (1999). A column generation based decomposition algorithm for a parallel machines just-in-time scheduling problem. *European Journal of Operational Research*, 116, 221-233.

Cheng, B., Wang, Q., Yang, S., & Hu, X. (2013). An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes, *Applied Soft Computing*, 13, 765–772.

Damodaran, P., & Vélez-Gallego, M.C. (2012). A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times, *Expert Systems with Applications*, 39, 1451–1458.

Damodaran, P., & Vélez-Gallego, M.C. (2012). A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times, *Expert Systems with Applications*, 39, 1451–1458.

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

Deng, X., Gu, N., Brecht, T., & Lu, K. (2000). Preemptive scheduling of parallel jobs on multiprocessors. *SIAM Journal on Computing*, 30(1), 145–160.

Drozdowski, M. (1996). Real-time scheduling of linear speedup parallel tasks. *Information Processing Letters*, 57(1), 35–40.

Du, J., & Leung, J. (1989). Complexity of scheduling parallel job system. *SIAM Journal on Discrete Mathematics*, 2, 473–487.

Dutot, P.F., Mounié, G., & Trystram, D. (2004). Scheduling parallel tasks: Approximation algorithms. In J.Y. Leung, editor, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, pages 26.1–26.24. *CRC Press*, Boca Raton.

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.

Emmons, H. (1987). Scheduling to a common due-date on parallel uniform processors. *Naval Research Logistics Quarterly*, 34, 803–810.

Esteve, B., Aubijoux, C., Chartier, A., & Tkindt, V. (2006). A recovering beam search algorithm for the single machine just-in-time scheduling problem. *European Journal of Operational Research*, 172, 798–813.

Feitelson, D.G., & Rudolph, L. (1998). Metrics and benchmarking for parallel job scheduling. In D.G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing. LNCS*, volume 1459, pages 1–24. Springer, Berlin.

Feldmann, A., Kao, M.-Y., Sgall, J., & Teng, S.-H. (1998). Optimal online scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4), 393–411.

Feldmann, A., Sgall J., & Teng, S.H. (1994). Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130(1), 49–72.

Frachtenberg, E., Feitelson, D.G., Petrini, F., & Fernandez, J. (2005). Adaptive parallel job scheduling with flexible coscheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(11), 1066–1077.

Glasgow, J., & Shachnai, H. (1997). Channel based scheduling of parallelizable tasks. *In Proceedings of 5th IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'97)*, pages 11–16. IEEE Computer Society, Los Alamitos, CA, USA.

Gordon, V., Proth, J.M., & Chu, C. (2002). A survey of the state-of-the-art of common due-date assignment and scheduling research. European Journal of Operational Research, 135, 1–25.

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

Herrmann, J.W., & Lee., C.Y. (1993). On scheduling to minimize earliness-tardiness and batch delivery costs with a common due date. *European Journal of Operational Research*, 70, 272-288.

Hoogeveen, J.A. (2005). Multicriteria scheduling. *European Journal of Operational Research*, 167, 592–623.

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

Jansen, K., & Porkolab, L. (2000). Preemptive parallel task scheduling in o(n) + poly(m) time. In D.T. Lee and S.H. Teng, editors, *Proceedings of ISAAC 2000. LNCS*, volume 1969, pages 398–409, Springer, Berlin.

Jansen, K., & Porkolab, L. (2002). Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmic*a, 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.

Johannes, B. (2006). Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5), 433–452.

Karger, D., Motwani, R., & Sudan, M. (1998). Approximate graph coloring by semidefiniteprogramming. *J. ACM*, 45(2), 246–265.

Krishnamurti, R., & Gaur, D.R. (1999). An approximation algorithm for nonpreemptive scheduling on hypercube parallel task systems. *Information Processing Letters*, 72(5–6), 183–188.

Krishnamurti, R., & Narahari, B. (1995). An approximation algorithm for preemptive scheduling on parallel-task systems. *SIAM Journal on Discrete Mathematics*, 8(4), 661–669.

Kubiak, W., Lou, S., & Sethi, R. (1990). Equivalence of mean flow time problems and mean absolute deviation problems. *Operations Research Letters*, 9, 371–374.

Kwon, O.H., & Chwa, K.-Y. (1999). Scheduling parallel tasks with individual deadlines. *Theoretical Computer Science*, 215(1-2), 209–223.

Lauff, V., & Werner, F. (2004). Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40, 637–655.

Li, K. (1999a). Analysis of an approximation algorithm for scheduling independent parallel tasks. *Discrete Mathematics and Theoretical Computer Science*, 3(4), 155–166.

Li, K. (1999b). Analysis of the list scheduling algorithm for precedence constrained parallel tasks. *Journal of Combinatorial Optimization*, 3(1), 73–88.

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, P., & Liu, Z. (2008). A report on approximate graph coloring by semidefinite programming, Instructor Kartik,K. Sivaramakrishnan, Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University.

Low, C., & Yuling, Y. (2009). Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. *Robotics and Computer-Integrated Manufacturing*, 2(25), 314-322.

Ludwig, W., & Tiwari, P. (1994). Scheduling malleable and nonmalleable parallel jobs. *In Proceedings of the 5th ACM-SIAM symposium on discrete algorithms (SODA*). pp. 167–176.

Lushchakova, I.N. (2012). Preemptive scheduling of two uniform parallel machines to minimize total tardiness, *European Journal of Operational Research*, 219(1), 27-33.

Montgomery, D.C. (2000. Design and Analysis of Experiments, Fifth ed. New York: *John Wiley & Sons.*

Naroska, E., & Schwiegelshohn, U. (2002). On an on-line scheduling problem for parallel jobs. Information Processing Letters, 81, 6, 297–304.

Pinedo, M.L. (2008). Scheduling: Theory, Algorithms, and Systems, Third Edition, *Springer* Science+Business Media, LLC, NY: USA.

Rapine, C.H., Scherson, I., & Trystram, D. (1998). On-line scheduling of parallelizable jobs. In D. Pritchard and J. Reeve, editors, *Proceedings of Euro-Par 1998. LNCS*, 1470, 322–327. Springer, Berlin.

Rocha, M., Gómez Ravetti, M., Mateus, G.R., & Pardalos, P.M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighborhood search. *IMA Journal of Management Mathematics*, 18, 101−115.

Sgall, J. (1996). Randomized on-line scheduling of parallel jobs. *Journal od Algorithms*, 21(1), 149– 175.

Shmoys, D., Wein, J., & Williamson, D. (1995). Scheduling parallel machines on-line, *SIAM Journal on Computing*, 24, 1313–1331.

Srinivasan, S., Subramani, V., Kettimuthu, R., Holenarsipur, P., & Sadayappan, P. (2002). Effective selection of partition sizes for moldable scheduling of parallel jobs. In S. Sahni, V.K. Prasanna, and U. Shukla, editors, *Proceedings of 9th International Conference on High Performance Computing (HiPC'02).* LNCS, 2552, 174–183.

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

Turek, J. Wolf, J.L. Pattipati, K.R., & Yu, P.S. (1992). Scheduling parallelizable tasks: Putting it all on the shelf. ACM *SIGMETRICS Performance Evaluation Review*, 20, 1, 225–236.

Turek, J., Ludwig, W., Wolf, J.L., Fleischer, L., Tiwari, P., Glasgow, J., Schwiegelshohn, U., & Yu, P. S. (1994). Scheduling parallelizable tasks to minimize average response time. *In Proceedings of 6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'94),* pages 200–209. ACM, New York, NY, USA.

Turek, J., Schwiegelshohn, U., Wolf, J.L., & Yu, P.S. (1994). Scheduling parallel tasks to minimize average response time. *In Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete algorithms (SODA'94),* pages 112–121. SIAM, Philadelphia PA, USA.

Turek, J., Wolf, J. L., & Yu, P. S. (1992). Approximate algorithms for scheduling parallelizable tasks. *In Proceedings of 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92),* pages 323–332. ACM, New York, NY, USA.

Wang, Q., & Cheng, K.-H. (1991). List scheduling of parallel tasks. *Information Processing Letters*, 37(5), 291–297.

Wang, Q., & Cheng, K.-H. (1992). A heuristic of scheduling parallel tasks and its analysis. *SIAM Journal on Computing*, 21(2), 281–294.

Ye, D., & Zhang, G. (2003). On-line scheduling of parallel jobs with dependencies on 2-dimensional mesh. In T. Ibaraki, N. Katoh, and H. Ono, editors, *Proceedings of 14th International Symposium Algorithms and Computation (ISAAC'03). LNCS*, 2906, 329–338.

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

This article can be cited: Behnamian, J. (2021). Parallel Jobs Scheduling with a Specific Due Date: a Semi-Definite Relaxation-Based Algorithm. *Journal of Optimization in Industrial Engineering*. 13 (2), 199-210.

http://www.qjie.ir/article_538024.html
DOI: 10.22094/JOIE.2017.599.1385